Computation of Hierarchical Transition Systems to Document Refined Event-B Models

Nicolas Stouls

▶ **To cite this version:**

Nicolas Stouls. Computation of Hierarchical Transition Systems to Document Refined Event-B Models. [Technical Report] 2008, pp.10. inria-00271164

HAL Id: inria-00271164
https://inria.hal.science/inria-00271164

Submitted on 8 Apr 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Computation of Hierarchical Transition Systems to Document Refined Event-B Models

Nicolas Stouls
INRIA Futurs, ProVal, Parc Club d'Orsay, F-91893
Université Paris-Sud, CNRS, Laboratoire LRI, UMR 8623, Orsay, F-91405
INP Grenoble, CNRS, Laboratoire LIG, UMR 5217, Saint Martin d'Hères, F-38402
Email: Nicolas.Stouls@lri.fr

***Abstract*—To document a formal model and to explain it to non specialists in formal methods , the usual approach is to use another formalism which does not require any expert skill and which gives a second viewpoint of the same model. According to this approach, we proposed in a previous paper to build a symbolic labelled transition system (SLTS) to describe behaviours of a B model. Extending this work, we now introduce hierarchies in SLTS in order to take into account the refinement process and to exhibit links between description levels. In this paper, we propose a formal definition of hierarchical-SLTS (HSLTS) and we extend the SLTS generation method in order to build a HSLTS which represents behaviours of an event-B refined model.**

***Index Terms*—Formal Methods, Symbolic Labelled Transition Systems, Refinement, Hierarchy, Computation**

I. Introduction

Formal methods, such as the B method [2], are used in developments demanding high security requirements. For instance, such a formalism can be used to describe the security policy of systems that will be certified. However, formal methods description languages are usually hard to understand by non-specialists. Since a specifier may have to describe its model to non-specialists in formal method, who can be specialists from other domain, a usual method is to rewrite the model in a different language in order to have a more understandable viewpoint on the same system.

A first approach [11] consists in computing a UML [18] class diagram from the B model. This technique gives a representation of the model architecture, but no functional information.

Another approach is to exhibit the behaviours of the model, that give functional information to non-specialists. Such a representation can also be used by the specifier, as it gives a complementary representation of the system, since the B language is data-based (description in terms of data modifications and not in terms of system behaviours). However, the main motivations of existing work [4], [7], [10] is oriented towards the verification of properties, e.g. by model-checking, while our objective is to provide a representation which aids in understanding a formal specification. Particularly, the refinement process is an important concept of the B method and that is not taken into account in cited works.

In a previous paper [5], we introduced a symbolic labelled transition systems formalism (SLTS), which precisely describes behaviours of an event-B model, as well as a method to compute an SLTS associated to an event-B model. An originality in this proposition is the split of any transition cross-condition in two parts: enabledness and reachability. As represented in Fig. 1, the enabledness condition (written D) characterises the set of elements from the starting state q_1 from which the event ev can be launched, while the reachability condition (written A_i) characterises, among the elements of q_1 which allow ev to be launched, these from which ev can reach q_2 and those from which ev can reach q_3.

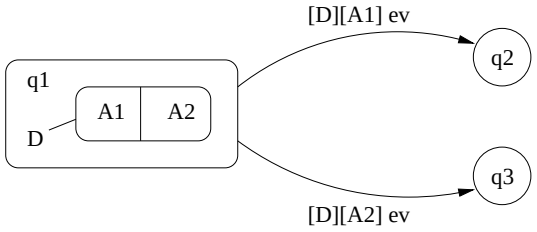

Fig. 1. Meaning of Split Cross-Conditions

Now, we propose to take into account additionally the refinement process and to visualise each design step (each refinement level). In B, a refinement component is a specification that is more concrete than its abstraction. We then propose to represent the refinement behaviours in terms of the abstract ones by introducing some hierarchy in the representation, such that each depth level is associated to a refinement level. Thus, the global structure of an abstract representation is preserved in order to describe the refinement behaviours. The link between abstract and refined data are underlined by state inclusions. Our computation method is based on the modification of an SLTS associated to the abstract specification. Super-states are those states that originate from the abstract representation, while substates are introduced to precise some refined behaviours. Finally, this approach allows to re-use, in the refinement, the understanding efforts performed on the abstract case.

To illustrate this paper, we use the following running example, which models a *very* abstract communication channel. Figure 2 is the abstract description level where a message is represented by its size ($MessageSize$) and Figure 4 is a refined description where a buffer has been introduced in the channel. In this system, a message is sent by the $Send$ event and received by the $Treat$ event. Each occurrence of the

$Treat$ event treats only one element of the message. The full treatment then needs $MessageSize$ occurrences of the $Treat$ event. Finally, the treatment can be cancelled at any time with the $Reset$ event. The B syntax will be described in the next section. Here, we just propose to give an intuition of our approach which consists in computing behaviours of such a model.

SYSTEM *Communication_Channel*
VARIABLES $MessageSize$
INVARIANT $MessageSize \in \mathbb{N}$
INITIALISATION $MessageSize := 0$
EVENTS
$Send \;\hat{=}$ **SELECT** $MessageSize = 0$ **THEN** $MessageSize :\in \mathbb{N}_1$ **END**;
$Treat \;\hat{=}$ **SELECT** $MessageSize > 0$ **THEN** $MessageSize := MessageSize - 1$ **END**;
$Reset \;\hat{=}$ **SELECT** $MessageSize > 0$ **THEN** $MessageSize := 0$ **END**
END

Note that $\mathbb{N}$ *is the unbounded set of naturals, that* $\mathbb{N}_1 \;\hat{=}\; \mathbb{N} - \{0\}$. *and that* $MessageSize :\in \mathbb{N}_1$ *assigns to* $MessageSize$ *any value in* $\mathbb{N}_1$.

Fig. 2. Abstract Description of a Communication Channel

Figure 3 represents the communication channel behaviours. It is a SLTS produced by the GénéSyst tool, which implements our method. On the diagram, we note $[D]\ [A]\ ev$ the label of a transition that can be crossed by the event ev under the enabledness condition D and the reachability condition A. A condition equivalent to true is denoted $[\ ]$. For instance, the split of the cross-condition of event $Treat$ exhibits that the event is always enabled in the state $MessageSize > 0$, while the state reached depends on the system valuation. Thus, $MessageSize$ is set to 0 by $Treat$ only if $MessageSize = 1$ in the previous state.

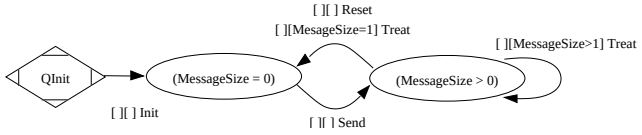

Fig. 3. Example of SLTS Associated to Communication Channel.

In the refined description, a buffer is introduced and a message can be sent block by block by the event $SendNext$, which is introduced through the refinement process. $Send$ then just initialises the emission. This means that sending and treatment can be interlaced, while $Reset$ can still occur at any time. Finally, the buffer size ($BufferSize$) is limited, but not defined (just constrained to be at least 1). The link between the abstract data $MessageSize$ and the refined data $BufferSize$ and $ToSend$ is given in the invariant clause.

Figure 5 is a Hierarchical SLTS (HSLTS) associated to the refined communication channel. We can read on it that the abstract set of valuations $MessageSize > 0$ is equivalent, in the refinement, to the union of $ToSend > 0$ and $ToSend = 0 \ \wedge\ InBuffer > 0$. Moreover, the use of hierarchy allows to structure the diagram by choosing initial-substates and final-substates and by merging transitions. For instance, $Reset$ is always enabled from each substate of

REFINEMENT *Communication_Channel_With_Buffer*
REFINES *Communication_Channel*
CONSTANTS $BufferSize$
PROPERTIES $BufferSize \in \mathbb{N}_1$
VARIABLES $InBuffer$, $ToSend$
INVARIANT $ToSend \in \mathbb{N} \ \wedge\ InBuffer \in 0..BufferSize \ \wedge\ InBuffer + ToSend = MessageSize$
INITIALISATION $InBuffer := 0 \ ||\ ToSend := 0$
OPERATIONS
$Send \;\hat{=}$ **SELECT** $ToSend = 0 \ \wedge\ InBuffer = 0$ **THEN** $ToSend :\in \mathbb{N}_1$ **END**;
$SendNext \;\hat{=}$ **SELECT** $ToSend > 0 \ \wedge\ InBuffer < BufferSize$ **THEN** $ToSend := ToSend - 1 \ ||\ InBuffer := InBuffer + 1$ **END**;
$Treat \;\hat{=}$ **SELECT** $InBuffer > 0$ **THEN** $InBuffer := InBuffer - 1$ **END**;
$Reset \;\hat{=}$ **SELECT** $ToSend{>}0 \vee InBuffer{>}0$ **THEN** $ToSend{:=}0 \ ||\ InBuffer{:=}0$ **END**
END

Fig. 4. Refinement of the Communication Channel

$MessageSize > 0$. All associated transitions can thus be merged in a single factorised transition.

Concretely, the contributions of this paper are a definition of a SLTS extension including hierarchy and method to build a HSLTS from a refined event-B model, by extending the one proposed in the case of an abstract specification.

In Sect. II and III a first part, we briefly introduce the event-B method and we recall the main results of [5] about SLTS representation and computation from abstract event-B systems. In Sect. IV, we introduce our Hierarchical-SLTS formalism. After that, we present in Sect. V, our methodology to compute a HSLTS from a refined event-B model and we finish, in Sect. VII and VI, by describing some implementation work and experiments.

II. Event-B

B [2] is a formal development method as well as a specification language, where components can be built through a refinement process. The consistency and correctness of models can be validated by proof obligations. Event-B [1] is an extension of B in which models are described by events instead of operations. Each event is composed of a guard G and an action S such that if G is enabled, then S can be executed. If several guards are enabled at the same time then the triggered event is chosen in a nondeterministic way. Events and initialisation are described with generalised substitutions, which act as instructions of programming languages (Table I).

Substitution name	Notation
Do nothing	skip
Assignment	$x := e$
Guard	**SELECT** P **THEN** S **END**
Unbounded choice	**ANY** x **WHERE** P **THEN** S **END**
Become element of	$x :\in E$

where x a variable, e an expression, P a predicate, S a substitution and E a set

TABLE I
Generalised Substitutions Used in this Paper

The generalised substitutions semantics is defined by the

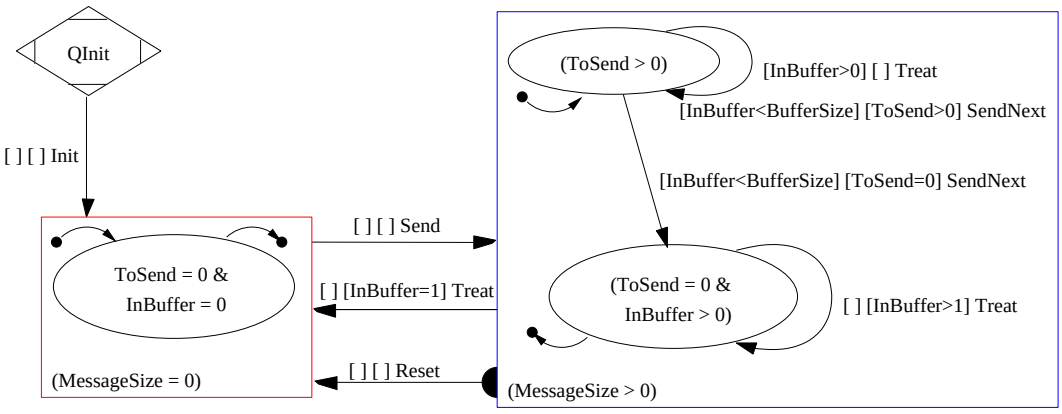

Fig. 5. HSLTS Associated to the Communication Channel Refinement.

weakest precondition calculus ($\mathcal{WP}$). We note $[S]R$ the weakest precondition such that the execution of the substitution S establishes the predicate R. $\mathcal{WP}$ definition is given in [2].

The logical guard of an event ev, in contrast with its syntactical guard, is defined in [1] as its feasibility condition, i.e. the set of all valuations for which ev can be executed and establish another valuation.

Definition 1 (Feasibility): *The feasibility of a generalised substitution S, written $\mathsf{fis}(S)$, is defined by:*

$$\mathsf{fis}(S) \mathrel{\hat{=}} \neg[S]\neg\mathsf{true}$$

Intuitively, $\neg\mathsf{true}$ is the absence of solution and $[S]\neg\mathsf{true}$ is the set of valuation that have no solution after execution of S. Then $\mathsf{fis}(S)$ is the set of valuations for which there exists at least one solution after execution of S.

Each event $ev \mathrel{\hat{=}} S$ can then be written in the following normalised form, where the syntactical guard is also the logical one:

$$ev \mathrel{\hat{=}} \textbf{SELECT } \mathsf{fis}(ev) \textbf{ THEN } S \textbf{ END}$$

We then write $\mathsf{Guard}(ev)$ for its logical guard ($\mathsf{Guard}(ev) \mathrel{\hat{=}} \mathsf{fis}(S)$) and $\mathsf{Action}(ev)$ for its action ($\mathsf{Action}(ev) \mathrel{\hat{=}} S$). In order to simplify the presentation, we consider in this paper that the syntactical guard is also the logical one.

We defined in [5] the semantics of event-B systems in terms of the events sequence that can be observed, as follows:

Definition 2 (Trace of an Event-B System): *A finite sequence of event occurrences $Init; e_1; \ldots; e_n$ is a trace of a system $\mathcal{A}$ if and only if $Init$ is the initialisation of $\mathcal{A}$, $\{e_1, \ldots, e_n\}$ are events from $\mathcal{A}$ and $\mathsf{fis}(Init; e_1; \ldots; e_n) \Leftrightarrow \mathsf{true}$.*

In the B method, the refinement process is done by parts. The initialisation is refined by a more concrete initialisation and each event is refined by its concrete definition. If an event ev_A is refined by an event ev_R, then the refined guard has to imply the abstract one. Moreover, some events can be introduced during the refinement process (by refining the skip event), according to the same principles as *stuttering* in TLA [12]: all traces of a refinement R of S are included, modulo some new events, into the traces of S. Data representation can also change. A predicate L (called *gluing invariant*) describes the relationship between abstract and concrete variables. L is the conjunction of the invariant I from the specification and the invariant J from the refinement. For instance, $InBuffer + ToSend = MessageSize$ is the gluing invariant of Fig. 4 and links the abstract variable $MessageSize$ to the refined ones $InBuffer$ and $ToSend$.

III. Behaviours of Abstract Event-B Systems

In this section, we briefly cover the approach introduced in [5], that we extend in this paper. We first present the defined transition system and then introduce the generation method.

A. Transition Systems for Abstract Event-B Systems

States are defined by first-order logic predicates and each transition is labelled by an event name ev. A transition represents a state change that can be done by an execution of its associated event. We also specify in each transition label two conditions under which the transition can be crossed: *enabledness* and *reachability*. The use of two conditions instead of a single one increases the understanding that we have of the system behaviour, thanks to more precise information about the origin of conditions. Indeed, an enabledness condition is due to the guard of the event, while a reachability condition is associated to data modification. These two conditions are called *cross-conditions*.

The SLTS are then defined by:

Definition 3 (Symbolic Labelled Transition Systems): *A SLTS $(\mathbb{V}, \mathbb{E}, \mathbb{Q}, \mathsf{q}_{Init}, \mathcal{D}ef, \mathbb{L}, \mathbb{R})$ is defined by:*

- *$\mathbb{V}$: a set of variables;*
- *$\mathbb{E}$: a set of event names;*
- *$\mathbb{Q}$: a set of state names;*
- *q_{Init}: an initial state ($\mathsf{q}_{Init} \in \mathbb{Q}$);*
- *$\mathcal{D}ef$: a function which associates a definition predicate to each state name ($\mathcal{D}ef \in \mathbb{Q} \rightarrow \mathbb{P}$);*
- *$\mathbb{L}$: a set of labels ($\mathbb{L} \subseteq \mathbb{P} \times \mathbb{P} \times \mathbb{E}$);*
- *$\mathbb{R}$: a transition relation ($\mathbb{R} \subseteq \mathbb{Q} \times \mathbb{L} \times \mathbb{Q}$).*

Where $\mathbb{P}$ is the set of all predicates on variables $\mathbb{V}$ and $\mathcal{D}ef(\mathsf{q}_{Init}) = \mathsf{true}$.

Finally, the semantics of a SLTS is given by its set of paths. A path is an events sequence starting from the initial state and going over the SLTS through legal transitions crossings. Let $t = (q_1, (D, A, ev), q_2)$ be a transition from q_1 to q_2 by ev, under the enabledness condition D and the reachability condition A. t is said to be *legal* if there exists two system valuations x_1 and x_2 establishing respectively $\mathcal{D}ef(q_1)$ and $\mathcal{D}ef(q_2)$, and if ev can reach x_2 from x_1.

B. Generation of a SLTS from an Abstract B-System

The state space is given by the user. It allows him/her to manage the generated view in order to focus on a particular property. Hence, to compute a SLTS from a B-system it is sufficient to determine the cross-conditions D and A of each transition $(q_1, (D, A, ev), q_2)$. If one of the conditions is false then the transition can not be pass through. The proposed method then consists in determining these predicates by proof obligations, by examining three cases: true *(always enabled or reachable)*, false *(never enabled or reachable)* or conditioned. If no proof obligation can be established for condition, then we speak about *lack of proof*. This case exist because of the first order logic non-determinism. For instance, if $\forall x \cdot (\mathcal{D}ef(q_1) \Rightarrow \mathsf{Guard}(ev))$ is established then ev is always enabled from q_1, while if $\forall x \cdot (\mathcal{D}ef(q_1) \Rightarrow \neg\mathsf{Guard}(ev))$ is established then ev is never enabled from q_1.

IV. Proposition of Transition Systems for Refinements

In this section, we introduce the first proposal of this paper: a formal definition of hierarchical-SLTS (called HSLTS). This formalism will be used to represent behaviours of a refined event-B model. This section ends with a discussion about related works.

A. Hierarchical-SLTS Definition

We propose to represent data from each refinement level of a B model as a hierarchical level of states in a HSLTS diagram. Hence, states inclusion represents the relationship between abstract and refined data. Since a valuation can be potentially reached from any other valuation, we have to allow transitions between all states, including states from different clusters[1]. In our approach, we define a hierarchical-SLTS as a single SLTS including a state hierarchy function $\mathcal{S}up$, which associates, to each state, its super state. There is then a unique transition relation in a HSLTS, allowing transitions to connect any couple of states, including states from different clusters. Since a state can not be its own substate, the $\mathcal{S}up$ function, which associates each substate to its superstate, has to be cycle-free. Moreover, each hierarchical state can include an initial and a final substate. These states are respectively characterised by the $\mathbb{Q}_{Init}$ and $\mathbb{Q}_{Final}$ sets. In order to define the HSLTS as an extension of the SLTS, the roots (states that are not substates) can not be in these sets and the initial state system is q_{Init}.

Definition 4 (Hierarchical SLTS): *A hierarchical SLTS* $(\mathbb{V}, \mathbb{E}, \mathbb{Q}, \mathsf{q}_{Init}, \mathbb{Q}_{Init}, \mathbb{Q}_{Final}, \mathcal{D}ef, \mathbb{L}, \mathbb{R}, \mathcal{S}up)$ *is defined by:*

- $(\mathbb{V},\ \mathbb{E},\ \mathbb{Q},\ \mathsf{q}_{Init},\ \mathcal{D}ef,\ \mathbb{L},\ \mathbb{R})$ *is a SLTS, such that* $\mathbb{Q}$ *is the set of* ***all*** *state names,* ***including all levels of hierarchy****;*
- $\mathcal{S}up$ *is a cycle-free function, associating each substate to its superstate* $(\mathcal{S}up \in \mathbb{Q} \mapsto \mathbb{Q} \ \wedge\ \mathsf{id}(\mathbb{Q}) \cap \mathcal{S}up^+ = \emptyset)$*;*
- $\mathbb{Q}_{Init}$ *is the set of initial substates* $(\mathbb{Q}_{Init} \subseteq \mathsf{dom}(\mathcal{S}up))$*;*
- $\mathbb{Q}_{Final}$ *is the set of final substates* $(\mathbb{Q}_{Final} \subseteq \mathsf{dom}(\mathcal{S}up))$.

[1]Also called *hierarchical state*, a *cluster* is a state including substates.

A SLTS is then a HSLTS without hierarchy ($\mathcal{S}up = \emptyset \ \wedge\ \mathbb{Q}_{Final} = \emptyset \ \wedge\ \mathbb{Q}_{Init} = \emptyset$). Notice than states without superstate are called roots and that states without substate are called leaves. Hence, we define the following predicates:

$$\mathsf{Leaves}(\mathcal{S}up) \mathrel{\hat{=}} \mathsf{dom}(\mathcal{S}up) - \mathsf{ran}(\mathcal{S}up)$$
$$\mathsf{Roots}(\mathcal{S}up) \mathrel{\hat{=}} \mathsf{ran}(\mathcal{S}up) - \mathsf{dom}(\mathcal{S}up)$$

Note than each B refinement level has its own variables set, while a HSLTS is defined on a single variables set. This point will be discussed in Sect. V-A.

Finally, in order to keep a coherence between states-hierarchy and states definition, we impose that:

Property 1 (Hierarchical Coherence): *For each cluster* q*, such as* $\mathcal{S}up^{-1}[\{q\}] = \{q_1, \ldots, q_n\}$*:* $\mathcal{D}ef(q) \Leftrightarrow \bigvee_{i=0}^{n} \mathcal{D}ef(q_i)$ *where* $\mathcal{S}up^{-1}[\{q\}]$ *is the set of substates of* q*.*

B. HSLTS semantics

Transition boundaries can be partitioned in three types (Fig. 6): factorised (in place of several transitions), initial or final (in relation with an initial or a final substate) and single (on leaves). We define each of them, and then the semantics of a HSLTS, in terms of flattening in a SLTS. Since these diagrams have to aid in understanding, we consider a syntactical flattening, that can be easily done by a user. Hence, we consider there is loss of information if the syntactical flattening includes some cross conditions that can be reduced to true or false, but that are nevertheless conditioned.

1) Factorised Transitions: A transition boundary is factorised if the associated state is not a leaf. Such a transition is equivalent to the same transition copied out on each substate (with the same conditions). Moreover, each copy has to be legal (crossable). This restriction keeps cross-conditions through syntactical flattening.

2) Initial and Final Transitions: If an external[2] transition starts on a leaf which is also a final substate, then the transition can be represented as starting from the superstate. We then distinguish the definition of such a transition which starts from the leaf and the representation, which starts from the superstate. In the same manner, if an external transition ends on a leaf which is also an initial substate, then the transition can be represented as ending to the superstate.

3) Single Transitions: A transition boundary is single if the associated state is a leaf. In the particular case of an external transitions, the substate also have to be neither initial nor final. The semantics is then the one of the SLTS transitions.

4) Semantics of a HSLTS: Hence, we define the semantics of a HSLTS by the semantics of the SLTS in which it is flattened:

Definition 5 (HSLTS Semantics): The HSLTS $(\mathbb{V},\ \mathbb{E},\ \mathbb{Q}_H,\ \mathsf{q}_{Init},\ \mathbb{Q}_{Init},\ \mathbb{Q}_{Final},\ \mathcal{D}ef_H,\ \mathbb{L}_H,\ \mathbb{R}_H,\ \mathcal{S}up)$ semantics is defined, through flattening, by the SLTS

$$(\mathbb{V},\ \mathbb{E},\ \mathbb{Q}_S,\ \mathsf{q}_{Init},\ \mathcal{D}ef_S,\ \mathbb{L}_S,\ \mathbb{R}_S)$$

such as:

$\mathbb{Q}_S \mathrel{\hat{=}} \mathsf{Leaves}(\mathcal{S}up)$ */* Keep only leaves */*
$\mathcal{D}ef_S \mathrel{\hat{=}} \mathbb{Q}_S \lhd \mathcal{D}ef_H$ */* Keep only definition of leaves */*

[2]Transition crossing a state boundary (between two states from different clusters).

/ Transition relation flattening */*

$$\mathbb{R}_S \triangleq \left\{ (q,(D,A,ev),r) \;\middle|\; \exists (q',ev,r') \cdot \begin{pmatrix} (q',(D,A,ev),r') \in \mathbb{R}_H \\ \wedge\ q \in (\mathcal{S}up^{-1})^*[\{q'\}] \\ \wedge\ r \in (\mathcal{S}up^{-1})^*[\{r'\}] \\ \wedge\ q \in \mathbb{Q}_S \ \wedge\ r \in \mathbb{Q}_S \end{pmatrix} \right\}$$

$\mathbb{L}_S \triangleq \{(D,A,ev) \mid \exists (q,r) \cdot (q,(D,A,ev),r) \in \mathbb{R}_S\}$ */* Keep labels */*

Notation:

(Domain restriction) $A \lhd R \triangleq \{(x,y) \mid (x,y) \in R \ \wedge\ x \in A\}$

(Kleene closure) $R^* \triangleq \mathsf{id}(A) \cup R \cup (R;R) \cup (R;R;R) \cup \ldots$, where $R \in A \leftrightarrow A$

5) Graphical Representation of Transitions: Figure 6 (next page) gives the graphical representation of each transition type.

C. Related works

The proposed semantics is not the most usual approach to define hierarchical transition systems. Indeed, such systems are usually [14], [22] described by a set of SLTS and a composition function, such that a transition can only be defined between states from the same cluster. For instance, in the proposed semantics, the hierarchical transition system Fig. 7 is defined by the SLTS Fig. 8, where the hierarchy is used to structure the representation. In the usual approach, Fig. 9, each SLTS is independently defined and the composition function associates each SLTS to its parent state.

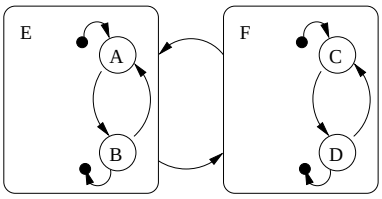

Fig. 7. Example of Hierarchical Transition System

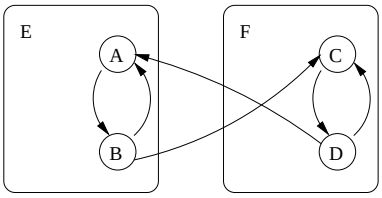

Fig. 8. Proposed Semantics of Fig. 7

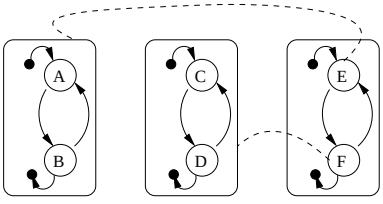

Fig. 9. Usual Semantics of Fig. 7

We argue than the proposed approach is more expressive than the usual one. Indeed, in the usual definition a transition cannot be defined between states from different clusters, because each transition relation is defined in terms of its local states. Thus, if only one initial state is allowed per cluster, then this state is the only one that can be reach from external transitions. Because of this restriction, behaviours of all B models cannot be represented in usual cases. This is why we choose this original definition. However, the main interest of the usual case is to simplify the partwise verification of properties on the model (by model checking for instance). Hence, the proposed approach is not enhanced for verification, but does not impose restriction on the choice of states space to represent the model behaviours.

In the following section, we describe how to generates a HSLTS, by exploiting the B refinement properties.

V. HSLTS Generation from Refined B Model

Our objective is to generate an HSLTS which represents the behaviours of a refined event-B model, such that each depth level of the hierarchy is associated to data from a refinement level.

In order to lighten the presentation, we consider in this section that $T_{\mathcal{A}}$ is a SLTS associated to an event-B model $\mathcal{A}$, that $\mathcal{R}$ is a refinement of $\mathcal{A}$ according to the gluing invariant L and that $T_{\mathcal{R}}$ is a HSLTS associated to $\mathcal{R}$.

We propose to compute $T_{\mathcal{R}}$ as a modification of $T_{\mathcal{A}}$ in three steps, detailed respectively in the three following sections:

- **State Space Definition** by projecting the definition predicates of T_S states on R variables and by adding a new hierarchy level;
- **Computation of Transition Relation between Leaves** with the method introduced for SLTS and taking into account the refinement properties;
- **Reduction of External Transition Amount** by factorisation and choosing initial and final substates.

A. State Space Definition

A HSLTS is defined by a single set of variables $\mathbb{V}$, while each description level of a B model includes its own variables set. Since a B refined model is characterised by its last refinement level, we define $\mathbb{V}$ with the set of variables from this last level. We then need to characterise each state of T_S by a predicate on $\mathbb{V}$:

Definition 6 (State Projection): *Let q be a state name defined, on variables x of $\mathcal{A}$, by the predicate $\mathcal{D}ef_{\mathcal{A}}(q)$ and let L be a gluing invariant between variables x and y. The projection $\mathsf{Proj}_L(q)$ of q according to L is defined by:*

$$\mathsf{Proj}_L(q) \triangleq \{y \mid \exists x \cdot (L \ \wedge\ \mathcal{D}ef_{\mathcal{A}}(q))\}$$

Like in a SLTS computation, the user has to provide the state space describing the new hierarchy level (definition predicates $\mathcal{D}ef_U$, state names $\mathbb{Q}_U$ and composition function $\mathcal{S}up_U$). In the present work, we propose to use, as state name, the textual representation of the predicate used to describe the state initially. For instance, on Fig. 5 clusters are labelled with predicates in terms of abstract variables ($MessageSize$), while the substates are labelled with predicates in terms of refined variables ($BufferSize$ and $ToSend$). In this case, abstract states are those of SLTS Fig. 3, while the leaves have to be provided by the user, in addition to the hierarchy function.

If $T_{\mathcal{A}}$ is the HSLTS associated to the abstract system $\mathcal{A}$, then the state space of $T_{\mathcal{R}}$ associated to the refinement $\mathcal{R}$ of $\mathcal{A}$ according to L is computed as follows:

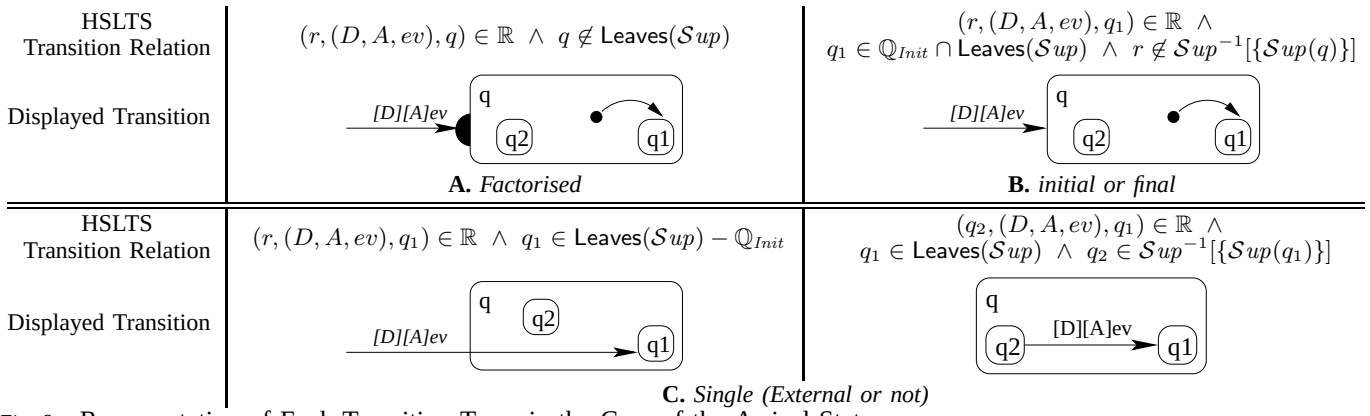

Fig. 6. Representation of Each Transition Type, in the Case of the Arrival State

$\mathbb{Q}_{\mathcal{R}} := \mathbb{Q}_{\mathcal{A}} \cup \mathbb{Q}_U \ \|\|$	*/* Keeping state names */*
$\mathcal{D}ef_{\mathcal{R}} := \{(\mathsf{q}_{Init} \mapsto \mathsf{true})\}$	*/* Keeping q_{Init} */*
$\cup \{E \mapsto \mathsf{Proj}_L(E) \mid E \in \mathbb{Q}_{\mathcal{A}} - \{\mathsf{q}_{Init}\}\}$	*/* Abstract states */*
$\cup \mathcal{D}ef_U \ \|\|$	*/* New leaves */*
$\mathcal{S}up_{\mathcal{R}} := \mathcal{S}up_{\mathcal{A}} \cup \mathcal{S}up_U$	*/* Composition computation */*

B. Computation of Transition Relation Between Leaves

The computation of the transition relation is split into two steps: in the first one we propose to compute the relation between leaves and in the second one we search for initial and final substates, as well as for factorised transitions. In this part, we propose to build a transition relation between leaves (Fig. 10). By taking into account refinement properties, we minimise the number of generated proof obligations and thus the number of lack of proof. In the next section we propose a post-processing in order to factorise transitions and choose initial or final substates.

To compute transitions between leaves, it is sufficient to compute the cross conditions D and A for each pair of leaves (q_1, q_2) and each event ev, by using the method proposed in Sect. III. However, this naive approach can be enhanced by taking into account event-B refinement properties. Indeed, if no transition goes from q_1 to q_2 by ev in $T_{\mathcal{A}}$ then no transition goes from a substate of q_1 to a substate of q_2 by ev in $T_{\mathcal{R}}$.

Property 2 (Cross Conditions Simplification): *Let $(q_1,(D_{\mathcal{R}},A_{\mathcal{R}},ev),q_2)$ be a transition from $T_{\mathcal{R}}$ and let $(\mathcal{S}up(q_1),(D_{\mathcal{A}},A_{\mathcal{A}},ev),\mathcal{S}up(q_2))$ be a transition from $T_{\mathcal{A}}$. If $\mathcal{R}$ refines $\mathcal{A}$ according to L then:*

$$L \ \wedge \ \mathcal{D}ef_{\mathcal{A}}(\mathcal{S}up(q_1)) \ \wedge \ D_{\mathcal{R}} \Rightarrow D_{\mathcal{A}} \tag{2.1}$$

$$L \ \wedge \ \mathcal{D}ef_{\mathcal{A}}(\mathcal{S}up(q_1)) \ \wedge \ D_{\mathcal{R}} \ \wedge \ A_{\mathcal{R}} \Rightarrow A_{\mathcal{A}} \tag{2.2}$$

Property (2.1) can be easily established using by the following hypothesis, which is given by the B refinement:

$$L \ \wedge \ \mathsf{Guard}_{\mathcal{R}}(ev) \Rightarrow \mathsf{Guard}_{\mathcal{A}}(ev)$$

However, Property (2.2) is slightly more difficult to establish. The interested reader can find this proof in the Appendix.

Finally, if new events are introduced through refinement, the associated transitions cannot be simplified with the previous property. Nevertheless, since a new event ev_n refines skip, we can establish that:

Property 3 (New Events Reflexivity): *Let q_1 and q_2 two states from $T_{\mathcal{R}}$. If their superstates are disjointed, then no event ev_n introduced through refinement can pass through a transition from q_1 to q_2.*

Hence, if two clusters q_1 and q_2 are disjointed[3], then no new event can connect a q_1 substate to a q_2 substate. Beyond the proof obligations shrinkage, this proof obligation is interesting because it depends neither on the event definition nor on the substates predicates. Hence, in several cases, the automatic proof can be easier.

C. Reduction of External Transition Amount

At this point, we know how to generate, from a refined event-B model, an HSLTS with transitions stated between leaves. Now we propose to reduce the amount of external[4] transitions in order to ease the system understanding. We then introduce a method to factorise transitions and some heuristics to choose initial and final substates.

1) Transitions Factorisation: According to our definition of factorised transitions (Sect. IV-B4), a transition factorised on a cluster q is equivalent to the same transition copied, with its cross conditions, on each substate of q, such that all these transitions are legal. Some transitions can then be factorised only if they establish the following condition:

Condition 1 (Factorisation Cases):

Factorisation on Starting State *If a legal transition $(q_i,(D_i,A_i,ev),r)$ exists for each substate q_i of q, then we can replace these transitions by $(q,(D',A',ev),r)$ if there exists D' and A' such as, for each q_i:*

$$\begin{array}{c} \mathcal{D}ef(q_i) \Rightarrow (D' \Leftrightarrow D_i) \\ \mathcal{D}ef(q_i) \ \wedge \ D_i \Rightarrow (A' \Leftrightarrow A_i) \end{array} \tag{1.1}$$

Factorisation on Ending State *If a legal transition $(q,(D_i,A_i,ev),r_i)$ exists for each substate r_i of r, then we can replace these transitions by $(q,(D,A',ev),r)$ if there exists A' such as, for each r_i:*

$$\mathcal{D}ef(q) \ \wedge \ D \Rightarrow (A' \Leftrightarrow A_i) \tag{1.2}$$

Note that, even if the r state or its substates r_i are not explicitly present in Condition (1.2), they are included in the A_i predicate [5, Cond. 1].

To factorise transitions on the superstate of their originating state (1.1), D' and A' can be syntactically computed as follows:

[3]With the single proof obligation: $\mathcal{D}ef_{\mathcal{R}}(\mathcal{S}up(q_1)) \Rightarrow \neg\mathcal{D}ef_{\mathcal{R}}(\mathcal{S}up(q_2))$

[4]Transition crossing a cluster boundary (between states from two clusters).

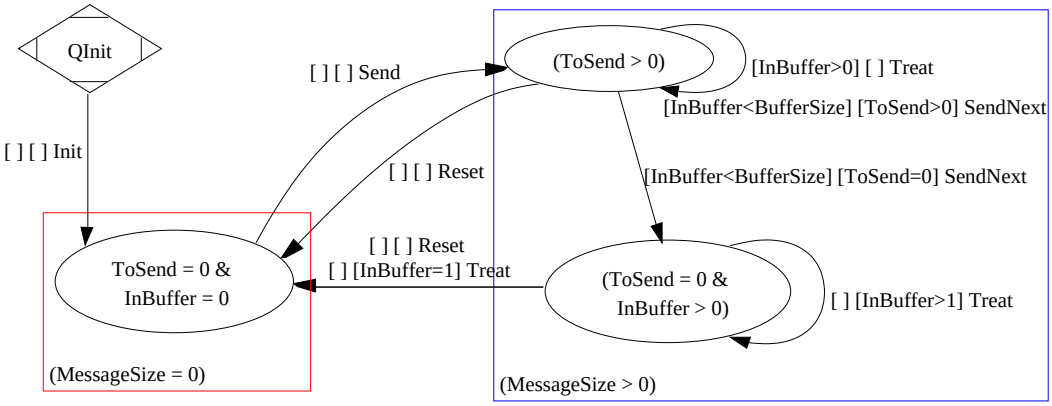

Fig. 10. HSLTS with Transition Relation between Leaves

- $D' \mathrel{\hat{=}} \bigwedge_i (\mathcal{D}ef(q_i) \Rightarrow D_i)$
- $A' \mathrel{\hat{=}} \bigwedge_i (\mathcal{D}ef(q_i) \ \wedge \ D_i \Rightarrow A_i)$

To factorise transitions on the superstate of their ending state (1.2), A' has to be equivalent to each A_i under the same hypothesis. Hence, transitions factorisation on ending states needs to establish, for each pair of substates (q_i, q_j), that: $\mathcal{D}ef(q) \ \wedge \ D \Rightarrow (A_i \Leftrightarrow A_j)$. A' can then be any of the A_i conditions:

- $A' \mathrel{\hat{=}} A_1$ */* For instance A_1 */*

A syntactical solution is to factorise only transitions that do not have a reachability condition. This choice can be restrictive but allows to syntactically flatten the HSLTS, and thus to provide more information to the user.

For instance, the state $ToSend = 0 \ \wedge \ InBuffer = 0$ can be reached by the $Reset$ event from each substate of $MessageSize > 0$, without any condition (Fig. 5). Transitions associated to $Reset$ are then factorised on their starting state without any crossing condition.

Finally, the correctness of the propositions can be easily proved by using [5, Cond. 1], which defines D and A cross-conditions.

2) Computation of Initial and Final Substates: There is not necessarily a single substate reached by all external transition coming into a state. In this part, we propose some heuristics to build $\mathbb{Q}_{Init}$. The set of final substates $\mathbb{Q}_{Final}$ can be built in the same way and is not detailed. Moreover, there is almost no interaction between the transitions factorisation and the choice of initial and final substates. This is why we consider that transitions are not factorised.

Since an initial transition which reaches a cluster effectively reaches its initial substate (Section IV-B4), every initial cluster has to include an initial substate. In the same way, if a cluster includes many initial substates, then it is impossible to know which one is reached by an initial transition and under which condition. Hence, we limit the number of initial (and final) substates to one per cluster.

Condition 2 (Constraints on Initial and Final Substates): *Each cluster includes at most one initial and one final substate:*

$$\forall (q_1, q_2) \cdot \left(\begin{array}{l} q_1 \in \mathbb{Q}_{Init} \ \wedge \ q_2 \in \mathbb{Q}_{Init} \ \wedge \ q_1 \neq q_2 \\ \Rightarrow \mathcal{S}up(q_1) \neq \mathcal{S}up(q_2) \end{array} \right)$$

$$\forall (q_1, q_2) \cdot \left(\begin{array}{l} q_1 \in \mathbb{Q}_{Final} \ \wedge \ q_2 \in \mathbb{Q}_{Final} \ \wedge \ q_1 \neq q_2 \\ \Rightarrow \mathcal{S}up(q_1) \neq \mathcal{S}up(q_2) \end{array} \right)$$

Each initial cluster includes an initial substate:

$$\forall q.(q \in \mathbb{Q}_{Init} \cap \mathsf{ran}(\mathcal{S}up) \Rightarrow \mathcal{S}up^{-1}[\{q\}] \cap \mathbb{Q}_{Init} \neq \emptyset)$$

Each final cluster includes a final substate:

$$\forall q \cdot (q \in \mathbb{Q}_{Final} \cap \mathsf{ran}(\mathcal{S}up) \Rightarrow \mathcal{S}up^{-1}[\{q\}] \cap \mathbb{Q}_{Final} \neq \emptyset)$$

The choice of initial and final substate is not deterministic and is then linked to some heuristics. We then propose several criteria to select a set of initial substate candidates. By using successively each of these criteria, we obtain a single initial substate, but others heuristics can be added:

1) *A cluster reached by no external transition does not have initial substate;*
2) **Else** *Substates reached by the maximum number of external transitions;*
3) **In case of equality:** *substates originating in the minimum of external transitions;*
4) **In case of equality:** *substates reached by in the minimum of internal transitions;*
5) **In case of equality:** *non deterministic choice.*

For instance (Fig. 5), the state $ToSend > 0$ is initial, because it is the only one which is reached by an external transition. In a same way, the state $ToSend = 0 \wedge InBuffer > 0$ is chosen as final substate since it is the state from which the maximum number of external transitions originates.

Finally, the approach described can be used to describe the behaviours of event-B systems including several refinement levels. Each refinement is then associated to a hierarchical level. The computation method differs only during the reduction of external transitions, where each level has to be treated one by one from leaves to roots.

VI. Implementation and Experimentations

A. An Implementation: the GénéSyst Tool

The GénéSyst [17] tool[5] implements the approach described. Starting from an event-B system and a refinement including the states space description, it produces an HSLTS describing the model behaviours. We choose to let

[5]Developed under the CeCILL licence (French equivalent of the GNU licence) and available at: http://www-lsr.imag.fr/users/Nicolas.Stouls/

the user describe the chosen states space as a disjunction in the **ASSERTIONS** clause. For instance, the clause **ASSERTIONS** $P_1 \vee P_2 \vee P_3$ is provided for the states space composed of three states with the definition predicates P_1, P_2 and P_3. Hence, when the user proves his model before generating its behaviours, he has to prove the correctness of the assert clause which corresponds to the completeness of the states space according to the invariant.

GénéSyst is based on the usage of automatic provers to establish the validity of a formula. Hence, the main restriction is the non-decidability of the first order logic used. Some transitions can then be considered as partially crossable, while they can never or always be passed through. In this case, we call it a *lack of proof*. However, note that this is only a syntactical restriction, since it is sufficient to manually evaluate the cross condition to obtain the correct result.

In order to minimise the number of lack of proof and to clarify the HSLTS produced, we introduce some computation optimisations, such as:

- heuristics to decrease the complexity and the number of proof obligations;
- prediction of reachability conditions, if an event can reach only one state from a given state;
- enhancement of the automatic prover efficiency by reducing hypothesis.

Table II gives some results obtained with the GénéSyst tool on published models. The communication channel is the one described in Fig. 2 and 4. The channel with big states space is a representation of the communication channel with a states space composed of 102 symbolic states. This bench is used to show that the tool is sufficiently strong to compute some diagrams too large to be used by humans as understanding aid. The DEMONEY model is an electronic purse model, described in section VII.

Model	Ev.	St.	Tr.	PO	LP	Dur.
Chocolate Machine [15]	5	3	10	57	0	15s
Refinement	7	9	16	114	0	26s
Parking [19]	4	3	4	32	0	6s
Refinement	4	4	6	29	0	6s
Communication Channel	3	2	4	20	0	4s
Refinement	4	4	12	54	0	15s
Communication Channel *(With large states space)*	3	102	304	21425	0	42min
DEMONEY	3	4	10	56	0	52s
Refinement	7	6	46	336	5	79min

Ev.: number of events / St.: number of states / Tr.: transitions built / PO: generated proof obligations / LP: number of lacks of proof / Dur.: generation duration (on a 2.8GHz P-IV with 1GB of RAM).

TABLE II
SOME GénéSyst BENCHS

Finally, GénéSyst is based on several other tools:

- the jBTools [21] parser, developed in the LIFC laboratory;
- the **Bo**B [20] tool box (notably used for $\mathcal{WP}$ calculus), developed in the LIG laboratory;
- the *AtelierB* or *B4free* automatic prover, developed by ClearSy;
- the GraphViz [8] diagram generator, developed in AT&T laboratory;
- the Grappa [16] **Java** interface for GraphViz, developed in AT&T laboratory.

B. Experimentations

1) Models Development and Properties Verification: The GénéSyst tool has been developed in the context of the GECCOO[6] [9] project, as a prototype providing an assistance to the specifier during the development process. Based on the complementarity of the B view (in terms of data modification) and the behavioural view (in terms of authorised event sequences), the GénéSyst tool provides a second point of view of the model. This view can be used to manually verify the model or to check its correctness with respect to dynamical properties, that can not be verified by invariant proof obligations. Indeed, the produced diagrams have the properties to include all authorised behaviours, contrarily to the usual approach by animating the specification.

More recently, some participants of the POSÉ[7] project have proposed to used GénéSyst to manage and limit the process of test cases generation. This technique is complementary to the generation by animation, because it provides an upper bound to the generation, while the animation gives a lower bound.

2) Aid for Documentation and Understanding: One of our requirements is to aid in understanding a model, such that the diagrams produced can be used as a communication support for non expert audience in formal methods. Hence, we based our choices on the understanding aid.

For instance, in the proposed definition of the transitions factorisation, we started from the goal that a syntactical rewriting on leaves is sufficient to obtain the HSLTS semantics, including cross conditions, without validating any proof obligations. However, this choice has a cost, since it reduces the possibility for factorisation, especially in the case of factorisation on ending super-state.

According to this motivation, GénéSyst were used by participants of the EDEMOI [3] project in order to illustrate documents. This project aimed at modelling and validating the regulations relative to civil airport security. Starting from official documents, formal models have been developed to describe all authorised behaviours of each subject (passenger, bagages, aircraft, etc.) and to exhibit if some sequences of authorised event can lead to a security failure.

In order to present the results to the ICAO experts (International Civil Aviation Organisation), that are not formal methods experts, a document was built using natural language descriptions and some diagrams. A part of those diagrams were produced by the GénéSyst tool, including all these present in the documentation of the Amdt11 model [6] (describing the aircraft flows, passengers flows and bagages flows). In conclusion to this project, the ICAO experts have appreciated and understood the diagrams, and the associated formal model.

[6]From the french ACI sécurité.
[7]French national project of the RNTL network.

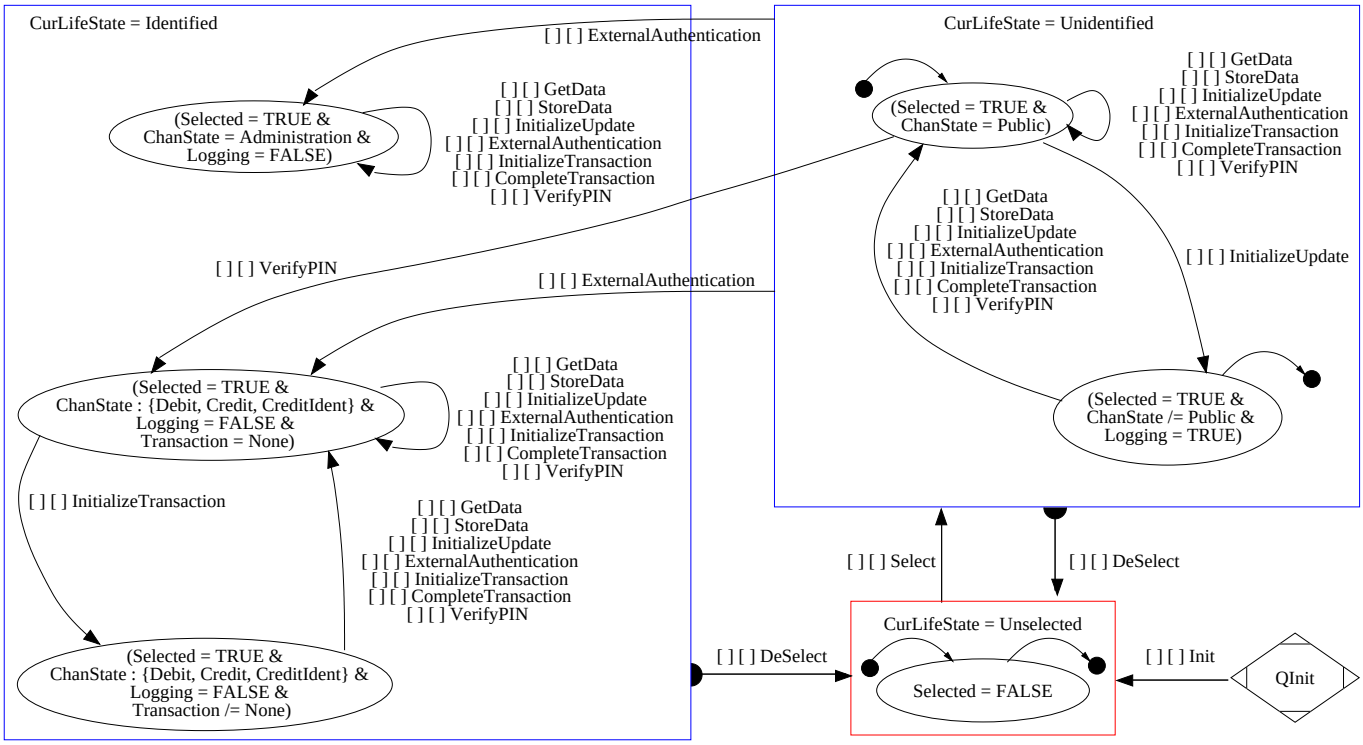

Fig. 11. DEMONEY Electronic Purse Behaviours

VII. Yet another concrete example

In this section, we describe a B formal model with only one light representation of its behaviours (Fig. 11). Due to space restrictions, only a part of the properties can be observed on the figure, but the interested reader can find the original model in file *Exemple_Demoney_V7.tar.gz* at http://www.lri.fr/~stoulsn/. This model is an abstract description of an electronic purse, based on the public specification of the DEMONEY [13] **JavaCard** electronic purse. First of all, note than no transition of Fig. 11 has any cross condition, because of a good states space choice and because the model still includes a part of non-determinism.

According to Fig. 11, the applet can be in three global states: *Unselected*, *Selected* or *Identified*. The initial state is Unselected and a *Select* event is the only one that can occur in that state (hypothesis on the **JavaCard** virtual machine – JVM). Once the applet is *Selected*, no transaction can occur, without the applet being *Identified*. To do that, there are two methods:

- with a **PIN code**. This gives access to a Debit and Credit actions (CreditIdent is a particular case of credit where the money is deduced from a bank account) ;
- with a **cryptographic key exchange**. According to the terminal used, this method can gives access either to Debit and Credit actions or to Administration actions.

In the second case, the authentication needs a sequence of two operations: *InitializeUpdate* and *ExternalAuthentication*. This mechanism is a protection to prevent attacks by removing or shutting down the card, and ensures the card integrity. The same mechanism is used to do a transaction with a sequence of two operations *InitializeTransaction* and *CompleteTransaction*.

Now, we argue that this short description paired with Fig. 11 is sufficient to understand all the concepts and actions that can be done by this applet, while the B model needs advanced skills in B.

VIII. Conclusion

The motivation for this work is to represent refined model behaviours with the maximum amount of information, in order to help in its understanding. We argued that an HSLTS is easier to understand if its representation is described with the same general structure as its abstraction. Moreover, the use of hierarchies allows to exhibit links between variables from abstract and refined descriptions.

In this paper, we formally introduced the notion of hierarchical-SLTS that we defined in terms of its flattened SLTS and then in terms of event-B model behaviours. In opposition to some usual definitions of hierarchical transition systems [14], [22], our proposal is based on the use of a single transition relation. This feature allows transitions between substates from any clusters. We propose a method to compute, by solving proof obligations, an HSLTS from a refined B model. By using refinement properties, we limit the number of proof obligations and thus the number of lack of proof.

Finally, we conclude this article with a new example which illustrates that a well chosen HSLTS can be sufficient to describe a formal model to non formal method experts. The example is an electronic purse used in the GECCOO project as a case study.

Thereafter, this approach can be integrated in a multi-view development method combining a behavioural representation (such as HSLTS) and a data-based representation (such as

B language). Finally, HSLTS can be extended with more annotations to describe and understand more precisely the system. For instance, it can be interesting to compute, for each transition, the set T of valuations that can be reached in the arrival state q_2 from a state q_1, by the event ev. Intuitively, T can be defined as:

$$[x := x']\mathcal{D}ef(q_2) \Rightarrow \begin{pmatrix} [x := x']T \Leftrightarrow \\ \exists x \cdot (\mathcal{D}ef(q_1) \wedge D \wedge A \wedge \mathsf{prd}_x(ev)) \end{pmatrix}$$

Where $[x := x']P$ is the substitution of non-free occurrences of x variable by x' and $\mathsf{prd}_x(ev)$ is the functional description of event ev (in terms of variable x, such that x' is the value of x after execution of ev).

REFERENCES

[1] J.R. Abrial. Extending B without Changing it. In Nantes H. Habrias, editor, *First Conference on the B method*, pages 169–190, 1996.

[2] J.R. Abrial. *The B-Book*. Cambridge University Press, 1996.

[3] D. Bert, F. Bouquet, Y. Ledru, and S. Vignes. Validation of Regulation Documents by Automated Analysis of Formal Models. In *REMO2V'06)*, 2006.

[4] D. Bert and F. Cave. Construction of Finite Labelled Transition Systems from B Abstract Systems. In W. Grieskamp, T. Santen, and B. Stoddart, editors, *Integrated Formal Methods*, volume 1945 of *LNCS*. Springer-Verlag, 2000.

[5] D. Bert, M-L. Potet, and N. Stouls. GénéSyst: a Tool to Reason about Behavioral Aspects of B Event Specifications. Application to Security Properties. In H. Treharne, S. King, M. C. Henson, and S. A. Schneider, editors, *ZB 2005: Formal Specification and Development in Z and B, 4th International Conference of B and Z Users*, volume 3455 of *Lecture Notes in Computer Science*, pages 299–318. Springer-Verlag, 2005.

[6] Didier Bert. Modèle formel B de l'aéroport : Amdt11, projet aci sécurité informatique : Edemoi. Rapport interne, 2006.

[7] D. Cansell, D. Méry, and S. Merz. Predicate Diagrams for the Verification of Reactive Systems. In *IFM*, volume 1945 of *LNCS*, 2000.

[8] John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull. Graphviz - Open Source Graph Drawing Tools. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Graph Drawing, 9th International Symposium, GD 2001 Vienna, Austria, September 23-26, 2001, Revised Papers*, volume 2265 of *LNCS*, pages 483–484. Springer, 2001.

[9] GECCOO. Génération de code certifié pour des applications orientées objet. spécification, raffinement, preuve et détection d'erreurs. Rapport final - http://geccoo.lri.fr, 2007.

[10] S. Graf and H. Saïdi. Construction of Abstract State Graphs with PVS. In *Computer-Aided Verification (CAV'97)*, volume 1254 of *LNCS*, 1997.

[11] A. Idani and Y. Ledru. Dynamic Graphical UML Views from Formal B Specifications. *International Journal of Information and Software Technology*, 48(3):154–169, Mars 2006. Elsevier.

[12] Leslie Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, may 1994.

[13] R. Marlet and C. Mesnil. DEMONEY : A demonstrative Electronic Purse - Card Specification. Public technical report, SECSAFE project, 11 2002. http://www.doc.ic.ac.uk/~siveroni/secsafe/docs/secsafe-tl-007-0.8.pdf.

[14] E. Mikk, Y. Lakhnech, and M. Siegel. Hierarchical automata as model for statecharts. In R. Shyamasundar and K. Ueda, editors, *Advances in Computing Science - ASIAN '97*, volume 1345 of *LNCS*, pages 181–196, December 1997.

[15] R. Milner. *Communication and concurrency*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1989.

[16] John Mocenigo. Grappa: A Java Graph Package. http://www.research.att.com/ john/Grappa/, 2006.

[17] Xavier Morselli, Marie-Laure Potet, and Nicolas Stouls. Génésyst : Génération d'un système de transitions étiquetées à partir d'une spécification B événementiel. In J. Julliand, editor, *AFADL'04, session outil*, pages 317–320, June 2004.

[18] OMG. *Unified Modeling Language Specification*, septembre 2001. Version 1.4.

[19] Marie-Laure Potet and Nicolas Stouls. Explicitation du contrôle de développement B événementiel. In J. Julliand, editor, *AFADL'04*, pages 13–27, June 2004.

[20] Nicolas Stouls. Compte rendu de stage de maîtrise. Chapitre 4, Université Joseph Fourier / Laboratoire LSR, équipe VASCO, rue de la Passerelle, 38402 St Martin d'hères, juillet 2002. Encadrant : Marie-Laure Potet.

[21] J-C. Voisinet, B. Tatibouet, and A. Hammad. jBTools: An Experimental Platform for the Formal B Method. In *PPPJ'02*, pages 137–140. Trinity College, Dublin, Ireland, Juin 2002.

[22] Mihalis Yannakakis. Hierarchical State Machines. In *Exploring New Frontiers of Theoretical Informatics, International Conference IFIP TCS*, volume 1872 of *LNCS*, pages 315–330. Springer, 2000.

APPENDIX

In case of HSLTS generation, we want establish Prop. 2.2 which is:

$$L \wedge \mathcal{D}ef_{\mathcal{A}}(\mathcal{S}up(q_1)) \wedge D_{\mathcal{R}} \wedge A_{\mathcal{R}} \Rightarrow A_{\mathcal{A}}$$

We propose the following proof:

Proof: Property 2.2 According to the definition of a refined event coming from [2] and by using the normalised form, we have:

$$L \wedge \mathsf{prd}_y(\mathsf{Action}_{\mathcal{R}}(ev)) \Rightarrow \exists x' \cdot (\mathsf{prd}_x(\mathsf{Action}_{\mathcal{A}}(ev)) \wedge [y, x := y', x']L)$$

Consequently to the Prop 1, page 4, the leaves space is complete according to the invariant J from the refinement[8]. Hence, there exists necessarily a leaf-state q_2 in $\mathbb{Q}$ such that $[y := y']\mathcal{D}ef_{\mathcal{R}}(q_2)$. Since $[y, x := y', x']L$, thus the super-state of q_2 verify $[x := x']\mathcal{D}ef_{\mathcal{A}}(\mathcal{S}up(F))$ in $T_{\mathcal{A}}$. We then have :

$$L \wedge \mathsf{prd}_y(\mathsf{Action}_{\mathcal{R}}(ev)) \Rightarrow \exists x' \cdot \begin{pmatrix} \mathsf{prd}_x(\mathsf{Action}_{\mathcal{A}}(ev)) \wedge [y, x := y', x']L \wedge \\ [x := x']\mathcal{D}ef_{\mathcal{A}}(\mathcal{S}up(q_2)) \wedge [y := y']\mathcal{D}ef_{\mathcal{R}}(q_2) \end{pmatrix}$$

That implies :

$$L \wedge \mathsf{prd}_y(\mathsf{Action}_{\mathcal{R}}(ev)) \Rightarrow \begin{pmatrix} \exists x' \cdot (\mathsf{prd}_x(\mathsf{Action}_{\mathcal{A}}(ev)) \wedge [x := x']\mathcal{D}ef_{\mathcal{A}}(\mathcal{S}up(q_2))) \\ \wedge [y := y']\mathcal{D}ef_{\mathcal{R}}(q_2) \end{pmatrix}$$

That implies that the following formula holds :

$$L \wedge \mathsf{prd}_y(\mathsf{Action}_{\mathcal{R}}(ev)) \wedge [y := y']\mathcal{D}ef_{\mathcal{R}}(q_2) \Rightarrow \exists x' \cdot (\mathsf{prd}_x(\mathsf{Action}_{\mathcal{A}}(ev)) \wedge [x := x']\mathcal{D}ef_{\mathcal{A}}(\mathcal{S}up(q_2)))$$

Although, there exists necessarily a state q_1 in $\mathbb{Q}$ such that y verify $\mathcal{D}ef_{\mathcal{R}}(q_1)$ (Prop. 1). Since L is in hypothesis, thus x verify $\mathcal{D}ef_{\mathcal{A}}(\mathcal{S}up(q_1))$.

$$\begin{pmatrix} L \wedge \mathcal{D}ef_{\mathcal{A}}(\mathcal{S}up(q_1)) \wedge \\ \mathsf{Guard}_{\mathcal{R}}(ev) \wedge \langle \mathsf{Action}_{\mathcal{R}}(ev) \rangle \mathcal{D}ef_{\mathcal{R}}(q_2) \end{pmatrix} \Rightarrow \langle \mathsf{Action}_{\mathcal{A}}(ev) \rangle \mathcal{D}ef_{\mathcal{A}}(\mathcal{S}up(q_2))$$

That is finally equivalent to :

$$L \wedge \mathcal{D}ef_{\mathcal{A}}(\mathcal{S}up(q_1)) \wedge D_{\mathcal{R}} \wedge A_{\mathcal{R}} \Rightarrow A_{\mathcal{A}} \qquad (2.2)$$

■

[8] In B, the link invariant L is the conjunction of the invariant I from abstraction and the invariant J from refinement.