Encoding a process algebra using the Event B Method. Application to the validation of user interfaces

Yamine AIT-AMEUR, Mickael BARON and Nadjet KAMEL
LISI-ENSMA and University of Poitiers
BP 40109, 86961 Futuroscope Cedex, France
{yamine, baron, kamel}@ensma.fr
http://www.lisi.ensma.fr/ihm

ABSTRACT

This paper presents the use of the B technique in its event based definition. We show that it is possible to encode, using Event B, the models i.e. transition systems associated to a process algebra with asynchronous semantics. The Event B obtained encoding considers that the Event B model associated to the left hand side of a BNF rule defining the algebra expressions is refined by a model corresponding to the right hand side of the same rule. The translation rules of each operator of a basic process algebra are given. Then, an example illustrating each translation rule is given. This approach is based on a proof technique and therefore it does not suffer from the state number explosion problem occurring in classical model checking. The interest of this work is the capability to validate user tasks or scenarios when using a given system and particulary a critical system. Finally, we discuss the application of this approach for validating user interfaces tasks in the Human Computer Interaction (HCI) area.

Keywords. Event B method, events refinements, process algebra, application to HCI

INTRODUCTION

When performing syntactic analysis of formal languages, the classical approach consists in deriving abstract representations from a formal BNF grammar. Usually, these abstract representations are abstract syntax trees. The construction of an abstract syntax tree consists in applying the derivation rules of the BNF description. A hierarchical derivation tree is obtained. This tree may be built either ascending (bottom-up approach) or descending (top-down approach). The trees are augmented by other semantic information. Among this information, one can cite attributes, typing, or code generation routines and so on.

We claim that it is possible to associate to BNF grammars a hierarchy of Event B models in a top-down approach. The refinement relationship of Event B is used to encode the hierarchy provided by the abstract syntax tree. Each derivation rule is represented by a refinement and the obtained hierarchy describes a tree that is augmented by models. Moreover, these models are enriched by relevant properties (safety, reachability, robustness, ...). Obviously, the interest of such a transformation is to allow the possibility to perform proofs of these relevant properties associated to these models.

The particular case of a language of processes (a process algebra which looks like CCS) is shown in this paper. A BNF grammar defining the studied process algebra named CTT (ConcurTaskTrees [32]) is used as an example illustrating how our approach works. We give a translation rule for each BNF rule. This process algebra is used for the specification and validation of User Interfaces (UI). Indeed, validation of critical UI usually requires user scenarios which describe different usages of an UI. Nominal and non nominal scenarios are defined and checked on the designed UI. Moreover, these scenarios or tasks are checked at the specification and/or design level.

This paper is structured as follows. Next section recalls the basic definitions of the Event B method. Section 3 is the kernel of our proposal. It gives the principle of the translation of a BNF grammar to a hierarchy of Event B models. It also shows how this approach works on a BNF of the CTT process algebra and gives a practical example for each basic operator of this algebra. Section 4 describes the usage of this translation for validating UI. Section 5 discusses the interest of this approach in comparison with classical model checking.

THE EVENT B METHOD

Among the increasing number of formal methods that have been described, model oriented methods, such as VDM [15], Z [35] or B [2] [20], seem to have proved their applicability and efficiency. These methods are based on model description. They consist in defining a model by variable attributes which characterize the state of the described system, the invariants and other properties that must be satisfied and the different operations that alter these variables. Starting from this observation, Z method uses set theory notations and allows to encode the specifications in a structure named schema. Like VDM, it is based on preconditions and post-conditions [25, 27, 26]. Moreover, VDM allows the generation of a set of proof obligations. On the other hand, B is based on the weakest precondition technique of Dijkstra [21]. Starting from this method, J.R. Abrial [2] has defined a logical calculus, named the Generalized Substitution Calculus. Proof obligations are generated and need to be proved in order to ensure the correctness of developments and refinements.

In the recent years, J.R. Abrial has suggested a new definition of the B method: the Event B method [1]. This method is adapted to the development of interactive systems as well as sequential systems. Our choice of B is motivated by the fact that B Method is supported by tools which allow a complete formal development and is adapted to the description of interactive systems [17].

Event B models

The basic element of any development achieved with the Event B method is the *model*. A model is defined as a set of variables, defined in the **VARIABLES** clause that evolve thanks to events defined in the **EVENTS** clause. The notion of Event B model encode a state transition system where the variables represent the state and the events represent the transitions from one state to another. Moreover, the refinement capability offered by Event B allows to decompose a model (thus a transition system) into another transition system with more and more design decisions moving from an abstract level to a less abstract one. Refinement technique allows to preserve the proved properties and therefore it is not necessary to prove them again in the refined transition system (which is usually more complex). The structure of an Event B model is given by the following elements.

```
MODEL nameM
REFINES nameR
    ...
    VARIABLES ...
    INVARIANT ...
    ASSERTIONS ...
    INITIALISATION ...
    EVENTS ...
END
```

A model *nameM* is defined by a set of clauses. It may refine another model *nameR*. Briefly, the clauses mean:

- **VARIABLES** clause represents the variables of the model of the specification. Refinement may introduce new variables in order to enrich the described system.
- **INVARIANT** clause describes, thanks to first order logic expressions, the properties of the attributes defined in the clause VARIABLES. Typing information and safety properties are described in this clause. These properties shall remain true in the whole model and in further refinements. Invariants need to be preserved by the initialisation and events clauses.
- **ASSERTIONS** are logical expressions that can be proved from the invariants. They do not need to be proved for each event like for the invariant. Usually, they contain properties expressing that there is no deadlock nor livelock.
- **INITIALISATION** clause allows to give initial values to the variables of the corresponding clause. They define the initial states of the underlying transition system.
- **EVENTS** clause defines all the events that may occur in a given model. Each event is described by a body thanks to generalized substitutions defined below. Each event is characterized by its guard (i.e. a first order logic expression involving variables). An event is fired when its guard evaluates to true.

Semantics of generalized substitutions

The initialisation and the events occurring in a B model are described thanks to generalized substitutions. Generalized substitutions are based on the weakest precondition calculus of Dijkstra. Formally, several substitutions are defined in B. If we consider a substitution S and a predicate P representing a post-condition, then $[S]P$ represents the weakest precondition that establishes P after execution of S. The substitutions occurring in Event B models are inductively defined by the following expressions [1, 2, 29].

$$[\textbf{SKIP}]\,P \iff P \tag{1}$$
$$[\textbf{S1} \mid\mid \textbf{S2}]\,P \iff [S1]\,P \wedge [S2]\,P \tag{2}$$
$$[\textbf{ANY}v\textbf{WHERE}E\textbf{THEN}S\textbf{END}]\,P \iff \forall\, v(P \Longrightarrow [S]\,P) \tag{3}$$
$$[\textbf{SELECT}E\textbf{THEN}S\textbf{END}]\,P \iff E \Longrightarrow [S]P \tag{4}$$
$$[\textbf{BEGIN}S\textbf{END}]\,P \iff [S]\,P \tag{5}$$
$$[\textbf{x:=E}]\,P \iff P(x/E) \tag{6}$$

$P(x/E)$ represents the predicate P where all the free occurrences of x are replaced by the expression E.

Substitutions 1, 2, 5 and 6 represent respectively the empty statement, the parallel substitution expressing that $S1$ and $S2$ are performed in parallel, the block substitution and the affectation. Substitutions 3 and 4 are the guarded substitutions where S is performed under the guard E.

In all the previous substitutions, the predicate E represents a guard. Each event guarded by a guard E is fired iff the guard is true and when it is fired, the post-condition P is established (feasibility of an event). The guards define the feasibility conditions given by the Fis predicate defined in [2].

Semantics of Event B models

The new aspect of the Event B method, in comparison with classical B, is related to the semantics. Indeed, the events of a model are atomic events. The associated semantics is an interleaving semantics.

Therefore, the semantics of an Event B model is trace based semantics with interleaving. A system is characterized by the set of licit traces corresponding to the fired events of the model which respects the described properties. The traces define a suite of states that may be observed by properties. All the properties will be expressed on these traces.

This approach has proved to be able to represent event based systems like interactive systems. Moreover, decomposition (thanks to refinement) allows building of complex systems gradually in an incremental manner by preserving the initial properties thanks to the gluing invariant preservation.

Refinement of Event B models

Each Event B model can be refined. A refined model is defined by adding new events, new variables and a gluing invariant. Each event of the abstract model is refined in the concrete model by adding new information by expressing how the new set of variables and the new events evolve. All the new events appearing in the refinement refine the *skip* event of the refined model. Each new event corresponds to an $\epsilon-$transition in the abstract model.

The gluing invariant ensures that the properties expressed and proved at the abstract level (in the **ASSERTIONS** and **INVARIANTS** clauses) are preserved in the concrete level. Moreover, **INVARIANT**, **ASSERTIONS** and **VARIANT** clauses allow to express deadlock and livelock freeness.

1. They shall express that the new events of the concrete model are not fired infinitely (no livelock). *A decreasing variant is introduced for this purpose.*
2. They shall express that at any time an event can be fired (no deadlock). *This property is ensured by asserting (in the* **ASSERTIONS** *clause) that the disjunction of all the abstract events guards implies the disjunction of all the concrete events guards.*

Moreover, in the refinement, it is not needed to re-prove these properties again while the model complexity increases. Notice that this advantage is important if we compare this approach to classical model checking where the transition system describing the model is refined and enriched.

A full simple example [16]

Let us consider below, the specifications of the clock example[16]. The abstract specification $Clock$ uses one variable h describing the hours of the clock. Two events are described. The first ($incr$ event) allows to increment the hour variable. The second event is the $zero$ event. It is fired when $h = 23$ to initialize the hour variable.

```
MODEL
  $Clock$
VARIABLES
  $h$
INVARIANT
  $h \in 0..23$
ASSERTIONS
  $h < 100$
INITIALISATION
  $h := 13$
EVENTS
  $incr =$ SELECT $h \neq 23$ THEN $h := h + 1$ END;
  $zero =$ SELECT $h = 23$ THEN $h := 0$ END.
```

In the refinement specification ($ClockWMinute$) we introduce a new variable m and a new event $ticTac$. We enhance the guards of the $incr$ and $zero$ events in introducing the description of minutes. **ASSERTIONS** clause allows to ensure that the new events of the description system can be fired.

```
REFINEMENT
  $ClockWMinute$
REFINES
  $Clock$
VARIABLES
  $h, m$
INVARIANT
  $m \in 0..59$
ASSERTIONS
  $(h \neq 23) \vee (h = 23) \Rightarrow (h \neq 23 \wedge m = 59) \vee (h = 23 \wedge m = 59) \vee (m \neq 59)$
VARIANT
  $59 - m$
INITIALISATION
  $h := 13 \parallel m := 14$
EVENTS
  $incr =$ SELECT $h \neq 23 \wedge m = 59$ THEN $h := h + 1 \parallel m := 0$END;
  $zero =$ SELECT $h = 23 \wedge m = 59$ THEN $h := 0 \parallel m := 0$END;
  $ticTac =$ SELECT $m \neq 59$ THEN $m := m + 1$ END.
```

ENCODING CTT ALGEBRA IN EVENT B MODELS

This section is the kernel of our proposal. It presents the informal BNF translation rule and its application on a particular language describing a process algebra.

BNF rules translation principle

Our claim is that it is possible to parse BNF grammars into Event B models. The translation principle is defined as follows.

Each BNF rule of the form T ::= E OP F is translated into two Event B models. The first one is associated with the left hand side of the rule and contains only one event $eventT$ associated with the non terminal T. The second model is a refinement of the first one and corresponds to the right hand side of the BNF rule. Two new events $eventE$ and $eventF$ associated with the non terminals E and F are added in the refinement. These events carry the semantics of the *op* terminal and of the right hand side of the BNF rule. The new events are fired and when they are completed, the refined event $eventT$ is fired.

The firing order of the events is determined by introducing a decreasing variant. A variant is a natural number which decreases to zero. When the variant is zero, the events of the described refined model can no longer be fired again. Events of the abstract model can be fired, this corresponds to a return at the previous level in the decomposition tree. This possibility to return to the events of the abstract level is offered by the refinement relationship. In practice, this variant corresponds to a decreasing enumeration of states in a trace thanks to logical expressions. Remember that the events allow to go from an initial state to a target state defining a trace in the underlying described transition system.

In order to illustrate our approach, let us consider the CTT (ConcurTaskTrees) language which defines a classical process algebra. This language is widely used by the user interface community for specifying and/or validating user interfaces. User interfaces tasks are described thanks to this language. We have used this language to validate user tasks on user interfaces designs expressed with Event B. This point is discussed later in the paper in next section.

The task modelling language CTT

CTT [32] is defined by its authors as a notation for task model specifications to overcome limitations of notations used to design interactive applications. Its main purpose is to provide with an easy-to-use notation, which permits to describe tasks expressions combining CTT temporal operators and atomic tasks (atomic events). A CTT task model is based on a hierarchical structure of tasks represented by a tree-like structure. It requires identification of temporal relationships between other subtasks of the same tree level.

Below, a potential grammar describing the syntax of the CTT language is given. It presents temporal operators (from classical process algebra) and task characteristics of CTT.

T ::=	$T >> T$	- - Enabling
|	$T[]T$	- - Choice
|	$T\|\|T$	- - Concurrent
|	$T \models\| T$	- - Order independency
|	$[T]$	- - Optional process
|	$T[> T$	- - Disabling
|	$T\| > T$	- - Interruption
|	$T^*[> T$	- - Disabling infinite process
|	T^N	- - Finite process iteration
|	T_At	- - Atomic process

Next sections, we show how the semantics of CTT can be formally described in Event B allowing to translate, in a generic manner, with generic translation rules, *every* CTT construction (interruption and disabling included) in Event B. This approach uses the refinement capability offered by Event B. For all the translation rules presented below, we will note var_i for the state variables, T_i for processes, G_i for event guards and S_i for any Event B generalized substitution. S_i corresponds to actions executed by a process T_i. It represents the captured semantics.

Generic rules for the translation of the basic CTT constructions

The rules for translating basic operators (*enabling*, *choice*, *iteration*, *concurrency* and *atomic process*) into Event B models are given below. They will be used to translate the remaining operators.

MODEL T_0
INVARIANT
  $I(var_i)$
INITIALISATION
  $Init(var_i)$
EVENTS
$Evt_0 =$
SELECT
  G_0
THEN
  S_0
END;

For the description of the transformation rules, we will use a process T_0 as the root process to be decomposed into another process expression corresponding to the CTT BNF given in above section. The set var_i describes all the useful state variables that characterize the process T_0. Other variables may be added after refinement if it is needed observe new elements while decomposing T_0 in the process tree. S_0 is the substitution that expresses, under the guard G_0, the state variables changes due to the process T_0. These elements are semantic features and are not represented in the syntax.

Basic illustrating example

The sum Sum of two natural numbers aa and bb will be used to illustrate how these rules work. We will give several different refinement possibilities to compute the sum of two natural numbers. The first Event B model Sum_{T0} corresponds to the instantiation of the previous generic model T_0. It contains the event Evt_0 whose corresponding guard (G_0) is equivalent to true (**BEGIN** . . . **END** generalized substitution).

```
MODEL $Sum_{T0}$
INVARIANT
  $Sum \in NAT \wedge aa \in NAT \wedge bb \in NAT$
INITIALISATION
  $Sum :\in NAT \parallel aa :\in NAT \parallel bb :\in NAT$
EVENTS
$Evt_0 =$
BEGIN
  $Sum := aa + bb$
END;
```

Here $aa :\in NAT$ means that aa becomes any natural number.

For the refinement of this example, we will use $RSum,\ AA\ BB$ as the new variables of the refinement. They correspond to refinement variables of the abstract variables Sum, aa and bb respectively. These variables are linked by the same gluing invariant $RSum + AA + BB = aa + bb$ which guarantees the correctness of the refinement and therefore of the CTT operators encoding. The variable initialization $aa, AA :\in (aa \in NAT \wedge AA = aa)$ of the refinement ensures that the variables aa and AA are arbitrarily chosen natural numbers and are equal. The same applies for bb and BB.

Notice that the kernel of the *semantic part* of the translation consists in finding such an invariant. The other part, is related to the firing order of events.

Enabling ">>"

Let us consider $T_0 ::= T_1 >> T_2$ for the activation of T_1 followed by T_2 (sequence). The translation to Event B is given by a model with two events $EvtT_1$ and $EvtT_2$ corresponding to T_1 and T_2.

The translation uses a decreasing variant $StateEna$ initialized to 2. $EvtT_1$ is fired if its guard G_1 is true and so its substitution S_1 is performed, the variant decreases. $EvtT_2$ is fired in sequence if its guard is true and if the variant value is set to 1 by Evt_1.

```
REFINEMENT $RefEnabling_{T0}$
REFINES $T_0$
INVARIANT
  $J(var_i, var_j) \wedge StateEna \in \{0, 1, 2\}$
ASSERTIONS
  $G_0 \Rightarrow ((StateEna = 2 \wedge G_1) \vee (StateEna = 1 \wedge G_2) \vee \ldots$
VARIANT
  $StateEna$
INITIALISATION
  $StateEna := 2 \parallel \ldots$
EVENTS
$EvtT_1 =$                      $EvtT_2 =$                      $EvtT_0 =$
SELECT                          SELECT                          SELECT
  $StateEna = 2 \wedge G_1$       $StateEna = 1 \wedge G_2$       $StateEna = 0 \wedge G'_0$
THEN                            THEN                            THEN
  $StateEna := 1 \parallel S_1$   $StateEna := 0 \parallel S_2$   $S'_0$
END;                            END;                            END;
```

The disjunction of guards is given in the **ASSERTIONS** clause. The set var_j defines the state variables of the refinement. They are linked to the abstract state variables of the abstract model thanks to the gluing invariant $J(var_i, var_j)$. This gluing invariant is defined in all the refinements given below. Notice that the event $EvtT_0$ ends the enabling of the two processes, it gives the refinement of the event corresponding to process T_0.

Example of the use of the enabling operator translation - Let us consider that the sum of aa and bb is performed in a sequential manner. First the variable aa is added to $RSum$ by event $EvtT_1$ and then the variable bb is added to $RSum$ by event $EvtT_2$. These two events of the refinement work for $EvtT_0$ which collects the results and refines the event Evt_0 of the abstraction.

```
REFINEMENT $RefEnabling_{T0}$
REFINES $Sum_{T0}$
INVARIANT
  $(RSum + AA + BB) = (aa + bb) \wedge \ldots$
ASSERTIONS
  $(AA = 0 \wedge BB = 0 \wedge aa + bb \in NAT) \vee \ldots$
VARIANT
  $AA + BB$
INITIALISATION
  $RSum := 0 \parallel Sum :\in NAT \parallel aa, AA :\in (aa \in NAT \wedge AA = aa) \parallel \ldots$
EVENTS
$Evt_1 =$                      $Evt_2 =$                      $Evt_0 =$
SELECT                         SELECT                         SELECT
  $AA \neq 0 \wedge BB \neq 0 \wedge \ldots$    $AA = 0 \wedge BB \neq 0 \wedge \ldots$    $AA = 0 \wedge BB = 0 \wedge \ldots$
THEN                           THEN                           THEN
  $RSum := RSum + AA \parallel$     $RSum := RSum + BB \parallel$     $Sum := RSum$
  $AA := 0$                      $BB := 0$                    END;
END;                           END;
```

Variables AA and BB are used to implicitly represent the variant $StateEna$. The **ASSERTIONS** clause ensures that the new events are fired and the variant guarantees the sequential ordering.

Choice "[]"

Let us consider $T_0 ::= T_1 [] T_2$ defining a non deterministic choice between processes T_1 and T_2 i.e. either T_1 or T_2 is fired. The translation to Event B is given by a model with three guarded events $EvtT_1$, $EvtT_2$ and $Evt_{InitChoice}$.

The variant $StateCho$ is initialized to 3. According to the guard value of each event, one of Evt_1 or Evt_2 is fired. Each event decreases immediately the variant to value 0 forbidding the other events to be fired. The first event to be fired is arbitrarily chosen by the **ANY WHERE THEN** substitution. The refined event $EvtT_0$ ends the process T_0, it allows to fire again the event Evt_0 of the abstract model T_0.

```
REFINEMENT $RefChoice_{T0}$
REFINES $T_0$
INVARIANT
  $J(var_i, var_j) \wedge StateCho \in \{0, 1, 2, 3\}$
ASSERTIONS
  $G_0 \Rightarrow ((\exists(p).(p \in \{1, 2\} \wedge$
  $StateCho = 3)) \vee (G_1 \wedge StateCho = 1) \vee$
  $(G_2 \wedge StateCho = 2) \vee (StateCho = 0 \wedge G_0))$
VARIANT
  $StateCho$
INITIALISATION
  $StateCho :\in \{1, 2\} \parallel \ldots$
EVENTS
$EvtChoiceT_1 =$              $EvtChoiceT_2 =$              $EvtT_0 =$
SELECT                        SELECT                        SELECT
  $StateCho = 1 \wedge G_1$       $StateCho = 2 \wedge G_2$       $StateCho = 0 \wedge G'_0$
THEN                          THEN                          THEN
  $StateCho := 0 \parallel S_1$     $StateCho := 0 \parallel S_2$     $S'_0$
END;                          END;                          END;
```

Event Evt_0 ends the firing of the choice between two processes, it gives the refinement of the event corresponding to process T_0.

Example of the use of the choice operator translation - Let us consider that the sum of aa and bb is performed using a non deterministic choice. This possibility is offered by the semantics of Event B which allows a non deterministic event firing.

Two events are defined. One $EvtChoice_1$ computes the result $RSum = AA + BB$ and the second $EvtChoice_2$ computes the result $RSum = BB + AA$. These two events of the refinement are working for the event Evt_0 which collects the results and refines the event Evt_0 of the abstraction.

Here the variant is the natural number $AA + BB$. Notice that the non deterministic choice is performed thanks to the presence of the variant expression $AA \neq 0 \wedge BB \neq 0$ in the two events $EvtChoice_1$ and $EvtChoice_2$.

```
**REFINEMENT** $RefChoice_{T0}$
**REFINES** $Sum_{T0}$
**INVARIANT**
  $(RSum + AA + BB) = (aa + bb) \wedge \ldots$
**ASSERTIONS**
  $(AA = 0 \wedge (BB = 0 \vee BB \wedge aa + bb \in NAT) \vee \ldots$
**VARIANT**
  $AA + BB$
**INITIALISATION**
  $RSum := 0 \parallel Sum :\in NAT \parallel aa, AA :\in (aa \in NAT \wedge AA = aa) \parallel \ldots$
**EVENTS**
$Evt_0 =$                        $EvtChoice_1 =$                   $EvtChoice_2 =$
**SELECT**                       **SELECT**                        **SELECT**
  $AA = 0 \wedge BB = 0 \wedge \ldots$    $AA \neq 0 \wedge BB \neq 0 \wedge \ldots$    $AA \neq 0 \wedge BB \neq 0 \wedge \ldots$
**THEN**                         **THEN**                          **THEN**
  $Sum := RSum$                    $RSum := AA + BB \parallel$         $RSum := BB + AA \parallel$
**END;**                           $AA := 0 \parallel BB := 0$         $BB := 0 \parallel AA := 0$
                                 **END;**                          **END;**
```

The $InitChoice$ event of the choice translation rule is not needed in this example. It is directly encoded in the **INITIALISATION** clause of the refinement.

Iterative process "T^*"

Let us consider a loop process $T_0 ::= T_1^N$. The principle of encoding a loop in Event B consists making possible to fire, N times, the events associated to the process T_1. A decreasing variant initialized to N is used. The translation into a loop encoded in Event B requires three events : a first one $Evt_{InitLoop}$ for initializing the variant $StateLoop$, a second one Evt_{Loop1} for the body of the loop and decreasing of the variant, and a third one Evt_0 for ending the loop and returning to the event of the abstract level.

```
**REFINEMENT** $RefIterative_{T0}$
**REFINES** $T_0$
**INVARIANT**
  $J(var_i, var_j) \wedge StateLoop \in NAT \wedge Start \in \{0, 1\}$
**ASSERTIONS**
  $G_0 \Rightarrow Start = 0 \vee (StateLoop > 0 \wedge G_1 \wedge Start = 1) \vee (StateLoop = 0 \wedge G'_0 \wedge Start = 1)$
**VARIANT**
  $StateLoop + Start$
**INITIALISATION**
  $Start := 1 \parallel \ldots$
**EVENTS**
$Evt_{InitLoop} =$               $Evt_{Loop1} =$                                   $EvtT_0 =$
**SELECT**                       **SELECT**                                        **SELECT**
  $Start = 1$                      $G_1 \wedge StateLoop > 0 \wedge Start = 0$           $G'_0 \wedge StateLoop = 0 \wedge Start = 0$
**THEN**                         **THEN**                                          **THEN**
  $StateLoop :\in NAT \parallel$     $StateLoop := StateLoop - 1 \parallel$                $S'_0$
  $Start := 0$                     $S_{Loop}$                                        **END;**
**END;**                         **END;**
```

The variant corresponding to the number of iteration is initialized by the $Evt_{InitLoop}$ event and then the Evt_{Loop1} event is fired $StateLoop$ times. When the loop terminates, $EvtT_0$ is fired. The variant $StateLoop$ decreases from its arbitrary initial value to 0. The **ASSERTIONS** clause states that one of the events guards is always true.

The initial value of the variant is arbitrary fixed thanks to the $:\in$ operator. The advantage of such an approach is the possibility to encode an arbitrary number of loop steps without increasing the complexity of the proof process. Compared to model checking techniques, increasing the number of loop steps may lead to the combinatorial explosion problem.

Example of the use of the iterative process translation - Let us consider the sum of two numbers obtained by performing the sum of aa and bb using a loop operator. The idea consists in first performing $RSum := AA$ and then adding, BB times, the value 1 to $RSum$. In this case, the variant will be the variable BB. Therefore, this example does not use an explicit variable $StateLoop$ to represent the variant.

```
**REFINEMENT** $RefIterative_{T0}$
**REFINES** $Sum_{T0}$
**INVARIANT**
  $(RSum + AA + BB) = (aa + bb) \wedge \ldots$
**ASSERTIONS**
  $(AA = 0 \wedge BB = 0 \wedge aa + bb \in NAT) \vee \ldots$
**VARIANT**
  $AA + BB$
**INITIALISATION**
  $RSum := 0 \parallel Sum :\in NAT \parallel aa, AA :\in (aa \in NAT \wedge AA = aa) \parallel \ldots$
**EVENTS**
$EvtT_0 =$
**SELECT**
  $AA = 0 \wedge BB = 0 \wedge \ldots$
**THEN**
  $Sum := RSum$
**END;**

$Evt_{InitLoop} =$
**SELECT**
  $AA \neq 0 \wedge BB \neq 0 \wedge \ldots$
**THEN**
  $RSum := RSum + AA \parallel$
  $AA := 0$
**END;**

$Evt_{Loop1} =$
**SELECT**
  $AA = 0 \wedge BB \neq 0 \wedge \ldots$
**THEN**
  $RSum := RSum + 1 \parallel$
  $BB := BB - 1$
**END;**
```

The previous Event B model uses three events. The event Evt_{Loop_1} decreases the variable BB until its value becomes 0. When the variant equals to 0, $EvtT_0$ is fired to return to the events of the abstraction.

Concurrency "||"

Let us consider $T_0 ::= T_1 || T_2$. The semantics of concurrency in interleaving semantics imposes to describe all the possible behaviors. Therefore, this process is described by all the possible traces. It uses the interleaving underlying Event B semantics i.e. if two events have their guard to true, they are fired in parallel, in an interleaving manner.

Two events $EvtT_1$ and $EvtT_2$ corresponding to processes T_1 and T_2 are defined. They can be fired at any time.

```
**REFINEMENT** $RefConcurrency_{T0}$
**REFINES** $T_0$
**INVARIANT**
  $J(var_i, var_j) \wedge StateConc_1 \in \{0,1\} \wedge StateConc_2 \in \{0,1\}$
**ASSERTIONS**
  $G_0 \Rightarrow ((G_1 \wedge StateConc_1 = 1) \vee (G_2 \wedge StateConc_2 = 1) \vee \ldots$
**VARIANT**
  $StateConc_1 + StateConc_2$
**INITIALISATION**
  $StateConc_1 :\in NAT \parallel StateConc_2 :\in NAT1 \parallel \ldots$
**EVENTS**
$Evt_1 =$
**SELECT**
  $G_1 \wedge StateConc_1 = 1$
**THEN**
  $StateConc_1 = 0 \parallel S_1$
**END;**

$Evt_2 =$
**SELECT**
  $G_2 \wedge StateConc_2 = 1$
**THEN**
  $StateConc_2 = 0 \parallel S_2$
**END;**

$Evt_0 =$
**SELECT**
  $G'_0 \wedge StateConc_1 = 0 \wedge StateConc_2 = 0$
**THEN**
  $S'_0$
**END;**
```

Once the events $EvtT_1$ and $EvtT_2$ are fired, they cannot be fired again. Their variant ($StateConc_1$ and $StateConc_2$ respectively) is decreased from any natural number ($:\in$) to 0 and the abstract event of the refined model can be fired again (Evt_0).

Example of the use of the concurrency operator translation - For the same example, we have imagined a refinement with a concurrent operator. Indeed, two concurrent events are defined. $EvtT_1$ adds the value of AA to $RSum$ when the event $EvtT_2$ adds the value BB to $RSum$. When these two events are fired, the last event $EvtT_0$ is fired to compute the final result for the abstraction.

```
REFINEMENT $RefConcurrency_{T0}$
REFINES $Sum_{T0}$
INVARIANT
  $(RSum + AA + BB) = (aa + bb) \wedge \ldots$
ASSERTIONS
  $(AA = 0 \wedge BB = 0 \wedge aa + bb \in NAT) \vee \ldots$
VARIANT
  $AA + BB$
INITIALISATION
  $RSum := 0 \parallel Sum :\in NAT \parallel aa, AA :\in (aa \in NAT \wedge AA = aa) \parallel \ldots$

EVENTS
$EvtT_0 =$
SELECT
  $AA = 0 \wedge BB = 0 \wedge \ldots$
THEN
  $Sum := RSum$
END;

$EvtT_1 =$
SELECT
  $AA \neq 0 \wedge \ldots$
THEN
  $RSum := RSum + AA \parallel$
  $AA := 0$
END;

$EvtT_2 =$
SELECT
  $BB \neq 0 \wedge \ldots$
THEN
  $RSum := RSum + BB \parallel$
  $BB := 0$
END;
```

The variant is the sum of variables AA and BB. However, if the model requires not to change AA nor BB then, the developer could have used explicit variants.

Translation of other CTT constructions

Up to now, $>>$, $||$, [ ] and $*$ operators have been described in Event B models. All the other CTT constructions (*order independency*, *optional process*, *disabling*, *interruption*) are described below. Thanks to the interleaving semantics of the Event B method, the translation of these operators uses the basic operators defined previously.

Order independency "$|=|$"

Let us consider $T_0 ::= T_1 \models| T_2$. In trace based semantics, order independency is interpreted by:

$$T_0 ::= T_1 \models| T_2 \; is\; translated\; to\; T_0 ::= (T_1 >> T_2) \, [] \, (T_2 >> T_1) \tag{7}$$

The enabling and choice basic operators are used for this translation. They define the interleaved traces encoding the order independency operator.

Optionality "$[Task]$"

Let us consider $T_0 ::= [T_1]$. The optional process indicates that process T_1 may be accomplished. In this case, we introduce the atomic empty process, namely T_{Skip}. Then, optionality is translated to a choice between the process T_1 or the empty process. We get:

$$T_0 ::= [T_1] \; is\; translated\; to\; T_0 ::= T_1 \, [] \, T_{Skip} \tag{8}$$

The generalized substitution of the process T_{Skip} is simply $skip$.

Disabling ”[>”

Let us consider $T_0 ::= T_1[> T_2$ where T_1 is disabled by T_2. Disabling requires a case based processing: either T_1 is atomic (i.e. cannot be refined or decomposed) or not.

Indeed, if T_1 is an atomic process, then it means that either T_1 is performed or T_2 is performed. We get a translation by a choice operator:

$$T_0 ::= T_1[> T_2 \; is\; translated\; to\; T_0 ::= T_1 \,[]\, T_2 \tag{9}$$

When T_1 is not an atomic process (i.e. it involves CTT operators defining observable states) disabling can occur in the trace defined by T_1. If T_1 is decomposed into $T_{1,1}\ op_1\ T_{1,2}\ op_2 \cdots op_n\ T_{1,n+1}$, then the disabling translation is defined by the choice of all the possible traces resulting from the disabling i.e. disabling can occur at each observable state of the T_1 decomposition and shall be propagated in the further decompositions. We get:

$$\begin{aligned} T_0 ::= T_{1,1}\ op_1\ T_{1,2}\ op_2 \quad & \cdots op_n\ T_{1,n+1}[> T_2 \; is\; translated\; to \\ & T_0 ::= T_2 \\ & []\, T_{1,1} >> T_2 \\ & []\, T_{1,1} op_1 T_{1,2} >> T_2 \\ & []\, T_{1,1} op_1 T_{1,2} \cdots op_i T_{1,i+1} >> T_2 \\ & []\cdots >> T_2 \\ & []\, T_{1,1} op_1 T_{1,2} \cdots op_n T_{1,n+1} \end{aligned} \tag{10}$$

Remark. When disabling occurs, T_1 is stopped. The system may be in a corrupted state which may correspond to a bad state of the system.

When we use the Event B method to encode the disabling operator, it may happen that the proof obligations cannot be proved. Indeed, since a state is corrupted it does not correspond to a correct process decomposition and the gluing invariant is not preserved by the refinement which encodes the disabling. The use of Event B allows to generate proof obligations that cannot be proved. In this case, a repairing event can be added in the disabling decomposition. The repairing event will artificially repair the system in order to return to a non corrupted state. This event enforces the preservation of the gluing invariant and provides for a formal debugging.

Let us consider a small decomposition of $T_0 ::= T_1[> T_2$ where T_1 is not an atomic process. We give below a partial refinement specification of the T_0 process.

```
REFINEMENT
  $DisablingRef$
INITIALISATION
  $DisablingState := 3 \parallel \ldots$

EVENTS
$EvtT_{11} =$
SELECT
  $G_{11} \wedge DisablingState = 2$
THEN
  $S_{11}$
END;

$EvtT_{12} =$
SELECT
  $G_{12} \wedge DisablingState = 1$
THEN
  $S_{12} \parallel DisablingState := 0$
END;

$Evt_{Disabling} =$
ANY $pp$
WHERE
  $pp \in 1, 2 \wedge$
  $\neg DisablingState = 0$
THEN
  $DisablingState := pp$
END;

$EvtT_1 =$
SELECT
  $G_1 \wedge DisablingState = 0$
THEN
  $S_1$
END;
```

Disabling infinite loop process " $^*[>$"

Let us consider $T_0 ::= T_1^*[> T_2$. In this case, the translation uses the same reasoning as the previous classical disabling. We use the previous translation to define intermediate disabling in each iteration. We get:

$$\begin{gathered} T_0 ::= T_1^*[> T_2 \; is\; translated\; to\; T_0 ::= (T_1)^N[> T_2 \\ where\; N\; is\; any\; arbitrary\; natural\; number \end{gathered} \tag{11}$$

Notice that the Event B method allows to define such an arbitrary natural number and to perform proofs on this basis. This capability represents one advantage on model checking techniques where such a reasoning is not possible.

Finally, if the process T_1 is not atomic, then disabling can occur in each observable state of the T_1 decomposition as defined in disabling operator subsection.

Interruption "$|>$"

Let us consider $T_0 ::= T_1| > T_2$. The process T_1 is interrupted by the process T_2. As for the disabling operator, interruption translation requires a case based reasoning.

If T_1 is an atomic process (not involving CTT operators), then it means that T_2 can be performed an arbitrary number of times (may be 0) and then T_1 is performed. In this case, we get:

$$T_0 ::= T_1| > T_2 \; is\; translated\; to\; T_0 ::= T_2^N >> T_1 \tag{12}$$

When T_1 is not an atomic process involving CTT operators the interruption can occur many times in each observable state of the trace defined by T_1. If T_1 is written as $T_{1,1}\; op_1\; T_{1,2}\; op_2 \cdots\; op_n\; T_{1,n+1}$, then the interruption translation is defined by the choice off all the possible traces resulting from the interruption. We get:

$$\begin{gathered} T_0 ::= T_{1,1}\; op_1\; T_{1,2}\; op_2 \cdots\; op_n\; T_{1,n+1}| > T_2 \\ is\; translated\; to \\ T_0 :: T_2^{N_1} >> T_{1,1} >> T_2^{N_2} op_1 T_{1,2} \cdots >> T_2^{N_{n+1}} op_n T_{1,n+1} \end{gathered} \tag{13}$$

Here N_i are arbitrary natural numbers showing that the process T_2 associated to interruption can occur zero or several times. Notice that we have chosen to interpret the interruption operator using this approach. One could have chosen to activate interruption exactly once ($N_i = 1$).

APPLICATION FOR THE SPECIFICATION AND VALIDATION OF UI

One of the major aspects, in the User Interface (UI) development activity, is the capability to take into account usability of the UI. In general, usability is captured a posteriori through experimentation and/or a priori through the description of a set of tasks representing scenarios of use. The validation of UI in critical systems like plane cockpits or machine factory interfaces use such scenarios.

The work we have performed with the previously described approach deals with the latter. We have represented tasks by decomposable processes. We consider that a set of tasks is described using a user task notation and the designed UI shall meet the requirements expressed within these tasks.

Notations and models for design and validation of HCI

In general, the development of UI is concerned by two important interleaving phases.

1. A *design phase* which allows to produce the code implementing the suited UI and its link with the functional core (heart of the application). In this phase, architectural notations, verification, validation, specification, refinement and programming techniques are used by UI developers. Among the design notations and techniques we can cite the Seeheim model [33], PAC[18], ARCH[14, 13], hybrid models [24, 22] and so on. Usually this phase allows to establish *robustness properties*.

2. A *task validation phase* which consists in validating user needs. This phase is not well mastered by UI designers since most of these validations are issued from non computer scientists like psychologists and ergonomists. A set of tasks, defining scenarios, is described at the requirements level and shall be supported by the final UI product. Among the description oriented techniques and notations one can cite MAD [34], XUAN [23] and CTT [31]. This list is not exhaustive and may be completed. Usually this phase allows to establish *validation properties*.

Unfortunately, the different models and notations we outlined above do not give enough formal representation to allow the complete validation and verification, at the specification and design levels, of the UI design with respect to given properties and user tasks. In general, verification and validation are reported at the testing phase when all the software development is completed.

Therefore, representing formally both design and tasks, at early stages of the development (abstract levels) will permit such verification and validation. However, thanks to the possibility of describing formal abstract models provided by formal techniques, it becomes possible to handle a large amount of validation and verification efforts early at the specification and design phases.

The proof based techniques we are using help to increase the quality of UI software developments. Indeed, our approach uses the Event B formal technique for representing, verifying and refining specifications [10, 5] [3] and [4]. In [5, 6], we presented our approach, based on Event B, handling the design phase.

Applications to Human Computer Interaction area

User requirements validation is performed by way of the validation of a set of user tasks. A task can be seen as a scenario of use of the interface. It can be validated using several techniques: running a prototype, simulating or animating a model or model checking or proof techniques.

For validating user interfaces tasks, we have used the CTT (ConcurTaskTrees) process algebra and its representation in Event B previously presented. Two main applications have been developed : one consists in validating WIMP interfaces (Windows, Icons, Mouse and Pointers) and the second one deals with multi-modal interfaces. Before describing these two applications, we overview the way we have represented the dialog controller of an UI which represents the kernel of any UI.

Encoding the dialog controller with Event B

A set of B models is described. They allow to encode all the parts of an UI software from the functional core of the application to the toolkit, presentation and dialog controller.

The dialog controller contains all the events that can be fired by an user while using the interface. These events are *atomic*. In their turn, these events fire other events from the presentation, toolkit or from the functional core. The guards of these events define their firing order and how these events interleave.

We do not give the details of this approach for the description of the UI in this paper but more details can be found in [10, 5, 4, 3] and [8].

Application to the WIMP user interfaces[7, 9, 12]

This first application consists in describing the whole CTT operators using Event B models for the WIMP user interfaces. Each decomposition of an upper task in the CTT tree corresponds to an Event B refinement. A CTT task is described by an initial state and a final state. It is refined into a sequence of atomic 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 atomic events, of the *dialog controller* are reached in the leaves. When the atomic events of the dialog controller are reached by the refinement, the validation process is completed.

The previous encoding of CTT in Event B has been experimented on several task models for WIMP applications. Complete examples may be found in [8]. Moreover, the developed examples have shown that it is possible to have a generic translation of CTT task trees allowing task validation also for plastic interfaces.

Let us consider our approach on a small WIMP case study which allows to convert euros to dollars and vice-versa. The user enters, in a textfield component, the value he/she wants to convert then he/she makes the conversion by pushing either the €>> $ button component to convert euros to dollars, or $ >> € button component to convert dollars to euros. The converted value is displayed thanks to a textfield component.

The dialog controller - A list of atomic events defining the transition system of the dialog controller of the exchange currency application is given below.

Event Name	Description
Evt_{Exit}	Click on exit button
$Evt_{ExitApplication}$	Close exchange application
$Evt_{InputValue}$	Input any convert value
$Evt_{ReadOutputValue}$	Updating all views after a conversion
$Evt_{ConvertInEuro}$	Click on dollar to euro button
$Evt_{ConvertInDollar}$	Click on euro to dollar button

An user task model - A potential user task model, giben below, of the exchange currency application has been experimented from our encoding CTT. Notice that the leaves of this user task model correspond of the atomic avents of the dialog controller.

$$\begin{array}{ll} ExchangeApplication & = ExchangeTask^{*}[> Evt_{Exit} >> Evt_{ExitApplication} \\ ExchangeTask & = Evt_{InputValue} >> ChoiceOfConversion >> Evt_{ReadOutputValue} \\ ChoiceOfConversion & = Evt_{ConvertInEuro}[]Evt_{ConvertInDollar} \end{array}$$

Application to the multi-modal user interfaces [28, 11]

In this kind of application, several modalities (i.e. Speech, Gesture or Direct Manipulation) are available. They may be used in order to build an interaction with the system. The building process associated to the interaction is based on a composition of sub-interactions which are themselves compositions of other sub-interactions and so on, until basic events (from dialog controller) are reached. Composition operators are needed in order to build such multi-modal interactions. So, the application to the multi-modal user interfaces consists in applying the whole CTT operators translation of previously introduced to compose multi-modal interactions. Regarding the WIMP applications, multi-modal applications need to express and check by Event B method another kind of properties, the CARE (Complementary, Assignment, Redundancy and Equivalence) properties [19].

As a case study, we have specified and validated the multimodal MATIS (Mulitmodal Airline Travel Information System) application defined in [30]. This application allows an user to retrieve information about flights schedules using speech and/or direct manipulation with keyboard and mouse, or a combination of these modalities. This interaction mode supports individual and synergistic use of multiple input modalities. The information about flights schedules correspond to the parameters of the request, such as the departure and arrival city names and the min and max departure hours.

The dialog controller - Below the list of the atomic events of the dialog controller MATIS case study is given. It corresponds to the transition system defining the multi-modal interactive application.

Event Name	Description
$Evt_{CityDepartDM}$	Input the departure city thanks to the direct manipulation modality
$Evt_{CityDepartSpeech}$	Input the departure city thanks to the voice modality
$Evt_{CityDepartGesture}$	Input the departure city thanks to the gesture modality
$Evt_{CityDepartDMSG}$	Input the departure city thanks to all the three modalities
$Evt_{CityArrivalDM}$	Input the arrival city thanks to the direct manipulation modality
$Evt_{CityArrivalSpeech}$	Input the arrival city thanks to the voice modality
$Evt_{CityArrivalGesture}$	Input the arrival city thanks to the gesture modality
$Evt_{CityArrivalDMSG}$	Input the arrival city thanks to all the three modalities
$Evt_{ResultOfRequest}$	Display the result of request
$\ldots$	$\ldots$

An user task model - We have experimented our encoding CTT user tasks model on the MATIS application. A potential user task model is shown below. This user task model expresses the capability to modify the different parameters before processing the request (iterative process) and the capability to input parameter values in different orders using a concurrent interaction mode involving all the three modalities (Speech, Direct Manipulation and Gesture).

$$\begin{array}{ll}
SearchFly & = InputData^{*} [> Evt_{ResultOfRequest} \\
InputData & = CityDeparture || CityArrival || MinHourDepart || MaxHourDepart \\
CityDeparture & = Evt_{CityDepartDM} [] Evt_{CityDepartSpeech} [] \\
 & \quad Evt_{CityDepartGesture} [] Evt_{CityDepartDMSG} \\
CityArrival & = Evt_{CityArrivalDM} [] Evt_{CityArrivalSpeech} [] \\
 & \quad Evt_{CityArrivalGesture} [] Evt_{CityArrivalDMSG} \\
MinHourDepart & = \ldots \\
MaxHourDepart & = \ldots
\end{array}$$

USE OF THE EVENT B METHOD. PROOF TECHNIQUE VERSUS MODEL CHECKING TECHNIQUE

The previous translation rules cover the whole CTT language for user tasks modelling and description. These translation rules give not only a syntactical translation, but also give a formal semantics using the Event B method semantics for the CTT language. All these rules are implemented and are tool supported. The Atelier B [17] tool is used for this purpose. We have developped several examples of CTT tasks using this approach.

When a CTT task is defined, the corresponding decomposition tree corresponds to a set of models and refinements designed using event B. These models contain **ASSERTIONS** clauses expressing the soundness of events occurrence thanks to the variant behavior and to the guard disjunction property. In addition to the task model validation provided by the event B technique, these clauses allow to express other properties and to prove them thanks to the used technique. It means that it is possible to validate or invalidate properties on the CTT tasks descriptions.

Finally, the use of arbitrary natural numbers in the interruption and disabling operators is possible using the **ANY** . . . **WHERE** . . . **THEN**. . . **END** or $:\in$ event B operators. The possibility of using arbitrary natural numbers allow to deal with all the possible cases for tasks descriptions and modelling. Moreover, the proving system supported by event B method allows to prove all the properties expressed in these models. Notice that this is almost impossible in model checking techniques, where a fixed value for the natural numbers is required. Usually the state number explosion problem arises when these natural numbers increase.

Nevertheless, we claim that model checking techniques and proof based techniques shall be used in conjunction in order to get the benefits of both: automatic proving for model checking techniques and state number explosion avoidance for proof based techniques. Proof based techniques have been used to validate user needs in WIMP (Windows Icons Menus and Pointers) applications[8] and to express and to check multi-modal properties[11].

CONCLUSION

This paper has presented an approach allowing to translate a BNF grammar to a set of Event B models. The translation rule we defined consists in associating a first event B model to the left hand side of a BNF rule and a second one to the right hand side of this rule. The second B model refines the first one. Variants are defined to ensure the correct firing order of events in these models. The abstract syntax tree describes the hierarchy of refined models. When these models are built, it is possible to enrich them by defining state variables, transitions and invariants expressing relevant properties. The interest of such models is to establish properties and to check them a priori allowing early validation/verification. Moreover, we have applied this translation rule to the BNF grammar defining process algebra expressions. These expressions were used to describe tasks and scenarios used to validate user interfaces.

Although the proposed approach produced satisfactory results, principally for user interfaces validation, it needs a deep study. Indeed, there is a need to address the following points :

1. study of other BNF examples and other languages for which useful proofs could be performed. These examples will improve the translation rule definition;
2. study the theoretical aspects. What are the necessary and/or sufficient conditions that shall be fulfilled by a BNF grammar to be translatable in Event B models;
3. and finally study and/or develop an automatic (or semi-automatic) tool that allows parsing of BNF grammars to a hierarchy of Event B models corresponding to an abstract syntax tree.

References

[1] J-R Abrial. Extending b without changing it (for developing distributed systems). In H Habrias, editor, *First B Conference, Putting Into Pratice Methods and Tools for Information System Design*, page 21, Nantes, France, 1996.

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

[3] Y. Aït-Ameur, B. Bréholée, L. Guittet, F. Jambon, and P. Girard. Formal Verification and Validation of Interactive Systems Specifications. Technical report, LISI/ENSMA, March 2000.

[4] Y. Aït-Ameur and P. Girard. Specification, Design, Refinement and Implementation of Interactive Systems: the B Method. Technical report, LISI/ENSMA, March 2001.

[5] Y. Aït-Ameur, P. Girard, and F. Jambon. Using the B Formal Approach for Incremental Specification Design of Interactive Systems. In S. Chatty and P. Dewan, editors, *IFIP TC2/WG2.7 Engineering for Human-Computer Interaction*, pages 91–110. Kluwer Academic Publishers, 1998.

[6] Yamine Aït-Ameur. Cooperation of formal methods in an enngeneering based software development process. In Spring Verlag LNCS 1945, editor, *Second International conference on Integrated Formal Methods, IFM*, pages 136–155, Schloss Dagstuhl, 2000.

[7] Yamine Aït-Ameur and Mickaël Baron. Bridging the gap between formal and experimental validation approaches in hci systems design : use of the event b proof based technique. In Department of Computer Science University of Cyprus, editor, *ISOLA 2004 - 1st International Symposium on Leveraging Applications of Formal Methods*, pages 74–81, Paphos, Cyprus, 2004.

[8] Yamine Aït-Ameur and Mickaël Baron. Bridging the gap between formal and experimental validation approaches in hci systems design : use of the event b proof based technique (revue). *Accepted in International Journal on Software Tools for Technology Transfer Special Section*, 2005.

[9] 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.

[10] Yamine Aït-Ameur, Patrick Girard, and Francis Jambon. A uniform approach for the specification and design of interactive systems: the b method. In Panos Markopoulos and Peter Johnson, editors, *Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'98)*, volume Proceedings, pages 333–352, Abingdon, UK, 1998.

[11] Yamine Aït-Ameur and Nadjet Kamel. A generic formal specification of fusion of modalities in a multimodal hci. In Ren Jacquart, editor, *IFIP World Computer Science*, pages 415–420, Toulouse, France, 2004. Kluwer Academic Publishers.

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

[13] L. Bass, E. Hardy, K. Hoyt, R. Little, and R. Seacord. The arch model : Seeheim revisited, the serpent run time architecture and dialog model. Technical Report CMU/SEI-88-TR-6, Carnegie Melon University, 1988.

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

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

[16] Dominique Cansell. *Assistance au dveloppement incrmental et sa preuve*. Habilitation diriger les recherches, Universit Henri Poincar, 2003.

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

[18] 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.

[19] J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, and R.M. Young. Four easy pieces for assessing the usability of multimodal interaction: the CARE properties. In *Proceedings of Human Computer Interaction - Interact'95*, pages 115–120. Chapman and Hall, 1995.

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

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

[22] Jean-Daniel Fekete. *Un modèle multicouche pour la construction d'applications graphiques interactives*. Doctorat d'université (phd thesis), Université Paris-Sud, 1996.

[23] 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 UAN : - Aspect symtrique USER/SYSTEM - Contraintes temporelles - Paramtrisation des tches - Pr et Post-conditions.

[24] Laurent Guittet. *Contribution 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.

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

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

[27] C.A.R. Hoare and N. Wirth. An Axiomatic Definition of the Programming Language Pascal. *Acta-Informatica*, 23(2):335–355, 1973.

[28] Nadjet Kamel. Utilisation de smv pour la vrification de proprits d'ihm multimodales. In *16 Confrence Francophone sur l'Interaction Homme-Machine (IHM'2004)*, volume 1, pages 219–222, Namur, Belgique, 2004. ACM Press.

[29] K. Lano. *The B Language Method: A Guide to Practical Formal Development*. Springer Verlag, 1996.

[30] Laurence Nigay. *Conception et Modlisation Logicielle des Systmes Interactifs : Application aux Interfaces Multimodales*. Doctorat d'universit (phd thesis), Universit Joseph Fourier, 1994.

[31] F Patern, 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.

[32] Fabio Patern. *Model-Based Design and Evaluation of Interactive Applications*. Springer, 2001.

[33] Gnther E Pfaff, editor. *User Interface Management Systems, Proceedings of the Workshop on User Interface Management Systems held in Seeheim*. Eurographic Seminars. Springer-Verlag, Berlin, 1985.

[34] 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.

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