# Beyond Modelling: An Integrated Environment Supporting Co-Execution of Tasks and Systems Models

Eric Barboni, Jean-François Ladry, David Navarre, Philippe Palanque, Marco Winckler

# Beyond Modelling: An Integrated Environment Supporting Co-Execution of Tasks and Systems Models

**Eric Barboni, Jean-François Ladry, David Navarre, Philippe Palanque, Marco Winckler**
Institute of Research in Informatics of Toulouse (IRIT), University of Toulouse
{barboni, ladry, navarre, palanque, winckler}@irit.fr

**ABSTRACT**

This paper focuses on the articulations of task models and system models. Tasks models are meant to be used by human factor specialists whilst system models are supposed to be produced by software engineers. However, tasks models and systems models represent two different views on how users interacting with a computing system to reach a goal. This paper presents an integration framework aiming to take full advantage of task models and system models that have been developed initially in a separated manner and how these two views can be integrated at the model level and additionally at the tool level. The main contribution of the paper lies in the definition of such integration at the tool level to be used at runtime (while the user is operating the system). Indeed, thanks to this integration contextual help can be offered to the users supporting the construction of the mental bridge between what they have to do (defined in the tasks model) and what the interactive system allows (defined in the system model). The approach, the tools and the integration are presented on a case study of a Weather Radar System (WXR) embedded in aircraft cockpits.

**Author Keywords**
Task and systems models, models integration, tool support.

**INTRODUCTION**

In the Human-Computer Interaction (HCI) field, there is industrial and academic consensus on the importance and usefulness of building task and system models in the design and development process of interactive systems. Task models are required as they allow expressing users' goals and activities that should be performed to reach such goals. There are various notations (e.g. CTT [17, 18], MAD [22]) for modelling tasks, ranging from very abstract (i.e. declarative notations describing user knowledge and activities) to more concrete ones (i.e. procedural notations describing temporal relationships and the expected system feedback [5]). System models describe important aspects of the user interface such as the set of states the system can be in, the actions the system is able to perform, the events to which system is able to react and the state changes that occur when events or actions are performed. More generally, system models aims at helping designers to build the application before it is implemented. These two models can be embedded in the development process of interactive systems in a complementary way as they correspond to different views on the same world (one is centred on user behaviour, the other is centred on system's behaviour).

As tasks models and system models are needed to support the design of usable and reliable interactive systems, one should expect tools and techniques for checking whether if these models match. For example, both models should be cross-consistent, which means both descriptions refer to the same system human-computer system. This requires checking if, for each user action appearing in the system model, there is an actual counterpart in the task model, and each system output provided to the user has been represented and is expected by the user in the task model. This checking is particularly important when the models were built by different people and/or at different moment in time within the design and development process. Indeed, in real case studies it happens that sometimes the task models will be performed first while in other cases they might be built after the system model has already been constructed. In order to be able to support such a flexible design and development process, we need an approach which does not embed constraints on what is assumed to be available in which phase of the system design.

This paper presents such an approach and focuses on the possible articulations of task models and system models. We present how these two views can be integrated at the model level and additionally at the tool level. We focus on the latter that raises new challenging issues but also provides high benefits. Next section describes an overview of previous research work devoted to the integration of system models and task models. From that work we identify the requirements for articulating tasks and system models and we propose an approach fully supported by open source tools that meet them.

## STATE OF THE ART

Several tools have tried to deal with the integration of task models and system models. Mobi-D [19] and Trident [3] are the first generation of tools aiming at using information contained in models to support design and development of user interfaces. Whilst in Mobi-D it is possible to combine information contained in task and domain models to derive the user interface, in Trident abstract interaction description and guidelines are used to generate platform independent user interfaces. Much work has been also devoted to the generation of one model from another one such as in [8] where the authors generates the system model from a task model, or in [7] where the authors do the opposite. However, as discussed in [12], tasks models lack of much necessary information to reconstruct completely the system models. Some authors [6] [20] propose recreate the dialog part of the system models by compounding several models and applying transformation rules according to a Model-Driven Approach (MDA). Notwithstanding, without designer intervention transformation rules often lead to unrealistic descriptions of the user activity [23].

Whilst previous attempts focused on producing models, some authors investigated techniques for measuring tasks and system compatibility [21]. For example, in [16] authors check the compatibility of UAN tasks translated into Petri nets and system models described in Petri nets. In [15] it was presented the use of CTT for abstract task modelling and high level Petri nets for low-level task modelling. In that paper the low-level task model was used in order to evaluate the “complexity” of the tasks to be performed, by means of performance evaluation techniques available from Petri net theory. In order to provide a more synergistic integration of task models and system models, Palanque et al. [13] have introduced a method and a tool support for playing scenarios extracted from task models (based on CTT) into system models (based on the Interactive Cooperative Objects (ICOs)). That approach enables to check correspondence and completeness by means of concrete scenarios that are a kind of "lingua franca" to ensure actual correspondence between what has been described within the models and what has been described in the system model. The main drawback is that compatibility between task models and system models requires scenarios previously extracted from task models making that integration asynchronous and thus not exploitable while the system is in operation. In [9] the same authors envisioned that work so that the simulation of system was controlled by the on-the-fly execution of task models.

More recently, Blumendorf et al. [2] argued that a MDA framework could support co-executing of models, ensure consistency and bi-directional execution of models (i.e. changes in one model trigger changes in all counter models in the framework). However, the links and mappings necessary for executing task models and system models together are not fully described. So far, no tool supports cross-execution of models as proposed in [2].

## REQUIREMENTS FOR ARTICULATING TASK MODELS AND SYSTEM MODELS

The successful integration of task and system models relies on three main types of requirements:

1. Expressiveness power of the task models, including:

- The description of artefacts used to perform a task should be close to the representation of objects manipulated by the system;
- User tasks should include elements of the behaviour expected from the system; e.g. user providing an input to the system, requesting a feedback or any kind of system output, or both actions at the same time.
- Task models should be able to express both qualitative temporal relationships (e.g. task ordering) and quantitative temporal relationships (e.g. amount of time required to perform a task). These relationships are needed to describe time constraints applied during system execution;
- It must be possible to describe tasks models as unities that cooperate rather than monolithic models. This aspect would support a better mapping between tasks and different system’s modules.

2. Expressiveness power of the system models, including:

- The formal description of both the "input" aspects of the interaction (i.e. how user actions impact on the inner state of the application, and which actions are enabled at any given time) and its "output" aspects (i.e. when and how the application displays information relevant to the user)
- It should be able to represent all aspects of interactive systems (dynamic instantiation, multimodal, collaborative …)
- Models produced should be executable, so that system models can be used as a prototype of the expected final application.

3. Availability of tool support that should:

- Be open source, or at least provide a powerful API for services enabling to control the models; this is critical to make it possible for the research community to contribute, extend and re-use such tools;
- Provide visual feedback on the current execution; this is important too in order to support the assessment of the adequacy of the task model with users’ real tasks. Indeed, without tool support for simulation it is very cumbersome to understand how the model behaves;
- Supports simulation of scenarios which supports the compatibility assessment activities;
- Implement an API observed/observer of events. This is mandatory for connection task modelling tools and system modelling tools.

Many formal notations have been proposed for modelling interactive systems [4], however only a few can represent

fine-grained system behaviour which is a requirement allowing cross-execution of system and tasks models. ICO [11, 12] is such as a formal description technique that fulfils all the requirements related to system models requirements. Moreover, it is provided with an open source development environment called PetShop [1] that covers the requirements identified earlier for the integration with task models.

Current task modelling techniques partially fulfils the above requirements. Moreover most of the current available tools are hardly extensible. We thus have proposed and defined the HAMSTER notation which is briefly introduced hereafter. The implementation of HAMSTERS was done with the objective of making it easily extendable and it results in a CASE tool that contributes to the engineering of task models. In a nutshell, HAMSTERS is open source, featuring a task simulator and provides a dedicated API for observing editing and simulation events.

### The HAMSTERS task modelling technique

HAMSTERS is heavily inspired by existing notations and tools, including concepts such as abstract, system, user and interactive tasks (see Figure 1). Notwithstanding it makes explicit which tasks requests user input and/or system output.

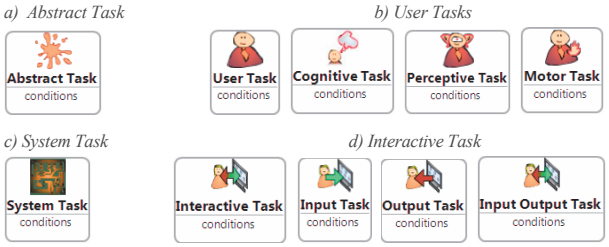



**Figure 1. Illustration of the task type within HAMSTERS**

HAMSTERS offers two types of relationships between tasks: the first one describes how tasks are related to other tasks and the second one represents the information flow between tasks. Objects (defined through a set of attributes) can be attached to tasks through relationships (as illustrated by Figure 2 where the PIN entered in the first task is conveyed to the next task by means of input and output ports).

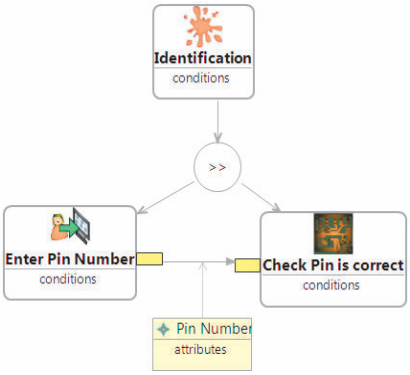



**Figure 2. Tasks relationships in HAMSTERS**

Similarly to MAD [22] and CTT [17, 18], qualitative time is expressed using Lotos-like temporal operators while quantitative time is represented by expressing task duration (such as with CTT) and delay before tasks availability.

## SYNERGY BETWEEN TASK AND SYSTEM MODELS

Interactive systems engineering can involve the production of various models such as task model, user model, domain model, dialog model, training model … that should not be considered independently as they usually co-evolve and represent different views of the same world. When formal description techniques are used, the process of verification and modification of models is iterative and iteration is conditioned by the result of formal verification. This allows proofs to be made on the system model in addition to classical empirical testing once the system has been implemented. Modelling systems in a formal way helps to deal with issues such as complexity, helps to avoid the need for a human observer to check the models and to write code. It allows reasoning about the models via verification and validation and also to meet three basic requirements notably: reliability (generic and specific properties), efficiency (performance of the system, the user and the two together) and finally to address usability issues (by means of tasks models for instance to assess effectiveness). Figure 3 presents an example of development process taking into account the integration of system and task models.

As stated above, such a process should be extended to take into account other types of models (e.g. training, requirement …) and this extension is already part of our current work, while the expression and verification of properties has been previously studied for formal notations in the field of interactive systems [4].

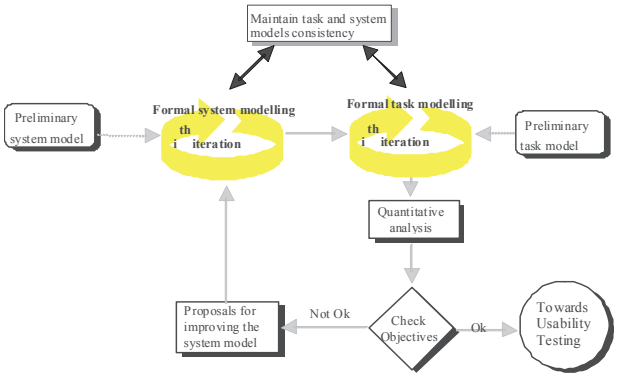



**Figure 3. The iterative model-based design life cycle using both tasks and system models**

### Principles

Making possible the integration between task and system models at tool level requires identifying basic bricks from both notations and supporting tools. As stated in [9] the integration at the tool level is divided into two parts: the first is the editing of the correspondence between the two models while the second consists in a co-simulation of these models.

### Correspondence between models

On the task side, the integration relies on the HAMSTERS environment that provides a set of tools for engineering task models (editing and simulation of models). Similarly, on the system side, the integration relies on the ICO environment (PetShop) that provides means for editing and simulating the system model:

- From the tasks specification we extract the set of interactive tasks (input and output tasks) representing a set of manipulations that can be performed by the user on the system and outputs from the system to the user.
- 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.

The principle of editing the correspondences between the two models is to put together interactive input tasks (from the task model) with system inputs (from the system model) and system outputs (from the system model) with interactive output tasks (from the task model). Setting up this correspondence may show inconsistencies between the task and system model such as interactive tasks not supported by the system or rendering information not useful for the task performance. The correspondence editing process is presented on the top part of Figure 4 where each tool feeds the correspondence editor with information from the API in order to notify it with modifications are done both in the task model and in the system model.

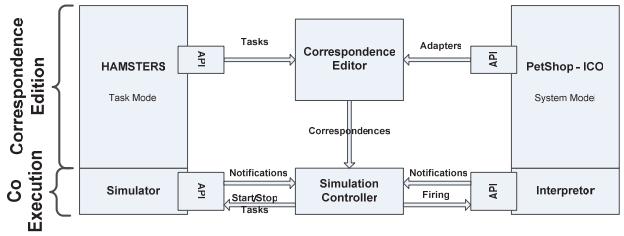



**Figure 4. Global architecture of the framework for the co-execution of task and system model.**

### Co-execution of task and system models

Our framework allows the co-execution of task and system models controlled by both the execution of the system model and the execution of the task model as shown in Figure 4 (where the top part represents the correspondence edition).

Figure 4 highlights the two way communication allowed by the services embedded within the four APIs:

- Through an API, HAMSTERS notifies the Simulation controller of changes in the current scenario.
- Through another API, the Simulation controller fires the corresponding activation adapter (according to the correspondence provided by the Correspondence editor).
- Through an extended API, the PetShop interpreter notifies the Simulation controller of the evolution of the current execution of the system model (notifications comes from both rendering and activation functions).
- Through an extended API, the HAMSTERS Simulation controller performs the corresponding task (according to the correspondence provided by the Correspondence editor), simulating the user action.

When the task simulator controls the execution of the system model, the framework behaves as follows: while building a scenario, if the task performed within the scenario is one of the identified interactive input tasks within the correspondence editor, an event is sent to the activation function (simulating the corresponding user event on the user interface), resulting in a user action on the interactive application (from the execution of the model). As a scenario describes a sequence of tasks and as we are able to define a correspondence between an interactive input task and an activation adapter, it is now possible to convert the scenarios into a sequence of firing of event handlers in the ICO specification. In other words, a scenario performed from these tasks can be converted into a sequence of firing of event handlers that directly drive the execution of the ICO specification in exactly the same way as user actions on the user interface would have triggered the same event handlers.

Symmetrically, when the execution is controlled by the execution of the system model, user actions are directly linked to the corresponding tasks from the task model and the user's action on the user interface of the application change the current state of the task model simulation.

## CASE STUDY

To illustrate our approach, we use the example of an interactive cockpit application (see Figure 5) called WXR (for Weather Radar System).

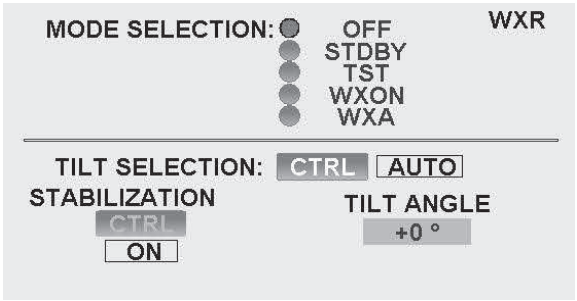



**Figure 5. Snapshot of the WXR application in civil commercial aircrafts**

The lower part of Figure 5 is dedicated to the adjustment of the weather radar orientation (called tilt selection), while the upper part allows the crew members changing the mode of the weather radar (independently from the tilt selection).

### Task modelling using Hamsters for WXR application

As shown in Figure 6, the high-level tasks for managing the weather radar (i.e. "manage_WXR") are decomposed into two tasks, "setup" and "switch_off". Task "setup" represents the two activities of adjusting the weather radar orientation

and mode, while task “switch_off” may interrupt it at any time. The two abstract tasks “change_mode” and “manage_tilt_mode” are detailed in Figure 7 and Figure 8. These two tasks are activated by the two cognitive tasks “need_to_change_mode” and “decide_tilt_mode” to represent the decision activity performed by the crew members before interacting with the system. The crewmembers can switch between five modes of the weather radar itself (on, off, standby, test and focus alert, and task “switch_off” as shown in Figure 6).

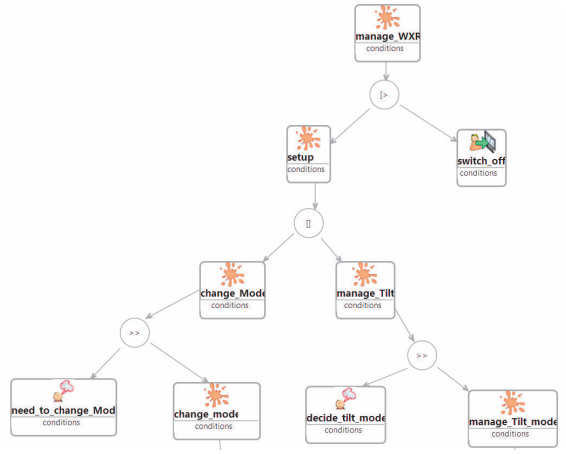



**Figure 6. High-level set of tasks for weather radar management.**

In Figure 7, the task “change_to_On_mode” is detailed whilst the other ones are folded (displayed as greyed out).

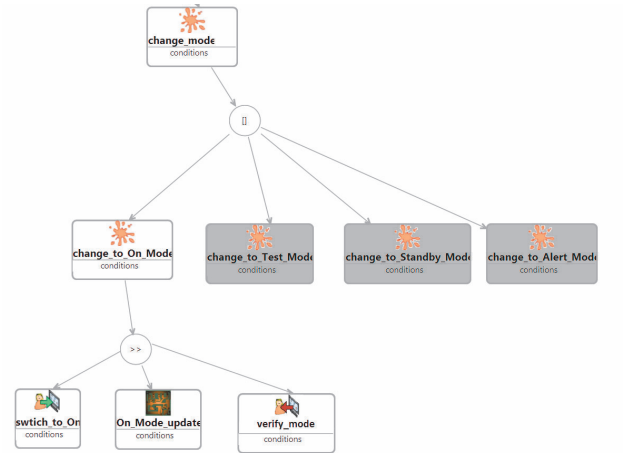



**Figure 7. Detailed set of subtasks for "change_mode" task**

The task “change_to_On_mode” encompasses the following sub-tasks:

- Interactive input task “switch_to_On” represents the crew members’ action on the system.
- System task “on_mode_update” represents the system inner activity to take the mode switching into account.
- Interactive output task “verify_mode” models the fact that crew members process system’s output to check that their action has been taken into account.

As shown in Figure 8, the crew members may adjust the orientation (the tilt angle) of the weather radar when required (the main idea being to use this feature only if necessary as, most of the time, the default behaviour is the correct one). There are three possible modes for tilt angle selection: auto adjustment, auto stabilization and setting up manually the tilt angle (represented by the three tasks “change_to_auto_mode”, “change_to_stabilization” and “change_angle_manually”).

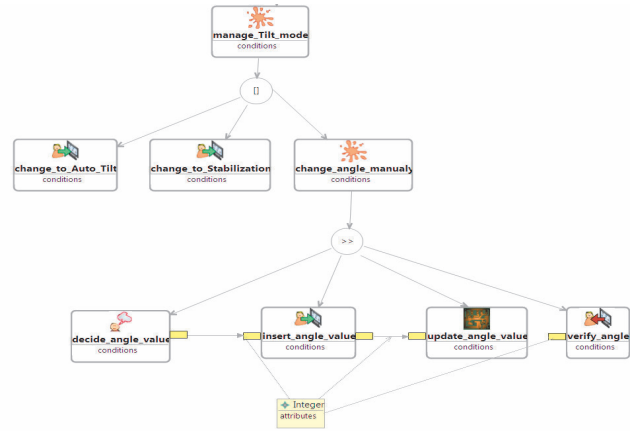



**Figure 8. Subtasks for "manage_tilt_mode" abstract task**

While the first two tasks are simple interactive input tasks, changing the tilt angle manually implies four sub-tasks:

- Cognitive task “decide_angle_value” is the choice of a value by the crew members.
- Interactive input task “insert_angle_value” is the interaction for editing the value in the aircraft system.
- System task “update_angle_value” is the inner activity of the system for checking the validity of the value.
- Interactive output task “verify_angle” provides crew members with output from the system to check that the entered value has been taken into account.

### System models using ICO

WXR system was modelled using the ICOs formal notation [11]. Hereafter we present the ICO modelling technique including the dialog part and the presentation part (according to arch architecture) and two functions (the activation and the rendering functions) that are connecting these two parts.

*Dialog part as an Interactive Cooperative Object*

Figure 9 shows the entire behaviour of page WXR which is made of two non connected parts:

- The upper part of the Petri net handles events received from the 5 CheckButtons (see Figure 5 for the presentation part). Even though they are CheckButtons,

the actual behaviour of that application makes it possible to select only one of them at a time. The current selection (an integer value from 1 to 5) is stored in the token of place MODE_SELECTION and corresponds to one the possible selected CheckButtons (OFF, STDBY, TST, WXON, WXA). The token is modified by the transitions (new_ms = 3 for instance) using variables on the incoming and outgoing arcs as formal parameters.

- The Petri net in the lower part handles events from the 2 PicturePushButton and the EditBoxNumeric, changing the state of the application. In the current state, this part of the application is in the manual state (i.e. a token is in place NOT_AUTO and a token is place STABILIZATION_OFF). This configuration of tokens is required in order for the edit box to be available to the user (visible on the model as transition change_Angle_T1 is in a darker colour).

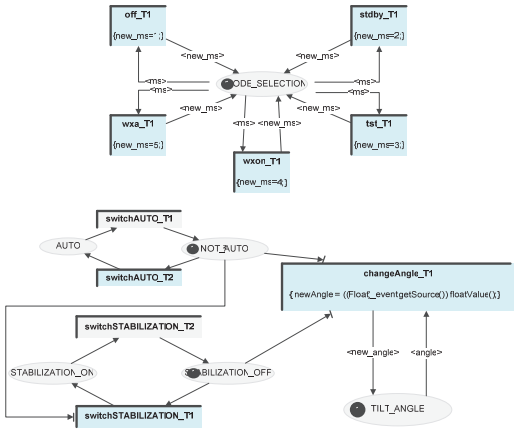



**Figure 9. Behaviour of the page WXR**

*Presentation part*

In an ICO description, the presentation part corresponds to the Logical Presentation, hidden by a set of rendering methods (in order to render state changes and availability of event handlers) and a set of user events, embedded in a software interface.

```
Public interface WXR_PAGE extends ICOWidget {
 //List of user events.
 public enum WXR_PAGE_events {asked_off, asked_stdby, asked_wxa,
asked_wxon, asked_tst, asked_auto asked_stabilization, asked_changeAngle}
 //List of activation rendering methods.
 void setWXRModeSelectEnabled(WXR_PAGE_events, List<ISubstitution>);
 void setWXRTiltSelectionEnabled (WXR_PAGE_events,
List<ISubstitution>);
 //List of rendering methods.
 void showModeSelection (IMarkingEvent anEvent);
 void showTiltAngle (IMarkingEvent anEvent);
 void showAuto (IMarkingEvent anEvent);
 void showStab (IMarkingEvent anEvent);
}
```

**Figure 10. Software interface of the page WXR from the user application MPIA**

*Activation function*

When considering WIMP interfaces, user → system interaction (inputs) only takes place through widgets. When a user event is triggered, the Activation function is notified and requires the ICO to fire the corresponding event handler embedding the value received in the user event. When the state of an event handler changes (i.e. becomes available or unavailable), the Activation function is notified (via the observer and event mechanism presented above) and calls the corresponding activation rendering method from the presentation part embedding the values from the event handler.

Figure 11 shows the activation function for page WXR.

| User Events | Event handler | Activation Rendering |
|---|---|---|
| asked_off | Off | setWXRModeSelectEnabled |
| asked_stdby | Stdby | setWXRModeSelectEnabled |
| asked_tst | Tst | setWXRModeSelectEnabled |
| asked_wxon | Wxon | setWXRModeSelectEnabled |
| asked_wxa | Wxa | setWXRModeSelectEnabled |
| asked_auto | switchAUTO | setWXRTiltSelectionEnabled |
| asked_stabilization | switchSTABILIZATION | setWXRTiltSelectionEnabled |
| asked_changeAngle | changeAngle | setWXRTiltSelectionEnabled |

**Figure 11. Activation Function of the page WXR**

Each line in this table describes the three objects taking part in the activation process. The first line, for instance, describes the relationship between the user event ask_off (produced by clicking on the CheckButton OFF), the event handler off (represented in the model by transition off_t1) and the activation rendering method setWXRModeSelectEnabled from the presentation part. More precisely:

- When the event handler off becomes enabled, the activation function calls the activation rendering method setWXRModeSelectEnabled providing it with data about the enabling of the event handler. On the physical interaction side, this method call leads to the activation of the corresponding widget (i.e. presenting the checkButton OFF as available to the user).
- When the button OFF of the presentation part is pressed, the presentation part raises the event called asked_off. This event is received by the activation function which requires the behaviour part to fire the event handler off (i.e. the transition off_T1 in the Petri net of Figure 9).

*Rendering function*

System → user interaction (outputs) presents to the user the state changes that occur in the system. The rendering function maintains the consistency between the internal state of the system and its external appearance by reflecting system states changes on the user interface. Indeed, when the state of the ICO changes (e.g. marking changes for at least one place), the Rendering function is notified (via the observer and event mechanism) and calls the corresponding rendering method from the presentation part with tokens or firing values as parameters.

| ObCS Node name | ObCS event | Rendering method |
|---|---|---|
| MODE_SELECTION | token_enter | showModeSelection |
| TILT_ANGLE | token_enter | showTiltAngle |
| AUTO | marking_reset | showAuto |
| AUTO | token_enter | showAuto |
| AUTO | token_remove | showAuto |
| STABILIZATION_ON | marking_reset | showStab |
| STABILIZATION_ON | token_enter | showStab |
| STABILIZATION_ON | token_remove | showStab |

**Figure 12. Rendering Function of WXR page**

Figure 12 presents the rendering function of the WXR application in a table where each line features the objects taking part in the rendering process. For instance, the first line shows the link between the place MODE_SELECTION, the event linked to this place (a token enters the place) and the rendering method showModeSelection from the presentation part component. It can be read as follows: when a token enters the place MODE_SELECTION, the rendering function is notified and the rendering method showModeSelection is invoked with data concerning the new marking of the place that is used as parameters of the rendering method.

### Demonstration of co-execution

In this section we illustrate the synergistic modelling framework using the WXR case study. We present the correspondence edition between the models and then the co-execution of these models exploiting that correspondence. Then we discuss validation and verification possibilities of this framework.

*The Correspondence editor*

The edition of correspondences between the two models are done by a dedicated editor (see Figure 13) making it possible to put together interactive input tasks (from the task model) with system inputs (from the system model) and system outputs (from the system model) with interactive output tasks (from the task model).

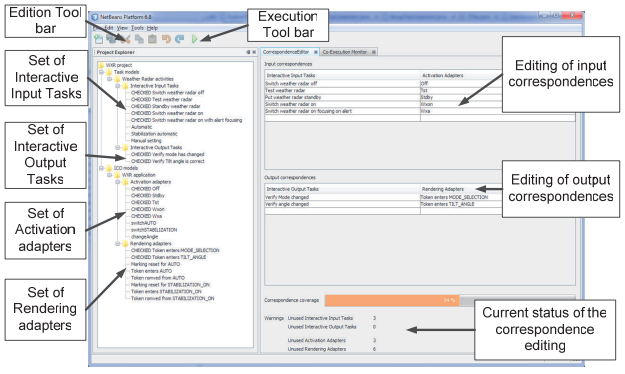



**Figure 13. Snapshot of the correspondence editor**

The left-hand side contains a tree structure with the relevant items from both the task model and the system model (interactive input and output tasks, activation and rendering adapters). The case study only features one role (only one task model) and only one ICO model; but the editor is able to handle larger set of models at a time. The top right-hand side is made up with two tables representing the current state of the editing of the correspondence:

- The table on top represents the input correspondences e.g. the link made between an input task and an activation adapter (a user event). In the example, five tasks are already connected to five user events (e.g. "switch_off" is connected to user event "Off").
- The bottom table represents the output correspondences e.g. the link made between an output task and a rendering adapter. In the example, two tasks are already connected to two rendering events ("verify_mode" has changed is connected to rendering event Token enters SELECTION_MODE and "verify_angle" has changed is connected to rendering event Token enters TILT_ANGLE…).

The bottom right-hand part represents the current status of the editing of correspondences. It is made up with a progress bar showing the current coverage of task and system items by the edited correspondences (i.e. the rate of used items: the current editing of Figure 13 shows 14 items used among 26). Below the progress bar, a set of warnings are displayed, showing how many tasks and system items are not currently used (for instance, in Figure 13, three).

At any time, the co-execution of models may be launched (via a tool bar icon), even if correspondence editing is not completed.

*The co-execution monitoring interface*

The execution of models in the framework can start either by task models or system models. When the co-execution is launched from the correspondence editing, a new set of components allows to control and to monitor this co-execution as presented by Figure 14 that can be decomposed in three parts:

- The left-hand part is a set of tabs containing the ICOs involved in the co-execution showing their evolution during the execution (one tab per model).
- The central part contains on its top part a view of the task model and at the bottom part the simulation controller of HAMSTERS with an empty panel on its right side that contains when necessary the means to provide values for the task execution (i.e. numerical values typed in a text field, or more complex objects selected using a list box).
- The right-hand part contains a table featuring a logging for events occurring during the execution and their counter partner input or output correspondences.

Additionally, the window of the executed application (WXR) is visible at the bottom of Figure 14, ready to react to user events.

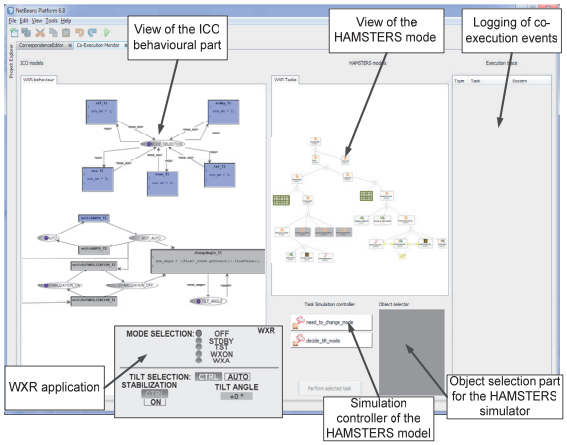
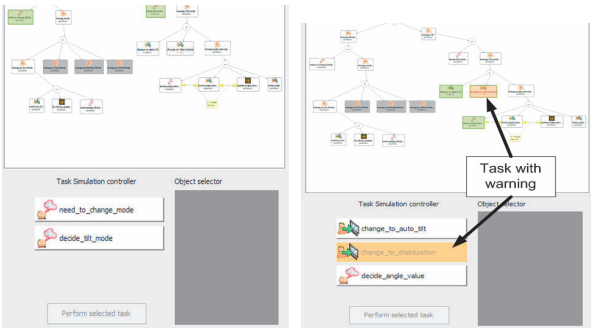



**Figure 14. Snapshot of the co-execution monitoring interface**

*Task models execution controlling the system execution*

The execution of a task model produces a sequence of tasks including interactive (input and output) tasks. Non interactive tasks are not related to the system execution as they involve user without interaction with system or system without feedback to the user. The correspondences identified within the editor, make possible to convert the sequence of interactive tasks into a sequence of user event triggering within the ICO specification, controlling the system execution as if the scenario played were a user.

For the case study, when the co-execution monitor starts, the initial set of available tasks contains "need_to_change_mode" and "decide_tilt_mode" (as shown on Figure 15.a). Performing one of these two tasks is made possible by double-clicking it or use the dedicated button.




a) available tasks b) unavailable tasks

**Figure 15. Excerpt of the co-execution monitor featuring task availability.**

Cognitive task "decide_tilt_mode" means that the user decides to setup the WXR orientation (such task is not related to any user event). Performing this task makes available the two interactive input tasks "change_to_auto_tilt" and "change_to_stabilization" and the cognitive task "decide_angle_value".

The execution of the system model driven by the task model is performed task after task within the HAMSTERS simulation controller until it reaches the end of the scenario.

If no system item corresponds to one of the available tasks then the co-execution monitor will display a warning (such as illustrated by Figure 15.b where tasks "change_to_stabilization" is available and corresponding user event is disabled until the user press the CTRL button). Such case could be normal as it would correspond to sequence of actions forced by the system for safety purpose for instance.

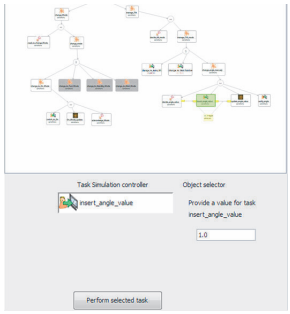



**Figure 16. Excerpt of the co-execution monitor presented on Figure 14 with object editing.**

Task "decide_angle_value" has an output port that represents the value the user wants to set the tilt angle with. Performing this task activates the interactive output task "insert_angle_value" that receives the tilt angle value through its input port. If the corresponding user event for task "insert angle value" is enabled (i.e. the editing of the tilt angle using the edit box), performing the interactive task requires runtime information. Such values may be system values (values within the system model) or free values (such as numbers). When performing such task, the co-execution monitor provides means to enter or select the corresponding value. The identification of the value type is done according to the artefact description attached to the output port of the corresponding interactive task within the HAMSTERS model and the corresponding activation adapter. An example of such execution is provided by Figure 16.

If none of the available task can be executed on the system model, the simulation is stopped and an error is notified. The simulation ends when there is a no longer available interactive task.

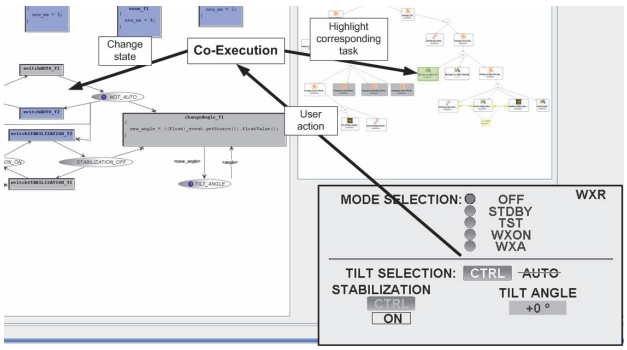



**Figure 17. Interaction between task model execution and system model.**

*When system execution controls the task execution*
A sequence of actions on the user application (played using the ICO model) is able to control the execution of a task model according to the edited correspondences, as each user action may be related to an interactive input task (see Figure 17 where task "change_to_stabilization" is highlighted on the task model after the user has pressed the button CTRL).

While interacting with the system, it is possible to point out which task from the task model is performed. This makes it possible to trace the system execution within the task model. Ambiguity in pointing tasks may appear if there is more than one task with the same name within the task model. In the current design of the framework, when such ambiguities appear, the co-execution monitor triggers warnings showing the set of potential corresponding tasks.

This policy of co-execution allows knowing at any time where a user is with respect to the described activity within the task model. Knowing this, it is possible to provide the user with a contextual help such as in [14].

Another possible use of the execution of task driven by the system model could be to determine if going from one interactive task to another (according to the system execution) is possible, using path analysis on the task model. An interesting output of such work is that it allows finding inconsistencies such as sequences of user actions allowed by the system model and forbidden by the task model.

## DISCUSSION

While previous research activities done on the topic of interactive systems modelling made ICO mature enough to be the basis of the proposed framework, they pointed out the need of an extension of current task modelling approaches making possible the synergistic support of task and system modelling activities. With Hamsters we propose a notation and a tool to answer these needs, making it an independent tool supporting task modelling activities, and enhancing it to be part of the framework (by means of interactive input and output tasks, explicit artifacts, dedicated API …).

The integration framework presented in this paper allows for property checking during verification and validation phases of the development process as described by Hix and Hartson [5]. Validation phase relates to the question "do we have modelled the right system?" while the verification phase addresses the question "do we have modelled the system right?" At notation and tool level, our approach provides the first bricks for the validation and verification of the synergistic exploitation of task and system modelling:

- It is possible to assess the structural compatibility between models while editing the correspondences between them.
- It is possible to verify if particular scenarios are playable on the system model. This make it possible to highlight or verify system behaviours that ensure the non occurrence of particular tasks scenarios as this impossibility could be required in the system (ex. Getting card before getting cash using ATM to avoid post completion errors or ensuring that an accident scenario cannot reoccur). This work could be extended to automatically extract scenarios from the task models and assess automatically too their compatibility with the system model.
- It is possible to execute the system model driven by the simulation of the task model and it is possible to build scenarios driven by the system execution, but, even if the framework makes it possible, we did not present the complete co-execution of the two models due to space constraints. One of the possible use of such complete co-execution could be to enhance the user providing contextual help at runtime:
    - While interacting with the system it is possible to identify the current task in the task model, it is thus possible to provide the user with information about this task (for instance how many actions and which actions are still required to reach the goal).
    - A scenario from task model can drive the execution of the system model, it is thus possible to extract scenario that illustrates how to perform a task, and play it on the system to interactively show to the user how to achieve her goal (as training material for instance).

Such model checking is part of the role of the correspondence editor that notifies any inconsistency between the HAMSTERS and the ICO specifications.

Amongst the advantages, such integration allows a real co-evolution of the two models, as the execution of one tool impacts the execution of the other tool. This integration can provide designers with shorter iterations in the task and system modelling process. It also represents an improvement for the end user as the execution of the system should support training and provide contextual help. As stated in previous work [14], it thus allows the use the task model as an input for providing the user support.

## CONCLUSION AND FUTURE WORK

This paper has presented a tool supported approach for bridging the gap between tasks and system views in the design of interactive systems. To this end we have briefly introduced a new notation called HAMSTERS for the description of tasks models. For the system side we used the ICO notation supported by the CASE tool PetShop.

While in earlier work [10] the bridge between task models and system models was performed in an asynchronous way by means of scenarios, the current paper has presented how the full integration of two dedicated tools can be performed and that it provides many benefits both for the verification of the compatibility of the models and at runtime by supporting users activity to reach their goals providing them with contextual information.

The work presented here belongs to a longer term research program targeting at the design of resilient interactive systems using model-based approaches. Future work targets at exploiting these two models to support the usability evaluation of interactive systems and to provide task-based training material in the field of satellite ground segments.

## REFERENCES


1. Bastide, R., Navarre, D., and Palanque, P. 2002. A model-based tool for interactive prototyping of highly interactive applications. In CHI '02 Extended Abstracts on Human Factors in Computing Systems. pp. 516-517.
2. Blumendorf, M., Lehmann, G., Feuerstack, S., and Albayrak, S. 2008. Executable Models for Human-Computer Interaction. In Proc. of DSV-IS 2008 Kingston, Canada. Springer LNCS v. 5136. pp. 238-251.
3. Bodart, F., Hennebert, A., Leheureux, J., and Vanderdonckt, J. 1994. Towards a dynamic strategy for computer-aided visual placement. In Proc. of the Workshop on Advanced Visual interfaces (Bari, Italy, June 01 - 04, 1994) at AVI '94. pp. 78-87
4. Dix, A. J., 1991 Formal Methods for Interactive Systems. s.l: Academic Press. 0-12-218315-0.
5. Hix, D. and Rex Harston, H. 1993. Developing User Interfaces: ensuring usability through product and process. s.l. : Wiley, 1993. 978-0-471-57813-0.
6. Limbourg, Q., Vanderdonck J., Michotter, M., Bouillon, L., Lopez-Jaquero, V., 2005. USIXML: A Language Supporting Multi-path Development of User Interfaces In proc. of EHCI-DSVIS 2004, LNCS 3425, pp. 200-220.
7. Lu S., Paris C., Vander Linden K. 1999. Towards the automatic generation of task models from object oriented diagrams. In Engineering for Human-Computer Interaction, Kluwer academic publishers, Boston, 1999.
8. Lu S., Paris C., Vander Linden K., and Colineau N. 2003. Generating UML Diagrams From Task Models. In Proc. of CHINZ'03, Dunedin, New Zealand, 2003.
9. Navarre, D., Palanque, P., Barboni E. and Mistrzyk, T. 2007. On the Benefit of Synergistic Model-Based Approach for Safety Critical Interactive System Testing. In Proc. of TAMODIA 2007. (Toulouse, France), Springer LNCS vol. 4849. pp. 140-154.
10. Navarre, D.; Palanque, P.; Bastide, R.; Paternó, F., and Santoro, C. A tool suite for integrating task and system models through scenarios. In DSV-IS'2001 (Glasgow, Scotland, June 13-15, 2001). LNCS 2220. Springer; 2001
11. Navarre, D., Palanque, P., Ladry, J., and Barboni, E. 2009. ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. ACM TOCHI. V. 16, 4, 1-56
12. Navarre, D., Palanque, P., Winckler, M. Task Models and System Models as a Bridge between HCI and Sofware Engineering. In: Human-Centered Software Engineering Software Engineering Models, Patterns and Architectures for HCI. Springer, June 2009, pages 357-385.
13. Palanque, P., Bastide, R. Synergistic Modelling of Tasks, Users and Systems Using Formal Specification Techniques. Elsevier. Interacting With Computers 9 N. 2, pp. 129-53, 1997.
14. Palanque, P., Bastide, R., Dourte L. Contextual Help for Free with Formal Dialogue Design. In Proc. of HCI International'93, Orlando, USA, 8-15 August 1993.
15. Palanque, P., Bastide, R. and Paternò, F. 1997. Formal Specification As a Tool for the Objective Assessment of Safety Critical Interactive Systems. In Proc. of Interact'97, Sydney, Autralia, 1997, 323-330.
16. Palanque, P., Bastide, R. and Sengès, V. 1995. Validating Interactive System Design Through the Verification of Formal Task and System Models. In Proc. of EHCI'95, Garn Targhee Resort, Wyoming, USA, August 14-18, 1995. Chapman et Hall, 1995.
17. Paternò F., Breedvelt-Schouten I., deKonig N. 1998. Deriving Presentations from Task Models, In Proceedings EHCI'98, Creete, Kluwiert Publisher.
18. Paternò, F., Mori, G., and Galiberti, R. 2001. CTTE: an environment for analysis and development of task models of cooperative applications. In CHI '01 Extended Abstracts, Seattle, Washington, March 31 - April 05, 2001. CHI '01. ACM, New York, NY, 21-22.
19. Puerta, A. and Eisenstein, J. 1999. Towards a general computational framework for model-based interface development systems. In Proc. of IUI'99, Los Angeles, CA, USA, January 05-08, 1999. pp. 171-178.
20. Reichart, D., Dittmar, A., Forbrig, P., Wurdel, M. Tool Support for Representing Task Models, Dialog Models and User-Interface Specifications. In Proc. of DSVIS'2008. Kingston, Canada, July 16-18 2008. Springer LNCS 5136. pp. 92-95.
21. Sawyer J. T., Minsk B., Bisantz A. M. 1996. Coupling User Models and System Models: A Modeling Framework for Fault Diagnosis in Complex Systems Interacting with computer 1996.
22. Scapin, D.and Pierret-Golbreich, C. 1989. Towards a method for task description: MAD. Work with DisplayUnits WWU'89, 27–34.
23. Winckler, M., Vanderdonckt, J., Trindade, F., Stanciulescu, A. Cascading Dialog Modeling with UsiXML. In Proc. of DSVIS'2008. Kingston, Canada, July 16-18 2008. Springer LNCS 5136. pp. 121-135