FORMAL VERIFICATION AND VALIDATION OF INTERACTIVE SYSTEMS SPECIFICATIONS

From Informal Specifications to Formal Validation

Yamine Aït-Ameur[1], Benoit Breholée[2], Patrick Girard[1],
Laurent Guittet[1] And Francis Jambon[3]
[1] *LISI/ENSMA, BP 40109, Téléport 2, 86961 Futuroscope cedex, France*
[2] *ONERA-CERT-DTIM, 2 Avenue Edouard Belin, BP 4025, 31055 Toulouse cedex, France*
[3] *CLIPS-IMAG, BP 53, 291 avenue de la bibltothèque, 38041 Grenoble cedex 9, France*

E-mail: {yamine, girard, guittet}@ensma.fr, breholee@cert.fr, francis.jambon@imag.fr

Abstract: This paper proposes a development process for interactive systems based both on verification and validation methods. Our approach is formal and use at first the B Method. We show in this paper how formal B specifications can be derived from informal requirements in the informal notation UAN. Then, these B specifications are validated using the data oriented specification language EXPRESS. Several scenarios can be tested against these EXPRESS specifications.

Key words: B Method; EXPRESS; UAN; interaction properties; verification; validation; formal specification of interactive systems.

1. INTRODUCTION

Graphical user interfaces relying mostly on software, are being more and more used for safety-critical interactive systems –for example aircraft glass cockpits– the failure of which can cause injury or death to human beings. Consequently, as well as hardware, the software of these interactive systems needs a high level of dependability. Besides, on the one hand, the design process must insure the reliability of the system features in order to prevent disastrous breakdowns. On the other hand, the usability of the interactive system must be carefully carried out to avoid user misunderstanding that can trigger similar disastrous effects. So, the software dependability of these

safety-critical interactive systems rely as well on safety as on usability properties. Our work focuses on the use of formal techniques in order to increase the quality of HCI software and of all the processes resulting from the development, verification, design and validation activities.

In past workshops and conferences, we presented our approach through papers dealing with formal specifications of HCI software (Aït-Ameur et al. 1998a), formal verification of HCI software (Aït-Ameur et al. 1998), test based validation of existing applications (Jambon et al. 1999). This paper addresses another topic not tackled yet by our approach: design and formal validation of formal specifications with respect to informal requirements. This work completes the whole development process of a HCI software. Indeed, our approach uses the B formal technique for representing, verifying and refining specifications (Aït-Ameur et al. 1998a, Aït-Ameur et al. 1998, Jambon et al. 1999), test based validation of existing applications (Jambon et al. 1999), secure code generation (Jambon 2002) and integration of formal approaches (Girard et al. 2003).

This paper starts from the translation of the requirements in the UAN notation (Hix and Hartson 1993) and shows how B specifications can be derived from. Then, the EXPRESS formal data modeling language (EXPRESS 1994) is put into practice for the validation of the derived B specifications. We show how the B specifications can be translated to EXPRESS code which allows validation.

This paper is structured as follows. Section 2 reviews the different notations and formal techniques that have been experienced on HCI. Our approach and the case study –used to illustrate our approach– are also described in this section. Next section gives the UAN representation of the case study requirements. Section 4 presents the B technique and the specifications of the case study in B. Section 5 is related to validation. It presents the formal data modeling technique EXPRESS which allows the validation of the B specifications. We show how an automatic translation from B to EXPRESS can be performed and how this technique is applied to our case study. The result is a set of EXPRESS entities that are checked against various scenarios. Last, we conclude on the whole suggested approach.

2. NOTATIONS AND TECHNIQUES IN HCI: A BRIEF STATE OF THE ART

2.1 Notations & Formal techniques

In order to express HCI software requirements, several notations were suggested. As examples, MAD (for "Méthode Analytique de Description") (Scapin and Pierret-Golbreich 1990) and HTA (for Hierarchical Task Analysis) (Shepherd 1989) use a hierarchical decomposition of user tasks. On the other side, a notation like UAN (Hix and Hartson 1993) and its extension XUAN (Gray et al. 1994) allow the description of not only the interface feedback, but of the user behaviors as well. UAN specifications record the state of the interface and tasks are described as state evolutions. This state orientation of UAN facilitates translation to state based formal techniques –B for example.

Several techniques were used in the HCI area. These techniques differ from some point of views: semantics –algebraic or state based– verification –incremental proof or fully automatic proof– etc. Some of these techniques can be summarized in the following.

On the one hand, the first techniques are state based. They were based on automata through statecharts (Wellner 1989) and ATN (Waserman 1981) (Guittet 1995), Petri Nets (Accot et al. 1996) (Navarre et al. 2000). They have been extended to support temporal logics to allow automatic model checking like in CTL* (Paternò and Mezzanotte 1995), XTL (Brun 1997) and SMV (Clarke et al. 1986, McMillian 1992), or with the Lustre language (Roché 1998). The previous techniques support code generation and automatic proving. Other techniques supporting state based semantics and incremental proving and refinement like Z (Johnson 1995), VDM (Marshall 1986) or B (Aït-Ameur et al. 1998) were suggested.

On the second hand algebraic techniques have been applied with LOTOS (Paternò and Faconti 1992) for describing HCI software. The proofs are achieved by rewriting and refinement is performed by transformation. Other techniques based on higher order type systems have been experienced.

All these techniques cover a limited part of the development of an HCI. Our approach does not use only one technique, but it suggests to use several techniques which cooperate, choosing each technique where it has proved to be most efficient.

2.2 Our approach

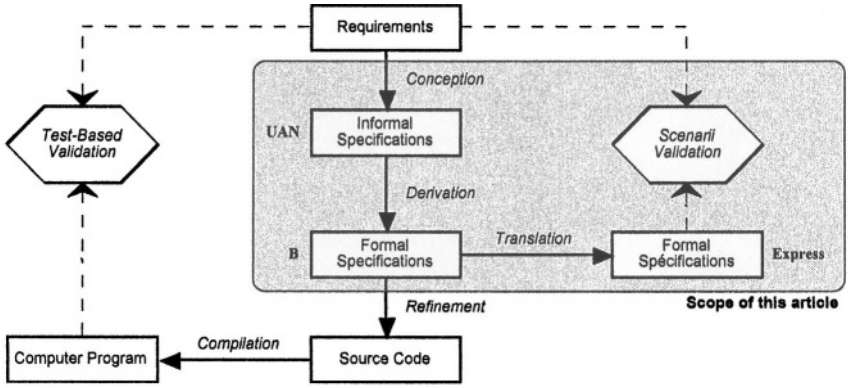

Figure 1: Scope of this article in the approach we suggest for handling the development and validation of HCI

Our approach uses the B technique. B supports formal specifications, refinement from specifications to code and property verification through the proof of the generated proof obligations. Specifications are derived from the informal UAN notation and are validated using the EXPRESS data modeling language.

Formal specifications, property verification and refinement from specification to code have been have been presented in (Aït-Ameur et al. 1998a ,Aït-Ameur et al. 1998, Jambon et al. 1999) respectively. This paper presents the last point: deriving specifications from semi-formal notations and their validation in EXPRESS. This paper completes the whole developed approach described in figure 1.

2.3 The case study: the *Rangeslider*

An usual slider –with a single cursor– is a graphical toolkit widget used by interface designers to allow the specification of a value in an interval. The *Rangeslider* (Ahlberg and Truve 1995) used by Spotfire™ (http://www.spotfire.com) is an enhanced version of this classical slider, i.e., it supplies two cursors –see fig. 2– in order to allow users to select not only a single value, but a range of values. This new widget is used by interface designers to implement easy-to-use zoom or filtering functions. A *Rangerslider* user can interact with the widget by the way of three different kinds of actions:

– **Move one cursor:** the user moves one of the two cursors, to the left or to the right. As a consequence, the area of the center zone expands or

reduces. The moved cursor cannot cover over the other cursor nor exceed the widget length.

- **Move the center zone:** the user moves the center zone, and at the same time both cursors come after it. So the area of the center zone remains unchanged. No cursor can exceed the widget length.
- **Select a value in outer zones:** the user clicks in one of the outer zones – MinZone or MaxZone– and the closest cursor moves at the selected point. As a consequence, the area of the center zone expands or reduces.

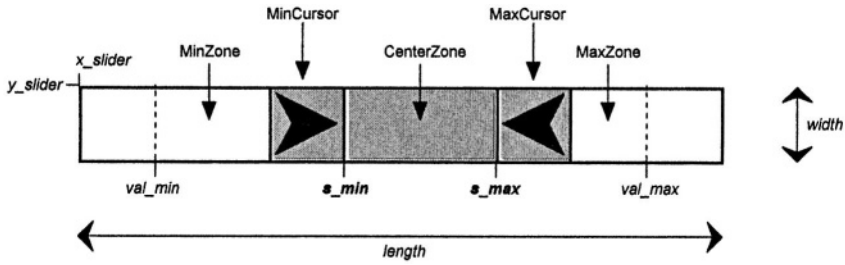

Figure 2: RangeSlider scheme —with variables names used both in the UAN, B and EXPRESS specifications.

This case study was proposed in the French working group ALF (Architectures, Languages and Formalisms) to study the expressiveness of a wide range of formalisms. Some interesting results, based on this case study, have already been proposed (Navarre et al. 2000).

3. THE USER ACTION NOTATION

The User Action Notation is an interaction-design notation. Hix et al. suggest that *"the UAN is intended to be written primarily by someone designing the interaction component of an interface, and to be read by all developers, particularly those designing and implementing the user interface software"* (Hix and Hartson 1993). The UAN is user- and task-oriented. A UAN specification describes, at the physical level, the user actions and their corresponding interface feedback and state changes.

The three tables below –table 2 to table 4– are the UAN specifications of the three user interactions described in §3.1. In fact, a full UAN specification must comprise five tables –one table for each user interaction. However the two pairs of UAN tables for cursor and outer zones are so similar that one table of each pair has been omitted. In these tables, *Rangeslider* is the name

of the whole slider object and Δx is the spatial increment on the abscissa axis.

In order to move the left cursor –MinCursor– the user must move the mouse button in the context of the MinCursor object. Then, he can depress the mouse button and drag the cursor. The MinCursor follows the mouse pointer and the center zone must be redisplayed. At each increment, the value of the `s_min` variable is updated.

Table 2: UAN specification of the task "move MinCursor"

TASK: **move MinCursor**		
USER ACTIONS	**INTERFACE FEEDBACK**	**INTERFACE STATE**
~[MinCursor] Mv	MinCursor !	
~[x,y in RangeSlider]*	0<x<s_max : MinCursor > ~ redisplay(CenterZone)	s_min=s_min+Δx
M^	MinCursor -!	

In order to move the center zone –CenterZone– the user must move the mouse button in the context of the CenterZone object. Then, he can depress the mouse button and drag the zone. The zone follows the mouse pointer and both cursors must be redisplayed. At each increment, the value of the `s_min` and `s_max` variables are updated.

Table 3: UAN specification of the task "move CenterZone"

TASK: **move CenterZone**		
USER ACTIONS	**INTERFACE FEEDBACK**	**INTERFACE STATE**
~[CenterZone] Mv	CenterZone !	
~[x,y in RangeSlider]*	s_min<x<s_max : CenterZone > ~ redisplay(MinCursor) redisplay(MaxCursor)	s_min=s_min+Δx s_max=s_max+Δx
M^	CenterZone -!	

In order to select a value in an outer zone –MinZone– the user must move the mouse button in the context of the MinZone object. Then, he depresses the mouse button. At this point, the left cursor –MinCursor– as well as the center zone must be redisplayed at the new position.

Table 4: UAN specification of the task "select a value in MinZone"

TASK: **select value in MinZone**		
USER ACTIONS	**INTERFACE FEEDBACK**	**INTERFACE STATE**

~[x,y in MinZone] Mv^	redisplay(MinCursor)@x,y redisplay(CenterZone)	s_min=s_min+Δx

These UAN tables, together with the figure 2 are the high level and informal specifications of the Rangeslider. These specifications are very useful for interface designers because they express in a rather short and precise way the behavior of the RangeSlider. However, the UAN is a notation to express requirements but cannot be used to prove or test the features of the interaction object we analyze. As an example, we cannot be sure that the cursor will never cover over the other cursor nor exceed the widget length. So we must now use formal methods to prove this kind of properties.

4. THE B TECHNIQUE: FORMAL SPECIFICATION

The B method, as VDM or Z, is based on model description. Like VDM, B uses preconditions and post-conditions (Hoare 1969, Hoare et al. 1987). Moreover, B is based on the weakest precondition technique of Dijkstra (Dijkstra 1976). Starting from this method, J.R. Abrial (Abrial 1996) has defined a logical calculus, named the generalized substitutions calculus. Our choice had been motivated by the fact that B is supported by tools (ClearSy 1997) which allow a complete formal development.

The following abstract machine describes what a set of range sliders is. It describes the set of all the sliders to be `SLIDERS` and two constants describing the length and the width of the screen. The `PROPERTIES` clause types these two constants and gives their corresponding values.

```
MACHINE
  the_slider
SETS
  SLIDERS
CONSTANTS
  screen_width, screen_height
PROPERTIES
  screen_width : NAT  ∧  screen_width = 800 ∧
  screen_height : NAT ∧  screen_height = 600
```

The model of this abstract machine is given by the attributes defined in the VARIABLES clause. The set `sliders` describes the set of the actually described range sliders. The other variables allow to access the attributes of a given range slider.

Informally, as described in figure 2, each range slider is characterized by:

- `x_slider` and `y_slider` are the coordinates of the up left corner of the window describing the range slider,
- `width` and `length` are respectively the width and the length of a given range slider,
- `val_min` and `val_max` are the minimal and maximal values associated to a range slider,
- and finally, `s_min` and `s_max` are the current low and up values of the described range slider.

```
VARIABLES
  sliders, x_slider, y_slider, width, length,
  val_min, val_max, s_min, s_max
```

All these variables are typed in the `INVARIANT` clause. This clause contains the properties that are always satisfied by the variables of the model. These properties shall be maintained by the operations that affect these variables. Two kinds of properties are described:

- typing properties that give types to the variables. The set `sliders` is declared as a subset of the set `SLIDERS`. Then, all the other variables are accessing functions and they are typed by their signature,
- safety properties which ensure a set of critical properties and model consistance. They are described in first order logic and are maintained by the B prover. They assert that the low (resp. Up) value of a slider shall be greater (resp. Lower) or equal to the minimal (resp. maximal) value of the range slider. Moreover, it states that the whole range slider is contained in the screen dimensions. This last assertion ensures visibility and reachability properties.

In the B language, these properties are described by:

```
INVARIANT
  sliders ⊂ SLIDERS ∧
  x_slider  ∈ sliders-->NAT ∧ y_slider  ∈ sliders-->NAT ∧
  width     ∈ sliders-->NAT ∧ length ∈ sliders-->NAT ∧
  val_min   ∈ sliders-->NAT ∧ val_max   ∈ sliders-->NAT ∧
  s_min     ∈ sliders-->NAT ∧ s_max     ∈ sliders-->NAT ∧
/* Safety properties of the slider */
  ∀ sl.(sl:sliders => (val_min(sl) >= 0)) ∧
  ∀ sl.(sl:sliders => (val_min(sl) <= s_min(sl))) ∧
  ∀ sl.(sl:sliders => (s_min(sl)<s_max(sl))) ∧
  ∀ sl.(sl:sliders => (s_max(sl) <= val_max(sl))) ∧
  ∀ sl.(sl:sliders => (val_max(sl) <= length(sl))) ∧
  ∀ sl.(sl:sliders => (x_slider(sl) ∈ 1..screen_width)) ∧
  ∀ sl.(sl:sliders => (y_slider(sl) ∈ 1..screen_height)) ∧
```

```
∀ sl.(sl:sliders => (x_slider(sl)+length(sl) ∈ 1..screen_width)) ∧
∀ sl.(sl:sliders => (y_slider(sl)+width(sl) ∈ 1..screen_height))
```

The first operation allows to create a range slider with XX, YY as coordinates of its left up corner. Its length and width are respectively given by the parameters LENGTH and WIDTH. Finally, VMIN and VMAX parameters indicates the minimal and maximal values of the range. The slider is created with VMIN and VMAX as initial minimal and maximal values. A preconditon ensures that the parameters are correctly typed and the invariant is maintained. It ensures that the creation of a range slider is correctly performed.

```
OPERATIONS
 create(XX,YY,LENGTH,WIDTH,VMIN,VMAX)=
 PRE
    sliders ≠ SLIDERS ∧
    XX ∈ NAT ∧ YY ∈ NAT ∧ WIDTH ∈ NAT ∧ LENGTH ∈ NAT ∧
    VMIN ∈ NAT ∧ VMAX ∈ NAT ∧ VMIN>=0 ∧ VMIN<VMAX ∧ VMAX<=LENGTH ∧
    XX ∈ 1..screen_width ∧    YY ∈ 1..screen_height ∧
    XX+LENGTH ∈ 1..screen_width ∧ YY+WIDTH ∈ 1..screen_height
 THEN
    ANY sl
    WHERE sl ∈ SLIDERS - sliders
    THEN
       sliders := sliders ∪ {sl}  ||
       x_slider(sl):=XX  ||  y_slider(sl):=YY  ||
       length(sl):=LENGTH  ||  width(sl):=WIDTH  ||
       val_min(sl):=VMIN  ||  val_max(sl):=VMAX  ||
       s_min(sl):=VMIN  ||  s_max(sl):=VMAX
    END
 END;
```

In order to keep this paper in a reasonable length, we show only one operation that manipulates the range slider. It allows to move the left value of the range slider to the left. In B this operation is described by:

```
move_left_slider(one_slider, new_left_min_value)=
PRE
   one_slider ∈ sliders ∧ new_left_min_value ∈ NAT ∧
   new_left_min_value > val_min(one_slider) ∧
   new_left_min_value < s_max(one_slider)
THEN
   s_min(one_slider) := new_left_min_value
END;
```

Other operations related to the range slider have been described in this abstract machine. Moreover, the whole application is represented by several

abstract machines not presented in this paper. Indeed, abstract machines related to the mouse management, to the direct manipulation and so on have been described. Finally, notice that the abstract machine described in B and presented in this paper has shown that it is possible to:

- ensure that a range slider remains in the screen limits,
- ensure that the low and up values of a range slider respect the definition of a range,
- move the low value, of a range slider to the left in order to decrease its left value, by running the corresponding operation.

For the whole developed abstract machine, the proof obligations have been generated. They all have been automatically proved. However, this specification has not been built at the first attempt. We had to enrich the preconditions and to remove other preconditions. Indeed, the prover behaves following:

- preconditions are not complete, therefore the proof cannot be achieved,
- preconditions are contradictory, then the user has to make new choices and to check the requirements.

Finally, about 40 proof obligations are generated for this application. We had to prove only 2 proof obligations using the interactive prover, i.e., "by hand". This shows that when the application is well specified following sound software engineering concepts, the proof phase can be considerably reduced.

All these properties are safety properties. In the next section we address the problem of the validation of such formal specifications that is not supported by the B formal technique.

5. THE EXPRESS LANGUAGE: VALIDATION

The EXPRESS language specifies formal data models. The language focuses on the definition of entities –types– which represent the objects –classes– we want to describe. EXPRESS is type oriented: entity types are defined at compile time and there is no concept of meta-class. Each entity is described by a set of characteristics called attributes. These attributes are characterized by a domain and constraints on these domains. An important aspect of these entities is that they are hierarchically structured allowing multiple inheritance as in several object oriented languages. This part of the specification describes the structural and the descriptive parts of the domain knowledge.

On the other hand, it is possible to describe processes on the data defined in the entities by introducing functions and procedures. These functions are used to define constraints, pre-conditions and post-conditions on the data. They are also used to specify how the values of some properties that may be derived from the values of other properties. This part of the specification describes the procedural part of the domain knowledge.

Finally, in EXPRESS, entities –data– and functions –processes– are embedded in a structure called a SCHEMA. These schemes may reference each other allowing a kind of modularity and therefore specification in the large possibilities. More details about the definition of this language can be found in (Schenck, Wilson 1994, Bouazza 1995).

5.1 Translation of B specifications to EXPRESS

The translation from B specifications to EXPRESS code is based on the semantics of generalized substitutions on which B is built. The idea consists in:

- representing the state variables of the model by an EXPRESS entity. This entity describes a state in the underlying transition system. According to the B semantics, this transition system describes the semantic model of the developed application,
- representing the invariant properties by global EXPRESS rules. Indeed, the properties that are described in the `INVARIANT` B clause are global properties that need to be satisfied at each state,
- and finally, representing operations by entities expressing the initial and the final states with local rules that express the relationship between the initial and the final states.

All the objects that are defined in an abstract machine are translated into EXPRESS. Each abstract machine corresponds to one EXPRESS schema.

5.2 The case study in EXPRESS

The following EXPRESS entity defines the model associated to the abstract machine described in B. It is obtained by a translation of all the variables that are described in the `VARIABLES` B clause.

```
SCHEMA The_Slider;

ENTITY Slider;
  x_slider,y_slider  :INTEGER;
  width, length      :INTEGER;
```

```
  val_min, val_max  :INTEGER;
  s_min, s_max      :INTEGER;
END_ENTITY;
```

For a given range slider, the previous entity describes the `x_slider` and `y_slider` representing its coordinates, its width and length, its minimal and maximal values and finally its low and up values. The instanciation of this entity allows to create a range slider. This entity preserves the identifiers introduced in the B abstract machine. Moreover, it encodes all the invariant properties which are related to typing of the variables.

The other invariant clauses that are related to the universally quantified properties, which express safety properties, are represented by a global EXPRESS rule. This rule expresses that all the instances of the entity `slider` satisfy the expressed logical properties. It states that the set of all the instances of a range slider satisfying these properties is exactly the set of all instances of a range slider. It is given by:

```
RULE coord FOR (Slider);
LOCAL
  sliders_ok, ens_sliders :
  SET OF Slider := [] ;
END_LOCAL;
  sliders_ok := QUERY(s <* Slider | ((s.val_min >=0) AND
  (s.val_min <= s.s_min) AND (s.s_min < s.s_max) AND
  (s.s_max <= s.val_max) AND (s.val_max <= s.length) AND
  (s.x_slider >= 0) AND (s.x_slider + s.length < 800) AND
  (s.y_slider >= 0) AND (s.y_slider + s.width < 600)));
  ens_sliders := QUERY(s <* Slider | true);
WHERE
  sliders_ok = ens_sliders ;
END_RULE;
```

Finally, operations are also transformed into an EXPRESS entity. The translation principle is based on the semantics of B. Indeed, the entity slider expresses the state of the described system (state based formal semantics). So, an operation, acting on a state, transforms an initial state `Ei` to a final state `Ef`.

The operation `move_left_slider` considers two states: the initial state `Ei` and the final state `Ef` and its input parameter, namely `new_left_min_value`. The description of this entity is given by:

```
ENTITY Move_Left_Slider;
  -- states
  Ei, Ef : Slider ;
  -- input parameter
  new_left_min_value : INTEGER;
```

The next part completes the description of an operation by an entity. It translates the precondition part (expressed by the B keyword `PRE`), the effect of the operation by expressing the change of `s_min` in the final state and finally it states the unchanged attributes in final state. The result gives the following `WHERE` rules.

```
WHERE
-- Translation of preconditions
      pre1: new_left_min_value >= Ei.val_min ;
      pre2: new_left_min_value < Ei.s_max ;
-- Translation of perations
    ope1: Ef.s_min = new_left_min_value;
-- Translation of unchanged state variables
      cst1: Ef.x_slider = Ei.x_slider ;
      cst2: Ef.y_slider = Ei.y_slider ;
      cst3: Ef.width = Ei.width ;
      cst4: Ef.length = Ei.length ;
      cst5: Ef.val_min = Ei.val_min ;
      cst6: Ef.val_max = Ei.val_max ;
      cst7: Ef.s_max = Ei.s_max ;
END_ENTITY;
....
END_SCHEMA;
```

This approach shows that it is possible to automatically translate B specifications into EXPRESS data modeling specifications. This translation will allow to give data models that represent specification tests.

5.3 Validation scenarios

In order to describe tests of B specifications –recall that validation and test are not supported by B– we need to describe instantiations of the EXPRESS data model.

As an illustration consider two rangesliders that are described by the same coordinates (`x_slider` =20 and `y_slider`=30), the same length and width (equal to `length`=100 and `width` = 10), the same minimal and maximal values (equal to `val_min`= 40 and `val_max`= 80) and the same up value (equals to `s_max` = 60). Consider that the first range slider RS1 corresponding to the initial state has a low value (equals to `s_min`= 50) and the second range slider RS2 has a low value (equals to `s_min` = 45). In fact this situation corresponds to a moving of the left value of a range slider. It can be expressed as `move_left_slider (RS, 45)`. Here we consider that the range sliders `RS1` and `RS2` corresponds respectively to the range

sliders of the initial and final states. In EXPRESS, this situation corresponds to the description of the three following instances:

```
#1=SLIDER (20, 30, 10, 100, 40 , 80, 50 , 60) ;
#2=SLIDER (20, 30, 10, 100, 40 , 80, 45 , 60) ;
#3=MOVE_LEFT_SLIDER(#1, #2, 45) ;
```

The previous set of instances represent a test case for the `move_left_slider` operation. The method can be generalized to other operations and to compositions of these operations that allow the description of a wide range of user scenarios. The test sequences can then be produced using the UAN specifications described in §3.3. Thanks to these specifications, a wide coverage can be achieved.

6. CONCLUSION

This paper shows a formal technique that allows to derive, verify and validate formal B specifications of HCI software. The informal requirements are expressed using the semi-formal notation UAN which is used as the basis for writing formal specifications. This process is proved helpful for writing formal specifications. Indeed, the direct derivation of these specifications from informal requirements is a hard task. This approach bridges the gap between user oriented specifications which feed the formalization process, the B formal development and verification techniques.

As a second step this paper addresses a crucial issue related to formal validation of formal specifications. It suggests to use a data oriented modeling language, namely EXPRESS, which allows to represent validation scenarios. This approach increases the efficiency of the HCI software development process since validation is not performed at the programming language level but at higher and abstract specifications. This approach allows to validate scenarios of application earlier in the development process. The result increases the efficiency of the development and decreases its cost.

Finally, to end the whole development process we suggest there is a need for taking into account user tasks descriptions and user tasks validations. This topic has not been addressed in this paper but it will be tackled in future developments. Indeed, we think that task representations and validations are possible within the framework we have developed.

REFERENCES

Abrial, J.-R. (1996) *The B Book: Assigning Programs to Meanings.* Cambridge University Press.

Accott, J., Chatty, S. and Palanque, P. (1996) A formal description of low level interaction and its application to multimodal interactive systems. In *Proceedings of Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'96)* (5-7 June, Namur, Belgium), Springer-Verlag, pp. 92-104.

Ahlberg, C. and Truve, S. (1995) Tight Coupling: Guiding User Actions in a Direct Manipulation Retrieval System. In *Proceedings of HCI'95 Conference on People and Computers X,* pp. 305-321.

Aït-Ameur, Y., Girard, P. and Jambon, F. (1998a) A Uniform approach for the Specification and Design of Interactive Systems: the B method. In *Proceedings of Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'98)* (3-5 June, Abingdon, UK), pp. 333-352.

Aït-Ameur, Y., Girard, P. and Jambon, F. (1998) Using the B formal approach for incremental specification design of interactive systems. In *Proc. of Engineering for Human-Computer Interaction,* Kluwer Academic Publishers, pp. 91-108.

Bouazza, M. (1995) *Le langage EXPRESS.* Hermès, Paris.

Brun, P. (1997) XTL: a temporal logic for the formal development of interactive systems. Palanque, P. et Paternò, F. (Ed.). In *Formal Methods for Human-Computer Interaction,* Springer-Verlag, pp. 121-139.

Clarke, E.M., Emerson, E.A. and Sistla, A. P. (1986) Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems.* 2, 8, pp. 244-263.

ClearSy.(1997) *Atelier B - version 3.5.* 1997.

Dijkstra, E. (1976) *A Discipline of Programming.* Prentice Hall, Englewood Cliff (NJ), USA.

EXPRESS. (1994) *The EXPRESS language reference manual.* ISO, 1994 ISO 10303-11.

Girard, P., Baron1, M. and Jambon, F. (2003) Integrating formal approaches in Human-Computer Interaction. In *Proceedings of INTERACT 2003 - Bringing the Bits togETHer - Ninth IFIP TC13 International Conference on Human-Computer Interaction - Workshop Closing the Gaps: Software Engineering and Human-Computer Interaction,* (September 1-5, Zurich, Switzerland).

Gray, P., England, D. and McGowan, S. (1994) *XUAN: Enhancing the UAN to capture temporal relation among actions.* Department of Computing Science, University of Glasgow, February, Department research report IS-94-02.

Guittet, L. (1995) *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.

Hix, D. and Hartson, H.R. (1993) *Developping user interfaces: Ensuring usability through product & process.* John Wiley & Sons, inc., Newyork, USA.

Hoare, C.A.R. (1969) An Axiomatic Basis for Computer Programming. *CACM.* 12, 10, pp. 576-583.

Hoare, C.A.R., Hayes, I.J., Jifeng, H., Morgan, C.C., Sanders, A.W., Sorensen, I.H., Spivey, J.M. and Sufrin, B.A. (1987) Laws of Programming. *CACM.* 30, 8.

Jambon, F. (2002) From Formal Specifications to Secure Implementations. In *Proceedings of Computer-Aided Design of User Interfaces (CADUI'2002)* (May 15-17, Valenciennes, France), Kluwer Academics, pp. **43-54.**

Jambon, F., Girard, P. and Boisdron, Y. (1999) Dialogue Validation from Task Analysis. In *Proceedings of Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'99)* (2-4 June, Universidade do Minho, Braga, Portugal), Springer-Verlag, pp. 205-224.

Johnson, C.W. (1995) Using Z to support the design of interactive, safety-critical systems. *IEE/BCS Software Engineering Journal.* 10, 2 (March), pp. 49-60.

Marshall, L.S. (1986) *A Formal Description Method for User Interface.* Ph.D Thesis : University of Manchester.

McMillian, K. (1992) *The SMV System.* Carnegie Mellon University, 1992.

Navarre, D., Palanque, P., Bastides, R. and Sy, O. (2000) Structuring interactive systems specifications for executability and prototypability. In *Proceedings of 7th Eurographics workshop on Design, Specification and Verification of Interactive Systems, DSV-IS'2000* (Limerick, Ireland), Springer Verlag.

Paternò, F. and Faconti, G.P. (1992) On the LOTOS use to describe graphical interaction. In Cambridge University Press, pp. 155-173.

Paternò, F. and Mezzanotte, M. (1995) Formal verification of undesired behaviours in the CERD case study. In *Proceedings of IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction (EHCI'95)* (14-18 August, Grand Targhee Resort (Yellowstone Park), USA), Chapman & Hall, 1995, pp. 213-226.

Roché, P. (1998) *Modélisation et validation d'interface homme-machine.* Doctorat d'Université (PhD Thesis) : École Nationale Supérieure de l'Aéronautique et de l'Espace.

Scapin, D.L. and Pierret-Golbreich, C. (1990) Towards a method for task description : MAD. Berliguet, L. et Berthelette, D. (Ed.). In *Working with display units,* Elsevier Science Publishers, North-Holland, pp. 371-380.

Schenck, D. and Wilson, P. (1994) *Information Modelling The EXPRESS Way.* Oxford University Press.

Shepherd, A. (1989) Analysis and training in information technology tasks. Diaper, D. (Ed.). In *Task Analysis for Human-Computer Interaction,* Ellis Horwood, Chichester, USA, pp. 15-55.

Waserman, A. (1981) User Software Engineering and the design of Interactive Systems. In *Proceedings of 5th IEEE International Conference on Software Engineering* ,IEEE society press, 1981, pp. 387-393.

Wellner, P. (1989) StateMaster : a UIMS based on Statecharts for prototyping and target implementation. In *Proceedings of Human Factors in Computing Systems (CHI'89)* (30 April - 4 May, Austin, USA), ACM/SIGCHI, pp. 177-182.