Bridging the gap between formal and experimental validation approaches in HCI systems design: use of the event B proof based technique.

Y. AIT-AMEUR and M. BARON

LISI - ENSMA and University of Poitiers
BP 40109, 86961, Futuroscope Cedex, France
{yamine,baron}@ensma.fr

Abstract The development of User Interfaces (UI) needs validation and verification of a set of required properties. Different kinds of properties are relevant to the Human Computer Interaction (HCI) area. Not all of them may be verified using classical software engineering validation and verification tools. Indeed, a large part of properties is related to the user and to usability. Moreover, this kind of property usually requires an experimental validation. This paper addresses the cooperation between formal and experimental HCI properties validation and verification. This paper focuses on a proof based technique (event B) and a MBS (Model Based System) based technique (SUIDT). Moreover, this paper tries to bridge the gap between both approaches in order to reduce the semantic heterogeneity they lead to.
Keywords: formal methods, interaction properties verification and validation, user task modelling and validation, event B method, CTT (ConcurTaskTrees) language.

1 Introduction

Due to their wide class of possible application domains and to their presence in a big number of systems, the development of User Interfaces (UI) involves more and more methods, models, techniques, and processes leading to heterogeneous developments. When designing a given UI, these methods and techniques are set up at different development levels: design, specification, maintenance, etc. with different objectives: validation, verification, testing, etc. The resulting activities may be time consuming, inaccurate, cost effective and heterogeneous. The usual development process consists first in building, at a given development level, a model of the UI and second in performing validation, verification, testing, experiments, etc. on this model. Techniques are used to express both the model and the validation and verification procedure. Each technique is applied where it is the most efficient. For example formal techniques are useful for formal verification, proof, exhaustive testing by model checking etc. while experimental techniques are useful for testing, validation of user behavior, validation of ergonomic properties, etc.

The use of different techniques lead to different models and modelling processes and to a strongly heterogeneous development. At this stage, it becomes quite impossible to assert that for given UI requirements R, a property P established on a model M of an UI issued from R, is also valid on a model M' of an UI issued from the same R on which other properties may have been established. There is a need of relating M and M', and this relation cannot be defined if M and M' are not formally represented. For example, when M and M' are formalized, this relation can be identity, refinement, simulation, bi-simulation, abstraction, etc. The problem of heterogeneity is more or less studied when the techniques are formal techniques. Approaches based on category and institution theories or on synchronisation of logics have been studied by the formal methods community.

Nevertheless, in the HCI area, a crucial validation is the experimental validation. Indeed, this kind of validation is based on the user which plays a central role. Other actors like ergonomists or psychologists are involved in this validation activity. To perform such experiments, several experimental models have been developed to conduct these experiments. **M**odel **B**ased **S**ystems (MBS) are one of the techniques which give an operational behavior to a UI in order to perform experimental UI validation. Again, we are facing the problem of heterogeneity.

This paper suggests a general approach allowing to bridge the gap between formal techniques and experimental techniques. Currently, these two techniques are used in parallel or in a concurrent way in the development process but, they are not based on a common formal model and therefore, there is no guarantee that the results issued from formal verification on a formal model apply to the experimental model and vice versa: there is no guarantee that the results obtained from experiments on an experimental model are valid on the formal model. One solution is to use the final UI program as a central model and perform both formal verification (e.g. reverse-engineering) and an experimental validation (e.g. testing with user). This approach is heavy and costly since it requires to provide the final application before performing validation.

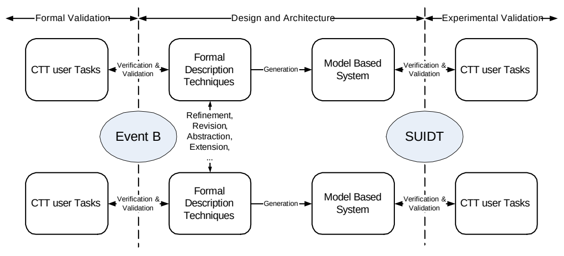

Formal Validation
Design and Architecture
Experimental Validation
CTT user Tasks
Verification & Validation
Formal Description Techniques
Generation
Model Based System
Verification & Validation
CTT user Tasks
Event B
Refinement, Revision, Abstraction, Extension, ...
SUIDT
CTT user Tasks
Verification & Validation
Formal Description Techniques
Generation
Model Based System
Verification & Validation
CTT user Tasks

Figure 1. Formal and experimental validation of HCI properties approach.

Our approach consists in performing both formal and experimental validation earlier in the development process, see figure 1. By earlier we mean at any stage of the development. The only necessary condition is to provide a formal constructive model of the UI to be developed.

This paper is structured as follows. Next section presents a brief overview of the development of UI. It focuses on both formal and experimental approaches. Section 3 presents our approach for design and architecture using a formal approach based on event B. The formal and experimental validations and their cooperation are discussed in section 4. A simple case study is used to illustrate this approach. Finally, we conclude and give some future issues to this work.

2 Development of UI

In general, two different development scenarios leading to different development processes can be defined. Either (1) a set of properties is provided and the specification and design conforming this property are built or (2) a constructive specification is provided and the properties shall be checked on the built specification. In both cases, properties validation is a fundamental activity. The properties shall be validated by any specification or design that claim conformance to the original requirements.

2.1 Design and architecture

As for classical program developments, the area of HCI software development has seen a growing number of design and description techniques and notations. Moreover, the involvement of different competencies like software engineering, ergonomy and psychology has made these notations heterogeneous. Most of these techniques are semi-formal and the graphical support is one of their important characteristics since it represents the only integrated notation. We have chosen to divide these techniques and notations into two parts:

- *Design oriented notations and techniques* allow to express the static structure of the software implementing the described UI. These techniques and notations separate the UI from the core application. Examples of such techniques are: PAC [Cou87], ARCH [BPR$^+$91] and hybrid models [Gui95].
- *Description oriented notations and techniques* allow to describe the user needs in terms of usability of an UI. They are usually issued from non computer scientists (ergonomy and psychology for example). The majority of these notations are user centered. The user needs are expressed by a set of tasks that allow to reach a given goal from an input state. Examples of such techniques are: MAD [SPG89], XUAN [GEM94] and CTT [PMG01].

2.2 Properties in HCI : verification and validation

In the HCI development area, two categories of properties[1] have been distinguished by several authors [DFAB93] [DH95].

[1] Notice that in terms of UI developers, the words defining properties do not have the same meanings like the ones used by formal methods users. These properties are used to characterize the quality of UI software.

- *Behavioral properties* characterize the behavior of the UI about the user. In this category of properties, we can distinguish completeness (achievement of an objective in different manners), flexibility for information representation, task achievement (multiple users, state reachability, non preemptive tasks), flexibility for information representation, properties related to time, psychological and ergonomic aspects, and so on.
- *Robustness properties* are related to the reliability of the UI in particular and of the system in general. Among these properties, we find properties related to visualisation like observability, insistance and honesty.

The classical properties issued from software engineering are checked using the well known verification techniques, but there is a need of experimental techniques to check some of the user related properties. Moreover, the software engineering techniques and the experimental techniques shall remain consistent with respect to the specification and design of the system.

In the remaining of this paper, we focus on formal validation techniques for the software engineering properties and on the MBS based techniques for the experimental ones.

2.3 Formal approaches in HCI area

Among the several used formal description techniques in software engineering, model oriented approaches play a major role in the HCI area. These methods are based on the description of the model by a set of variables which are modified by the operations and events of the model. Generally, these techniques are divided into two categories: automatic proving by model checking and proof systems. Both of these two techniques have been applied to interactive systems [CH97].

The first category is based on the evaluation of logical properties on the state transition system which is obtained from the evolving variables. Among these techniques, we can find temporal logics, Petri nets and so on. In the area of interactive systems, these techniques are assumed to have been used first in formal verification of interactive systems [CH97]. For example, [PF92] uses LOTOS to write interactors specifications, and analyses translated finite state machines using ACTL (Action-based Temporal Logic). Model checking is also used by [PBS95] who model user and system by the way of object-oriented Petri nets ICO. More recently, [D'A98] used the Lustre dataflow language for the automatic validation of user interface systems.

The second category is based on proof systems where the model is described by variables, operations, events, temporal properties and invariants. The operations must preserve these invariants and a set of other properties (safety, liveness, deadlock freeness etc.). To ensure the correctness of these specifications a set of proof obligations are generated and shall be proved. According to its implementation, the proof system can achieve some of the proofs automatically. Among these techniques, we can find Z, based on set theory [Spi88], VDM [Bjo87], based on preconditions and postconditions calculus [Hoa69], and B, based on the weakest precondition calculus [Abr96] [Dij76]. In the HCI field VDM and Z have been used for defining atomic structures like interactors [DH93a] [DH93b].

2.4 Experimental approaches in HCI area

Human computer interaction tools for building interactive software are numerous. On the one hand, GUI-Builders[2] and tools like Visual Basic® or JBuilder® do not support any kind of external model. On the other hand, Model-Based tools[3] [Pue96] deal with models, but are not usable for actual software development. MBS are the evolution of primitive User Interface Management Systems (UIMS). [Sze96] gives a generic overview of architectural description of MBS components.

First, the model is the most important component of MBS environments. It represents all the different views of the interactive application and may be decomposed into sub-models such as the domain, the dialog, and/or the presentation models. Three abstraction layers are identified. (1) The higher layer is made of the *domain and task models*. (2) The intermediate layer, the *abstract specification layer*, may involve abstract interaction objets and abstract data. Finally, (3) the lower layer describes the *concrete specification*. Interaction objets are concrete widgets that come from toolkits.

The second category of components of MBS environments is a set of tools able to manage the different models. *Modelling tools* help the designer to edit the models as MASTERMIND[SSC$^+$95], or more user-centered, such as forms in MECANO[Pue96]. *Automatic design tools* are able to complete and/or to concretize some abstract specifications in order to produce new and more concrete specifications as JANUS[BHKN96]. *Implementation tools* allow direct production of code from the models as MOBI-D[PE98]. *Validation tools* are the really added value of MBS compared from non MBS approaches. Reasoning upon model is easy, and enforcing properties through interactive systems is much more easy when external models are available. A new tendency incorporates formal approaches into MBS in order to get the benefits of formal validation. PETSHOP[Nav01] and our suggested approach push forward this tendency.

2.5 Conclusion and our proposal

The problem of cooperation between formal and experimental approaches remain unsolved. Indeed, on the one hand formal approaches use abstract models of the UI system and allow to check properties of this abstract model and/or supports its refinement to code, abstraction, extension, etc. On the other hand, experimental techniques use rapid prototyping approaches through MBS systems that allow to animate a given UI system. At this level of the paper, we can conclude that:

1. usually the semantics of the formal and experimental techniques may be different and therefore, there is no guarantee that the properties formally checked apply on the developed MBS for experimentation and vice versa. We are facing a classical problem of semantic heterogeneity in system development: **horizontal heterogeneity.**

[2] **G**raphical **U**ser **I**nterfaces Builders
[3] currently called **MBS** for **M**odel **B**ased **S**ystem

2. the level of application of these techniques (either experimental or formal) are different. While formal techniques may be applied at any allowed abstraction level (due to their possibility to express different abstract models), experimental techniques based on MBS systems usually require a more concrete level near to programming level and toolkits. We are facing another kind of heterogeneity related to the level of abstraction: **vertical heterogeneity.**

Our objective is to reduce the gaps resulting from these heterogeneities and we suggest to use a common central formal model of the UI system. All the formal and experimental validations are performed on this central model. Proceeding this way contributes to reduce both vertical and horizontal heterogeneities. So, our proposal consists in designing a central model for representing the core of the UI system to be validated. We have chosen the B event based technique to represent this central model of the system. Then, starting from this common formal representation, we extract two models for formal (CTT with event B) and experimental (CTT with SUIDT) validation (see figure 1).

2.6 The case study

We illustrate our proposal on a small case study : French francs to euro conversion. Figure 2 shows a potential user interface. The user enters in the left **textfield** component (tag 1) the value he/she wants to convert, he/she makes the conversion by pushing either the $F >> E$ **button** component (tag 2) to convert in euro, or $E >> F$ **button** component (tag 3) to convert in French francs. The converted value is displayed thanks to the right **textfield** component (tag 4).

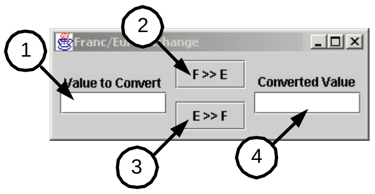


Figure 2. A potential UI for the case study.

3 Design and architecture.

This section reviews the use of the formal and experimental approaches applied to design the UI to be validated later.

3.1 Formal approach

We use the B formal method to show that it is possible to handle a complete UI formal development. This approach permits specifying, verifying and refining the formal B specifications. Particularly, programs written in Ada/Tk have been generated within this technique, demonstrating that our approach may be used in real size systems [AAGJ98][AABG03] [AABG$^+$04].

The B approach. We use the B method [Abr96] in its event based definition. It allows to describe distributed, parallel, multi-model, reactive and interactive systems.

An event B model is composed of a set of atomic events described by particular generalized substitutions (assignment, ANY, BEGIN and SELECT). Each event Ev is fired if the guard P associated to this event is true. For the purpose of this paper, we will only use the $SELECT$ substitution $Ev = SELECT\ P\ THEN\ S\ END$. The event Ev is fired and S is executed when P is true. Moreover, a B model contains a set of properties i.e invariants, liveness, safety and reachability properties which can be proved during the development thanks to the embedded proof system associated to B and to the B tool supported by *Atelier B*[Cle97]. Finally, B models can be refined into other B models which can be enriched by new events and new properties. The refinement process leads a developer to the final UI design after finite refinement steps which provide different abstract levels. The associated semantics is a trace based semantics where the events are interleaved. Thus, there is no parallel firing of events (asynchronous modelling).

Design and architecture with B. A set of events is described to define a transition system that allows to represent the dialog controller of the UI to be specified. In the case, of an interactive application described by several systems, our approach uses the refinement technique to introduce the events of the composed automata. Each system is then described progressively by refinements in an incremental way. Robustness and reachability properties were expressed and checked according to the B method.

The event B technique is applied for the design of the whole case study. We do not describe this development in this paper, it can be found in [Bar03]. We focus on the dialog controller which contains the set of events that are fired while using the UI and do not present events related to other parts. A list of events defining the automaton of the dialog controller of our case study is given below.

Event Name	Description
$Evt_{ClickQuitButton}$	Click on exit button
$Evt_{ClickQuitApplication}$	Close exchange application
$Evt_{InputValue}$	Input any convert value
$Evt_{DisplayValue}$	Updating all views after a conversion
$Evt_{ClickEuro}$	Click on franc to euro button
$Evt_{ClickFranc}$	Click on euro to franc button

Below the *ExchangeController* model obtained at a given refinement level is described. It represents the decomposition of the whole application into a set of events. The obtained model is sufficient to perform task validations presented in this paper.

REFINEMENT
VARIABLES
but_quit_event, *textfield_input_event*, *but_convert_event*, *but_euro_event*, *but_franc_event*, *exit_state*, *focus_state*, *convert_state*, *view_state*, *input_value*, *output_value*, *intput_value_display*, *output_value_display*, ...
INVARIANT ...
ASSERTIONS ...
INITIALISATION ...

The first five variables (*but_quit_event*, ..., *but_franc_event*) are used to identify the user interactions which occur. For example, if *but_franc_event* is $TRUE$, the user has clicked on $E >> F$ **button** component. *view_state* is used to know if the user interfaces are notified about modifications from the functional core. The conversion state (no conversion, French franc conversion or euro conversion) is returned by *convert_state* variable. We are able to know which user interface owns the current focus (exchange application or another application). Finally, *exit_state* allows to know if the exchange application is running or not.

The **INVARIANT** clause represents robustness properties and the **ASSERTIONS** clause ensures that the new events of the composed automata can be fired (no deadlock). Their description can be found in [AAB04][Bar03].

```
EVENTS
Evt_ClickQuitButton =
SELECT
  but_quit_event = TRUE ∧
  focus_state = convertappl ∧
  exit_state = FALSE ∧ ...
THEN
  but_quit_event := FALSE ||
  exit_state := TRUE || ...
END;

Evt_ClickQuitApplication =
SELECT
  but_quit_event = FALSE ∧
  focus_state = convertappl ∧
  exit_state = TRUE ∧ ...
THEN
  convert_state := no_conversion ||
  exit_state := FALSE || ...
END;

Evt_InputValue =
SELECT
  textfield_input_event = TRUE ∧
  input_value_display = FALSE ∧
  but_convert_event = FALSE ∧
  view_state = FALSE ∧ ...
THEN
  output_value_display := FALSE ||
  textfield_input_event := FALSE ||
  input_value_display := TRUE ||
  input_value :∈ NAT || ...
END;

Evt_DisplayValue =
SELECT
  textfield_input_event = FALSE
  output_value_display = FALSE
  ∧ view_state = TRUE ∧ ...
THEN
  view_state := FALSE || ...
  output_value_display := TRUE ||
  value_output :∈ NAT ||
  input_value_display := FALSE || ...
END;

Evt_ClickEuro =
SELECT
  convert_state ∈ {euro, franc} ∧
  but_euro_event = TRUE ∧
  but_franc_event = FALSE ∧
  view_state = TRUE ∧
THEN
  but_euro_event := FALSE ||
  but_franc_event := FALSE ||
  convert_state := euro
END;

Evt_ClickFranc =
SELECT
  convert_state ∈ {euro, franc} ∧
  but_euro_event = FALSE ∧
  but_franc_event = TRUE ∧
  view_state = BOOL ∧
THEN
  but_euro_event := FALSE ||
  but_franc_event := FALSE ||
  convert_state := franc
END;
```

For example, $Evt_{ClickQuitButton}$ is fired if *but_quit_event* is $TRUE$ and when the user clicks on *quitbutton*. This event modifies *exit_state* value in order to fire the *Evt_ClickQuitApplication* which permits to close application.

$Evt_{ClickEuro}$ and $Evt_{ClickFranc}$ are fired when the user clicks on franc to euro ($but\_euro\_event = TRUE \wedge \ldots$) or euro to franc ($but\_franc\_event \wedge \ldots$) button.

3.2 Experimental approach

The experimental approach starts from a formally developed functional core (FC). However, we do not expect that the user of the experimental tool becomes a formal method specialist. The SUIDT environment hides most of the formal aspects. We assume that the FC and the dialog controller, which has been specified and generated thanks to the formal approach, delivers the same services through an API[4]. Indeed a *get_xxx* and *set_xxx* and *run_evt* API is built. It is possible to automatically link such a B model to a tool that exploits function signatures and formal specifications to help building interactive software.

In figure 3, we can see a screen capture of some SUIDT elements. On the left (tag 1), the animator consists in a fully generated interface that allows to interactively run the functional core. Every function and/or event of the functional core is usable through button activation. When function parameters (variables) are required, a dialog box appears to allow the user to enter them. Functions are textually described and current state of the functional core can be estimated through the result of all functions.

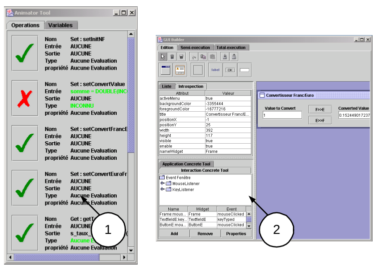


Figure 3. Experimental approach : SUIDT design point of view.

SUIDT covers the entire cycle of interactive development. It allows to interactively build graphical user interfaces through a GUI-Builder to which it is connected. Nowadays, it is related to the functional core through the domain model. In the right part of the figure 3 (tag 2), we can see the GUI-Builder view, where widgets can be dropped to build the concrete user interface. In the center, as in any GUI-Builder, we can see a property window. It allows the user to concretize the presentation. The last window associates events to functions from the functional core and/or dialog controller.

Two major advantages can be outlined. First, at any time, we can switch from design mode to test mode. The events of functional core can be fired from the presentation or from the animator. Second, the system is based on formal proved specifications to ensure that event firing are correct.

4 Validation and verification

This section shows how formal and experimental validations are performed on the design presented section.

4.1 Formal approach

Our approach consists in describing the whole CTT operators using event B models. Each decomposition of an upper task (a node) in the CTT tree corresponds to an event B refinement. A CTT Task is described by an initial state

[4] Application Program Interface

and a final state. It is refined into a sequence of basic events which lead from the initial state to the final state. The refinement preserves all the properties of the initial task. This process is repeated until basic events, present in the design are reached. When the basic events of the dialog controller are reached by the refinement, the validation process is completed. The full description of the CTT language by event B models can be found in [AAB04][Bar03]. Two validation aspects are addressed at this point. First, the final sequence of events shows that there exists a sequence of basic elements implementing the upper abstract task (the root task in the CTT tree). *This is a task validation aspect.* Second, if one of the basic events of the dialog controller is not present and/or some proof obligation related to the basic events cannot be proved, in the B models of the design, then, we can assert that some of basic events are missing and/or wrongly specified and therefore, the design shall be updated and/or completed. *This is a design and architectural decomposition validation aspect.*

Application to the case study. In order to illustrate our approach, let us develop an example of a generic task model described using CTT and built with the CTTE tool [PMG01] (figure 4). From a textual point of view, the hierarchical decomposition of task T_0 can be written as:

$$T_0 = T_1^*[> (T_2 >> T_3)$$
$$T_1 = T_4 >> T_5 >> T_6$$
$$T_5 = T_7[\ ]T_8$$

It uses three decomposition levels corresponding to three refinements. The leaves of the task model are represented by the tasks T_4, T_7, T_8, T_6 and T_3 and they correspond to events of the dialog controller (see figure 4). To illustrate this decomposition process, we give below the first refinement level for task T_0.

```
REFINEMENT $T_0$
REFINES $Main$
INVARIANT
  $StateDesT_0 \in (0..3) \wedge StateLoopT_0 \in NAT \wedge StartT_0 \in \{0,1\} \wedge \dots$
ASSERTIONS
  $\dots$)
INITIALISATION
  $StateDesT_0 := 3 \parallel StartT_0 := 0 \parallel \dots$

EVENTS
$Evt_0 =$
SELECT
  $G_0' \wedge$
  $StateDesT_0 = 0$
THEN
  $S_0'$
END;

$Evt_{InitLoop1} =$
SELECT
  $StateDesT_0 = 3 \wedge$
  $StartT_0 = 0$
THEN
  $StateLoopT_0 :\in NAT$
  $\parallel StartT_0 := 1$
END;

$Evt_{Loop1} =$
SELECT
  $StateDesT_0 = 3 \wedge$
  $StateLoopT_0 > 0 \wedge$
  $G_{Loop} \wedge StartT_0 = 1$
THEN
  $StateLoopT_0 := StateLoopT_0 - 1 \parallel S_{Loop}$
END;

$Evt_1 =$
SELECT
  $G_1 \wedge StartT_0 = 1 \wedge$
  $StateDesT_0 = 3 \wedge$
  $StateLoopT_0 = 0$
THEN
  $StateDesT_0 := 2 \parallel$
  $S_1$
END;

$Evt_2 =$
SELECT
  $G_2 \wedge$
  $StateDesT_0 = 2$
THEN
  $S_2$
  $\parallel StateDesT_0 := 1$
END;

$Evt_3 =$
SELECT
  $G_3 \wedge$
  $StateDesT_0 = 1$
THEN
  $S_3 \parallel$
  $StateDesT_0 := 0$
END; $\dots$
```

The S_i event B substitutions and the whole properties in the invariant and assertions clauses are not given to keep this paper short enough. The variable $StateDesT_0$ plays the role of a variant ensuring the right events firing order. The expression $StateLoopT_0 :\in NAT$ allows to initialize the loop variant with any natural number corresponding to the arbitrary iteration (T^*).

This CTT task model is translated into B event in a set of property preserving refinements. Three refinement steps which correspond to each level of the tree task model are necessary. The first B model describes T_0. The first refinement decomposes this task into $(T_1^*)[> T_2 >> T_3$ and the second refinement describes the refinement of $(T_1)^*$ by task $(T_3 >> T_4 >> T_5)^*$. Finally the last refinement describes the refinement of T_4 by $T_6[]T_7$.

Notice that we could immediately describe only one decomposition level for T_0. This decomposition of T_0 is the one we could describe in formal techniques that do not support hierarchical structuring nor refinement nor decomposition operators. As a consequence, dealing with such complex expressions makes the proof hard. However, one could also use a binary decomposition following the structure of the CTT operator, but it will need more refinements steps.

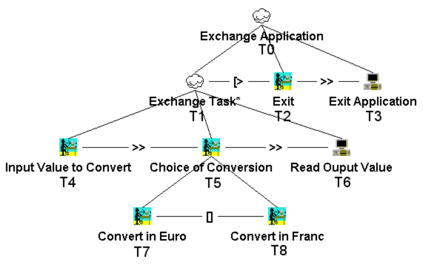


Figure 4. Exchange application's CTT task model.

4.2 Experimental approach

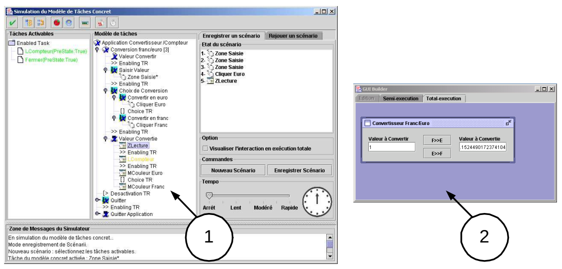
User validation with the experimental approach leans on the formal specification issued from the previously defined formal approach. SUIDT incorporates task-based analysis into our system by the way of a two level task model based on the CTT language. These two levels correspond to the intermediate and to the lower levels of the generic MBS architecture. They allow to take into account the user needs and to realize a successive validation in two steps.

Functional Validation. The intermediate level of a task model allows to validate the user point of view on the functional core features. In fact, in SUIDT, this validation level corresponds to higher level models and allows modelling the goals of the user over the interactive application. More concretely, SUIDT links together the domain model and the task model.

The dynamics of the task model is based on precedence constraints (temporal operators), on the guards (playing the role of pre-conditions) of the functional core functions and on the task post-conditions which permit to modify the state of the functional core expressed at the leaves of the task model.

At this intermediate level of the task model, two main results are obtained. On the one hand, it is possible to test the functionalities of the functional core in order to evaluate if the user needs are satisfied at any abstraction level of specification. On the other hand, it is possible to record scenarios for further tests of integration.

User Interface Validation. In a second step, a graphical interface, designed thanks to a classical GUI-Builder, is associated with the intermediate task model level in order to obtain a lower level model. This model is a refinement of the state of the previous task model, where every interactive or application task of the CTT is described in terms of concrete interaction or application objects. Refinement of a state consists in adding other state variables related to interaction (the event B dialog controller of section 3.1). Since we deal with a single environment, it becomes possible now to relate these interactive or application tasks of the CTT to the concrete GUI level.



En simulation du modèle de tâches concret...
Mode enregistrement de Scénarii.
Nouveau scénario : sélectionnez les tâches activables.
Tâche du modèle concret activée : Zone Saisie*

Figure 5. Experimental approach : SUIDT validation and verification point of view.

As result of the SUIDT approach, every model is an executable model since the functional core plays a central role. In fact, every model is related to the functional core, which allows to run an actual program each time the task model is run. Thus, the running context (during test phase) is not re-initialized or lost when the designer switches to the design mode. So, it is easier for him/her to validate the prototype. Moreover, as for tools like MASTERMIND or Petshop, our application allows to test the program under construction.

In figure 5, we can see a view of the simulation tool. On the tag 1, the simulator tool permits to animate the task model in concordance with the user interface (tag 2). The designer can directly test the interactive application either by interaction on user interface (tag 2) or by animation of the task model (tag 1).

Application to the case study. To illustrate our experimental approach for the validation and verification process, the SUIDT approach is applied on the case study. In the first place, the SUIDT approach allows to check some of the properties available in the formal approach. We can verify with a simulator tool (available on the SUIDT tool) the correctness of the task model (reachability for example). More concretely, in this case study, thanks to the state of the conversion button (disabled or not), we can check if the convert interaction is available or not. The state variables of the dialog controller can be observed.

Meanwhile, this experimental technique is useful for validation of user behavior and validation of ergonomic properties. We could have checked the user behavior by spying the users with a camera device. From the analysis (need of psychologist expertise) of the recording step, we could updated the UI design in setting the interface design. The execution of the application makes it possible to apply different ergonomic criteria [Van94] to check ergonomic properties. For example verifying if the coma for the euro or French franc digit is satisfiable. This kind of properties can not be checked using formal method, therefore experimental validation is required as well as formal validation.

5 Conclusion

We have shown in this paper that it is possible to perform formal and experimental UI validation starting from a single and common core (kernel) model. This model is a formal representation of the dialog controller of an UI. The benefits of this approach can be summarized as follows.

First, the wide spectrum of HCI properties is covered using this approach. Indeed, not all the properties can be checked using a single approach. The fact that formal and experimental validation are performed simultaneously and at the same abstraction level ensures the properties preservation. The use of different validation techniques contributes towards a consistent cooperation of heterogeneous techniques. Moreover, since one single formal model is used for both formal and experimental validation, there is no doubt about the appliance of the validation and/or invalidation properties. Indeed, both formal and experimental properties are checked on the central model. The proposed approach preserves consistency of the set of properties. This result addresses the problem of **horizontal heterogeneity**.

Second, the model of the dialog controller may be formalized at different abstraction levels provided that the modelling language allows these abstraction levels. This is the case for event B where refinement allows to go from one abstract level to a less abstract one (may be a program). Validation can be performed at any development stage and particularly at an early development step allowing savings at development. This result is contribution to solve **vertical heterogeneity.**

Finally, this approach is tool supported. Indeed, the Atelier B tool and the SUIDT model based tool are used to illustrate this approach. We are aware that the presented case study is simple, but it shows its feasibility, its extensibility and the possibility to scale up since the developed approach is generic.

This approach can be easily extended in order to handle testing of formal specification at any abstract level. A first step was described in [AABG$^+$04]. Moreover, when other MBS are used for experimentation, it seems that it is possible to check the plasticity of UI and extract sufficient logical conditions, in the form of properties, that make it possible to use a given UI presentation that uses the same dialog controller in place of another one.

References

[AAB04] Yamine Aït-Ameur and Mickaël Baron. Formal validation of ctt user task models using the proof based formal event b method. *Submitted for journal publication in Interacting with Computers. Journal of Human-Computer Interaction*, 2004.

[AABG03] Yamine Aït-Ameur, Mickaël Baron, and Patrick Girard. Formal validation of hci user tasks. In Al-Ani Ban, Arabnia H.R, and Mum Youngsong, editors, *The 2003 International Conference on Software Engineering Research and Practice - SERP 2003*, volume 2, pages 732–738, Las Vegas, Nevada USA, 2003. CSREA Press.

[AABG$^+$04] Yamine Aït-Ameur, Benoît Bréholée, Patrick Girard, Laurent Guittet, and Françis Jambon. Formal verification and validation of interactive systems specifications. from informal specitications to formal validation. In *Conference of Human Error, Safety and Systems Development, HESSD, Toulouse*, 8 2004.

[AAGJ98] Yamine Aït-Ameur, Patrick Girard, and Francis Jambon. Using the b formal approach for incremental specification design of interactive systems. In Stéphane Chatty and Prasun Dewan, editors, *Engineering for Human-Computer Interaction*, volume 22, pages 91–108. Kluwer Academic Publishers, 1998.

[Abr96] J.R. Abrial. *The B Book. Assigning Programs to Meanings.* Cambridge University Press, 1996.

[Bar03] Mickaël Baron. *Vers une approche sûre du développement des Interfaces Homme-Machine (Thesis).* Thèse de doctorat, Université de Poitiers, 2003.

[BHKN96] H. Balzert, F. Hofmann, V. Kruschinski, and C. Niemann. The janus application development environment-generating more than the user interface. In Jean Vanderdonckt, editor, *Computer-Aided Design of User iterface (CADUI'96)*, pages 183–206, Namur, Belgium, 1996. Presse Universitaire de Namur.

[Bjo87] D. Bjorner. VDM a Formal Method at Work. In Springer-Verlag. LNCS, editor, *Proc. of VDM Europe Symposium'87*, 1987.

[BPR$^+$91] l. Bass, R. Pellegrino, S. Reed, S. Sheppard, and M. Szezur. The arch model : Seeheim revisited. In *User Interface Developper's Workshop*, 1991.

[CH97] J.C. Campos and M.D. Harrison. Formally Verifying Interactive Systems: A Review. In *Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'97)*, pages 109–124. Springer Verlag, 1997.

[Cle97] ClearSy. Atelier b - version 3.5, 1997.

[Cou87] J. Coutaz. Pac, an implementation model for the user interface. In *IFIP TC13 Human-Computer Interaction (INTERACT'87)*, pages 431–436, Stuttgart, 1987. North-Holland.

[D'A98] B. D'Ausbourg. Using Model Checking for the Automatic Validation of User Interface Systems. In *Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'98)*, pages 242–260. Springer Verlag, 1998.

[DFAB93] Alan Dix, Janet Finlay, Gregory Abowd, and Rusell Beale. *Human-Computer Interaction.* Prentice Hall, 1993.

[DH93a] D. Duke and M. D. Harrison. Abstract Interaction Objects. In *Proceedings of Eurographics conference and computer graphics forum*, volume 12, pages 25–36, 1993.

[DH93b] D.J. Duke and M.D. Harrison. Towards a Theory of Interactors. Technical report, Amodeus Esprit Basic Research Project 7040, System Modelling/WP6, 1993.

[DH95] D Duke and M. D. Harrison. Event model of human-system interaction. *IEEE Software*, 1(10):3–10, 1995.

[Dij76] E.W. Dijkstra. In *A Discipline of Programming*. Prentice-Hall Englewood Cliffs, 1976.

[GEM94] Phil Gray, David England, and Steve McGowan. Xuan: Enhancing the uan to capture temporal relation among actions. Department research report IS-94-02, Department of Computing Science, University of Glasgow, February 1994. Modifications par rapport à UAN : - Aspect symétrique USER/SYSTEM - Contraintes temporelles - Paramètrisation des tâches - Pré et Post-conditions.

[Gui95] Laurent Guittet. *Contribution à l'Ingénierie des Interfaces Homme-Machine - Théorie des Interacteurs et Architecture H4 dans le système NODAOO*. Doctorat d'université (phd thesis), Université de Poitiers, 1995.

[Hoa69] C.A.R. Hoare. An Axiomatic Basis for Computer Programming. *CACM*, 12(10):576–583, 1969.

[Nav01] David Navarre. *Contribution à l'ingénierie en Interaction Homme-Machine*. Doctorat d'université (phd thesis), Université Toulouse 3, 2001.

[PBS95] P. Palanque, R. Bastide, and V. Sengès. Validating Interactive System Design Through the Verification of Formal Task and System Models. In *IFIP TC2/WG2.7 Engineering for Human-Computer Interaction*, pages 189–212, 1995.

[PE98] A. Puerta and J. Eisenstein. Interactively mapping task model to interfaces in mobi-d. In Panos Markopoulos and Peter Johnson, editors, *Eurographics Workshop on Design, Specification and Validation of Interactive Systems (DSV-IS'98)*, volume Proceedings, pages 261–274, Abingdon, UK, 1998.

[PF92] F. Paterno and G. Faconti. On the LOTOS Use to Describe Graphical Interaction. In *Proceedings of HCI, People and Computer*, pages 155–173. Cambridge University Press, 1992.

[PMG01] F Paternò, G Mori, and R Galimberti. Ctte: An environment for analysis and development of task models of cooperative applications. In *ACM CHI 2001*, volume 2, Seattle, 2001. ACM/SIGCHI.

[Pue96] Angel Puerta. The mecano project : comprehensive and integrated support for model-based interface development. In Jean Vanderdonckt, editor, *Computer-Aided Design of User interface (CADUI'96)*, pages 19–35, Namur, Belgium, 1996. Presse Universitaire de Namur.

[SPG89] D L Scapin and C Pierret-Golbreich. Mad : Une méthode analytique de description des tâches. In *Colloque sur l'Ingénierie des Interfaces Homme-Machine (IHM'89)*, pages 131–148, Sophia-Antipolis, France, 1989.

[Spi88] J M. Spivey. *The Z notation: A Reference Manual*. Prentice–Hall Int., 1988.

[SSC$^+$95] P. Szekely, P. Sukaviriya, P. Castells, J. Muthukumarasamy, and E. Salcher. Declarative interface models for user interface construction tools : the mastermind approach. In Leonard J Bass and Claus Unger, editors, *IFIP TC2-WG2.7 Working Conference on Engineering for Human-Computer Interaction (EHCI'95)*, pages 120–150, Grand Targhee Resort (Yellowstone Park), USA, 1995. Chapman and Hall.

[Sze96] P. Szekely. *Retrospective and challenge for Model Based Interface Development*, pages 1–27. SpringerComputerScience. Springer-Verlag, Namur, Belgium, 1996.

[Van94] Jean Vanderdonckt. *Guide ergonomique des interfaces homme-machine*, volume 1. Presses Universitaires de Namur, 1994.