Validation d'IHM3 par animation de modèles B

SP 3
LOT 4
LISI/ENSMA
SILICOMP/AQL

Auteurs : Y. Aït-Ameur, I. Aït-Sadoune, J-M Mota et L. Van Aertryck
Adresse : LISI / ENSMA - Téléport 2 - 1, avenue Clément Ader - B.P. 40109 - 86960 Futuroscope Cedex - France
Emails : {yamine, idir.aitsadoune, mota}@ensma.fr, lionel.van.aertryck@aql.fr

---

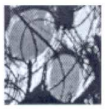

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Introduction

Jusqu'à présent tous les développements conduits dans le cadre de Verbatim portent sur le raffinement et la preuve.

Dans un soucis de validation par l'expérimentation nous nous sommes penchés sur l'animation et le test de modèles B événementiel. En effet, il nous paraît indispensable de permettre aux concepteurs de savoir si les modèles événementiels obtenus sont conformes aux besoins utilisateurs en utilisant le test ou bien l'animation de modèles B en respectant les modèles de tâches qui ont été définis.

Pour cela nous avons réalisé un travail consistant à animer et à tester les modèles B événementiel en deux étapes.

La première étape consiste à animer les modèles B événementiel. Le principe est de suivre les traces des événements déclenchés. Nous avons pour cela traduit les modèles B dans le langage de modélisation de données EXPRESS. Les traces des exécutions sont rangées dans des instances d'objets EXPRESS. L'outil résultant est dénommé B2EXPRESS. Ce travail est présenté dans le chapitre 2 de ce document.

La seconde étape est liée à la génération de tests. Nous avons interfacé l'outil B2EXPRESS avec l'outil CASTING d'AQL, partenaire du projet. CASTING a permis de générer des tests sous forme d'instances d'objets événements qui peuvent être lus par B2EXPRESS. Ce travail est présenté dans le chapitre 3 de ce document.

Notons que ces travaux ont été réalisés dans la cadre de la validation d'IHM3, mais ils restent génériques et applicables à tout modèle B événementiel.

Chapitre 2

Validation d'IHM3 par animation de modèles B

2.1 Introduction

Nous avons proposé dans le LOT3 une démarche pour la validation des tâches utilisateurs par un modèle événementiel de tâche construit à partir d'une tâche CTT. C'est une démarche qui se base essentiellement sur le raffinement et la preuve. Pendant l'activité de preuve, il arrive que l'on trouve des difficultés à prouver l'invariant. Une question se pose : est ce que c'est un problèmes au niveau de la preuve ou c'est un problème au niveau de la tâche. Pour cela, on propose de compléter l'activité de preuve par une activité de test et d'animation des modèles B obtenus pour pouvoir détecter les blocages et les invariants non satisfaits. En effet, il nous paraît indispensable de permettre aux concepteurs de savoir si les modèles événementiels obtenus sont conformes aux besoins utilisateurs en utilisant le test ou bien l'animation de modèles B en respectant les modèles de tâches qui ont été définis.

La méthode d'animation proposée repose essentiellement sur le langage $EXPRESS$. Dans ce chapitre nous présentons la méthode d'animation en commençant par la définition du langage $EXPRESS$, nous détaillons par la suite les règles de passage entre B événementiel et $EXPRESS$ et le principe d'animation [AS05]. Comme exemples, nous appliquons cette méthode à nos deux études de cas $Matis$ et $PagesJaunes$. Enfin, nous présentons $B2Express$ un outil développé dans le but d'automatiser la méthode d'animation présentée.

2.2 Le langage de modélisation EXPRESS

EXPRESS [ISO 10303.11-94] [IS094, SW94] est un langage de modélisation conceptuelle et de spécification des données conçu dans le cadre du projet STEP (STandard for the Extange of Product model data), officiellement connu sous la référence ISO 10303. Son principal objectif est la description de modèles de données. Dans ce langage, l'accent est mis sur la précision du modèle et tout particulièrement sur les contraintes respectées par les éléments des modèles. C'est aussi un formalisme de spécification.

2.2.1 Les concepts

Les éléments de base dont dispose le langage $EXPRESS$ pour décrire un schéma sont : les entités, les attributs, les types, les contraintes et les fonctions/procédures. En général, un modèle $EXPRESS$ est un ensemble d'**entités** regroupées dans un **schéma**.

Les entités

Une entité désigne un élément identifiable dans le domaine étudié et elle a souvent une existence autonome et peut représenter une catégorie. Des relations d'héritage et d'association entre entités peuvent exister. En plus de ces concepts, on peut remarquer la notion d'abstraction d'une entité. Dans ce cas, l'entité ne peut-être instanciée qu'au travers de ses sous entités.

Nous montrons dans le code suivant une partie d'un schéma $EXPRESS$. Cet exemple montre une entité abstraite E et qui est un super type. Il montre aussi deux entités $E1$ et $E2$ qui sont dérivées de l'entité E. L'entité $E2$ est à son tour un super type d'une entité ($E22$) qui n'apparaît pas sur l'exemple.

```
ENTITY E
  ABSTRAST SUPERTYPE OF (E1, E2);
END_ENTITY;

ENTITY E1
  SUBTYPE OF (E);
  ...
END_ENTITY;

ENTITY E2
  SUPERTYPE OF (E22)
  SUBTYPE OF (E);
  ...
END_ENTITY;
```

Les attributs

Chaque entité est décrite par un ensemble d'attributs. Leurs domaines de valeurs peuvent être des domaines simples (les entiers, les réels), des collections (listes, ensembles), des entités (associations entre entités) ou des domaines construits à partir d'un type utilisateur. L'ensemble de ces domaines est décrit plus en détail dans la partie suivante consacrée aux types. Les attributs quant à eux possèdent plusieurs statuts pour une entité, ils peuvent être libres, optionnels ou dérivés à partir d'autres attributs.

L'exemple suivant montre deux entités dont les attributs sont de types différents. Dans l'entité A, nous avons un attribut att_A de type simple et un attribut att_I de type entité inverse. Pour l'entité B, les attributs sont de type simple pour att_1, de type collection pour att_2, de type entité pour att_3 et de type simple dérivé pour att_4.

```
ENTITY A;
  att_A : INTEGER
INVERSE
  att_I : FOR att_3
END_ENTITY;
...
ENTITY B;
  att_1 : REAL;
  att_2 : LIST [0 :?] OF STRING;
  att_3 : A;
DERIVE
  att_4 : BOOLEAN := (SELF.att_3 A.att_A = SIZEOF(SELF.att_2));
END_ENTITY;
```

Les types

Les types définissent les domaines auxquels appartiennent les attributs, les variables ou les paramètres d'une fonction ou une procédure. Les types peuvent être de plusieurs formes : les types simples ($BOOLEAN$, $INTEGER$, $REAL$, $STRING$), les collections ($LIST$, SET, BAG, $ARRAY$), les types nommés (définis par l'utilisateurs), les types $SELECT$ et les types $ENUMERATION$.

Les contraintes

Le langage $EXPRESS$ offre un langage de description de contraintes très élaboré. On distingue deux types de contraintes : les contraintes locales qui s'appliquent individuellement sur chacune des instances de l'entité où elles sont définies ($WHERE$) et les contraintes globales qui nécessitent une vérification globale sur l'ensemble des instances d'une entité donnée ($UNIQUE$, $INVERSE$ et $RULE$).

L'exemple suivant montre le cas d'une contrainte locale définie sur les attributs de l'entité A. Dans la clause $WHERE$, un prédicat exprime des propriétés sur l'attribut att_A de type $INTEGER$ et dont la valeur doit être comprise entre 1 et 10. Il donne aussi un cas d'une contrainte globale définie sur l'ensemble des instances de l'entités A ($RULE\ Card\ FOR\ A$), elle exprime le fait que les instances dont l'attribut att_A est égale à 1, sont au nombre de 2.

```
ENTITY A;
  att_A : INTEGER;
WHERE
  (SELF.att_A >= 1) and (SELF.att_A <= 10);
END_ENTITY;
...
RULE Card FOR A;
WHERE
  SIZEOF(QUERY(inst < *A|(inst.att_A = 1))) = 2;
END_RULE;
```

Les fonctions et procédures

Les fonctions et les procédures sont introduites respectivement par les mots clés $FUNCTION$ et $PROCEDURE$. Le langage de description s'apparente au langage $PASCAL$. Des variables peuvent être déclarées localement et les structures de contrôle classiques ($REPEAT$, IF,...) sont disponibles.

L'exemple suivant montre la structure d'une fonction en $EXPRESS$ dont les paramètres d'entrée sont x et y de types $type\_1$ et $type\_2$ respectivement. Le paramètre de sortie est de type $type\_3$.

```
FUNCTION F ( x : type_1; y type_2 ) : type_3;
  (*Function_Body;*)
END_FUNCTION;
```

2.2.2 Représentation des instances :le fichier physique

Une population de données instances d'entités conforme à un modèle de données $EXPRESS$ est représentée au sein d'un fichier physique. Celui-ci est un fichier de caractères répondant à une syntaxe définie dans [ISO10303-21] précisant, en particulier, les règles de traduction des définitions $EXPRESS$ pour décrire des instances.

Les instances d'un modèle $EXPRESS$ sont décrites et échangées à travers un fichier textuel. Chaque instance est identifiée par un OID (#i). Elle est caractérisée par le nom de l'entité instanciée et est décrite par la suite des valeurs de ses attributs représentées entre parenthèses. Les collections d'éléments sont définies entre parenthèses. Les attributs optionnels sont désignés par '$' et les types énumérés par

une notation encadrant l'identificateur par un point à chacune de ses extrémités. Les attributs dérivés et inverses ne sont pas représentés.

L'exemple suivant donne un aperçu de la structure d'un fichier physique.

```
# 1 = A (3, #2);
# 2 = B (4 , ('hello' , 'bye') , #1);
```

Après avoir donné un aperçu du langage $EXPRESS$ et ses différents concepts, nous présentons dans ce qui suit le principe utilisé pour animer les modèles B avec le langage $EXPRESS$ et le différentes règles utilisées pour passer du modèle B vers un modèle $EXPRESS$.

2.3 Approche proposée

B événementiel permet de décrire un système dont le modèle peut-être vu comme un système de transitions. L'état est défini dans la clause **VARIABLES** et les transitions sont représentées par les événements déclarés dans la clause **EVENTS**. Pour animer un modèle B, nous traduisons ce dernier en un modèle $EXPRESS$. Pour cela, on définit un méta modèle des systèmes de transitions, puis on définit des règles de passage entre le modèle B événementiel et le modèle $EXPRESS$. Le modèle final obtenu est dérivé du méta-modèle et préserve les propriétés du modèle B événementiel.

Enfin, pour pouvoir animer et tester un modèle B, nous exploitons le fait qu'il est possible d'instancier un modèle $EXPRESS$ pour déclencher des événements particuliers.

2.3.1 Un méta modèle pour les automates

Le schéma du méta-modèle des systèmes de transitions comprend l'ensemble des éléments nécessaires à la description d'un système de transitions. Les différents états du système sont représentés par une entité sans attributs nommée STATE. Les transitions sont modélisées par une entité nommée TRANSITION qui possède deux attributs de type STATE. Ces derniers décrivent les valeurs de l'état de départ de la transition (DepState) et l'état atteint par la transition (ArrState). L'état initial du système est décrit dans l'entité INITIALISATION par l'attribut InitState.

```
ENTITY STATE
    ABSTRACT SUPERTYPE;
END_ENTITY;

ENTITY EVENTS
    ABSTRACT SUPERTYPE OF(ONEOF(INITIALISATION, TRANSITION));
END_ENTITY;

ENTITY INITIALISATION
    ABSTRACT SUPERTYPE
    SUBTYPE OF(EVENTS);
    InitState : STATE;
END_ENTITY;

ENTITY TRANSITION
    ABSTRACT SUPERTYPE
    SUBTYPE OF(EVENTS);
    DepState : STATE;
    ArrState : STATE;
END_ENTITY;
```

Pour exploiter ce modèle dans nos travaux d'animation et de test, le schéma a été enrichi par une entité TEST qui contient la séquence de test représentée par une liste ordonnée de transitions. L'objectif est de voir si cette séquence représente un chemin possible de notre système de transitions. Une contrainte locale de l'entité TEST est définie pour garantir que pour chaque séquence, pour deux transitions qui se succèdent l'état de départ de la $(N+1)^{eme}$ est l'état d'arrivée de la $(N)^{eme}$. Cette contrainte vérifie aussi que la séquence commence par l'état initial de notre système.

```
ENTITY TEST
    ABSTRACT SUPERTYPE;
    SEQUENCE : LIST [1 :?] OF EVENTS;
WHERE
    wr1 : VERIF_TEST( SEQUENCE ) = TRUE;
END_ENTITY;

FUNCTION VERIF_TEST ( t : LIST [1 :?] OF EVENTS ) : BOOLEAN;
    -- fonction qui vérifie que pour deux transitions qui se succèdent, l'état de départ
    -- de la N+1 ème est l'état d'arrivée de la N ème
    -- elle vérifie aussi que la séquence commence par l'état initial de l'automate
END_FUNCTION;
```

Nous détaillons dans la section suivante les règles de transformation d'un modèle B événementiel vers un modèle $EXPRESS$ qui va servir pour l'animation. Le modèle résultant exploite aussi les ressources définies dans le méta-modèle.

2.3.2 Règles de transformation d'un modèle B en un modèle EXPRESS

Les grandes lignes du processus de traduction d'un modèle B événementiel vers un modèle EXPRESS reposent sur les lignes suivantes :

- représenter les variables de l'état du modèle B (Clause VARIABLES) par des attributs dans une entité EXPRESS dérivée de l'entité STATE du méta-modèle ;
- les propriétés décrites dans la clause INVARIANT sont traduites par des contraintes globales définit sur l'entité STATE ;
- la clause INITIALISATION est traduite par une entité dérivée de l'entité INITIALISATION du méta-modèle ;
- les événements de la clause EVENTS sont traduits par des entités dérivées de l'entité TRANSITION du méta-modèle.

Nous détaillerons le processus de traduction pour chaque clause.

La clause VARIABLES et la partie typage de la clause INVARIANT

Les variables de l'état sont récupérées à partir de la clause VARIABLES et leurs types de la partie typage de la clause INVARIANT. Chaque variable est représentée par un attribut du même nom et du type équivalent en EXPRESS dans une entité nommée STATE_REF dérivée de l'entité STATE du méta-modèle. Le code suivant montre un exemple de transformation de cette clause.

VARIABLES *queryReady*, *querySending*, ... **INVARIANT** $queryReady \in INTEGER \wedge$ $querySending \in INTEGER \wedge$ ...	**ENTITY** *STATE_REF* **SUBTYPE OF** (*STATE*); $queryReady : INTEGER$; $querySending : INTEGER$; ... **END_ENTITY;**

La clause INVARIANT

L'invariant est d'abord re-écrit sous la forme disjonctive. Les parties qui décrivent les propriétés de typage ne sont pas prises en compte. Chaque sous expression obtenues de la forma conjonctive est introduite à l'intérieur d'une contrainte globale au niveau du modèle EXPRESS. Cette contrainte est appliquée sur l'entité STATE_REF, du coup, l'invariant au niveau du modèle B doit être vérifié par l'état du système et les contraintes obtenues en EXPRESS doivent être respectées

par l'ensemble des instances de l'entité STATE. Le code suivant montre un exemple de transformation d'une partie d'un invariant.

```
...                                          ...
INVARIANT                                    RULE INVARIANT_0 FOR (STATE_REF);
    ...                                          WHERE
    queryRead ∈ {0,1}∧                           QUERY(s <* STATE_REF|TRUE) =
    querySending ∈ {0,1}∧                        QUERY(s <* STATE_REF|
    ...                                              ( s.queryReady = 0 OR s.queryReady = 1 ));
    queryReady ≠ querySending∧               END_RULE;
...
                                             RULE INVARIANT_1 FOR (STATE_REF);
                                                 QUERY(s <* STATE_REF|TRUE) =
                                                 QUERY(s <* STATE_REF|
                                                     ( s.querySending = 0 OR s.querySending = 1 ));
                                             END_RULE;
                                             ...
                                             RULE INVARIANT_5 FOR (STATE_REF);
                                                 QUERY(s <* STATE_REF|TRUE) =
                                                 QUERY(s <* STATE_REF|
                                                     ( s.queryReady <> s.querySending ));
                                             END_RULE;
                                             ...
```

La clause INITIALISATION

La clause INITIALISATION du modèle B représente l'état initial du système modélisé. Au niveau du modèle EXPRESS, une entité INIT est dérivée du méta modèle du système de transitions à partir de l'entité INITIALISATION. Cette dernière possède un attribut InitState de type STATE. Les substitutions de la clause INITIALISATION du modèle B sont représentées au niveau de l'entité INIT sous forme d'une contrainte locale sur l'attribut InitState et qui exprime les valeurs obtenues après l'application des différentes substitutions.

```
INITIALISATION                               ENTITY INIT
    ...                                          SUBTYPE OF (INITIALISATION);
    queryReady := 1 ||                           WHERE
    querySending := 0 ||                         ...
    ...                                          (InitState.queryReady = 1) AND
                                                 (InitState.querySending = 0) AND
                                                 ...
                                             END_ENTITY;
```

La clause EVENTS

Chaque événement de la clause EVENTS est traduit par une entité EXPRESS dérivée de l'entité TRANSITION du méta modèle. Elle en hérite les attributs DepState et ArrState. La garde et les substitutions de chaque événement sont exprimées

sous forme de contraintes locale à la transition. La garde est la contrainte que doit satisfaire l'état de départ de la transition (DepState) et les substitutions sont traduites sous forme d'une contrainte qui exprime les résultats de l'application des substitutions sur l'état d'arrivée de la transition (ArrState). Enfin, une contrainte supplémentaire est ajoutée pour garantir que les variables d'état qui n'ont pas été modifiées par l'événement gardent toujours leurs valeurs de l'état de départ.

... $QueryReady =$ **SELECT** $(EV1 = 0 \wedge EV2 = 0 \wedge EV3 = 0 \wedge EV4 = 0) \wedge$ $queryReady = 1$ **THEN** $queryReady := 0 \parallel$ $querySending := 1$ **END ;**	**ENTITY** $QueryReady$ **SUBTYPE OF** $(TRANSITION)$; **WHERE** $garde : ((DepState.EV1 = 0)AND(DepState.EV2 = 0)AND$ $(DepState.EV3 = 0)AND(DepState.EV4 = 0))\ AND$ $(DepState.queryReady = 1)$; $sub : (ArrState.queryReady = 0)AND$ $(ArrState.querySending = 1)$ $const : ...$ $(ArrState.EV1 = DepState.EV1)\ AND$ $(ArrState.EV2 = DepState.EV2)\ AND$ $(ArrState.EV3 = DepState.EV3)\ AND$ $(ArrState.EV4 = DepState.EV4)$; **END_ENTITY ;**

La suite est consacrée à la description du processus d'animation et l'utilisation du fichier d'instances $EXPRESS$ pour l'écriture des scénarios de tests. Nous expliquons comment les différentes clauses d'un modèle B sont vérifiées à partir d'un modèle $EXPRESS$ et enfin, nous donnons un aperçu de l'animation à des niveaux différents du raffinement.

2.4 Animation du modèle

La sémantique d'entrelacement des événements des modèles B permet d'écrire les scénarios de test qui vont servir pour l'animation sous forme d'une suite d'événements. Les différents scénarios de tests permettent de vérifier le respect des différentes clauses et contraintes du modèle B comme l'invariant et l'assertion. Dans cette section, nous détaillons le contenu du fichier d'instances, la validation des propriétés et des clauses et la prise en compte du raffinement dans l'animation.

2.4.1 Description de l'animation

Les scénarios de test sont des populations de données instances d'entités conformes au modèle de données $EXPRESS$ obtenu après traduction du modèle B à animer. Ces différents scénarios sont introduits sous forme d'une liste d'événements, comme

instances de l'attribut SEQUENCE de l'entité TEST (#30 et #31). Dans notre exemple, les événements sont représentés par leurs OID (entre #20 et #28).

ENTITY TEST
  ABSTRACT SUPERTYPE;
  SEQUENCE : LIST [1 :?] OF EVENTS;
  WHERE
  wr1 : VERIF_TEST(SEQUENCE) = TRUE;
END_ENTITY;
	...
	$\#30 = TEST([\#20, \#21, \#22, \#23, \#24])$
	$\#31 = TEST([\#20, \#25, \#26, \#27, \#28])$
	...

Les événements sont regroupés dans une entité abstraite EVENT de laquelle on dérive l'entité INITIALISATION et l'entité TRANSITION. L'entité INITIALISATION, a un attribut InitState, les entités TRANSITIONS quand à elles, ont deux attributs (DepState et ArrState). Tous ces attributs, sont de types STATE.

ENTITY *INIT*
  SUBTYPE *OF* (*INITIALISATION*);
  WHERE
  ...
ENTITY *SetName*
  SUBTYPE *OF* (*TRANSITION*);
  WHERE
  ...
ENTITY *SetAdress*
  SUBTYPE *OF* (*TRANSITION*);
  WHERE
  ...
ENTITY *QueryReady*
  SUBTYPE *OF* (*TRANSITION*);
  WHERE
  ...
	...
	$\#20 = INIT(\#1)$
	$\#21 = SETNAME(\#1, \#2)$
	$\#22 = SETADRESS(\#2, \#3)$
	$\#23 = QUERYREADY(\#3, \#4)$
	...
	$\#25 = SETADRESS(\#6, \#7)$
	$\#26 = SETNAME(\#7, \#8)$
	$\#27 = QUERYREADY(\#8, \#9)$
	...

Les différentes instances de l'état (#1-#9) sont définies à partir de l'état initiale jusqu'au dernier état envisagé par le scénario en respectant les différents changements provoqués par les transitions sur l'état.

ENTITY *STATE_REF*
  SUBTYPE OF (*STATE*);
  *EV*1 : *INTEGER*;
  *EV*2 : *INTEGER*;
  queryReady : *INTEGER*;
  querySending : *INTEGER*;
END_ENTITY;
	...
	$\#1 = STATE\_REF(1, 1, 1, 0)$
	$\#2 = STATE\_REF(0, 1, 1, 0)$
	$\#3 = STATE\_REF(0, 0, 1, 0)$
	$\#4 = STATE\_REF(0, 0, 0, 1)$
	...

Après avoir introduit les scénarios de test dans le fichier d'instances, nous vérifions si l'invariant et les différentes propriétés (contraintes locales, les gardes des événements) sont respectés.

2.4.2 Validation des propriétés

La validation des propriétés d'un modèle B par l'animation consiste à vérifier toutes les contraintes globales et locales définies au niveau du schéma $EXPRESS$ et qui en même temps représentent les propriétés du système modélisé telles que les invariants, les gardes des événements et les différentes substitutions des événements. En plus des contraintes relatives au modèle B, des contraintes ont été ajoutées pour vérifier que les différentes séquences de tests ont un format bien défini (commence toujours par l'entité $INIT$) et qu'à chaque niveau des différentes transitions on a l'état initial de la (i+1)ème transition égal à l'état final de la ième transition.

La vérification est assurée par l'outil *ECCO Toolkit* [SM97], cet outil lit les différents scénarios de test depuis le fichier d'instances et réalise une vérification (checking) pour contrôler si les différentes instances correspondent bien au modèle décrit dans le schéma $EXPRESS$. Dans le cas contraire un message d'erreur est renvoyé en plus de la localisation de la contrainte non respectée sur le code source du schéma $EXPRESS$.

Vérification de l'invariant

Dans le schéma $EXPRESS$, l'invariant est défini par des contraintes globales sur l'entité $STATE\_REF$ qui représente l'état. Ces contraintes sont vérifiées sur l'ensemble des instances de cette entité. En d'autres termes, l'invariant est une contrainte qui doit être vérifiée par tous les états du système au niveau de toutes les transitions.

Comme nous l'avons expliqué dans la section 2.3.2, l'invariant est découpé en sous expressions dans des contraintes globales. Le but de ce découpage et de pouvoir localiser la sous expression de l'invariant qui n'est pas vérifiée s'il y a une erreur provoquée par le test. Cela permet aussi de détecter les variables impliquées et les substitutions qui ont provoqué cette erreur à l'intérieur de l'événement responsable de cette erreur.

Vérification des gardes et des substitutions des événements

Au niveau de chaque entité de type TRANSITION, les contraintes locales définies sur l'attribut DepState et ArrState sont vérifiées, elles correspondent respectivement aux gardes et aux substitutions des événements dans le modèle B.

Vérification des propriétés du système de transition

Une contrainte locale définie au niveau de l'entité $TEST$ sert à vérifier qu'un scénario de test correspond à une séquence du système de transitions que représente le modèle $EXPRESS$. Elle vérifie également que le scénario de test commence par une initialisation et est suivi par une suite de transitions dont l'état initial de la (i+1)ème est égal à l'état final de la ième transition.

2.4.3 Animation et raffinement

En général, un développement en B événementiel comporte une abstraction et plusieurs niveaux de raffinements. Pour vérifier l'ensemble des invariants et des propriétés de chaque niveau, on procède à la traduction de l'ensemble du développement en plusieurs modèles $EXPRESS$ correspondant à chaque niveau de raffinement. On définit également les scénarios de tests correspondants à chaque niveau. Ainsi, on se place au niveau de n'importe quel raffinement et on procède à son animation et à la vérification de ses invariants et de ses différentes propriétés.

Dans le cadre du projet VERBATIM, cette méthode d'animation a été appliquée sur deux études de cas qui sont $Matis$ et $PagesJaunes$. Les résultats de cette application sont détaillées dans la section suivante.

2.5 Application à l'étude de cas

Pour les besoins de l'animation, nous reprenons les modèles B événementiel des études de cas $Matis$ et $PagesJaunes$ obtenus avec le scénario 4 de l'approche expliqué dans le SP3-LOT2. La présentation des deux études de cas y est également. Nous présentons d'abord le modèle $EXPRESS$ obtenu, puis nous étudions des exemples de scénarios de test.

2.5.1 Application à "Matis"

Pour caractériser l'état des modalités utilisées par l'application $Matis$, un nouveau type nommé $STATE$ est crée au niveau de la clause $SETS$ du modèle B. Ce nouveau type représente un ensemble qui comporte deux éléments on et off. Il est traduit en un nouveau type utilisateur énuméré au niveau du modèle $EXPRESS$.

... **SETS** $STATE = \{on, off\}$	... **TYPE** $STATE$ = **ENUMERATION OF** (on, off); **END_TYPE**;

Les variables de l'état de l'application *Matis* représentées dans la cause **VARIABLES** et la partie typage de l'**INVARIANT** du modèle B, sont représentées au niveau des attributs de l'entité $STATE\_REF$ dérivée de l'entité $STATE$ qui modélise l'état.

... **VARIABLES** $queryReady, querySending,$ $MouseModal,$ $KeyboardModal,$ $SpeechModal,$ $EV1, EV2, EV3, EV4$	... **ENTITY** $STATE\_REF$ **SUBTYPE OF** (STATE); $queryReady : INTEGER;$ $querySending : INTEGER;$ $MouseModal : STATE;$ $KeyboardModal : STATE;$ $SpeechModal : STATE;$ $EV1 : INTEGER;$ $EV2 : INTEGER;$ $EV3 : INTEGER;$ $EV4 : INTEGER;$ **END_ENTITY**;

Les propriétés du système décrites dans la clause $INVARIANT$ sont traduites par des contraintes globales au niveau du modèle $EXPRESS$. Par exemple, la propriétés définie sur la variable $queryReady$ ($queryReady \in \{0,1\}$), se traduit par la contrainte globale $INVARIANT\_0$. Les autres propriétés qui définissent le domaine des valeurs possibles des variables $querySending$, $EV1$, $EV2$, $EV3$ et $EV4$, sont traduites de la même façon. La contrainte $INVARIANT\_6$ traduit une partie de l'invariant qui définit une propriété sur les variables $queryReady$ et $querySending$ ($queryReady \neq querySending$).

... **INVARIANT** ... $queryReady \in \{0,1\} \wedge$ $querySending \in \{0,1\} \wedge$ $EV1 \in \{0,1\} \wedge$ $EV2 \in \{0,1\} \wedge$ $EV3 \in \{0,1\} \wedge$ $EV4 \in \{0,1\} \wedge$ $queryReady \neq querySending$ ...	... **RULE** $INVARIANT\_0$ FOR $(STATE\_REF)$; **WHERE** $QUERY(s <* STATE\_REF \mid TRUE) =$ $QUERY(s <* STATE\_REF \mid$ $s.queryReady = 0$ OR $s.queryReady = 1)$; **END_RULE**; ... **RULE** $INVARIANT\_6$ FOR $(STATE\_REF)$; **WHERE** $QUERY(s <* STATE\_REF \mid TRUE) =$ $QUERY(s <* STATE\_REF \mid$ $s.queryReady <> s.querySending)$; **END_RULE**;

Au niveau du modèle B, l'état initial de *Matis* est défini dans la clause **INITIALISATION**. En ce qui concerne le modèle $EXPRESS$, les substitutions de cette clause sont exprimées sous forme d'une contrainte locale à l'entité $INIT$. Cette contrainte s'applique sur l'attribut $InitState$ qui représente l'état initial.

INITIALISATION $queryReady := 1 \parallel$ $querySending := 0 \parallel$ $MouseModal := on \parallel$ $Keyboard := on \parallel$ $SpeechModal := on \parallel$ $EV1 := 1 \parallel$ $EV2 := 1 \parallel$ $EV3 := 1 \parallel$ $EV4 := 1$	**ENTITY** $INIT$ **SUBTYPE OF** $(INITIALISATION)$; **WHERE** $(InitState.queryReady = 1)\ AND$ $(InitState.querySending = 0)\ AND$ $(InitState.MouseModal = on)\ AND$ $(InitState.KeyboardModal = on)\ AND$ $(InitState.SpeechModal = on)\ AND$ $(InitState.EV1 = 1)\ AND$ $(InitState.EV2 = 1)\ AND$ $(InitState.EV3 = 1)\ AND$ $(InitState.EV4 = 1)$; **END_ENTITY**;

Vu le nombre élevé d'événements que contient le modèle B de *Matis*, on se contente de présenter un seul exemple avec l'événement $EV_{InputDeparture}$. La garde de l'événement est représentée sous forme d'une contrainte locale à l'entité $EV_{InputDeparture}$ étiquetée **garde** et les substitutions sont représentées dans une contrainte locale étiquetée **sub**. Les variables d'état, qui ne sont pas modifiées par cet événement, sauvegardent leur état initial.

$EV_{InputDeparture}$ = **SELECT** $(MouseModal = on \vee$ $KeyBoardModal = on \vee$ $SpeechModal = on) \wedge$ $EV1 = 1$ **THEN** $EV1 := 0$ **END;**	**ENTITY** $EV_{InputDeparture}$ **SUBTYPE OF** $(TRANSITION)$; **WHERE** $GARDE : ((DepState.MouseModal = on)\ OR$ $(DepState.KeyboardModal = on)\ OR$ $(DepState.SpeechModal = on))\ AND$ $(DepState.EV1 = 1)$; $SUB : ArrState.EV1 = 0$; $cst0 : ArrState.EV4 = DepState.EV4$; $cst1 : ArrState.EV3 = DepState.EV3$; $cst2 : ArrState.EV2 = DepState.EV2$; $cst3 : ArrState.MouseModal = DepState.MouseModal$; $cst4 : ArrState.KeyboardModal = DepState.KeyboardModal$; $cst5 : ArrState.SpeechModal = DepState.SpeechModal$; $cst6 : ArrState.queryReady = DepState.queryReady$; $cst7 : ArrState.querySending = DepState.querySending$; **END_ENTITY**;

Après la description du modèle $EXPRESS$ obtenu par la traduction du modèle B, nous animerons ce modèle à travers l'expression de scénarios utilisateurs dans le fichier d'instances $EXPRESS$. Le premier scénario consiste en l'exécution de la tâche suivante, elle commence par la saisie des paramètres de la requête (ville de départ, ville arrivée, heure départ et heure d'arrivée dans l'ordre), puis lancer la requête :

$$Evt_{InputDeparture} >> Evt_{InputArrival} >> Evt_{HoureDeparture} >> Evt_{HoureArrival} >> QueryReady >> QuerySending$$

Les différents états de ce scénario sont définis par les instances de l'entité $STATE\_REF$ identifiées entre #1 et #7. L'entité $INIT$ (#8) représente l'état initial et elle a comme attribut l'état #1. L'ensemble des entités qui représentent les événements du scénario (de #8 à #14), ont comme attributs deux états ($DepState$ et $ArrState$) et prennent leurs valeurs dans l'ensemble des instances de l'entité $STATE\_REF$. Ces événements sont regroupés dans l'entité $TEST$ (#15) pour former la séquence qui correspond au scénario.

$\#1 = STATE\_REF\ (1, 0, on, on, on, 1, 1, 1, 1);$
$\#2 = STATE\_REF\ (1, 0, on, on, on, 0, 1, 1, 1);$
$\#3 = STATE\_REF\ (1, 0, on, on, on, 0, 0, 1, 1);$
$\#4 = STATE\_REF\ (1, 0, on, on, on, 0, 0, 0, 1);$
$\#5 = STATE\_REF\ (1, 0, on, on, on, 0, 0, 0, 0);$
$\#6 = STATE\_REF\ (0, 1, off, off, off, 0, 0, 0, 0);$
$\#7 = STATE\_REF\ (1, 0, on, on, on, 1, 1, 1, 1);$
$\#8 = INIT\ (\#1);$
$\#9 = Evt_{InputDeparture}\ (\#1, \#2);$
$\#10 = Evt_{InputArrival}\ (\#2, \#3);$
$\#11 = Evt_{HoureDeparture}\ (\#3, \#4);$
$\#12 = Evt_{HoureArrival}\ (\#4, \#5);$
$\#13 = QueryReady\ (\#5, \#6);$
$\#14 = QuerySending\ (\#6, \#7);$
$\#15 = TEST\ ([\#8, \#9, \#10, \#11, \#12, \#13, \#14]);$

2.5.2 Application à "Pages Jaunes"

L'état de l'application $PagesJaunes$ est représenté par un ensemble de variables décrivant l'interface de l'application et les différentes modalités utilisées. Ces variables sont reprises comme des attributs au niveau de l'entité $STATE\_REF$ qui représente l'état dans le schéma $EXPRESS$ obtenu par la traduction du modèle B.

...	...
VARIABLES	**ENTITY** $STATE\_REF$
$querySending, EV1$	**SUBTYPE OF** (STATE);
$Nn, Na, Nn2, lock,$	$querySending : INTEGER;$
$map, name, name2, adress,$	$EV1 : INTEGER;$
$V\_speech\_speech,$	$Nn : INTEGER;$
$V\_mouse\_speech,$	$Na : INTEGER;$
$V\_mouse\_keyboard,$	$Nn2 : INTEGER;$
$V\_speech\_keyboard$	$lock : BOOLEAN;$
	$map : BOOLEAN;$
	$name : STRING;$
	$name2 : STRING;$
	$adress : STRING;$
	$V\_speech\_speech : INTEGER;$
	$V\_mouse\_speech : INTEGER;$
	$V\_mouse\_keyboard : INTEGER;$
	$V\_speech\_keyboard : INTEGER;$
	END_ENTITY;

L'invariant est écrit sous la forme conjonctive et présente quatre sous expressions. Chaque sous expression est reprise dans une contrainte globale au niveau du schéma $EXPRESS$ obtenu par la transformation du modèle B vers le modèle $EXPRESS$.

... **INVARIANT** ... $(not(lock = FALSE) \vee (Nn2 = Nn - 1)) \wedge$ $(not(lock = TRUE) \vee (Nn2 = Nn)) \wedge$ $(not(EV1 \neq 2) \vee (lock = TRUE)) \wedge$ $(not(Nn = 0) \vee (lock = TRUE)) \wedge$	... **RULE** $INVARIANT\_0$ FOR $(STATE\_REF)$; **WHERE** $QUERY(s <* STATE\_REF \mid TRUE) =$ $QUERY(s <* STATE\_REF \mid$ $(NOT(s.lock = FALSE)\ OR\ (s.Nn2 = s.Nn - 1)))$; **END_RULE**; **RULE** $INVARIANT\_1$ FOR $(STATE\_REF)$; **WHERE** $QUERY(s <* STATE\_REF \mid TRUE) =$ $QUERY(s <* STATE\_REF \mid$ $(NOT(s.lock = TRUE)\ OR\ (s.Nn2 = s.Nn)))$; **END_RULE**; ... **RULE** $INVARIANT\_3$ FOR $(STATE\_REF)$; **WHERE** $QUERY(s <* STATE\_REF \mid TRUE) =$ $QUERY(s <* STATE\_REF \mid$ $(NOT(s.Nn = 0)\ OR\ (s.lock = TRUE)))$; **END_RULE**;

L'état initial de système qui est décrit au niveau de la clause INITIALISATION du modèle B, se traduit en une entité INIT dont l'attribut InitState est de type STATE hérité de l'entité INITIALISATION. Les différentes substitutions de l'initialisation sont traduites sous formes d'une contrainte locale (clause WHERE) sur l'attribut InitState.

INITIALISATION $querySending := 1 \parallel$ $map := FALSE \parallel$ $EV1 := 3 \parallel$ $name :\in STRING \parallel$ $name1 :\in STRING \parallel$ $adress :\in STRING \parallel$ $Nn := 1 \parallel$ $Na := 1 \parallel$ $Nn2 := 1 \parallel$ $lock := TRUE \parallel$ $V\_speech\_speech :\in NATURAL \parallel$ $V\_mouse\_speech :\in NATURAL \parallel$ $V\_mouse\_keyboard :\in NATURAL \parallel$ $V\_speech\_keyboard :\in NATURAL \parallel$	**ENTITY** $INIT$ **SUBTYPE OF** $(INITIALISATION)$; **WHERE** $(init.querysending = 1)\ AND$ $(init.map = FALSE)\ AND$ $(init.EV1 = 3)\ AND$ $('STRING'\ IN\ TYPEOF(init.name))\ AND$ $('STRING'\ IN\ TYPEOF(init.name1))\ AND$ $('STRING'\ IN\ TYPEOF(init.adress))\ AND$ $(init.Nn = 1)\ AND$ $(init.Na = 1)\ AND$ $(init.Nn2 = 1);$ $(init.lock = TRUE)\ AND$ $('INTEGER'\ IN\ TYPEOF(init.V\_speech\_speech))\ AND$ $('INTEGER'\ IN\ TYPEOF(init.V\_mouse\_speech))\ AND$ $('INTEGER'\ IN\ TYPEOF(init.V\_mouse\_keyboard))\ AND$ $('INTEGER'\ IN\ TYPEOF(init.V\_speech\_keyboard));$ **END_ENTITY;**

Vu le nombre très important des événements qui forment ce raffinement, nous avons choisi de décrire l'événement qui $EV_{InputAdress}$ dont le rôle est de récupérer la valeur de l'un des paramètres de la requête de l'application $PagesJaunes$ qui est l'*adresse*. L'événement est repris sous forme d'une entité $EXPRESS$ qui hérite de l'entité $TRANSITION$ les attributs $DepState$ et $ArrState$ qui sont de type $STATE$. La garde de l'événement est présentée comme une contrainte locale étiquetée $GARDE$ définie sur l'attribut $DepState$. Par contre les substitutions sont au niveau de la contrainte locale de l'événement étiqueté SUB appliquée à l'attribut $ArrState$. Cette contrainte présente les différentes transformations que provoque l'événement sur l'état.

$EV_{InputAdress}$ = **SELECT** $EV1 = 2 \wedge$ $Na \neq 0$ **THEN** $Na := Na - 1 \parallel$ $adress :\in STRING$ **END ;**	**ENTITY** $EV_{InputDeparture}$ **SUBTYPE OF** ($TRANSITION$); **WHERE** $GARDE : (DepState.EV1 = 2)\ AND$ $(DepState.Na <> 0)$; $SUB : (ArrState.Na = DepState.Na - 1)\ AND$ $('STRING'\ IN\ TYPEOF(ArrState.adress))$); $cst0 : ArrState.QuerySending = DepState.QuerySending$; $cst1 : ArrState.EV1 = DepState.EV1$; $cst2 : ArrState.Na = DepState.Na$; $cst3 : ArrState.Nn2 = DepState.Nn2$; $cst4 : ArrState.lock = DepState.lock$; $cst5 : ArrState.map = DepState.map$; $cst6 : ArrState.name = DepState.name$; $cst7 : ArrState.name2 = DepState.name2$; $cst8 : ArrState.V\_speech\_speech = DepState.V\_speech\_speech$; $cst9 : ArrState.V\_mouse\_speech = DepState.V\_mouse\_speech$; $cst10 : ArrState.V\_mouse\_keyboard = DepState.V\_mouse\_keyboard$; $cst11 : ArrState.V\_speech\_keyboard = DepState.V\_speech\_keyboard$; **END_ENTITY** ;

Nous proposons de tester le scénario suivant qui consiste à lancer une requête pour les paramètres *name* et *adress* contenant les valeurs *idir* et *poitiers*. Ces paramètres de la requête sont introduits dans le système en utilisant les modalités $MOUSE$ et $KEYBOARD$. Ce scénario se traduit par la tâche suivante :

$NAME\_ADRESS >> MOUSE\_KEYBOARD$
$>> EV_{INPUTNAME} >> EV_{INPUTADRESS}$
$>> SEARCH >> QUERY >> RESULTQUERY$

Cette tâche est représentée par le fichier d'instances $EXPRESS$ suivant

$$
\begin{array}{l}
\#1 = STATE\_REF\ (0,3,1,1,1,'','','',.F.,.T.,0,0,0,0); \\
\#2 = STATE\_REF\ (0,2,1,1,1,'','','',.F.,.T.,0,0,1,0); \\
\#3 = STATE\_REF\ (0,2,1,0,1,'',' idir','',.F.,.F.,0,0,1,0); \\
\#4 = STATE\_REF\ (0,2,0,0,1,' idir',' idir','',.F.,.T.,0,0,0,0); \\
\#5 = STATE\_REF\ (0,2,0,0,0,' idir',' idir',' poitiers',.F.,.T.,0,0,0,0); \\
\#6 = STATE\_REF\ (0,1,0,0,0,' idir',' idir',' poitiers',.F.,.T.,0,0,0,0); \\
\#7 = STATE\_REF\ (1,0,0,0,0,' idir',' idir',' poitiers',.F.,.T.,0,0,0,0); \\
\#8 = STATE\_REF\ (0,3,0,0,0,' idir',' idir',' poitiers',.T.,.T.,0,0,0,0); \\
\#9 = INIT\ (\#1); \\
\#10 = NAME\_ADRESS\ (\#1,\#2); \\
\#11 = MOUSE\_KEYBOARD\ (\#2,\#3); \\
\#12 = EV_{INPUTNAME}\ (\#3,\#4); \\
\#13 = EV_{INPUTADRESS}\ (\#4,\#5); \\
\#14 = SEARCH\ (\#5,\#6); \\
\#15 = QUERY\ (\#6,\#7); \\
\#16 = RESULTQUERY\ (\#7,\#8); \\
\#17 = TEST\ ([\#9,\#10,\#11,\#12,\#13,\#14,\#15,\#16]);
\end{array}
$$

Cette méthode d'animation de modèle B est enrichie par un outil B2EXPRESS. Avec cet outil, le processus de traduction des modèle B vers un schéma EXPRESS et l'instanciation du modèle EXPRESS pour déclencher des événements B est complètement caché pour l'utilisateur. L'outil reçoit en entrée le modèle B et offre plusieurs modes pour générer des scénarios d'animation et de tests. La section suivante a pour but de décrire cet outil.

2.6 Description du prototype B2EXPRESS

L'outil $B2EXPRESS$ permet d'animer les modèles événementiels exprimés avec B. Un modèle B est traduit en un schéma EXPRESS qu'il est possible d'instancier pour déclencher des événements particuliers. Il est utilisé dans le but de contrôler si des scénarios correspondent à une utilisation licite ou non du système d'événements décrit avec B. B2EXPRESS permet d'animer ces modèles selon plusieurs modes, permettant de détecter des blocages ou bien des invariants non satisfaits.

L'outil peut être exploité dans deux activités distinctes :

- *Le test* : lire un fichier qui contient les scénarios de test générés manuellement ou par d'autres outils (CASTING [Aer98]) ;
- *L'animation* : déclencher une suite d'événements via l'interface de l'outil en cliquant sur les différents boutons étiquetés par les noms des événements du modèle B.

En ce qui concerne l'acivité de test, le chapitre suivant donnera plus de détails sur l'utilisation de l'outil pour l'éxécution des scénarios de test. Nous nous intéressons plus particulièrement dans la section suivante à l'animation des modèles B.

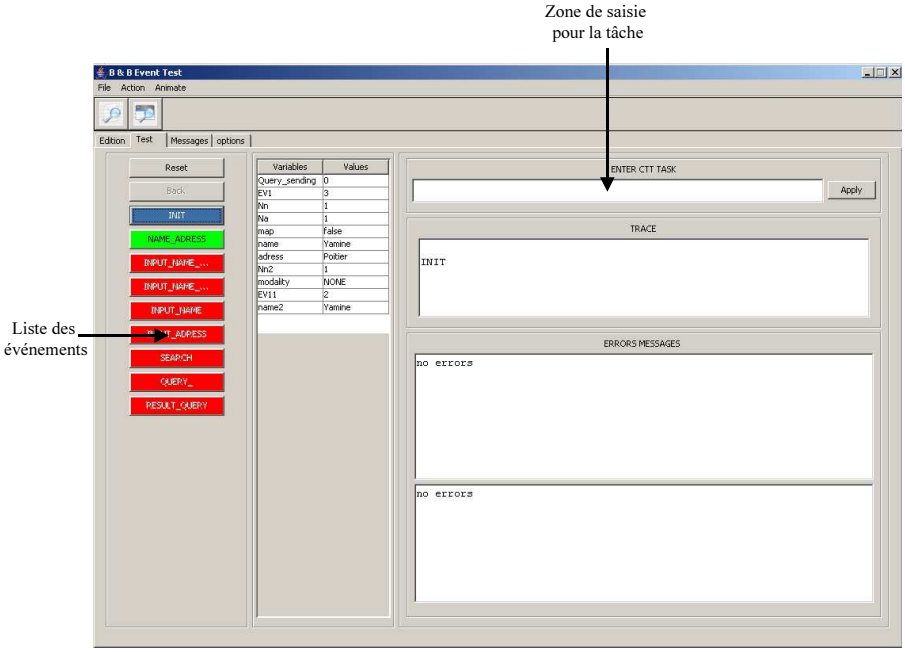

FIG. 2.1 – l'interface de B2Express

2.6.1 Modes d'animation

B2EXPRESS permet d'animer selon plusieurs modes (*mode correct, mode libre et mode tâche*). L'intérêt de chaque mode est différent. Le mode correct permet à l'utilisateur d'avoir un choix pour déclencher les événements B dont les gardes sont correctes. Le mode libre laisse le choix à l'utilisateur de déclencher n'importe quel événement, même ceux dont la garde est fausse, c'est une option qui permet de détecter des blocages ou bien des invariants non satisfaits. En fin le mode tâche, permet de déclencher des événements selon deux contraintes, la garde de l'événement doit être vérifiée et l'enchaînement des événements déclenchés doivent correspondre à une expression d'algèbre de processus passée en entrée de l'outil (EXEMPLE CTT). Il est possible de choisir d'autres options pour la vérification. L'outil permet également de restreindre les contraintes vérifiées en cours d'un test. Il est possible de vérifier l'invariant seul, l'assertion, les gardes des événements seulement ou de vérifier l'ensemble des contraintes.

2.6.2 Vérification des contraintes

B2EXPRESS interroge ECCO Toolkit [SM97] qui contrôle la conformité des scénarios d'animation et de test par rapport au modèle EXPRESS et renvoie des messages. Les messages renvoyés sont récupérés par l'outil B2EXPRESS qui dispose d'un interprèteur des messages d'erreur de ECCO. En cas de blocage au niveau des événements ou de détection d'un invariant non satisfait, l'outil affiche un message d'erreur conforme et localise l'erreur sur le code source du modèle B passé en entrée comme le montre la figure 2.2.

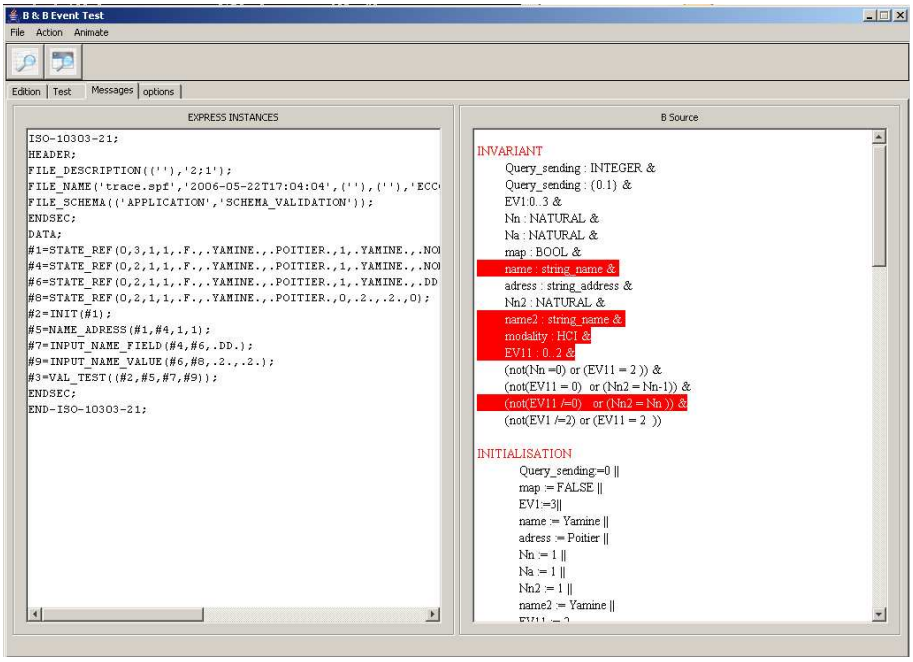

FIG. 2.2 – Localisation d'une erreur sur le source du modèle B

2.7 Conclusion

Nous avons proposé une méthode pour animer les modèles *B* événementiel avec un outil *B2Express* qui permet de générer des scénarios de tests selon plusieurs modes. Il offre aussi plusieurs options pour la vérification des clauses d'un modèle B événementiel. L'animation de modèles B nous permet de compléter l'activité de preuve et de validation de modèle de tâche par des scénarios d'animation et de test conformes aux besoins utilisateurs.

Comme perspective à ce travail, nous envisageons d'intégrer les propriétés propres au raffinement comme la prise en compte de l'invariant de collage. Il est aussi très intéressant de réfléchir autour d'une méthode qui permettra de jouer le même scénario de test à des niveaux différents du raffinement.

Chapitre 3

Génération de tests avec CASTING

3.1 Introduction

Nous avons montré qu'il est possible d'animer les modèles B selon différents modes. L'outil B2EXPRESS développé permet de déclencher des séquences d'événements de B correspondants à un entrelacement.

L'étape de travail suivante s'est intéressée à la génération de ces séquences en utilisant des techniques de génération de tests. Pour cela, nous avons, en collaboration avec AQL, interfacé l'outil CASTING avec l'outil B2EXPRESS. Dans ce chapitre nous abordons ce travail.

Remarque. Ce chapitre fait référence au SP3-LOT 5 qui décrit les détails de la génération de tests.

3.2 Casting

Pour la description de l'outil CASTING [Aer98], nous renvoyons le lecteur à la section 2 du SP3-LOT 5.

3.3 Connexion entre B2EXPRESS et Casting

Lorsque l'on soumet à CASTING le code source de B événementiel, l'outil CASTING fournit automatiquement des séquences de test. Ces séquences obéissent à des stratégies définies dans CASTING concernant le taux de couverture et autre. Ces différents éléments sont décrits dans le SP3-LOT5. L'outil B2EXPRESS est ensuite en mesure d'exécuter ces séquences de test.

3.3.1 Démarche de tests de modèles B événementiel

Les tests générés par CASTING, à partir des modèles B événementiel en entrée, sont exprimés sous-forme d'une séquence d'états. Chaque état est décrit par les valeurs courantes des variables d'état. Ces variables sont modifiées d'un état à un autre par l'occurrence d'une événement.

L'adaptation de CASTING développée par AQL permet de générer automatiquement ces séquences de test sous forme d'instances de modèles EXPRESS qui correspondent au modèle B événementiel analysé en entrée de CASTING. Les instances générées sont lues par l'animateur B2EXPRESS pour être exécutées. Il est alors possible de valider les modèles B événementiel a priori.

3.3.2 Application à l'étude de cas "Pages Jaunes"

Nous avons choisi de montrer l'utilisation de cette approche sur l'application "pages jaunes". CASTING adapté à B événementiel a permis de générer automatiquement plusieurs séquences de test correspondant à plusieurs critères. Nous montrons ci-dessous l'ensemble des instances EXPRESS correspondant au modèle EXPRESS généré par l'animateur B2EXPRESS.

```
DATA;
#1=STATE_REF(.F.,.T.,.F.,.F.,.Lionel..F.,.Rennes.,3);
#2=STATE_REF(.F.,.T.,.F.,.F.,.Lionel..F.,.Rennes.,2);
#3=STATE_REF(.F.,.T.,.F.,.T.,.Lionel..F.,.Rennes.,2);
#4=STATE_REF(.F.,.T.,.F.,.F.,.Lionel..F.,.Rennes.,2);
#5=STATE_REF(.F.,.T.,.F.,.T.,.Lionel..F.,.Rennes.,2);
#6=STATE_REF(.F.,.T.,.F.,.F.,.Lionel..F.,.Rennes.,2);
#7=STATE_REF(.F.,.T.,.F.,.F.,.Lionel..T.,.Rennes.,2);
#8=STATE_REF(.F.,.T.,.F.,.F.,.Lionel..F.,.Rennes.,2);
#9=STATE_REF(.F.,.T.,.F.,.F.,.Lionel..T.,.Rennes.,2);
#10=STATE_REF(.F.,.T.,.F.,.F.,.Lionel..F.,.Rennes.,2);
#11=STATE_REF(.F.,.T.,.F.,.F.,.Lionel..F.,.Rennes.,1);
#12=STATE_REF(.T.,.T.,.F.,.F.,.Lionel..F.,.Rennes.,0);
#13=STATE_REF(.F.,.T.,.T.,.F.,.Lionel..F.,.Rennes.,3);
#14=STATE_REF(.F.,.T.,.T.,.F.,.Lionel..F.,.Rennes.,3);
#15=STATE_REF(.F.,.T.,.T.,.F.,.Lionel..F.,.Rennes.,3);
#16=STATE_REF(.F.,.T.,.T.,.F.,.Lionel..F.,.Rennes.,3);
#17=STATE_REF(.F.,.T.,.T.,.F.,.Lionel..F.,.Rennes.,3);
#18=STATE_REF(.F.,.T.,.T.,.F.,.Lionel..F.,.Rennes.,3);
```

```
#19=INIT(#1);
#20=NAME_ADDRESS(#1,#2);
#21=INPUT_NAME_FIELD(#2,#3);
#22=INPUT_NAME_VALUE(#3,#4);
#23=INPUT_NAME_FIELD(#4,#5);
#24=INPUT_NAME_VALUE(#5,#6);
#25=INPUT_ADDRESS_FIELD(#6,#7);
#26=INPUT_ADDRESS_VALUE(#7,#8);
#27=INPUT_ADDRESS_FIELD(#8,#9);
#28=INPUT_ADDRESS_VALUE(#9,#10);
#29=SEARCH(#10,#11);
#30=QUERY(#11,#12);
#31=RESULT_QUERY(#12,#13);
#32=MAP_ZOOM(#13,#14);
#33=MAP_ZOOM(#14,#15);
#34=MAP_MOVE(#15,#16);
#35=MAP_MOVE(#16,#17);
#36=MAP_MOVE(#17,#18);
#37=VAL_TEST((#19,#20,#21,#22,#23,#24,#25,#26,#27,
              #28,#29,#30,#31,#32,#33,#34,#35,#36));
ENDSEC;
```

Dans cet ensemble d'instances correspondant à une séquence de test on observe :

- un ensemble de 18 états identifiés correspondants aux instances #1 à #18. Il s'agit ici d'une séquence de 17 événements déclenchés ;
- l'initialisation est définie en #19 par l'état #1 ;
- les événements menant du premier état au dernier sont décrits par les 17 instances de #21 à #35. Ils correspondent à la trace du test ;
- enfin, le tests est décrit par l'instance #37 qui définit la séquence proprement dite dans son ordre d'exécution.

Notons que pour un utilisateur, le modèle EXPRESS ainsi que l'ensemble des instances obtenues ne sont pas visibles des utilisateurs de ces deux systèmes. Seuls les codes des modèles B événementiel sont visibles par les utilisateurs.

Enfin, plusieurs séquences de test ont ainsi été générées par cette approche.

3.4 Conclusion

Plusieurs résultats ont été obtenus dans ce travail. D'une part, nous avons montré qu'il était possible d'animer des modèles B événementiel décrivant des IHM3. Nous avons montré que plusieurs options d'animation étaient possibles notamment avec l'expression de modèles de tâches sous forme d'expressions d'une algèbre de processus. D'autre part, le travail réalisé en commun avec AQL nous a permis de générer des séquences de tests à partir de CASTING et d'utiliser l'animateur B2EXPRESS pour exécuter ces séquences.

Nous obtenons ainsi une chaîne de validation complète avec preuve et tests en utilisant les modèles de tâches utilisateur.

Enfin, il faut noter que la démarche exposée dans ce document peut être étendue aux autres modèles B événementiel en général en prenant en compte la manipulation de tous les objets décrits en B (fonctions, contraintes, etc.) et que cette démarche est outillée. Elle peut donc être utilisée dans des domaines d'applications autres que les IHM3.

Chapitre 4

Bilan

4.1 Résultats obtenus

Le premier résultat de ce travail est la production d'un outil permettant l'animation de modèles B événementiel. Cet animateur permet de suivre des traces d'exécution de systèmes d'événements décrits en B. Plusieurs possibilités d'animation sont offertes (en ignorant l'invariant, les gardes, les événements etc.). Cette animateur permet aux développeurs utilisant B de mieux comprendre les modèles B en particulier lors de la preuve interactive des différentes obligations de preuve.

Le second résultat est la génération de tests pour les modèles B au travers de l'animateur. Le principe consiste à produire des traces d'événements, sous forme d'instances d'objets EXPRESS, qui peuvent être ensuite lus par l'outil B2EXPRESS pour animer les modèles B avec les tests générés. L'outil casting a été utilisé dans ce but.

Il reste néanmoins a compléter ces outils de génération avec la possibilité de manipuler la totalité de la syntaxe des modèles B. Il faudra entre autre identifier les bonnes représentations en EXPRESS des constructions de B qui ne sont pas encore prises en compte dans B2EXPRESS.

4.2 Conclusion générale du sous-projet 3

Pour ce qui est du sous-projet 3, les résultats obtenus par le projet Verbatim concernent deux volets. Le premier volet est lié au développement pas raffinement et par la preuve avec l'utilisation de la méthode B. Ces résultats se résument de la façon suivante.

- **Mise œuvre de la méthode B avec succès.** Nous avons, au travers d'études de cas, démontré que la méthode B dans sa version événementielle permettait de décrire des modèles formels pour la conception et la validation de systèmes interactifs multi-modaux. Différents scénarios de développement ont été étudiés et une évaluation de la complexité de ces développements a été réalisée en fonction de l'architecture logicielle choisie.
- **Modélisation de l'interaction entrée.** Le processus de fusion de modalités a été traité dans le cadre de cette étude. Nous avons montré comment plusieurs modalités et actions pouvaient être fusionnées pour déclencher une action du noyau fonctionnel. Nous avons décrits ce processus de fusion aussi bien par l'utilisation d'expressions d'une algèbre de processus que par l'utilisation de composants logiciels de la plate-forme ICARE.
- **Vérification de propriétés d'utilisabilité.** Outre les propriétés classiques telles que sûreté, etc. nous nous sommes intéressés aux propriétés d'utilisabilté des IHM3 au travers de la modélisation des propriétés CARE. A ce niveau également, nous avons proposé différentes approches, allant des expressions de tâches jusqu'aux composants ICARE, pour exprimer puis vérifier ces propriétés.
- **Modélisation de tâches utilisateurs en conception et/ou en validation.** L'expression de tâches est un aspect important dans le développement d'IHM. Elles permettent aux différents concepteurs (qu'ils soient ergonomes, psychologues ou informaticiens) de décrire les besoins des utilisateurs (conception centrée utilisateur). Nous avons défini différentes approches de validation des tâches utilisateurs en utilisant la mthode B.
- **Modélisation à base composants ICARE et démarche méthodologique.** La modélisation de tâches et le raffinement se font de façon descendante et compromettent la réutilisation de codes ou de modèles déjà existants. Pour répondre en partie à ce problème, nous avons modélisé les composants génériques d'une plate-forme multi-modale, à savoir ICARE en B. Tous les raffinements définis s'arrêtent lorsque les modèles faisant appel aux composants ICARE sont atteints. Cette approche permet également de garantir les propriétés CARE par construction.
- **Animation de modèles B.** Nous avons défini une approche outillée permettant d'animer les modèles B. Elle vient compléter la preuve par la possibilité d'exécuter ou de déclencher les événements dans une trace. Nous avons également associé une expression de tâches aux traces d'événements afin de suivre

le déroulement d'une tâche exprimée sur les événements du modèle B en cours d'animation.

- **Génération de tests.** L'animateur B2EXPRESS a été interfacé avec l'outil de génération de test CASTING en collaboration avec AQL. Nous avons montré que la génération de séquences de test qui ensuite étaient exécutées sur les modèles B événementiel correspondants était possible.
- **Démarche d'intégration.** Enfin, du point de vue méthodologique, notre approche a contribué à intégrer modèles et notations en IHM existants et largement utilisés en conception. C'est ainsi que nous avons exploité les modèles de composants, modèles de tâches et modèles d'architecture. Nous nous sommes efforcés à ne pas proposer de nouvelle notation mais plutôt a intégrer celles qui sont éprouvées dans un processus de développement rigoureux. Bien sûr, nous avons du faire des choix d'interprétation sémantique partout où des ambiguïtés nous sont apparues.

Le second volet concerne la démarche d'intégration des méthodes formelles dans le développement d'IHM Multi-modales. Ces résultats sont décrits dans le sous-projet 6.

Bibliographie

[Aer98] L. Van Aertryck. *Une méthode et un outil pour l'aide à la génération de jeux de tests de logiciels.* PhD thesis, Thèse de l'Université de Rennes I, Janvier 1998.

[AS05] I. Ait-Sadoune. *Vérification et validation formelle d'IHM Multimodales fondées sur la preuve. Utilisation de la Méthode B.* Mémoire d'ingénieur d'état en informatique, INI, Alger, juin 2005.

[IS094] IS010303.02. Product data representation and exchange - part 2 : Express reference manual. *ISO*, (055), 1994.

[SM97] G. Staub and M. Maier. Ecco tool kit - an environnement for the evaluation of express models and the development of step based it applications. *User Manual*, 1997.

[SW94] D. Schenck and P. Wilson. *Information Modelling The EXPRESS Way.* Oxford University Press, 1994.