



HAL
open science

A Generic Tool-Supported Framework for Coupling Task Models and Interactive Applications

Célia Martinie, David Navarre, Philippe Palanque, Camille Fayollas

► **To cite this version:**

Célia Martinie, David Navarre, Philippe Palanque, Camille Fayollas. A Generic Tool-Supported Framework for Coupling Task Models and Interactive Applications. 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2015), ACM SIGCHI: Special Interest Group on Computer-Human Interaction, Jun 2015, Duisburg, Germany. pp.244-253, 10.1145/2774225.2774845 . hal-04079370

HAL Id: hal-04079370

<https://hal.science/hal-04079370v1>

Submitted on 24 Apr 2023

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.



Open Archive TOULOUSE Archive Ouverte (OATAO)

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible.

This is an author-deposited version published in : <http://oatao.univ-toulouse.fr/>
Eprints ID : 15321

The contribution was presented at EICS 2015:
<http://eics2015.org/>

To cite this version : Martinie De Almeida, Celia and Navarre, David and Palanque, Philippe and Fayollas, Camille *A Generic Tool-Supported Framework for Coupling Task Models and Interactive Applications*. (2015) In: 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2015), 23 June 2015 - 26 June 2015 (Duisburg, Germany).

Any correspondence concerning this service should be sent to the repository administrator: staff-oatao@listes-diff.inp-toulouse.fr

A Generic Tool-Supported Framework for Coupling Task Models and Interactive Applications

Célia Martinie, David Navarre, Philippe Palanque, Camille Fayollas

ICS-IRIT, University of Toulouse 3

118, route de Narbonne

F-31062, Toulouse, France

{lastname}@irit.fr

ABSTRACT

Task models are a very powerful artefact describing users' goals and users' activity and contain numerous information extremely useful for designing usable interactive application. Indeed, task models is one of the very few means for ensuring effectiveness of the application i.e. that the application allows users to reach their goals and perform their tasks. Despite those advantages, task models are usually perceived as a very expensive artefact to build that has to be thrown away as soon as the interactive application has been designed, i.e. right after the early stages of the design process are performed. However, tasks models can also be of great help for instance when used to support training material production, for training of operators and for providing tasks and goals oriented contextual help while the interactive application is being used ... This paper proposes a tool-supported framework for exploiting task models throughout the development process and even when the interactive application is deployed and used. To this end, we introduce a framework for connecting task models to an existing, executable, interactive application. The main contribution of the paper lies in the definition of a systematic correspondence between the user interface elements of the interactive application and the low level tasks in the task model. Depending on the fact that the code of the application is available or not, the fact that the application has been prepared at programming time for such integration or not, we propose different alternatives to perform such correspondence (in a tool-supported way). This task-application integration allows the exploitation of task models at run time bringing in the benefits listed above to any interactive application. The approach, the tools and the integration are presented on a case study of a Flight Control Unit (FCU) used in aircraft cockpits.

Author Keywords

Interactive systems, task models, co-execution.

INTRODUCTION

Task models are considered by many as a corner stone in the development process of usable interactive applications. Indeed, they provide a unique mean for gathering information about users' roles, goals and activities either being about an extant or envisioned system. However, at the same time, task models are also considered as cumbersome, expensive to build and mainly useful in the early phases of the development process. When used throughout the development process as well as at operation time (when the system has been deployed and is currently used) task models bring many benefits such as:

- Support the assessment of the effectiveness factor of usability by identifying which tasks are supported by the interactive application and which ones are not;
- Support the assessment of the task complexity in terms of perception, analysis, decision and motor action of users in order to reach a goal [6], assessment of operators' performance to reach a goal [25] which can lead to predictive workload assessment [18];
- Support the construction of training material and training sessions of operators of complex systems [10];
- Support the structuring and the construction of user documentation [8];
- Support the heuristic evaluation of usability of interactive applications (better than when task models are not used) not only for single user applications [5] but also for multi user applications [24];
- Support the identification of user errors and their impact on the overall performance for reaching the goals [21] as well as preventing those user errors [22];
- Support the identification of tasks that are good candidate for migration towards an automation of the system [13] but also towards other users in the context of collaboration [28];
- Makes it possible to provide users with contextual help i.e. explicit information about how (which tasks to perform) to reach the goal both at design time [17] and from the current state of interaction while interacting with the system [19];
- Support the redesign of the extant system by analysis of extant task models and producing task models for the future system as promoted in ADEPT framework [26].

It is important to note that most of the potential benefits listed above require that the interactive application and the information represented in the task model are compatible. Such compatibility can be seen at lexical (for each interface

object corresponds a low level task in the task model and reciprocally), syntactic (the task structure and temporal operators are conformant with the availability of interface objects i.e. compatible with the dialogue of the UI) and semantic levels (the interactive application allows users to reach the goals identified in the task model) [20].

This paper argues that it is possible to **ensure compatibility between a task model and an interactive application**. Beyond previous work in which compatibility was ensured by generation of the application from the task model (as promoted in [26]) or by “connecting” a model of the application with the task model (as promoted in [20] and [1]). The proposed contribution is wider in the sense that it allows coupling task models with any existing interactive application (without having a model of it). By this coupling the paper provides support for taking advantage of the benefits of task models based approaches (listed above), even without following a model-based development of the interactive application.

The remainder of the paper is structured as follows. Next section provides a quick overview of related work on compatibility aspects between task models and interactive applications. Section III presents a stepwise process for extracting information about an interactive application and for connecting it to a task model to ensure consistency or to identify gaps. Section IV presents a co-execution environment making it possible to embed task models at execution time. Section V presents the application of the process and the use of the co-execution environment on a case study in the interactive cockpits of large civil aircraft domain. Last section summarizes the contributions of the paper, make explicit their benefits and limitations and highlights future work.

RELATED WORK ON TASK-APPLICATION COMPATIBILITY

There have been mainly two main ways of addressing the issue of compatibility between task models and interactive applications: by generation of the application from the task model and by defining a correspondence between a model (formal or not) of the application and the task model.

Generation of application from a task model

From the seminal work of ADEPT [27] and its design process based on tasks and domain models, many authors have followed and refined that approach as, for instance, in the CAMELEON framework [4]. The basic assumptions for these approaches is that task models are the preliminary source of information and that it is possible to generate an interactive application from such information (while adding other ingredients such as UI guidelines for instance). The main claim is that with such an approach it is possible to generate user interfaces for different platforms thus reducing the development costs. The main drawbacks are that it is difficult to integrate design and craft knowledge in such processes ending up with stereotyped user interfaces far away (in terms of design and interaction techniques) from leading edge applications.

Correspondence at models level

Another approach promoted was to perform integration at with a (possibly formal) model of the interactive system. Such an approach has been firstly introduced in [20]. However, such an approach requires a description technique able to encompass all the elements of the interactive application including input device information, interaction techniques as well as the non-interactive part (functional core) of the application. Another drawback is the high development costs for the construction of the application and interaction models limiting usually its use to safety critical applications. Besides, such approaches are very different from current processes in interactive application development where Rapid Application Developments toolkits (RAD) are common practice. Lastly, task models and system models must be consistent. It is another cumbersome task to perform as presented in [15] where such compatibility was assessed through scenarios extracted from the task models and executed on the system model.

Issue of having the task model at run time

An orthogonal issue with respect to the two approaches presented above, lies in the fact that some of the advantages (such as contextual help) can only be available if the task models are embedded during the use of the application. Embedding task models at run time while executing the models of an interactive application was proposed in [1] where the benefits were made available but again at the costs of building an interactive application via the construction of system models. This capability has also been used to modify the system behavior at run time. In [3], the current state of execution of task models are used to modify user interfaces distribution over several devices. At last, automated testing of GUIs has been proposed in [16] but is based on test cases (sequences of low level GUI events) and not on task models.

A SYSTEMATIC AND EXPLICIT APPROACH TO ENSURE CONSISTENCY BETWEEN USER TASKS AND INTERACTIVE APPLICATIONS

The proposed approach overcomes (as much as possible) the limitations identified above while keeping the potential benefits. In order to reach this goal, our approach is made of:

- A process based on the synergistic use of task models and interactive application.
- A technique for instrumenting an existing application in order to be able to co-execute it with task models at run time.

Proposed process

The process takes as input an existing interactive system and its software interactive applications. This process, illustrated in Figure 1, starts with 2 phases that can be lead concurrently: task modelling and instrumentation of the interactive application. The task analysis and modelling phase consists in understanding user's tasks with the interactive system and in describing them in task models. The particularity of this

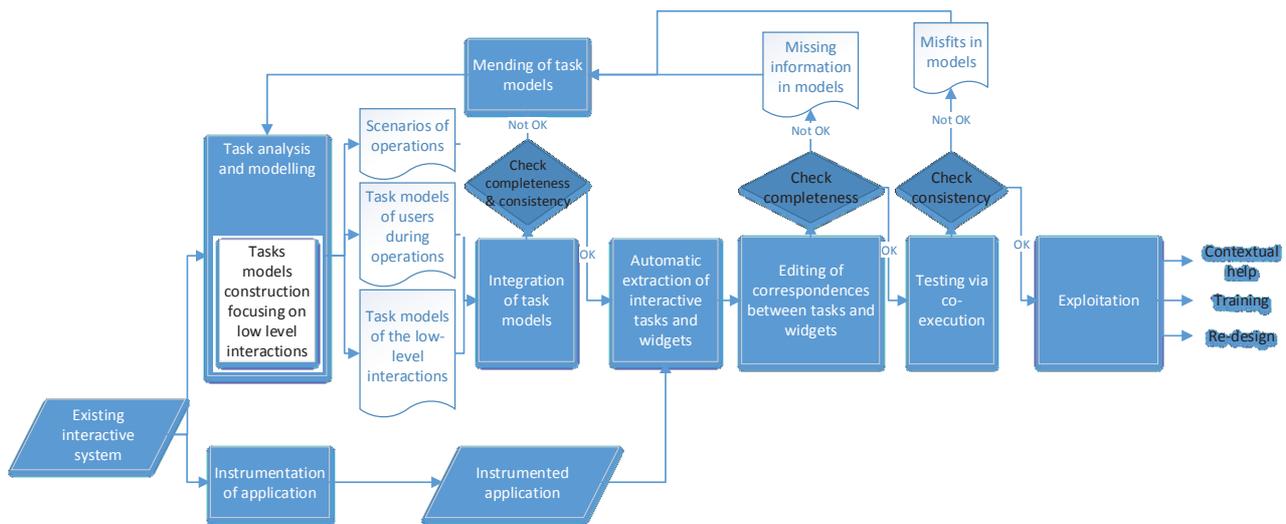


Figure 1. Process for validating the effectiveness of an interactive application

phase is that task models of the low-level interactions with the applications have to be produced, as these descriptions of the interactive tasks will be put in correspondence with the interactive application. The amount of tasks to be performed can be very important and thus there can be several models which describe tasks at various abstraction level. All these models have to be linked between each other and these phase is depicted as the “integration of task models” phase in Figure 1. In this phase the models are simulated in order to check their completeness and consistency with regards to operational scenarios.

On the interactive application side, the software is instrumented to enable the connection with task models. Then, a synergistic module automatically extracts widgets from the interactive application and interactive tasks from the task models. Interactive tasks are then put in correspondence with the widgets and their associated events (phase “Editing correspondences between tasks and widgets” in Figure 1). When trying to link interactive tasks to widgets and events, if tasks are not matching widgets or events, incompleteness is detected and the task models have to be mend (return loop in Figure 1). Once the task models are complete, the instrumented interactive application can be co-executed with the task models. During this co-execution, inconsistencies can be detected between task models and interactive application, for example if a widget is described to be used instead of another which should really be used. In this case, the task models have to be corrected (return loop from “testing with co-execution” box in Figure 1).

The required framework to support such correspondence edition and to support co-execution of the interactive application with task models is discussed in the next section.

Once the task models are complete and consistent with the instrumented interactive application, the co-execution framework can be exploited in order to provide support for training programs [10], contextual help [19] and user centered redesign. In the illustrative example, we describe

how this co-execution framework provides support for assessing effectiveness of interactive applications, which can be exploited for the re-design of an interactive application.

Approaches for co-execution of interactive application and task models

Co-execution of system models and task models can be used to provide support for the development of single user applications [1] and collaborative applications [12]. In this paper we propose a framework that enables the co-execution of an interactive application with its associated task models. For this purpose, a synergistic module, which consists of a correspondence editor and a co-execution controller (or simulation controller), has been built in order to be able to interface task models with an interactive application programmed with a textual programming language. Following the work proposed in [1] and [12], connecting task models and system models in a software development environment requires the following steps to be performed:

- On task modelling side:
 - A set of interactive tasks (both input and output) must be extracted from the tasks specification (as they represent actions performed by the user and feedback the system provides the user with).
 - The task models simulation environment notifies to the synergistic module the evolution of the scenario under construction using a dedicated API (sending data from the simulator).
- On system modelling side:
 - The activation and rendering functions have to be extracted as they represent the system inputs and outputs. They contain information about the widgets, the list of related events (relevant for correspondence), the activation rendering methods for each couple of widget and event (i.e. how enabled/disabled widget statuses are rendered) and lastly the rendering methods (graphical representation of data within widgets).

- The system modelling execution environment executes the models and notifies to a synergistic module about any change that happens using a dedicated API.

In the proposed approach, these mechanisms remain the same on the task models side, but on the system side, working without models requires to build a co-execution framework with equivalent mechanisms, i.e.:

- For editing correspondences, the synergistic module of the framework needs to acquire what is provided by both activation and rendering functions: a list of widgets, the list of related events, the activation rendering methods and the rendering methods.
- For co-execution of the interactive application and associated task models, the synergistic module of the framework requires to be notified about any change: activation status of the widget (enabled/disabled), rendering changes. The synergistic module also has to be capable of raising events (for example, to force widgets to trigger a particular event).

Providing such features depends on how the application has been built, i.e. whether or not it has been setup for co-execution.

Instrumentation of an existing interactive application

When preparing the setup for co-execution, the effort required by the developer depends on the choice of technology or architecture made:

- A first possibility would be to ask developers to provide the activation and rendering functions as explained in the previous section, implementing a dedicated API. This solution requires knowledge about the synergistic module and modifies the structure of the interactive application.
- Another possibility is to ask developers to instrument the code of the existing application with means to allow the access of pertinent widgets of the application for both user inputs and system graphical outputs. Such a solution requires less effort from developers but implies to precisely define how required features would be exposed (widgets, events, rendering methods...). With Java¹ for instance (since jdk 1.5), “annotations” are special software code elements that can be used both at compilation and at run time in order to point out objects, attributes or methods, and providing access to them.

The option which consists in providing the developers with a library of widgets already prepared for the correspondence editing is particular. Such a solution introduces a bias as it requires the developer to work with this predefined set of widgets, and prevent them from using new widgets. If this last case occurs, the developer will have to provide extra code to use these new widgets within the framework (which corresponds to the first two possibilities presented).

Table 1 provides an overview of the possibilities that the application developer will have to face to be able to connect an interactive application to task models.

Table 1. Work to be done on the interactive application developer's side

	Correspondence editing	System driven simulation	Task driven simulation
Dedicated API	Provide: <ul style="list-style-type: none"> . a list of widgets . a list of events . a list rendering events 	<ul style="list-style-type: none"> . Notify user events occurrence . Notify rendering events 	<ul style="list-style-type: none"> . Allow trigger user events . Notify rendering events
Code instrumentation	Provide access to widgets: <ul style="list-style-type: none"> . dedicated to user inputs . dedicated to system graphical outputs 	Nothing (if notification mechanism is embedded within the widgets)	Nothing (if triggering event mechanism is embedded within the widgets)
Resource introspection	No work to be done	No work to be done	No work to be done
Runtime environment introspection	No work to be done	No work to be done	No work to be done
Modification of runtime environment	No work to be done	No work to be done	No work to be done

Design and development of the synergistic module

If the application is not instrumented for co-execution with task models, the effort will have to be assumed by developers of the correspondence module. The complexity of such approach thus depends on the technology used to develop the application:

- The used technology may provide mechanisms (at design time or at runtime) to discover interactive widgets and their features. This could be resource files providing the widgets and their layout (such as in Microsoft Visual Studio, or with Java FX fxml files, Qt qml files...) or mechanism provided by the runtime environment such as with Java Swing application (at runtime it is possible to explore the widget tree of any Java Swing application).

Another possibility could be to modify the runtime environment and to enhance widgets with dedicated mechanisms (as described in previous section). A good example of such possibility is the architecture of Java Swing, where it is possible to change the Look&Feel manager at runtime of any application even if it has not been set up for (for each Swing component, the Look&Feel manager defines both the rendering and how user events are handled). But for these two aforementioned possibilities, there are two important limitations:

- Widgets that are not predefined within the framework but are used for the application cannot be easily used for the correspondence editing as they would be difficult to detect and difficult to analyze (to provide list of produced events

¹ <http://www.oracle.com/technetwork/java/javase/overview/>

Table 2. Work to be done on the synergistic module developer's side

		Correspondence editing	System driven simulation	Task driven simulation	
Prepared for correspondence and co-execution	Dedicated API	Nothing (except editing correspondence itself)	Nothing	Nothing	
	Code instrumentation	Translate application enhancements into compatible features (done once per feature)	Adapt notification mechanisms if they exist or create a pull mechanism (done once per feature)	Adapt notification and firing mechanisms if they exist or create a pull mechanism (done once per feature)	
Not prepared for correspondence and co-execution	Resource introspection	<ul style="list-style-type: none"> . Parse resource files. . Foreach known widgets prepare list of features for correspondence editing 	Adapt notification mechanisms if they exist or create a pull mechanism (done once per feature)	Adapt notification and firing mechanisms if they exist or create a pull mechanism (done once per feature)	<ul style="list-style-type: none"> . Does not work for unknown widgets. . Difficult to do if widgets are dynamically instantiated
	Runtime environment introspection	<ul style="list-style-type: none"> . Get and explore widgets tree. . Foreach known widgets prepare list of features for correspondence editing. . Requires means to graphically identify widgets to build the correspondence. 	Adapt notification mechanisms if they exist or create a pull mechanism (done once per feature)	Adapt notification and firing mechanisms if they exist or create a pull mechanism (done once per feature)	
	Modification of runtime environment	<ul style="list-style-type: none"> . Modify the runtime environment making possible to retrieve a widget list. . Foreach known widgets prepare list of features for correspondence editing. . Requires means to graphically identify widgets to build the correspondence. 	Adapt notification mechanisms if they exist or create a pull mechanism (done once per feature)	Adapt notification and firing mechanisms if they exist or create a pull mechanism (done once per feature)	

for instance). Fixing this point thus depends on the used technology.

- Widgets that appear at run time cannot be detected at boot strap.

In any case (except when a dedicated API for connection has been implemented in the application), a set of functionalities have to be developed in the synergistic module in order to be able to adapt what is provided by the interactive application and to connect it to task models. Table 2 provides an overview of the work that has to be done by the synergistic module developer depending on the way the interactive application has been prepared for correspondence and co-execution. In this article, we present a solution based on the code instrumentation as it: a) provides full support for connecting an interactive application with its associated task models; b) provides support for modularity of the framework; c) does not modify the architecture of the interactive application; d) is the more accurate with regards to recent advances in software engineering.

AN INTEGRATED ENVIRONMENT SUPPORTING THE CO-EXECUTION OF TASKS MODELS AND JAVA APPLICATIONS

The proposed process for validating the effectiveness of an interactive application presented in the previous section is supported by a modelling and simulation CASE tool for engineering user tasks and interactive applications.

Task modeling with HAMSTERS

Task models support gathering and structuring data from the analysis of users’ activities, and recording, refining and analyzing information about users’ activities. Several notations are available to describe tasks with varying expressiveness levels depending on targeted analysis. HAMSTERS (Human – centered Assessment and Modeling to Support Task Engineering for Resilient Systems) is a tool-supported graphical task modeling notation for representing

human activities in a hierarchical and ordered way. At the higher abstraction level, goals can be decomposed into sub-goals, which can in turn be decomposed into activities. Output of this decomposition is a graphical tree of nodes. Nodes can be tasks or temporal operators.

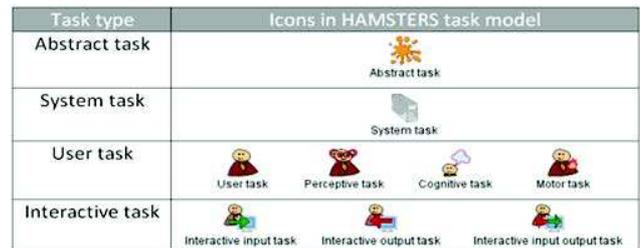


Figure 2. High-level Task Types in HAMSTERS

Tasks can be of several types (Figure 2) and contain information such as a name, information details, and critical level. Only the single user high-level task types are presented here but they are further refined. For instance the cognitive tasks can be refined in Analysis and Decision tasks [13] and collaborative activities can be refined in several task types [12]. Temporal operators are used to represent temporal relationships between sub-goals and between activities [12]. Tasks can also be tagged by temporal properties to indicate whether or not they are iterative, optional or both. HAMSTERS’ notation and tool provide support for task-system integration at the tool level [10] by:

- Structuring a large number and complex set of tasks introducing the mechanism of subroutines [11] and generic components [7]. These structuring mechanisms enables the breakdown of a task model in several ones. Subroutines are similar to functions or procedures. Generic components offer a set of properties and a set of functions and can be instantiated several times in several task models.

- Describing data that is required and manipulated [14] in order to accomplish tasks. Information (“I:” followed by a text box in Figure 8) may be required for execution of a system task, but it also may be required by the user to accomplish a task. The notation element “Object” (“O:” followed by a text box) is dedicated to the description of a software data object that is required by the system in order to accomplish a task. The notation element “Software Application” (“Sw A:” followed by a text box in Figure 8) provides support for describing a software application that is required in order to accomplish a task, and the notation element “Input/Output device” for an input and/or output device that is required (depicted in Figure 8).

Interactive application instrumentation and annotations processing

As stated above, we present an approach where developers of the interactive application instrument their code with annotations so that the interactive application can be integrated within the synergistic framework. Our framework and the interactive application presented in the case study are fully implemented using the Java technology. In particular, we propose a solution based on the Java type annotation mechanism (since JSE 1.8). Annotation in Java is a form of metadata (prefixed with a ‘@’), providing data about a program that is not part of the program itself. Annotations have no direct effect on the code instructions that they annotate but they are used by the compiler to detect errors. At compilation time, they provide information to generate the code. At run time they provide support for examining the annotated code. We use this last possibility in our framework. There are a lot of possible usage of such annotations:

- For instance, the following annotation may be used at run time to periodically schedule a method call.

```
@Schedule(dayOfWeek="Fri", hour="23")
public void doPeriodicCleanup() { ... }
```

- The following one is used at compilation time to detect a possible assignment to a null value (and raise an compilation error if so)

```
@NonNull String str;
```

The approach proposed is based on such annotations to point out within the code widgets that may be used to build the correspondence. There are two kind of annotations: one for widgets related to user inputs and the other one for graphical outputs from the system, any widget being possibly of the two kinds:

- Example of a simple button

```
@EventSource(name="Validate", event="actionPerformed")
private JButton btn1;
```

- Example of a label

```
@Renderer(name="Display", property="text")
private JLabel lbl1;
```

- Example of a text field

```
@EventSource(name="Name", event="actionPerformed")
@Renderer(name="Name", property="text")
private JTextField txt1;
```

Both types of annotation may be enhanced with extra data to provide information for the correspondence editing. In the two cases it provides a readable name for the correspondence (independent from the attribute name) and another information (the name of the related event or the name of the widget property that may change while using the application).

When editing the correspondence between the interactive application and task models, the synergistic framework uses the introspection mechanism of Java to find any annotated widget, building a list of widgets available for editing.

When co-executing the application and the task model, using the same introspection mechanism, the framework create at run time all necessary listeners or adapters required to trigger widget events.

Architecture of the framework for coupling task models and instrumented interactive application

The architecture of the synergistic software environment that supports both correspondence editing and co-execution is presented in Figure 3.

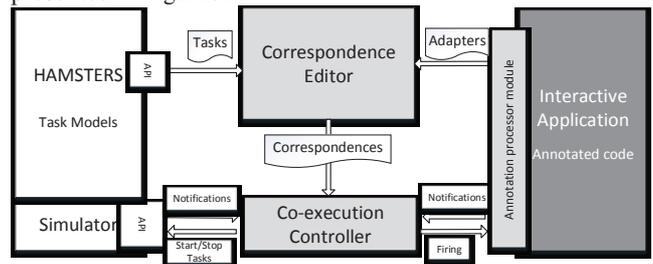


Figure 3. Architecture of the synergistic environment

This architecture is composed of: two modules for task edition and simulation (on the left side), the interactive application (on the right side) and of modules for connecting and co-executing task models with interactive application (grey shaded in the center in Figure 3). The colors of the architecture elements make explicit in light grey elements that have been either added or deeply redesigned to extend co-execution to applications not designed following a model-based approach [1]. The module in charge of interfacing the interactive application with the correspondence editor and simulation controller is called the “Annotation processor module”. This module is responsible of processing the annotations and of providing information about widgets, events, rendering to the correspondence editor and notifications to the simulation controller (as described in section “Approaches for co-execution of interactive application and task models”). The main differences with the architecture presented in [1] are that: the architecture presented here is not exclusively model-based; the information about widgets and events is provided by the annotation processor module instead of the PetShop CASE tool.

ILLUSTRATIVE EXAMPLE FROM THE FCU BACK UP CASE STUDY

The presented example has been extracted from a case study in the avionics application domain. In interactive cockpits, the Flight Control Unit (FCU) is a hardware panel composed of several electronic devices (such as buttons, knobs, displays...). It allows crew members to interact with the Auto-Pilot and to configure flying and navigation displays.

Presentation of the FCU Backup

The FCU Backup is an interactive application designed for the recovering of all FCU functions in case of FCU failure and is used in exclusion with the FCU. It is composed of two interactive pages:

- EFIS_CP: Electronic Flight Information System Control Panel for configuring piloting and navigation displays.
- AFS_CP: Auto Flight System Control Panel for the setting of the autopilot state and parameters.



Figure 4. EFIS control panel (w/o WPT button activated)

For example, this application is displayed on two of the eight cockpit LCD screens (in the Airbus A380), one for the Captain and the other for the First Officer. 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.

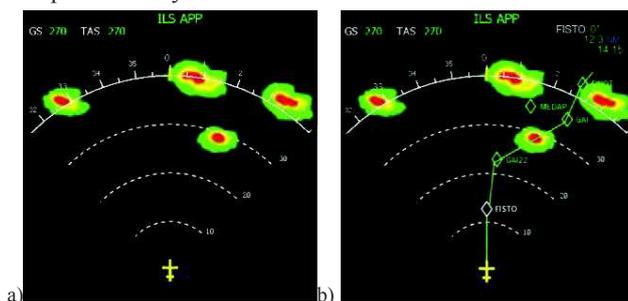


Figure 5. Navigation display (w/o waypoints displayed)

In this paper, we will focus on the EFIS_CP page depicted in Figure 4. The left panel is dedicated to the configuration of the Primary Flight Display while the two right panels are dedicated to the configuration of the Navigation Display (illustrated in Figure 5). The top right panel of the EFIS page enables the display of several navigation information while the bottom panel of the EFIS page enables to choose the display mode and scale. In this illustrative example, the focus

is set on the activities that have to be led in order to insert waypoints in the current route.

Inserting a waypoint in the current route with FCU Backup application

During the flight, crew members may ask permission or be asked to modify the current route of the aircraft. If the air traffic controller agrees, a clearance is ordered to the pilot to insert a waypoint. Figure 8a) details the tasks that have to be performed to reach this goal. First, thanks to the VHF radio device, the crew member receives a clearance from the air traffic controller indicating which waypoint has to be added in the current route (interactive output task “Receive a clearance from ATM for inserting a waypoint in flight plan” and perceptive task “Perceive waypoint insertion request” in Figure 8a)). Then, s/he decides to check the waypoints (cognitive decision task “Decide to check waypoints”) and has to perform a set of activities to display waypoints on the navigation display (abstract task “Display waypoints on ND” in Figure 8a)). In order to display waypoints on the navigation display, the crew member has to display the EFIS page and to configure the display options. The subroutine “Configure ND display options” describes the tasks that have to be performed to configure the navigation display options. The part of this subroutine task model which is relevant to our illustrative example is depicted in Figure 7. It shows that the crew member can iteratively configure the display of navigation information (iterative abstract task “Configure display of navigation information”) until (temporal ordering operator “disable”) s/he decides that the ND is setup correctly (cognitive task “Decide that ND setup if ok”). To configure the display of navigation information, the crew member has the choice between several actions to perform.

In order to configure the display of waypoints, s/he will press the WPT button widget (instantiated component “Press button widget to configure the display of [waypoints, WPT button] in Figure 7). Figure 8b) details the tasks that have to be performed to configure the display of waypoints. The crew member first has to locate the WPT button widget (user task “Locate <WPT button>” in Figure 8b)), then to perceive the current state of this button (perceptive task “Perceive current status of <WPT button>” in Figure 8b)). The button widgets of the EFIS page (illustrated in Figure 4) become green when they are pressed. Then, s/he decides that the current status of the button is the right one or not according to the targeted status (temporal ordering operator “choice”). If the crew member analyses that the current status of the button does not match the target status, s/he clicks on it (interactive input task “Click on <WPT> button” in Figure 8b)). The new status of the button is displayed (interactive output task “Display <WPT> button status” in Figure 8b)).

Correspondences and co-execution between task models and FCU Backup

Figure 6 presents one step of co-execution of the tool suite presented in this paper. The left part (resp. the right part) corresponds to the execution of the models and application before clicking the WPT button to display waypoints (resp. after clicking) using a task driven co-execution.

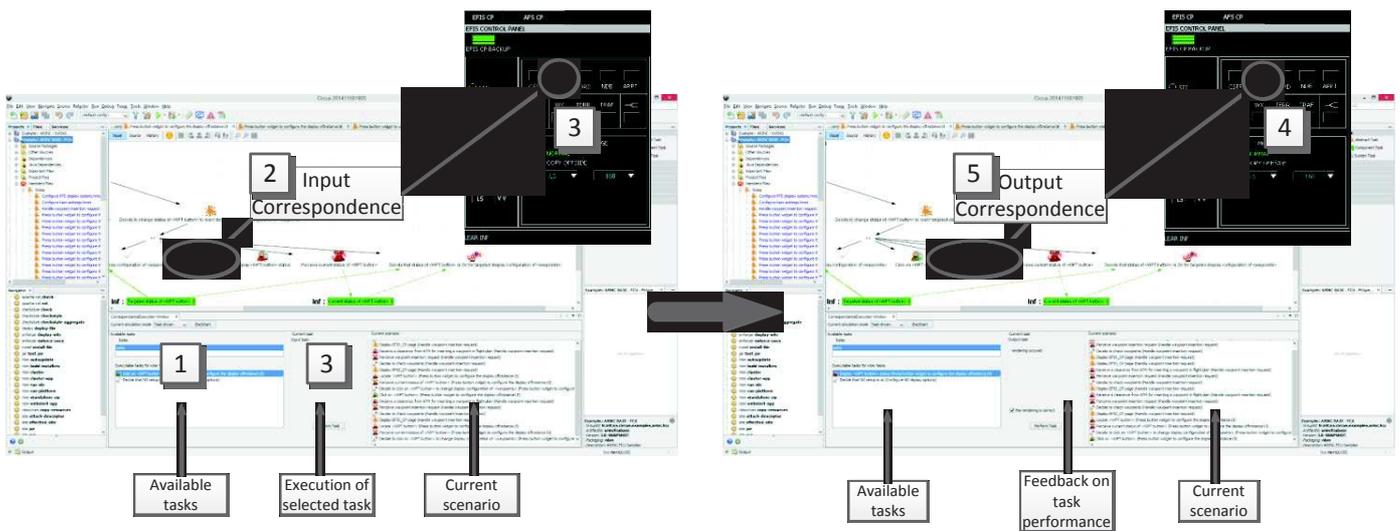


Figure 6. Illustration of a co-execution step between task models and FCU Backup application

A particular focus on the left part of the figure allows the understanding of the tool suite (which is a set of modules of the NetBeans IDE²). This tool may be divided into four parts:

- The top part is a set of classical IDE menu bars and tool bars buttons.
- The left part provides means to navigate amongst the project files (java sources, HAMSTERS and correspondence models).
- The right part shows properties of the sources or models (bottom part) and tools to modify the currently selected model (on the top part a specific toolbox appears depending on the kind models).
- The centre part allows the editing and execution control of both the sources and models. The layout of this part is fully reconfigurable as illustrated by the layout difference of the left and right part of the figure.
- To illustrate the synergistic exploitation of both task models and the FCU backup application, we use the models presented in the previous sections. As the correspondence editing between system and task models is a simple table editing already presented in [1], we only focus here on the task driven simulation.

Following the five numbered steps presented on the figure, the behaviour of the task driven simulation is as follows:

1. A set of available tasks is provided by the HAMSTERS environment that is selectable within the associated list box.
2. The selected task is connected to a widget event (the click on WPT button) by the correspondence editing.
3. Perform the task thus acts as if the user clicks on the corresponding button.
4. As if the button were clicked, a rendering may occur and this rendering may be related to an output HAMSTERS task, using the output correspondence edition (in our example, the button status changes to engaged and this property change is related to the task display <WPT button> status).
5. When an output task is selected, a dedicated panel appears at the bottom of the tool, showing whether a rendering occurs and if it corresponds to an output correspondence (panel “feedback on task performance”). In this panel it is possible to indicate if the rendering was effectively correct and perceive (for log purpose).

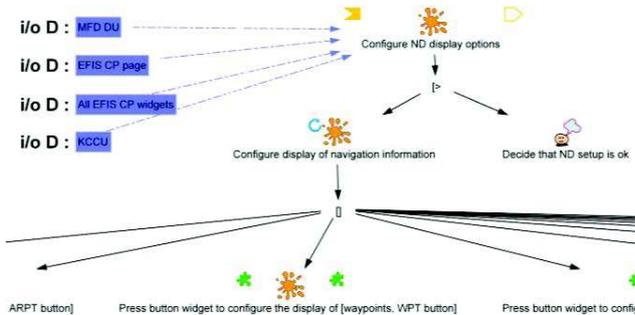


Figure 7. Subpart of “Config. ND display options” task model

Analysis of the effectiveness of the FCU Backup application with regards to crew members’ tasks

Thanks to the correspondence and co-execution, we have verified that the task models of the “Handle waypoint insertion request” activities are complete and fully consistent with regards to the FCU Backup interactive application. From the task models, we can analyse that the crew member has to move from one application to another in order to be able to modify the flight plan. S/he has to display the EFIS page to be able to configure the ND display options (“FCU Backup EFIS page” software application object in Figure 8a) and “FCU Backup FMS page” software application object in Figure 8a)).

² <https://netbeans.org/>

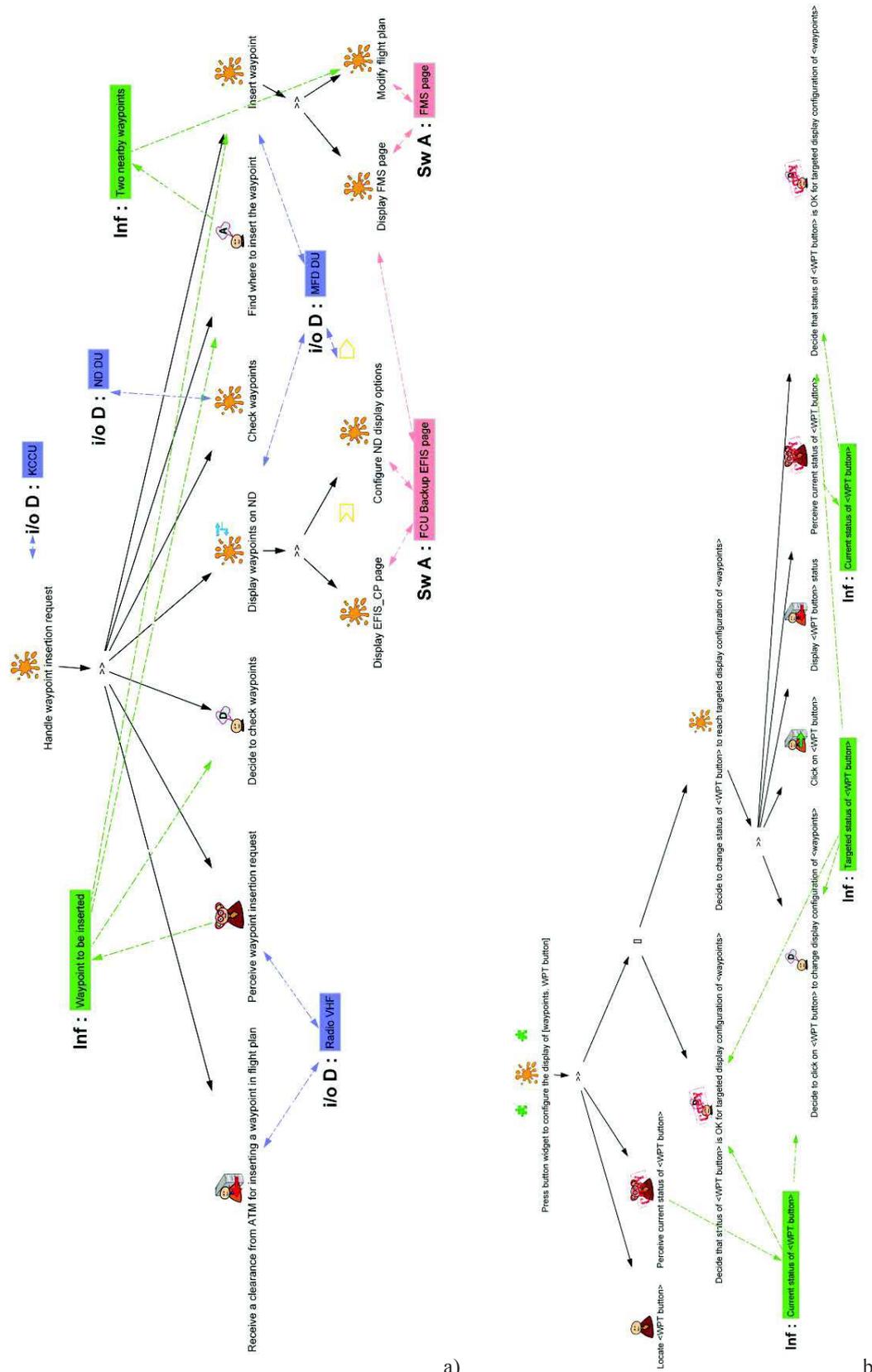


Figure 8. Task models for Handle waypoint insertion and Press button widget to configure display of waypoints

Thus, we have numbered 10 articulatory tasks that are only dealing with moving from one page to another (5 subtasks in the “Display EFIS_CP page” subroutine and 5 tasks in the “Display FMS page subroutine” in Figure 8a)).

Furthermore, this articulatory task load will also have to be added to the crew member’s activities as far as s/he will have to switch to another application in order to come back to her/his current activity. This analysis shows that the interactive application, as currently designed, reduces the crew members’ efficiency. One potential solution for re-design could be to integrate the Navigation Display panel with the FMS panel in one single interactive application.

CONCLUSION

The paper has proposed a tool-supported process for embedding task models in extant interactive application. The use of this process allows users to benefit from information available in the task model while interacting with the actual system. We have demonstrated that this process can be more or less time consuming depending on the application considered and whether or not it has been prepared for such an integration at development time. We believe such framework can be of great help for increasing the use of task models in interactive system development by providing benefits even for existing and already deployed applications. Of course, the usual benefits of using task models such as assessing work complexity, operators’ workload, identifying areas for improvement... are still present and even improved by possible storage of information about the actual use of the application under consideration in its real context. One limitation of the approach is that it is currently focusing on WIMP interactions (within the specific scope of JAVA SWING) but going to other platforms does not raise issues more difficult to solve than the ones already addressed. Another limitation is that going to more sophisticated interaction technique raises not trivial issues requiring extensions to task models providing more detailed representations of interaction as, for instance, in [9].

REFERENCES

- 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.
- Bernhaupt, R., Navarre, D., Palanque, P., Winckler, M. Model-Based Evaluation: A New Way to Support Usability Evaluation of Multimodal Interactive Applications. In *Maturing Usability: Quality in Software, Interaction and Quality*. Springer Verlag 2007, pp. 96-122.
- Blumendorf, M., Lehmann, G., Albayrak, S. Bridging models and systems at runtime to build adaptive user interfaces. In *Proc. of the EICS 2010*, pp. 9-18.
- Calvary G., Coutaz J., Thevenin D., Limbourg Q., Bouillon L., Vanderdonckt J. A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers* 15(3): 289-308 (2003)
- Cockton, G., & Woolrych, A. (2001). Understanding inspection methods: Lessons from an assessment of heuristic evaluation. *People and Computers*, Springer Verlag, pp. 171-192
- Fayollas C., Martinie C., Palanque P., Deleris Y., Fabre J-C., Navarre D: An Approach for Assessing the Impact of Dependability on Usability: Application to Interactive Cockpits. *EDCC 2014*: 198-209
- 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.
- Gong, R. & Elkerton, J. (1990). Designing minimal documentation using the GOMS model: A usability evaluation of an engineering approach. *CHI 90 Proceedings*. New York, ACM DL.
- Jourde F., Laurillau Y., Nigay L. COMM notation for specifying collaborative and multimodal interactive systems. *EICS 2010*: 125-134
- Martinie C., Palanque P., Navarre D., Winckler M. and Poupart E. Model-Based Training: An Approach Supporting Operability of Critical Interactive Systems: Application to Satellite Ground Segments, *Proc. of EICS 2011*, pp. 141-151, ACM DL.
- Martinie, C., Palanque, P., Winckler, M. Structuring and Composition Mechanism to Address Scalability Issues in Task Models. *Proceedings of the IFIP TC 13 INTERACT*, LNCS Springer Verlag, 2011.
- Martinie, C., Barboni, E., Navarre, D., Palanque, P., Fahssi, R., Poupart, E., Cubero-Castan, E. Multi-models-based engineering of collaborative systems: application to collision avoidance operations for spacecraft. *Proc. of EICS 2014*, pp. 85-94.
- Martinie C., Palanque P., Barboni E., Ragosta M. Task-Model Based Assessment of Automation Levels: Application to Space Ground Segments. *Proc. of the IEEE SMC*, Anchorage, 2011.
- Martinie C., Palanque, P., Ragosta, M., Fahssi, R. Extending Procedural Task Models by Explicit and Systematic Integration of Objects, Knowledge and Information. *Proc. of Europ. Conf. on Cognitive Ergonomics*, pp. 23-33.
- Navarre D., Palanque P., Paternò F., Santoro C., Bastide R: A Tool Suite for Integrating Task and System Models through Scenarios. *DSV-IS 2001*: 88-113.
- Nguyen, B., Robbins, B., Banerjee, I., Memon, A. GUITAR: an innovative tool for automated testing of GUI-driven software. *Automated Software Engineering*, March 2014, vol. 21 (1), pp 65-105.
- Pangoli S., Paternò F. Automatic Generation of Task-Oriented Help. *ACM Symposium on UIST 1995*: 181-187
- O'Donnell, R. D.; Eggemeier, F. T. Workload Assessment Methodology; In *Handbook of Perception and Human Performance (Vol. II Cognitive Processes and Performance*, pp. 42-41 - 42-49). Wiley & Sons, 1986.
- Palanque P., Martinie, C. Contextual Help for Supporting Critical Systems' Operators: Application to Space Ground Segments Activity in Context Workshop, AAAI conference on Artificial Intelligence.
- Palanque P., Bastide R., Sengès V. Validating interactive system design through the verification of formal task and system models. *EHCI 1995*: 189-212
- Palanque P., Basnyat S: Task Patterns for Taking Into Account in an Efficient and Systematic Way Both Standard and Erroneous User Behaviours. *HESSD 2004*: 109-130
- Paternò F., Santoro C. Preventing user errors by systematic analysis of deviations from the system task model. *Int. J. Hum.-Comput. Stud.* 56(2): 225-245 (2002).
- Paternò, F., Santoro, C., Spano, L.D. MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact.* 16, 4, Article 19 (November 2009), 30 pages.
- Pinelle, D., Gutwin, C., and Greenberg, S. Task Analysis for Groupware Usability Evaluation: Modeling Shared-Workspace Tasks with the Mechanics of Collaboration. *ToCHI*, 2003, 10(4), 281-311.
- Swearngin A., Cohen M., John B.E., Bellamy R. Human performance regression testing. *IEEE international conference on Software Engineering, ICSE 2013*: 152-161
- Wilson S., Johnson P. Bridging the Generation Gap: From Work Tasks to User Interface Designs. *CADUI 1996*: 77-94
- Wilson S., Johnson P., Kelly C., Cunningham J. and Markopoulos P. Beyond hacking: A model based approach to user interface design, In *Proceedings of HCI'93*, 217-23, University Press, BCS HCI.
- van Welie, M., van der Veer, G.C. Groupware task analysis. *Handbook of Cognitive Task Design*, LEA, NJ (2003), pp. 447-476.