Using the B formal approach for incremental specification design of interactive systems

Yamine Aït-Ameur, Patrick Girard & Francis Jambon
LISI / ENSMA, BP 109, Téléport 2, F-86960 Futuroscope cedex, France
Tel.: (+33/0) 5 49 49 80 63, Fax: (+33/0) 5 49 49 80 64
E-mail: {yamine, girard, jambon}@ensma.fr, Web: http://www.lisi.ensma.fr/cao.html

Abstract: This paper introduces a new technique for the verification of interactive systems. It first presents the use of a model oriented formal method for specifying interactive systems, i.e. the B method. Then, it suggests formally based solutions which allow solving difficulties that are inherent to interactive systems specification, like reachability, observability or reliability. Our claim is that this model-oriented technique that uses proof obligations can be used together with model checking techniques, where automatic proofs of properties can be performed.

Keywords: Model oriented notation, B method, specification refinement, interaction properties verification, specification of interactive system.

1. INTRODUCTION

Formal specification techniques become more and more used in the area of computer science and particularly for the development of secure systems namely "critical systems". These systems require a high level of correctness and consistency, which encourages formal development. Several approaches in formal methods have been proposed in the last decade. They are divided in different categories depending on the kind of semantics they are based on (Brun and Beaudouin-Lafon, 1995, Feather, 1987, Sannella, 1988). Conversely, formal specification of interactive systems can be split into different ways. Historically, the first way concerns the

Aït-Ameur Y., Girard P., & Jambon F. Using the B formal approach for incremental specification design of interactive systems. *IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'98), Heraklion (Crete), Greece*, 14-18 September 1998. p. {to be published}.

dialogue description (Dix et al., 1993). Related to the natural language decomposition, human-computer interactions have been broken down into three hierarchical levels, the *lexical* level, the *syntactic* level and the *semantic* level. The dialogue description mainly concerns the syntactic level. It involved several formalisms, grammars, state-based diagrams or event models (Green, 1986). More recently, the development of GUI-Builders (Graphical User Interface Builders) has considerably increased the graphical power of user interfaces. GUI-Builders allowed rapid prototyping of interactive systems, and authorised incremental development. Nevertheless, lack of precise specifications is becoming a more and more critical issue.

On the one hand, user-centred design leads to semi-formal but easy to understand notations, such as UAN (Hix and Hartson, 1993) and MAD (Scapin and Pierret-Golbreich, 1990) for interactive design, or GOMS (Card et al., 1983) for evaluation. On the other hand, adaptation of well-defined approaches, combined with interactive models, brings partial but positive results. These are for example the interactors and related approaches (Duke and Harrison, 1993b, Paternó, 1994, Palanque, 1992), using model oriented approaches (Duke and Harrison, 1993a), algebraic notations (Paternó and Faconti, 1992), Petri nets (Palanque, 1992) or Temporal Logic (Brun, 1997, Abowd et al., 1995)

In this paper we deal with model oriented formal methods. State transition networks use finite automata to describe the transition systems, Petri nets and temporal logic gave interesting results from the automatic verification point of view by providing automatic algorithms allowing to check critical properties of interactive systems like liveness, reachability, safety and so on. However, building these transition models incrementally is a heavy task. These approaches are based on the building, by composition, of a whole model (transition system) which represent the system to be described. Then, properties can be checked and the transition system is interpreted to generate a program implementing the behaviour it describes. The difficulty in such techniques is to build the whole (Dix et al., 1993). Other model-oriented approaches have been suggested. They are neither based on transition systems nor they provide an automatic procedure for the verification of the properties of the system. These systems are based on the generation of proof obligations that are checked by the user (with the help of a theorem prover). Among these techniques, VDM is based on preconditions and postconditions (Bjorner, 1987) and on the theory of partial functions, Z is based on set theory (Spivey, 1988), and B is based on the weakest precondition calculus (Abrial, 1996). They allow the description, in high abstraction level language, of the different components of a given system. However, these approaches require an important work to describe the different functions of the system.

Our claim is that neither the first approaches nor the second ones solve the whole problems, but the merging of both techniques seems to be a promising approach. Indeed, the first one allows the control of properties by model checking and neglects

Aït-Ameur Y., Girard P., & Jambon F. Using the B formal approach for incremental specification design of interactive systems. *IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'98), Heraklion (Crete), Greece*, 14-18 September 1998. p. {to be published}.

the development aspects. In opposite, the second approach allows a complete development of a system and controls all the properties by the verification of all the proof obligations (automatically or by the user). So, several interaction properties can be proved. The more important advantage of this second technique is that it avoids the state explosion problem that becomes crucial when systems grow and it allows the development of the whole system by refinement. As we stated above, our paper deals with the last category of formal methods. We focus on the B method (Abrial, 1996, Lano, 1996). Compared to VDM and Z, it allows the definition of a constructive process to build whole applications, with the respect of all the rules by the use of a semi-automatic tool (Steria méditerranée, 1997). Finally, notice that we did not address the other approaches mainly the ones based on algebraic techniques, rewriting, or concurrent languages.

This paper is organised as follows. In the next section, we briefly present a case study, which stems from the French group on formal specification for highly interactive systems FLASHI (Palanque and Girard, 1996), and we outline the major properties we want to ensure. Section 3 explains the B-method and presents a short overview of proving methods for interactive properties. Section 4 outlines some aspects of the B language over a well-known example, the mouse. Last, section 5 gives examples of B analysis in the context of our example, and details the usage of the B-tool, that allows proving properties.

2. CASE STUDY

Our case study is a co-operative version of a Post-It®[1] Note software. It appears on the screen as a *block* that can be iconified, but cannot be moved. Clickable areas allow direct manipulation of the block, for iconification, and close (quit). It is possible to enter text into the Post-It® Note on the block, which always exists. Last, it is possible to detach the upper Post-It® Note from the block, and then to drag it anyway. Post-It® Notes have a similar behaviour, that differs in four ways: (1) the detach area is restricted to a drag command area, (2) the close area becomes a kill area, (3) a resize area is defined in the lower-right corner, corresponding to a standard resize behaviour, and (4) a send behaviour is defined. This behaviour consists in emitting the Post-It® Note to a receiver that is visualised as a special icon. The interactive trigger is a drop (mouse up event) onto the icon.

[1] "Post-it" is a registered trademark of 3M

Aït-Ameur Y., Girard P., & Jambon F. Using the B formal approach for incremental specification design of interactive systems. *IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'98), Heraklion (Crete), Greece*, 14-18 September 1998. p. {to be published}.

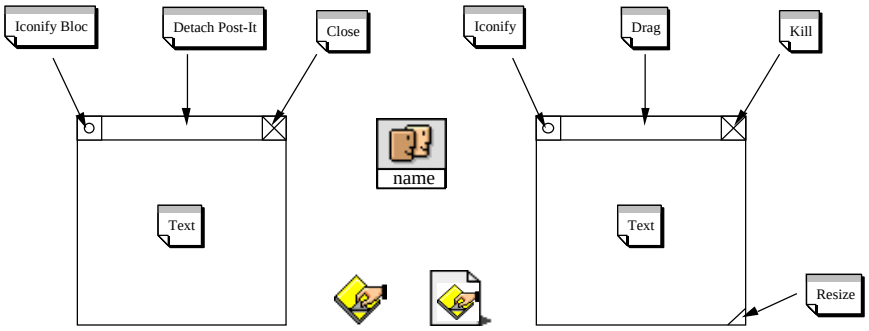

Figure 1. From the left to the right, The Post-It® block, the three icons (Post-It® block, User, and Post-It®), and the Post-It® Note itself, together with his active areas.

Defining good properties of Interactive Systems is an open problem. Several approaches have proposed classifications of properties. For example, (Dix et al., 1993) define three main categories of principles to support usability, which are then refined into about fifteen more specific principles. In their article, J. Campos and M. Harrison (Campos and Harrison, 1997) give a good framework for classifying user interface properties, separating them among three classes that address the user (*visibility*), the user interface (*reachability*), and the underlying system (*reliability*):

- **Visibility** relates to feedback and information perception by the user. In our case study, can we ensure that the user can perceive internal state of the system? Does every Post-It® Note move rely to hidden/exposed status of all the perceivable Post-It® Notes?
- **Reachability** deals with what can be done at the user interface and how can it be done. In our application, it concerns for example the way we can move the Post-It® Notes. How can the mouse be taken into account? Is it always possible to move a Post-It® Note? Can it move away from the screen?
- **Reliability** concerns the way the interface works with the underlying system. Does some predicate on the state of the system always hold? Is it possible to ensure that an empty Post-It® Note cannot be sent?

3. THE B METHOD AND INTERACTION PROPERTIES

The B method allows the description of different modules, i.e., abstract machines that are combined by programming in the large operators. This combination allows building complex systems incrementally and correctly when all

the proof obligations are proved. Moreover, the interest of this method in our case is the tool it provides.

3.1 The abstract machine notation

The abstract machine notation is the basic mechanism of the B method. J. R. Abrial defined three kinds of machines identified by the keywords MACHINE, REFINEMENT and IMPLEMENTATION. The first one represents the high level of specification. It expresses formal specification in a high abstract level language. The second one defines the different intermediate steps of refinement and finally the third one reaches the implementation level. Note that the development is considered to be correct only when every refinement is proved to be correct with respect to the semantics of the B language. Gluing invariants between the different MACHINES of a development are defined and sets of proof obligations are generated. They are used to prove the development correctness. A theorem prover including set theory, predicate logic and the possibility to define other theories by the user, achieves the proof of these proof obligations. The proving phase is achieved either automatically, by the theorem prover, or by the user with the interactive theorem prover.

3.2 Description of abstract machines

J. R. Abrial has described several important clauses for the definition of abstract machines. Depending on the clauses and on their abstraction level, these clauses can be used at different levels of the program development. In this paper, a subset of these clauses has been used for the design of our specifications. We will only review these clauses. A whole description can be found in the B-Book (Abrial, 1996). The typical B machine starts with the keyword MACHINE and ends with the other keyword END. A set of clauses can be defined in between. In our case, these clauses appear in the following order:

- EXTENDS is a programming in the large clause that allows to import instances of other machines. Every component of the imported machine becomes usable in the current machine. This clause allows modularity capabilities.
- SETS defines the sets that are manipulated by the specification. These sets can be built by extension, comprehension or with any set operator applied to basic sets.
- CONSTANTS defines all the constants that are used in the machine. Notice that the constants described can have any type (naturals, elements of sets, constant functions, and so on).
- PROPERTIES are logical expressions that are satisfied by the constants described in the previous clause.

Aït-Ameur Y., Girard P., & Jambon F. Using the B formal approach for incremental specification design of interactive systems. *IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'98), Heraklion (Crete), Greece*, 14-18 September 1998. p. {to be published}.

- VARIABLES is the clause where all the attributes of the described model are represented. In the methodology of B, we find in this clause all the selector functions which allow accessing the different properties represented by the described attributes.
- DEFINITIONS is a set of definitions introduced by the user. They are rewritten everywhere they are used in a machine. It allows simplification of machine notation.
- INVARIANT clause describes the properties of the attributes defined in the clause VARIABLES. The logical expressions described in this clause remain true in the whole machine and they represent assertions that are always valid.
- INITIALISATION clause allows giving initial values to the variables of the corresponding clause. Note that the initial values must satisfy the invariant.
- OPERATIONS clause is the last clause of a machine. It defines all the operations (functions and procedures) that constitute the abstract data type represented by the machine. Depending on the nature of the machine, the OPERATIONS clause authorises particular generalised substitutions to specify each operation. The substitutions used in our specifications and their semantics is described below.

Other syntax possibilities are offered in B, and we do not intend to review them in this paper, in order to keep its length short enough.

3.3 Semantics of generalised substitutions.

The calculus of explicit substitutions is the semantics of the abstract machine notation and is based on the weakest precondition approach of Dijkstra (Dijkstra, 1976). Formally, several substitutions are defined in B. If we consider a substitution S and a predicate P representing a postcondition, then [S]P represents the weakest precondition that establishes P after the execution of S. The substitutions of the abstract machine notation are inductively defined by the following equations. Notice that we restricted ourselves to the substitutions used for our development. The reader can refer to the literature (Abrial, 1996, Lano, 1996) for a more complete description:

```
[SKIP]P                          <==> P
[S1 || S2]P                      <==> [S1]P and [S2]P
[PRE  E  THEN S  END ]P          <==> E and [S]P
[ANY v WHERE E THEN S END]P      <==>   v (P =>[S]P)
[x:=E]P                          <==> P(x/E)
```

The last one represents the predicate P where all the free occurrences of x are replaced by the expression E. Notice that when a given substitution is used, the B checker generates the corresponding proof obligation, i.e., the logical expression on

the right hand side of the operator "<==>". This calculus propagates a precondition that must be implied by the precondition set by the user. If not, then the user proves the precondition or modifies it.

3.4 Interaction properties

Proving interaction properties can be achieved by the way of *model checking* or *theorem proving* (Campos and Harrison, 1997). Theorem proving is a deductive approach to the verification of interactive properties. Unless powerful theorem provers are available, proofs must be made "by hand". Consequently, they are hard to find, and their reliability depends on the mathematical skills of the designer. On the opposite, model checking is based on the complete verification of a finite state machine, and may be fully automated. However, one of the main drawbacks of model checking is that the solution may not be computed due to the high number of states (Campos and Harrison, 1997). The last sessions of EHCI'95 and DSV-IS'97 show a wide range of examples of these two methods of verification.

As an example, *model checking* is used by Palanque et al. who model user and system by the way of object-oriented Petri nets –ICO– (Palanque et al., 1995). They argue that automated proofs can be done to ensure that: there is no cycle in the task model; a specific task must precede another specific task (enter_pin_code and get_cash in the ATM example); and that the final functional core state is the final user task (get_cash and get_card). These proofs are relatives to reachability. Lauridsen uses the RAISE formalism to show that an interactive application –functional core, dialogue control and logical interaction– can be built using translations from the functional core adapter specification (Lauridsen, 1995). Then, Lauridsen shows that this refinement method can prove interaction properties relatives to predictability, observability, honesty, and substitutivity.

In the meantime, Paternó and Mezzanotte check that unexpected interaction trajectories expressed in a temporal logic –ACTL– cannot be performed by the user. The system –a subset of an air traffic control application– is modelled by interactors specified with LOTOS (Paternó and Mezzanotte, 1995). Brun et al. uses the translation from a semi-formal task-oriented notation –MAD– (Scapin and Pierret-Golbreich, 1990) to a temporal logic –XTL– (Brun, 1997) and prove reachability (Brun and Jambon, 1997).

Our approach –using the B method– deals with the first method, i.e., *theorem proving*. Yet, the method does not suffer from the main drawbacks of theorem proving methods, i.e., proving all the system "by hand". Most of the proofs, regarding *visibility* or *reachability*, which are compulsory in the building process are checked by the B-Tool "Atelier B". Notice that in our case study, about 95% of the proofs obligations are automatically proved. Moreover, since the specification is incrementally built, the proofs are also incrementally built. Indeed, compositionality

Aït-Ameur Y., Girard P., & Jambon F. Using the B formal approach for incremental specification design of interactive systems. *IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'98), Heraklion (Crete), Greece*, 14-18 September 1998. p. {to be published}.

in B ensures that the proofs of the whole system are built using the ones of the subsystems. This technique considerably simplifies the interaction properties verification. And then, this incremental conception of applications asserts that the proofs needed at the low levels B-machines of the application, i.e. the functional core, are true at the higher levels, i.e. presentation. So, the *reliability* is checked by construction. The following sections give examples of verification of *visibility* and *reachability* properties in our case study using the B technique.

4. FORMALISATION OF THE MOUSE INTERACTION

This section presents a simple example of the use of the B notation for the specification and for the verification of the operations related to the mouse. It is derived from the case study described in the section 3. The complete formalisation of this case study in B can be found in (Aït-Ameur et al., 1998a, Aït-Ameur et al., 1998b). This specification has been designed using reverse engineering techniques starting from an already existing toolkit implementing mouse manipulation. This reverse engineering task to re-design the mouse specification is a crucial phase in our work. It allows using the B language in a uniform manner even for already designed programs. However, this step of development needs to be achieved for other parts of an interactive application (e.g. keyboard). Below, we specify the only operations we are interested in.

4.1 Specification

The machine we define is named POSTIT_MOUSE. It defines two sets: the POST_MOUSE representing all the possible values for a mouse (it behaves as a type) and a finite enumerated set MOUSE_STATE for the states of the mouse.

```
MACHINE POSTIT_MOUSE

SETS
   POST_MOUSE;
   MOUSE_STATE ={up, down, clicked}
```

For convenience, we have considered three states only. Without loss of generality, we have voluntarily omitted the other states because they are not used in our application. Notice that they could have been included.

Aït-Ameur Y., Girard P., & Jambon F. Using the B formal approach for incremental specification design of interactive systems. *IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'98), Heraklion (Crete), Greece*, 14-18 September 1998. p. {to be published}.

```
CONSTANTS
   max_post_mouse,
   x_mouse_position_default,
   y_mouse_position_default,
   max_mouse_position_wide,
   max_mouse_position_high
PROPERTIES
   max_post_mouse = card(POST_MOUSE) &
   x_mouse_position_default = 20 &
   y_mouse_position_default = 20 &
   max_mouse_position_wide = 300 &
   max_mouse_position_high = 250
```

The previous part defines a set of constants. The *max_post_mouse* constant is the maximum of mouse positions we can have. The *x_mouse_position_default* and *y_mouse_position_default* are the default coordinates of the mouse. The *max_mouse_position_high* and *max_mouse_position_wide* are the maximum values of the mouse coordinates. They correspond to the dimensions of the screen.

```
VARIABLES
the_post_it_mouse,
x_post_it_mouse_position, y_post_it_mouse_position,
post_it_mouse_state, post_it_mouse_creation
INVARIANT
the_post_it_mouse<:POST_MOUSE &
x_post_it_mouse_position :the_post_it_mouse --> NAT  &
y_post_it_mouse_position :the_post_it_mouse --> NAT &
post_it_mouse_state:the_post_it_mouse --> MOUSE_STATE &
post_it_mouse_creation:the_post_it_mouse --> BOOL &
!xx (xx:the_post_mouse ==>
(x_post_it_mouse_position(xx):
          1..max_mouse_position_wide) &          (1)
(y_post_it_mouse_position(xx):
          1..max_mouse_position_high) )
```

For a given mouse in the set *the_post_it_mouse*, the previous selectors define a set of functions that allow retrieving the coordinates of a mouse, its state and its creation. The last predicate expresses that every mouse position remains in the screen. Several operations on the mouse (creating, moving, clicking…) are defined below. We only give the details for creating and for moving the mouse (! xx stands for xx).

Aït-Ameur Y., Girard P., & Jambon F. Using the B formal approach for incremental specification design of interactive systems. *IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'98), Heraklion (Crete), Greece*, 14-18 September 1998. p. {to be published}.

```
OPERATIONS
   pp <--create_mouse_position=
        ...
   Move_mouse_with_drag(pp, aa,bb)=
PRE
   pp : the_post_it_mouse &
   aa : NAT &
   aa : 1..max_mouse_position_wide &
   bb : NAT &
   bb:1..max_mouse_position_high &
   post_it_mouse_state(pp)=down &
   post_it_mouse_creation(pp)=TRUE
THEN
   x_post_it_mouse_position(pp):=aa ||
   y_post_it_mouse_position(pp):=bb
END;
   move_mouse(pp, aa, bb)=
        ...
   mouse_up(pp)=
        ...
   mouse_down(pp)=
        ...
   mouse_clicked (pp)=
        ...
END
```

The *create_mouse_position* creates a mouse and gives it the status up, and default position as coordinates. The *move_mouse_with_drag* allows the moving of the mouse with its state to down and it moves it to the coordinates (*aa,bb*), where a precondition requires *aa* and *bb* to be in the screen. This precondition is **mandatory** to prove that the mouse never goes out of the screen. In other words this precondition is an example of a *visibility* property.

4.2 Proofs

The previous machine generates a set of 33 proof obligations. All of them were **automatically** proven by the "Atelier B" tool. Among these proof obligations, let us consider the ones of the *move_mouse_with_drag*. There are 6 proof obligations, and among them, the ones that says that the invariant assertion (1) is preserved. There, the precondition:

```
(aa : 1..max_mouse_position_wide & bb : 1..max_mouse_position_high)
```

is used to prove the invariant assertion (1) in the clause INVARIANT.

Aït-Ameur Y., Girard P., & Jambon F. Using the B formal approach for incremental specification design of interactive systems. *IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'98), Heraklion (Crete), Greece*, 14-18 September 1998. p. {to be published}.

Let us now, try to write the following for the *move_mouse_with_drag* operation. We obtain:

```
Move_mouse_with_drag(pp,aa,bb)=
PRE
   pp : the_post_it_mouse &
   aa : NAT &
   bb : NAT &
   post_it_mouse_state(pp)=down &
   post_it_mouse_creation(pp)=TRUE
THEN
   x_post_it_mouse_position(pp):=aa ||
   y_post_it_mouse_position(pp):=bb
END;
```

If we violate the precondition in the *move_mouse_with_drag* operation by omitting that:

```
( aa : 1..max_mouse_position_wide  &  bb : 1..max_mouse_position_high )
```

Then the system is not capable to prove the correctness of the operation. It generates a proof obligation saying that there is a need to prove the invariant assertion (1) of the INVARIANT clause which is violated.

5. INTEGRATION OF THE MOUSE INTERACTION

Linking two abstract machines is possible thanks to programming in the large operators offered by the B method. These operators allowed us to incrementally build complex and sound formal specifications. In this section we present an abstract machine which specifies a simple window manager, and we mainly focus on the moving operation of a window. Then, this abstract machine and the one corresponding to the mouse are used to build another complete and sound abstract machine, which specifies the window manager with an interaction by the mouse. We do not give the complete specification and we focus on the proof aspects. The whole-annotated abstract machine is defined in (Aït-Ameur et al., 1998b).

5.1 The window manager without mouse interaction

The POSTIT_VISU_WITHOUT_INT is the abstract machine that defines a simple window manager related to our case study.

5.1.1 Specification

The following abstract machine defines a set of windows (to support a post it). It represents the module of all the allowed window manipulations.

```
MACHINE POSTIT_VISU_WITHOUT_INT
EXTENDS POSTIT_VISUALIZATION

CONSTANTS
        ...
PROPERTIES
   max_post_visu=card(POST_VISU)            &
   max_post_it_wide=300                     &
   max_post_it_high=250                     &
VARIABLES
...
```

The previous properties define the values of the dimensions of the screen. The variables of the INVARIANT describe the accessors to the different components of a window. *the_post_it_visu* is the set of all the windows effectively created, and *get_new_post*, *block_state* and *post_it_window_stat*us are the functions which respectively imports a window from the POSTIT_VISUALIZATION machine, gives the state of the block and computes the status of the window from the one on the imported window.

```
INVARIANT
   the_post_it_visu <:POST_VISU &
   get_new_post :the_post_it_visu --> post_new &
   block_state :block_VISU &
   post_it_window_status:the_post_it_visu-->SCREEN_STATE &
   (!xx. (xx : the_post_it_visu =>
        (x_post_it_position(xx) : 1..max_post_it_wide &    (2)
         y_post_it_position(xx) : 1..max_post_it_high ) ) )
```

The previous logical expression (2) is an important invariant. It states that all (!) the windows of the set *the_post_it_visu* have their upper-left corner in the screen. This invariant is conserved and proved for all the operations. We will now describe the *move_window_position* action, and every element it uses. In this machine, *move_window_position* is not strictly constrained. For example, it is possible to partially quit the visualisation window while moving a Post-It® Note. This constraint will be put later on. We only take into account the constants, variables and invariants that are needed by *move_window_position*.

Aït-Ameur Y., Girard P., & Jambon F. Using the B formal approach for incremental specification design of interactive systems. *IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'98), Heraklion (Crete), Greece*, 14-18 September 1998. p. {to be published}.

```
OPERATIONS
move_window_position(pp, aa, bb)=
   PRE
        aa : 1..max_post_it_wide &
        bb : 1..max_post_it_high &
        pp : the_post_it_visu   &
        block_state = block_open   &
        (  post_it_window_status(pp)= displayed  or
           post_it_window_status(pp)= hidden )
```

The previous preconditions say that the new coordinates of the upper-left corner of a window (aa,bb) must belong to the screen, that the window *pp* is effectively created (belongs to the_post_it_visu set), that the block of Post-It® Notes is effectively open and that the window can be displayed or hidden.

```
   THEN
        x_post_it_position(pp):= aa          ||
        y_post_it_position(pp):= bb          ||
        update_hidden(get_new_post(pp))
   END
```

Here, a parallel substitution is used. It says that after executing this operation the new coordinates of the upper-left corner become (*aa,bb*) and that the operation *update_hidden* (of another non described machine) is called. This last operation is called in order to update the status of the windows that become hidden or displayed after the current window has been moved. We take into account here a visibility property we have expressed in section 2.

5.1.2 Proofs

From these previous preconditions, we can infer the following properties:

- Windows never can be moved when they are iconified,
- Windows never can be moved when the block is not opened,
- Windows never have their upper-left corner outside of the screen. This is very important for windows manipulation. It is directly derived from the invariant assertion (2).

All these proofs show that *reachability* properties can be checked in the top-level B-machine.

Aït-Ameur Y., Girard P., & Jambon F. Using the B formal approach for incremental specification design of interactive systems. *IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'98), Heraklion (Crete), Greece*, 14-18 September 1998. p. {to be published}.

5.2 Specification of the window manager with mouse interaction

Now, we get two abstract machines: one for the mouse and another for the window managing. The goal of the third abstract machine is to merge these two abstract machines in order to build a new one that integrates the manipulation of the windows using a mouse.

5.2.1 Specification

The following abstract machine, named POSTIT_VISU_WITH_INT_MOUSE extends (by a programming in the large operator) the abstract machines for mouse and for windows.

```
MACHINE POSTIT_VISU_WITH_INT_MOUSE

EXTENDS
   POSTIT_MOUSE , POSTIT_VISU_WITHOUT_INT
PROPERTIES
   ...
   screen_wide = max_mouse_position_wide          &
   screen_high = max_mouse_position_high          &
   max_post_it_wide=max_mouse_position_wide       &
   max_post_it_high=max_mouse_position_high
```

The previous properties are named gluing properties. They make connections between the constants of all the imported machines. Indeed, it specifies that the dimensions of a screen for a window are the same as the ones for a mouse.

```
INVARIANT
   the_post_it_visu_with_mouse <:
        POSTIT_VISU_WITH_MOUSE &
   post_it_visu_with_mouse_creation :
        the_post_it_visu_with_mouse --> BOOL &
   get_the_mouse :
        the_post_it_visu_with_mouse --> the_post_it_mouse &
   get_the_post_it_visu :
        the_post_it_visu_with_mouse --> the_post_it_visu
```

The set *the_post_it_visu_with_mouse* records all the windows manipulated by a mouse. The selectors define a set of functions that allow extracting a window by *get_the_post_it_visu* and the mouse by *get_the_mouse*. Let us give the details of the action *move_window_with_mouse* that allows moving a window combining the

interaction of the mouse. This action has a set of preconditions, which needs to be valid before the action is performed. They are commented below.

```
OPERATIONS
move_window_with_mouse(pp, aa, bb) =
PRE
   aa : NAT &
   bb : NAT &
   aa : 1..max_post_it_wide &
   bb : 1..max_post_it_high &
   pp : the_post_it_visu_with_mouse &
```

The new coordinates of the upper-left corner of the window must define a point in the limits of the screen. Notice that these preconditions imply:

- The ones of the *move_window* without the use of the mouse. It remains to assert that the window is displayed,
- and the ones of *move_mouse*. It remains to assert that the mouse is down to ensure that the drag is possible.

```
   post_it_window_status(get_the_post_it_visu(pp))
        = displayed &
   post_it_mouse_state(get_the_mouse(pp)) = down &
   block_state = block_open &
```

The block of Post-It® must be open, otherwise the window of a Post-It® Note cannot be moved.

```
   post_it_visu_with_mouse_creation(pp) = TRUE &
   post_it_visu_creation(get_the_post_it_visu(pp)) = TRUE &
   post_it_mouse_creation(get_the_mouse(pp)) = TRUE &
```

The different elements manipulated by the operation (mouse, a window, a window with mouse interaction) must be already created.

```
   x_post_it_position(get_the_post_it_visu(pp))+5 :
           1..max_mouse_position_wide &
   y_post_it_position(get_the_post_it_visu(pp))+5 :
           1..max_mouse_position_high &
   x_post_it_window(get_the_post_it_visu(pp))-5 :
           1..max_mouse_position_wide &
```

The coordinates delimiting the moving zone of the window must belong to the screen, i.e., must appear in the screen.

Aït-Ameur Y., Girard P., & Jambon F. Using the B formal approach for incremental specification design of interactive systems. *IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'98), Heraklion (Crete), Greece*, 14-18 September 1998. p. {to be published}.

```
   x_post_it_mouse_position(get_the_mouse(pp)):
        x_post_it_position(get_the_post_it_visu(pp))+5..
        x_post_it_window(get_the_post_it_visu(pp))-5  &
   y_post_it_mouse_position(get_the_mouse(pp)):
        y_post_it_position(get_the_post_it_visu(pp))..
        y_post_it_position(get_the_post_it_visu(pp))+5
```

The mouse position must be in the moving zone delimited previously. This is an additional precondition which is strictly related to the fact that there is a mouse interaction and that the mouse must be in the moving zone. This precondition could not be expressed in one of the imported machines.

```
THEN
   move_window_position(get_the_post_it_visu(pp),aa,bb)||
   Move_mouse_with_drag(get_the_mouse(pp),aa,bb)
END
```

Calling the *move_window_position* action of the POSTIT_VISU_WITHOUT_INT moves the window and the mouse is also moved by the action *move_mouse_with_drag* of the machine POSTIT_MOUSE. Notice that this action is one of the actions that combine the mouse toolkit and the window manager toolkit.

5.2.2 Proofs

From the PROPERTIES assertions, we derive that the coordinates of the upper-left corner of a window and the ones of the mouse take their values in the same domain. Therefore, we can prove that a mouse can manipulate every window. This is an important *reachability* property. The previous abstract machine gives supplementary preconditions on the objects. These preconditions imply the ones of the corresponding operations in the machines imported by extension. Therefore, the prover is capable to complete the proof of correctness. This is an important issue of the B method that shows that B is not only capable to structure the development of specifications, but it allows the structuration and the modularization of the proofs. Moreover, this approach shows that it is possible to structure the specifications hierarchically with respect to the implication of the preconditions. Here, the implication plays the role of an ordering relation from weak to strong. Indeed, the preconditions of the last machine are stronger than the ones of the imported machines. Finally, this mechanism of weakest precondition conserves the coherence of the whole specification. This previous machine has shown that it is not possible to:

- move a window with a mouse whose position is not in the moving zone,
- move a window if there is no mouse,
- move a window if it is eliminated: this property is inherited form the imported machine.

We can prove the previous properties by removing the corresponding assertions from the preconditions and then the prover will complain because we are not capable to prove some proof obligations.

6. CONCLUSION AND FUTURE WORK

This paper has shown the use of the B method on a non-trivial case study. It has illustrated the capability of B to handle different aspects of the software life cycle in the area of interactive systems. The described approach showed that:

- The approach is completely formalised and most of the proof obligations are proved automatically. The other ones needed only few steps of proof. They allow controlling the development of the whole program.
- It is possible to check properties of the specifications, thanks to the weakening of the preconditions (Aït-Ameur et al., 1998a).
- Reverse engineering aspects can be handled with this approach and the specifications of already existing programs can be used to develop new ones. Therefore, reusability issues appear.
- The specifications are incrementally built, in contrast with state based approaches. Indeed, programming in the large operator allows to compose abstract machines and therefore to build more complex specifications. Yet, this process needs to follow a given methodology issued from the area of interactive system design.

However, several open issues raised from this work:

- There is a need to merge both state models and B models in heterogeneous approach. We will get the benefits of both the developments aspects and the model checking ones.
- The validation aspects need to be tackled in the context of B. There seems to be a possibility to generate tests from such B specifications.
- The B specifications must be developed, in an ascending manner for linking the task analysis to the derivation of specifications and in a descending manner to show the refinement the specifications to implementation.

Finally, we plan to adapt the B approach to the different architectures proposed in the area of interactive systems for handling the formal specification design

process. We suggest to define generic abstract machines that implement a given architecture model in order to avoid the user to be a B-expert.

7. REFERENCES

Abowd, G. D., Wang, H.-M. and Monk, A. F. (1995) A Formal Technique for Automated Dialogue Development, in DIS'95 (Eds, Olson, G. M. and Schuon, S.) ACM, Ann Arbor, Michigan, pp. 219-226.

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

Aït-Ameur, Y., Girard, P. and Jambon, F. (1998a) A Uniform approach for the Specification and Design of Interactive Systems: the B method, in 5th International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'98) (Eds, Johnson, P. and Markopoulos, P.) , Cosener's House, Abingdon, UK, pp. 333-352.

Aït-Ameur, Y., Girard, P. and Jambon, F. (1998b) Using the B formal approach for incremental specification design of interactive systems. Laboratoire d'Informatique Scientifique et Industrielle (LISI/ENSMA). Research report. 98-001.

Bjorner, D. (1987) VDM a Formal Method at Work, in VDM Europe Symposium'87 Springer-Verlag. LNCS.

Brun, P. (1997) XTL: a temporal logic for the formal development of interactive systems, in Formal Methods for Human-Computer Interaction (Eds, Paterno, F. and Palanque, P.) Springer-Verlag, pp. 121-139.

Brun, P. and Beaudouin-Lafon, M. (1995) A taxonomy and evaluation of formalisms for the specification of interactive systems, in 10th annual conference of the British Human-Computer Interaction Group (HCI'95) , University of Huddersfeild, UK.

Brun, P. and Jambon, F. (1997) Utilisation des spécifications formelles dans le processus de conception des Interfaces Homme-Machine, in Neuvièmes journées francophones sur l'Interaction Homme-Machine (IHM'97) Cépaduès-Éditions, Poitiers-Futuroscope, France, pp. 23-24-27-25-26-28-29.

Campos, J. C. and Harrison, M. D. (1997) Formally Verifying Interactive Systems: A Review, in Eurographics Workshop on Design, Specification, Verification of Interactive Systems (Eds, Harrison, M. D. and Torres, J. C.) Eurographics, Springer-Verlag, Granada, Spain, pp. 109-124.

Card, S., Moran, T. and Newell, A. (1983) The psychology of Human-Computer Interaction. Lawrence Erlbaum Associates.

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

Dix, A. J., Finlay, J., Abowd, G. and Beale, R. (1993) Human-Computer Interaction. Prentice Hall.

*Duke, D. J. and Harrison, M. D. (1993a) Abstract Interaction Objects. Computer Graphics Forum, **12,** 25-36.*

Duke, D. J. and Harrison, M. D. (1993b) Towards a Theory of Interactors. Amodeus Esprit Basic Research Project 7040. System Modelling/WP6.

Aït-Ameur Y., Girard P., & Jambon F. Using the B formal approach for incremental specification design of interactive systems. *IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'98), Heraklion (Crete), Greece*, 14-18 September 1998. p. {to be published}.

Feather, M. S. (1987) A Survey and Classification of some program Transformation Approaches and Techniques, in IFIP world congress (Ed, Meertens, L. G. L. T.) Elsevier Science Publishers B.V. (North-Holland), pp. 165-195.

Green, M. W. (1986) A Survey of three Dialogue Models. ACM Transactions on Graphics, ***5,*** *244-275.*

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

Lano, K. (1996) The B Language Method: A guide to practical Formal Development. Springer.

Lauridsen, O. (1995) Systematic methods for user interface design, in IFIP TC2/WG2.7 working conference on engineering for human-computer interaction (EHCI'95) (Eds, Bass, L. J. and Unger, C.) Chapman & Hall, Grand Targhee Resort (Yellowstone Park), USA, pp. 169-188.

Palanque, P. (1992) Modélisation par Objets Coopératifs Interactifs d'interfaces homme-machine dirigées par l'utilisateur. PhD of Univ. Toulouse I, Toulouse.

Palanque, P., Bastide, R. and Sengès, V. (1995) Validating interactive system design through the verification of formal task and system models, in IFIP TC2/WG2.7 working conference on engineering for human-computer interaction (EHCI'95) (Eds, Bass, L. J. and Unger, C.) Chapman & Hall, Grand Targhee Resort (Yellowstone Park), USA, pp. 189-212.

Palanque, P. and Girard, P. (1996) Groupe de travail GP3-FLASHI (Formalismes et Langages Appliqués aux Systèmes Hautement Interactifs). GDR-PRC Communication Homme-Machine. Rapport d'activité.

Paternó, F. (1994) A Theory of User-Interaction Objects. Journal of Visual Languages and Computing, ***5,*** *227-249.*

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

Paternó, F. and Mezzanotte, M. (1995) Formal verification of undesired behaviours in the CERD case study, in IFIP TC2/WG2.7 working conference on engineering for human-computer interaction (EHCI'95) (Eds, Bass, L. J. and Unger, C.) Chapman & Hall, Grand Targhee Resort (Yellowstone Park), USA, pp. 213-226.

Sannella, D. (1988) A Survey of Formal Software Developement Methods. Laboratory of Fundamental Computer Science. LFCS Report series. ECS-LFCS-88-56.

Scapin, D. L. and Pierret-Golbreich, C. (1990) Towards a method for task description : MAD, in Work with display units 89 (Eds, Berliguet, L. and Berthelette, D.) Elsevier Science Publishers, North-Holland.

Spivey, J. M. (1988) The Z notation: A Reference Manual. Prentice Hall Int.

Steria méditerranée (1997) Atelier B version 3.0.

Aït-Ameur Y., Girard P., & Jambon F. Using the B formal approach for incremental specification design of interactive systems. *IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'98), Heraklion (Crete), Greece*, 14-18 September 1998. p. {to be published}.