Specifying a train system using ASTD and the B method : Technical Report

Thomas Fayolle

November 25, 2014

Contents

1 Introduction

In this report we detail the specification of a train system, to explore the complementarity between B-like specifications and ASTD for safety-critical system modeling. The majority of train systems are based on the communication between two sub-systems. The first one is called on-board controler and is the controler that actually drives the train. The second one is called the track controler. It controls the entire train system and authorizes the trains to move. The specification will be made step by step, using refinement relations between the steps. This introduction will detail the used methodology and explain and justify the technical choices.

1.1 Background

1.1.1 B and Event-B

B is a formal method [2] supporting the main stages of the software development life cycle. Specifications are composed of abstract machines, which encapsulate state variables, an invariant constraining the state variables, an initialization of all the state variables, and operations on the state variables. The invariant is a first-order predicate in a simplified version of the ZF-set theory, enriched by many relational operators. Abstract sets or enumerated sets are used for typing the state variables. In B, state variables are modified only by means of substitutions. The initialization and the operations are specified in a generalization of Dijkstra's guarded command notation, called the Generalized Substitution Language (GSL), that allows the definition of non-deterministic and preconditioned substitutions. An operation is generally a preconditioned substitution, of the form PRE P THEN S END, where P is the precondition and S is a substitution. The state transition specified by a preconditioned substitution is guaranteed only when the precondition is satisfied. The main substitutions that will be used in the case study are: assignment substitution (denoted by $:=$); substitution of the form $x :|(P)$, which states that state variable x is updated such that predicate P becomes true; and simultaneous substitutions ($||$).

Through refinement steps, the initial abstract machine is transformed, step by step, into a B model of the code. Translations tools are then available for synthesizing the final code. Proof activity consists in proving all the generated proof obligations for the abstract machine and for each refinement step. In that aim, the B method is supported by several tools like Atelier B[1], ProB[2] and RODIN[3].

Event-B [1] is an evolution of the B language to specify complex systems by using decomposition and event-based descriptions. In Event-B, specifications describe "closed" event systems, in order to consider a system and its interactions with its environment as a whole. The behaviour is then modeled by events on the system. An event is defined by a guard, a blocking condition that ensures the consistency of the system if the event is executed, and an action described by GSL as in B. An event is of the form ANY $x, y, ...$ WHERE $P(x, y, ..., v, w, ...)$ THEN $S(x, y, ..., v, w, ...)$ END, where $x, y, ...$ are local variables and $v, w, ...$ are constants or state variables of the event system, predicate P is the guard, and substitution S, the action. An event system may be refined. Refinement in Event-B not only refines data structures like in B, but also allows new events to be added. However, only new concrete variables can be modified by new events. The state refinement is expressed, like in B, with a gluing invariant between the abstract state and the concrete state.

[1] `http://www.atelierb.eu`
[2] `http://www.stups.uni-duesseldorf.de/ProB`
[3] `http://www.rodintools.org`

1.1.2 ASTD

ASTD [5] is a formal graphical notation, which is an extension of Harel's Statecharts [7] with process algebra operators. Each ASTD type corresponds to either a hierarchical automaton or a process algebra operator like sequence, choice, Kleene closure, guard, synchronization, choice and interleave quantification. One of the main important features of ASTDs is to allow parameterized instances and quantifications. Moreover, the graphical representation brings an important mean for communicating with stakeholders and for validating the system model. This formal language has been notably used in the context of secure web services for the security policy specification [9, 3]. For the sake of concision, we introduce only the ASTD operators that will be used in the case study: automaton, quantified parameterized synchronization, Kleene closure and weak synchronization. The complete operational semantics is in [4].

An ASTD automaton is similar to a classical automaton, except that its states can be of any ASTD type, and that its transition relation δ can refer to substates of automaton states. Hence, there are three kinds of arrows: local transition between two states n_1 and n_2 of the automaton, denoted by (loc, n_1, n_2); transition from n_1 to substate $n_{2_\flat}$ of n_2; and transition from substate $n_{1_\flat}$ of n_1 to n_2. A transition can also be guarded or considered as final (*i.e.* it is triggered only if its source state is final). Thus, a transition from δ is of the form $(t, \sigma, g, \mathit{final}?)$, where t denotes the arrow, σ is the event, g is the guard, and *final*? is a boolean denoting whether the transition is final. For the sake of concision, history states are here omitted. A state of an automaton is of the form $(\mathsf{aut}_\circ, n, s)$ where n is the name of its current state and s is the current substate of the state. For example, rule aut_1 describes the semantics of a local transition

$$\mathsf{aut}_1 \ \frac{\delta((\mathsf{loc}, n_1, n_2), \sigma', g, \mathit{final}?) \qquad \Psi}{(\mathsf{aut}_\circ, n_1, s) \xrightarrow{\sigma,\Gamma} (\mathsf{aut}_\circ, n_2, \mathit{init}(\nu(n_2)))}$$

Predicate Ψ is a premiss which checks if the source state is final for final transitions, if the guard holds, and if the event received, noted σ, is equal, under the current transition environment Γ, to the event specified in the transition relation, noted σ'. Expression $\mathit{init}((\nu(n_2)))$ represents the initial state of the ASTD whose name is n_2. Thus, the target state of the transition is the initial state of the destination state in δ.

Then we need Kleene closure: a Kleene closure ASTD is an ASTD that can be executed zero, one or more time. When the final state is reached, the ASTD can restart.

The next ASTD we consider is parameterized quantified synchronization. The behaviour is defined as follows. Many ASTDs are executed in parallel. For each event whose label belongs to a synchronization set Δ, all ASTDs must execute this event at the same time; otherwise, they are executed in interleaving. The first requirement may be too strong to satisfy in some situations. In particular, if a quantified parameterized synchronization is used to specify the behaviour of several entities in parallel, it would be very restrictive to prevent a large subset of entities from executing a synchronized event because some of them are not ready or can be considered as having stopped their activity.

To take such cases into account, a weak synchronization has been defined. A state of the ASTD is then of the form $(\Psi_\circ, f)$, where f maps an ASTD state to each quantification parameter value. In the type corresponding to this kind of ASTDs, Δ represents the set of the actions that synchronize as above, and predicate p characterizes which instances of the quantified ASTD must synchronize. There are two inference rules:

$$\Psi_1 \ \frac{\alpha(\sigma) \notin \Delta \qquad f(v) \xrightarrow{\sigma, ([x := v]) \lhd \Gamma} s'}{(\Psi_\circ, f) \xrightarrow{\sigma,\Gamma} (\Psi_\circ, f \lhd\!\!+ x \mapsto s')}$$

$$\Psi_2 \ \frac{\alpha(\sigma) \in \Delta \qquad \forall v \in T.((\neg([x := v]p) \wedge f(v) = f'(v)) \vee (f(v) \xrightarrow{\sigma, ([x := v])} f'(v)))}{(\Psi_\circ, f) \xrightarrow{\sigma, \Gamma} (\Psi_\circ, f')}$$

Rule Ψ_1 is applied when there is no synchronization. Rule Ψ_2 corresponds to the case with synchronization: all the ASTDs for which p is true execute the event at the same time and the state of the other ASTDs does not change.

1.2 Methodology

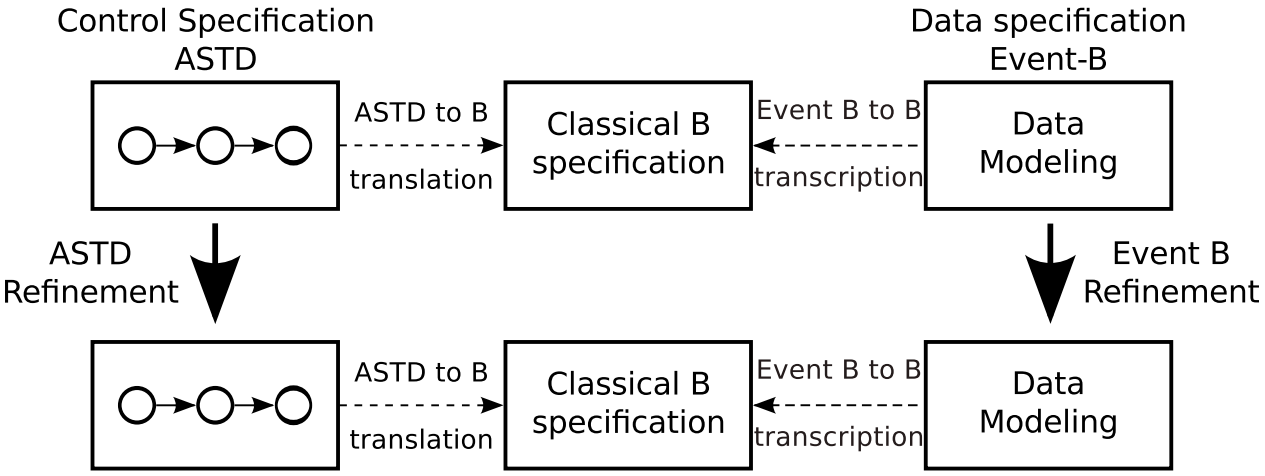

Figure 1: Methodology of the specification

Our approach uses a coupling of the graphical ASTD notation and Event-B to specify a system. The specification methodology is shown in Fig. 1. A system can be viewed in two parts. The first part models the dynamic behaviour of the system, and is specified in ASTD (box on the left in Fig. 1). The second part focuses on data, and is described in Event-B (box on the right in Fig. 1). Transitions constitute the link between the two parts: to each action label in ASTD corresponds an event in Event-B. To ensure the global consistency of the system, the ASTD and Event-B specifications are translated into classical B (middle box in Fig. 1). As we will explain later, the classical B is only used for technical reasons.

1.2.1 ASTD specification.

With graphical notation and process algebra operators, the ASTD specification models the ordering of actions. Since formal notations are not always easy to understand, ASTD provides a graphical visualisation which makes the model validation easier, while still remaining formal. Compared to Statecharts, the ASTD language is based on process algebra operators, like quantified parameterised synchronisation, which allows to represent many processes in parallel.

1.2.2 Event-B specification.

Event-B specification contains an event for each action label declared in ASTD. The ASTD part just describes the ordering of actions. In the Event-B part we specify the effects on the data model of each ASTD action. Static properties like safety and typing constraints are specified by means of Event-B invariants. Sometimes we need temporal properties which are not supported by the Event-B notation. In that case, we encode these temporal properties by using theorems. RODIN tools[4] are used to generate and prove the proof obligations associated to invariant preservation and additional theorems.

1.2.3 B specification.

The classical B specification contains two B machines. The first one is the translation of the ASTD specification, the second one is a transcription of the Event-B specification.

The ASTD to B translation can be summed up as follows. ASTD states are encoded by B state variables. To every ASTD action label corresponds a B operation. Its precondition checks that the state variable is in the initial state of the ASTD transition. Its postcondition assigns to the state variable the final state of the transition. Moreover, to link the resulting B operation with the data model, we would like to execute the events defined in the Event-B part during the transition. But technically, a B operation cannot call an Event-B event. That is why we also have to translate the events into B operations.

For the translation of the Event-B machine, variables and typing invariants remain unchanged. Events are rewritten into B operations: their guards are simply changed into preconditions and their postconditions remain identical. Grouping the two parts together in one unique B specification allows the global consistency (one horizontal level in Fig. 1) of the system to be proved: when we call an operation in B, the generated proof obligation checks that the precondition of the called operation is true before executing it. To prove the calling proof obligations, invariants are added in the B machine that translates the ASTD. This invariants link the variables of the Event-B description and the variables that encode the states of the ASTD.

Event-B provides the expressiveness and the refinement relation required for the system to be modeled, but it lacks some modularity features. There exist theoretical foundations for modularity in Event-B [8], but in practice, they are not yet supported by existing tools. B is then used for technical reasons.

1.2.4 Refinement of the model.

The methodology uses two refinements. On one side we refine the ASTD specification (left refinement arrow in Fig. 1), on the other side we refine the data specification in Event-B (right refinement arrow in Fig. 1).

A first definition of ASTD refinement is proposed in [6]. This refinement definition requires the traces to be preserved and three generic application patterns are described. By trying to apply this ASTD refinement relation on the case study described here, we realise that it is too restrictive. Consequently we have introduced new patterns that weaken the original definition but preserve behaviour consistency: the properties that are true in a state of the abstract ASTD specification have to be preserved in corresponding state of the concrete specification.

The Event-B part of the specification is refined using the Event-B definition of refinement. The proof obligations are automatically generated by the RODIN tool. The Event-B refinement guarantees the preservation of the invariants in the data part of the specification. This refinement definition is one of the reasons why we chose Event-B to specify the data part of the system: The classical B refinement does not allow events to be added, while ASTD refinement allows new transitions.

[4] www.event-b.org

1.3 Modeling choices

In this case study, a train system is modeled using ASTD. The most known train systems are CBTC (Communication Based Train System) and ERTMS (European Rail Traffic Management System). The first one is used for metros, the second one for trains. Those two systems are based on communications between two subsystems. The on-board controler is the system that drives the train. It has to respect the instructions given by the track controler that knows the positions of all the trains on the track and that computes limits for the train movement.

At the most abstract level, we consider a unique track on which a set of trains are moving in the same direction. This is realistic since, if there are trains in the opposite direction, they are blocked at switches. Furthermore, all the issues concerning switches, interlocking, etc... are not considered in this paper. They can be dealt with in subsequent refinement steps. Consequently, we check only the absence of front-to-rear collision.

The data concerning the track are defined in an Event-B context. It can be seen on figure 2. A set named $TRACK$ is defined. On this set, there is a total, strict, order relation (irreflexive, transitive and asymetric) called is_behind. $x1 \mapsto x2 \in is\_behind$ means that the element $x1$ is behind the element $x2$ on the track. The element *position0* represent the first element in the begining of the track.

The case study contains six levels of refinement. The first two ones specify a set of train moving autonomously. The third one introduces a control operation for each train. The fourth one regroups all the control operations in a unique control operation. The fifth one introduces communication operations. Those operations will allow us to decompose the system in the sixth level of refinement.

2 Specifications

2.1 First specification

We start by specifying the system behaviour in ASTD (see Fig. 3). The fact that the system contains many trains is represented by the quantified interleaving operator. Each train can start, then move zero or more times and then stop. Thanks to the Kleene closure, when it is stopped, a train can restart.

Since we assume that the trains are moving in the same direction on one track, the non-collision property can be expressed by the following predicate:

$$\begin{aligned}\forall t1, t2.(&t1 \in set\_of\_trains \wedge \\ &t2 \in set\_of\_trains \wedge \\ &t1 \neq t2 \Rightarrow \\ &\quad position\_of(t1) \neq position\_of(t2))\end{aligned} \tag{1}$$

Predicate (1) means that the positions of two distinct trains are different. We also need another predicate to express that a train cannot jump over another train.

$$\begin{aligned}\forall t1, t2.(&t1 \in set\_of\_trains \wedge \\ &t2 \in set\_of\_trains \wedge \\ &position\_of(t1) \mapsto position\_of(t2) \in is\_behind \Rightarrow \\ &\quad \mathsf{X}(position\_of(t1) \mapsto position\_of(t2) \in is\_behind))\end{aligned} \tag{2}$$

CONTEXT TRACK_DEF

SETS

$TRACK$

$TRAIN$

CONSTANTS

is_behind

$position0$

AXIOMS

axm1 : $is\_behind \in (TRACK \leftrightarrow TRACK)$

axm2 : $(\forall p1 \cdot (p1 \in TRACK \Rightarrow \neg(p1 \mapsto p1 \in is\_behind)))$

axm3 : $\forall p1, p2, p3 \cdot (p1 \in TRACK \wedge p2 \in TRACK \wedge p3 \in TRACK \wedge p1 \mapsto p2 \in is\_behind \wedge p2 \mapsto p3 \in is\_behind \Rightarrow p1 \mapsto p3 \in is\_behind)$

axm4 : $\forall p1, p2 \cdot (p1 \in TRACK \wedge p2 \in TRACK \wedge p1 \mapsto p2 \in is\_behind \Rightarrow \neg(p2 \mapsto p1 \in is\_behind))$

axm5 : $\forall p1, p2 \cdot (p1 \in TRACK \wedge p2 \in TRACK \wedge p1 \neq p2 \Rightarrow p1 \mapsto p2 \in is\_behind \vee p2 \mapsto p1 \in is\_behind)$

axm6 : $position0 \in TRACK$

axm7 : $\forall pp \cdot (pp \in TRACK \wedge pp \neq position0 \Rightarrow position0 \mapsto pp \in is\_behind)$

END

Figure 2: Event-B context defining the set $TRACK$

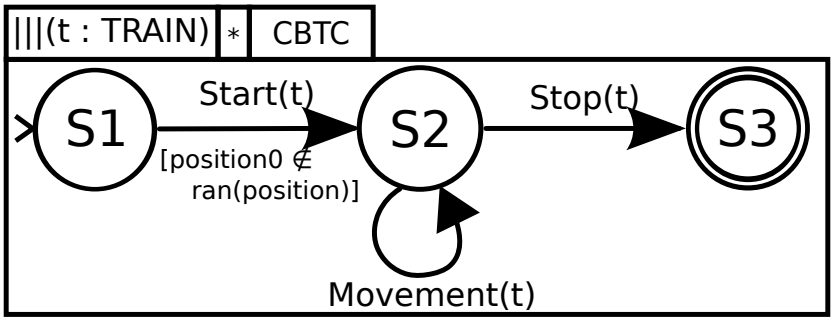

Figure 3: Train System ASTD Specification: First Specification

Predicate (2) checks that the order of trains does not change. Symbol X denotes the "next" operator from temporal logic [10].

To express these predicates in Event-B we introduce a set $TRAIN$ and a state variable *position* which is a partial function from $TRAIN$ to $TRACK$ ($position \in TRAIN \nrightarrow TRACK$). Variable *position* is set when the train starts and until it stops. The EventB event, corresponding to the movement action, that acts on the data is called $movement\_act$. It updates the variable such as:

- the new position is different from the positions of the other trains;
- the new position stays behind the position of the trains that were located before;
- the new position cannot be located behind the old position.

The *start* operation set the position value to the constant *position0*. This constant is only used for technical reasons: In the Event-B methods, when a variable is modified with a *Before/After* predicate, a proof of existence is needed. The idea of the proof is to prove that it exists a variable that satisfies the *Before/After* predicate. By setting the position to a value, this proof becomes trivial.

Predicate (1) is directly defined as an invariant of the data machine. Predicate (2) uses an operator coming from temporal logic and cannot be model checked by ProB. To avoid this issue, we translate temporal logic predicate (2) into assertions on the states, written as Event-B theorems. An Event-B theorem is an assertion that has to be proved with the invariants of the machine. A theorem is written for each event of the machine. For example, the theorem corresponding to the $movement\_act$ event checks that for all trains $train1$, $train2$ and $train3$:

- if $train1$ is behind $train2$
- if the precondition of the $movement\_act$ operation is true for $train3$
- if we execute $movement\_act(train3)$

then $train1$ stays behind $train2$. Note that this theorem includes the three possible cases: $train3 = train1$, $train3 = train2$ and ($train3 \neq train1$ and $train3 \neq train2$).

Figure 5 shows the Event-B theorem proved for the movement event. The idea is that we define a relation that links the state of the variables before the execution of the operation and the state of the variables after the execution of the operation (axm8_1 and axm8_2). Then we check that if the property was right for the

variables that are in the before-state of the event, it is still true for the variables that are in the after-state of the event (th_movement_act). The first two properties are axioms. The third one is a theorem and has to be proved.

Both ASTD and Event-B specifications are then translated into classical B. For each transition of the ASTD, a B operation is created. The precondition of the operation checks that the ASTD is in a state that allows the transition. The post-condition change the state of the ASTD: the state becomes the final state of the translation. In the post-condition, the translation of the Event-B operation is called.

The Event-B specification checks that the invariants do not change when each event is executed. This means that we check that the invariants do not change if the events are executed when their guards are true. By translating them and calling them in the classical B, we check that their guards are true when they are executed. This garantees the horizontal consistency of the model.

The Event-B operations of the first level of specification can be seen on figure 4.

2.2 First refinement

In the following parts, we will explain how the trains are moving while respecting the non-collision property. This refinement details the movement cycle as a Kleene closure ASTD. The new refinement can be seen in figure 6. The refinement we use does not exactly satisfy the definition of ASTD refinement.

Let's remind the definition of ASTD refinement. Let A be an abstract ASTD specification and C a concrete ASTD specification. We say that $A \sqsubseteq C$ iff:

$$\alpha(C) - \alpha(A) \neq \varnothing \tag{3}$$

$$\mathsf{traces}(A^{\setminus \bar{C}}) = \mathsf{traces}(C^{\setminus \bar{A}}) \tag{4}$$

$$\mathsf{deadlocks}(A^{\setminus \bar{C}}) = \mathsf{deadlocks}(C^{\setminus \bar{A}}) \tag{5}$$

$$\tau - \mathsf{enabled}(A^{\setminus \bar{C}}) \subseteq \tau - \mathsf{enabled}(C^{\setminus \bar{A}}) \tag{6}$$

where $A^{\setminus \bar{C}}$ is the ASTD A where all the transition that are in A and not in C are hidden (replaced by a silent τ-transition). It means that $A^{\setminus \bar{C}} = A \setminus (\alpha(A) - \alpha(C))$. If all the propsitions are true except the first one and if $\alpha(A) = \alpha(C)$, we say that the ASTD A and the ASTD C are equivalent.

In our refinement case, propositions (4), (5) and (6) are true and $\alpha(A) = \alpha(C)$. The concrete ASTD specification is a way to rewrite the abstract ASTD specification. Since the concret ASTD and the abstract one are equivalent and contain the same operations, the Event-B specification does not change.

2.3 Second refinement

In the previous specifications, the *movement* operation modifies the trains position such as the new position respect the non-collision properties. In this refinement, we explain the way such a position can be chosen. A computing operation is added. This operation computes a limit for each train. The limit is called *Movement Authority Limit* (*mal*). The movement operation updates the position of the train using the *mal*. The new position cannot overtake the limit.

The ASTD specification can be seen in figure 7. The refinement is proved using the general definition of the ASTD refinement from [6]. The propositions (3), (4), (5) and (6) are proved.

The limit is computed each time the position of the trains are updated, that means after the train starts and after every movement. The new limit is such that:

Event $start\_act \mathrel{\widehat{=}}$

any tt

where

gu1 : $tt \in TRAIN$

gu2 : $tt \notin dom(position)$

gu3 : $position0 \notin ran(position)$

then

act1 : $position := position \lhd\!\!+ \{tt \mapsto position0\}$

end

Event $movement\_act \mathrel{\widehat{=}}$

any tt

where

gu1 : $tt \in TRAIN$

gu2 : $tt \in dom(position)$

then

act1 : $position :\mid (\exists pp \cdot (pp \in TRACK \land$
$position' = position \lhd\!\!+ \{tt \mapsto pp\} \land$
$(\forall t2 \cdot (t2 \in dom(position') \land$
$t2 \neq tt \Rightarrow pp \neq position'(t2))) \land$
$(\forall t2 \cdot (t2 \in dom(position) \land$
$position(tt) \mapsto position(t2) \in behind \Rightarrow$
$pp \mapsto position(t2) \in behind)) \land$
$(pp = position(tt) \lor$
$position(tt) \mapsto pp \in behind)))$

end

Event $stop\_act \mathrel{\widehat{=}}$

any tt

where

gu1 : $tt \in TRAIN$

then

act1 : $position := \{tt\} \lhd position$

end

END

Figure 4: Train System Event-B Specification: First Specification

$$\texttt{axm8\_1} : movement\_relation \in TRAIN \leftrightarrow ((TRAIN \nrightarrow TRACK) \leftrightarrow (TRAIN \nrightarrow TRACK))$$

$$\begin{aligned}
&\texttt{axm8_2} : movement_relation = \{train \cdot train \in TRAIN \mid \\
&\quad train \mapsto \{position1, position2 \cdot (position1 \in TRAIN \nrightarrow TRACK \wedge \\
&\quad train \in dom(position1) \wedge \\
&\quad \exists pp \cdot (pp \in TRACK \wedge \\
&\qquad position2 = position1 \lhd\!\!\!\!- \{train \mapsto pp\} \wedge \\
&\qquad (\forall t2 \cdot (t2 \in dom(position2) \wedge t2 \neq train \Rightarrow pp \neq position2(t2))) \wedge \\
&\qquad (\forall t2 \cdot (t2 \in dom(position1) \wedge \\
&\qquad\quad position1(train) \mapsto position1(t2) \in is_behind \Rightarrow \\
&\qquad\qquad pp \mapsto position1(t2) \in is_behind)) \wedge \\
&\qquad (pp = position1(train) \vee position1(train) \mapsto pp \in is_behind)))\mid \\
&\qquad\quad position1 \mapsto position2\}\}
\end{aligned}$$

$$\begin{aligned}
&\texttt{th_movement_act} : \forall train3, position1, position2 \cdot \\
&\quad (position1 \mapsto position2 \in movement_relation(train3) \Rightarrow \\
&\qquad \forall train1, train2 \cdot (train1 \in dom(position1) \wedge \\
&\qquad\qquad train1 \in dom(position2) \wedge \\
&\qquad\qquad train2 \in dom(position1) \wedge \\
&\qquad\qquad train2 \in dom(position2) \wedge \\
&\qquad\qquad position1(train1) \mapsto position1(train2) \in is_behind \Rightarrow \\
&\qquad\qquad\quad position2(train1) \mapsto position2(train2) \in is_behind))
\end{aligned}$$

Figure 5: Expression Of Non-Collision Property In Event-B Theorem

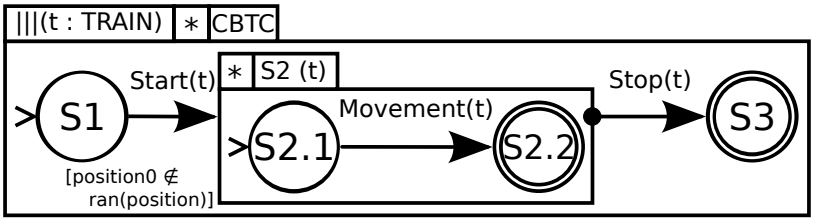

Figure 6: Train System ASTD Specification: First Refinement

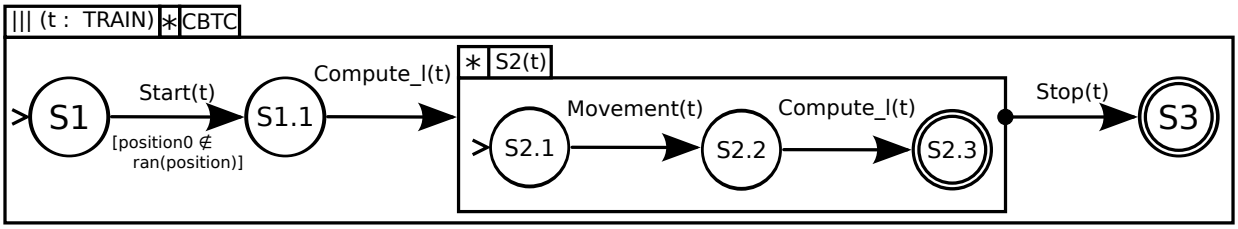

Figure 7: Train System ASTD Specification: Second Refinement

- The position of the train is behind the limit or equal to the limit
- The limit is behind the position of all trains that are ahead

The movement operation updates the position of the train with the *mal* variable. The new position has to be behind the limit. Figure 8 shows the specification of $compute\_l\_act$ and the new specification of $movement\_act$. $start\_act$ and $stop\_act$ remain unchanged.

The non-collision property is expressed with the two following predicates :

$$\begin{aligned}\forall t1, t2.(&t1 \in TRAIN \wedge \\ &t2 \in TRAIN \wedge \\ &position(t1) \mapsto position(t2) \in is\_behind \Rightarrow \\ &\quad mal(t1) \mapsto position(t2) \in is\_behind\end{aligned} \tag{7}$$

$$\begin{aligned}\forall tt.(&tt \in TRAIN \wedge \\ &tt \in dom(mal) \wedge \\ &tt \in dom(position) \Rightarrow \\ &\quad (position(tt) \mapsto mal(tt) \in is\_behind \vee \\ &\quad position(tt) = mal(tt)))\end{aligned} \tag{8}$$

Predicate (7) checks that the *mal* of a train is always behind the trains that are ahead it. Predicate (8) checks that a train cannot overtake its limit. The propositions are translated into B invariants and proved using the RODIN tools.

2.4 Third refinement

In this level of refinement, all the local $compute\_l$ operations are grouped into one global *compute* operation. This operation is synchronized for all the started train (Ψ operator). The new ASTD specification can be seen on figure 9.

This refinement transforms an interleaving ASTD $|||(t \in TRAIN)A(t)$ into a synchronised ASTD $\Psi(t \in TRAIN)C(t)$. The set of accepted traces is restricted. But we want to preserve behaviour consistency:

Event $compute\_l\_act \mathrel{\widehat{=}}$

any tt

where

gu1 : $tt \in TRAIN$

gu2 : $tt \in dom(position)$

then

act1 : $mal :| (\exists mm \cdot (mm \in TRACK \wedge$
$\forall t2 \cdot (t2 \in TRAIN \wedge t2 \in dom(position) \wedge$
$position(tt) \mapsto position(t2) \in is\_behind \Rightarrow$
$mm \mapsto position(t2) \in is\_behind) \wedge$
$(position(tt) \mapsto mm \in is\_behind \vee position(tt) = mm) \wedge$
$mal' = mal \Leftarrow \{tt \mapsto mm\}))$

end

Event $movement\_act \mathrel{\widehat{=}}$

any tt

where

gu1 : $tt \in TRAIN$

gu2 : $tt \in dom(position)$

gu3 : $tt \in dom(mal)$

then

act1 : $position :| (\exists pp \cdot (pp \in TRACK \wedge$
$position' = position \Leftarrow \{tt \mapsto pp\} \wedge$
$(position(tt) = mal(tt) \Rightarrow position(tt) = pp) \wedge$
$(\neg(position(tt) = mal(tt)) \Rightarrow (position(tt) \mapsto pp) \in is\_behind) \wedge$
$(pp = mal(tt) \vee (pp \mapsto mal(tt)) \in is\_behind)))$

end

Figure 8: Train System Event-B Specification: Second Refinement

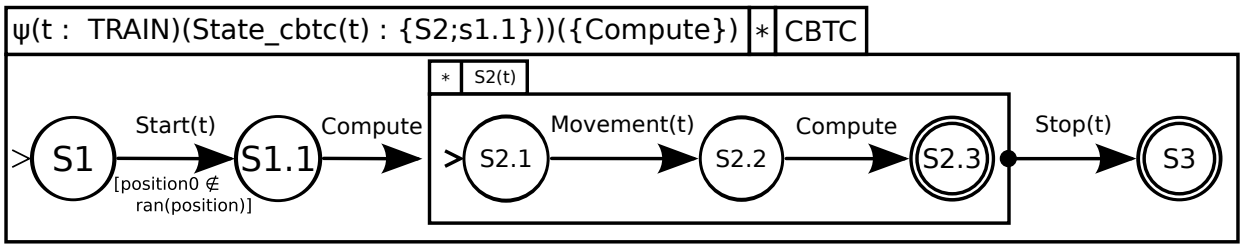

Figure 9: Train System ASTD Specification: Third Refinement

Event $compute\_act \mathrel{\widehat{=}}$

begin

act1 : $mal : |((dom(mal') = dom(position) \land$
$mal' \in TRAIN \nrightarrow TRACK \land$
$(\forall t1, t2 \cdot (t1 \in dom(position) \land t2 \in dom(position) \land$
$position(t1) \mapsto position(t2) \in is\_behind \Rightarrow$
$mal'(t1) \mapsto position(t2) \in is\_behind)) \land$
$(\forall tt \cdot (tt \in dom(mal') \Rightarrow (position(tt) \mapsto mal'(tt) \in is\_behind \lor$
$mal'(tt) = position(tt))))))$

end

Figure 10: Train System Event-B Specification: Third Refinement - compute_act event

For each train $t_n \in TRAIN$ the ASTD $C(t_n)$ is a refinement of the ASTD $A(t_n)$ according to the definition proposed in [6]. It means that if the global set of traces accepted by the ASTD specification is reduced, the local behaviour of each entities is preserved.

Since we change the local *compute_l* operation into a global one, we need to define this operation in the data part of the specification. This operation computes for all the started trains a limit such that the limit respects the invariants defined in section 2.3. The Event-B specification of *compute_act* can be seen in figure 10.

Proving this refinement is not possible in Event-B: a set of local events cannot be refined by a global one. To prove the consistency of our specification, we proved that executing *compute* operation is equivalent to execute any sequence of *compute_l* (which means an interleaving of *compute_l*).

To prove the refinement, events are expressed as relation between the state of a variable before and after executing the event. We write Rel_{Ev} the relation for an event Ev and $Rel_{Ev}(t)$ if the event has a parameter. We proved that:

$$\begin{aligned} \forall (t_1, t_2).(t_1 \in TRAIN \land \\ t_2 \in TRAIN \Rightarrow \\ Rel_{Compute\_l}(t_1); Rel_{Compute\_l}(t_2) = Rel_{Compute\_l}(t_2); Rel_{Compute\_l(t_1)} \end{aligned} \tag{9}$$

which means that for all couple of trains, the order in which we execute *Compute_l* event does not change the result. Using (9) and by induction on the set of trains, we prove that all the sequences of *Compute_l* execution are equivalents. Finally, we prove that executing an arbitrary sequence of *Compute_l* is equivalent to execute *Compute*. This implies that executing *Compute* is equivalent to execute an interleaving of *Compute_l* event.

To make this proof, the same solution that we used to prove the temporal property in section 2.1 is used. A context is defined (see fig. 11). In this context, we represent the operations as relations - the relation $Rel_{Compute\_l}$ (axioms axm8 and axm9 in fig. 11). Then a theorem is defined to prove the property (theorem th1 in fig. 11).

2.5 Fourth refinement

In this level of refinement, the communication operations are introduced. In the further refinement, we want to decompose our system into two sub-systems. The on-board system will do the *movement* and the *start*

AXIOMS

axm8 : $compute\_l\_relation \in$
$(TRAIN \leftrightarrow (TRAIN \nrightarrow TRACK)) \leftrightarrow$
$((TRAIN \nrightarrow TRACK) \leftrightarrow (TRAIN \nrightarrow TRACK))$

axm9 : $compute\_l\_relation = \{train, position1 \cdot train \in TRAIN \wedge$
$train \in dom(position1) \wedge position1 \in TRAIN \nrightarrow TRACK |$
$(\{train \mapsto position1\}) \mapsto \{mal1, mal2 \cdot mal1 \in TRAIN \nrightarrow TRACK \wedge$
$(\exists mm \cdot (mm \in TRACK \wedge$
$\forall t2 \cdot (t2 \in TRAIN \wedge t2 \in dom(position1) \wedge$
$position1(train) \mapsto position1(t2) \in is\_behind \Rightarrow$
$mm \mapsto position1(t2) \in is\_behind) \wedge$
$(position1(train) \mapsto mm \in is\_behind \vee position1(train) = mm) \wedge$
$mal2 = mal1 \Leftarrow \{train \mapsto mm\}))|mal1 \mapsto mal2\}\}$

th1 : $\forall t1, t2, position1 \cdot (position1 \in TRAIN \nrightarrow TRACK \wedge t1 \in dom(position1) \wedge$
$t2 \in dom(position1) \wedge$
$t1 \neq t2 \Rightarrow$
$compute\_l\_relation(\{t1 \mapsto position1\}); compute\_l\_relation(\{t2 \mapsto position1\}) =$
$compute\_l\_relation(\{t2 \mapsto position1\}); compute\_l\_relation(\{t1 \mapsto position1\}))$

END

Figure 11: Proving the refinement of a set of local operations by a global one

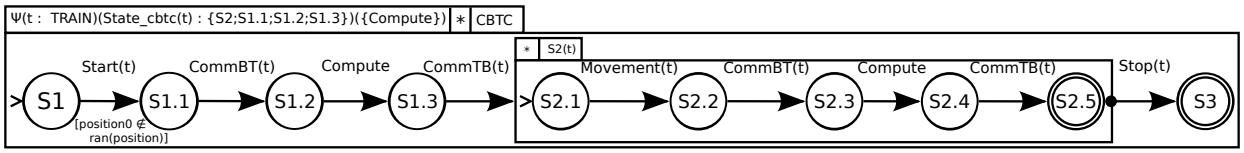

Figure 12: Train System ASTD Specification: Fourth Refinement

operation. The track system will do the *compute* operation. The *stop* operation will be a synchronized operation. Each time a sub-system will execute an operation, a communication operation will immediatly follow. This communication operation will communicate the new value of the variables. For exemple, when the on-board execute the *movement* operation, a communication from the board to the track ($CommBT$) will send the updated value of the *position* variable.

The new specification can be seen in figure 12. This refinement is proved using the general definition of ASTD refinement. The proof obligation given in section 2.2 are proved.

For each variable *var* of the abstract Event-B specification, two variables, on_board_var and $track\_var$ are defined. The "on board" variables can only be modified "on board" events, the "track" variables can only modified in "track" events. For exemple, an $on\_board\_position$ and a $track\_position$ are defined. The $on\_board\_position$ is the variable that will be used and modified in the "on board" events (*movement* and *start*). The specification of the $movement\_act$ event and the communications event can be seen in figure 13.

Event $commBT\_act \mathrel{\widehat{=}}$

any tt

where

gu1 : $tt \in TRAIN$

gu2 : $tt \in dom(on\_board\_position)$

then

act1 : $track\_position(tt) := on\_board\_position(tt)$

end

Event $movement\_act \mathrel{\widehat{=}}$

any tt

where

gu1 : $tt \in TRAIN$

gu2 : $tt \in dom(on\_board\_position)$

gu4 : $tt \in dom(on\_board\_mal)$

gu5 : $tt \in dom(track\_mal)$

gu3 : $on\_board\_mal(tt) = track\_mal(tt)$

then

act1 : $on\_board\_position :\mid (\exists pp \cdot (pp \in TRACK \land$
$on\_board\_position' = on\_board\_position \lhd\!\!\!\!- \{tt \mapsto pp\} \land$
$(on\_board\_position(tt) = on\_board\_mal(tt) \Rightarrow$
$on\_board\_position(tt) = pp) \land$
$(\neg(on\_board\_position(tt) = on\_board\_mal(tt)) \Rightarrow$
$(on\_board\_position(tt) \mapsto pp) \in is\_behind) \land$
$(pp = on\_board\_mal(tt) \lor (pp \mapsto on\_board\_mal(tt)) \in is\_behind)))$

end

Event $commTB\_act \mathrel{\widehat{=}}$

any tt

where

gu1 : $tt \in TRAIN$

gu2 : $tt \in dom(track\_mal)$

then

act1 : $on\_board\_mal(tt) := track\_mal(tt)$

end

END

Figure 13: Train System Event-B Specification : Fourth Refinement

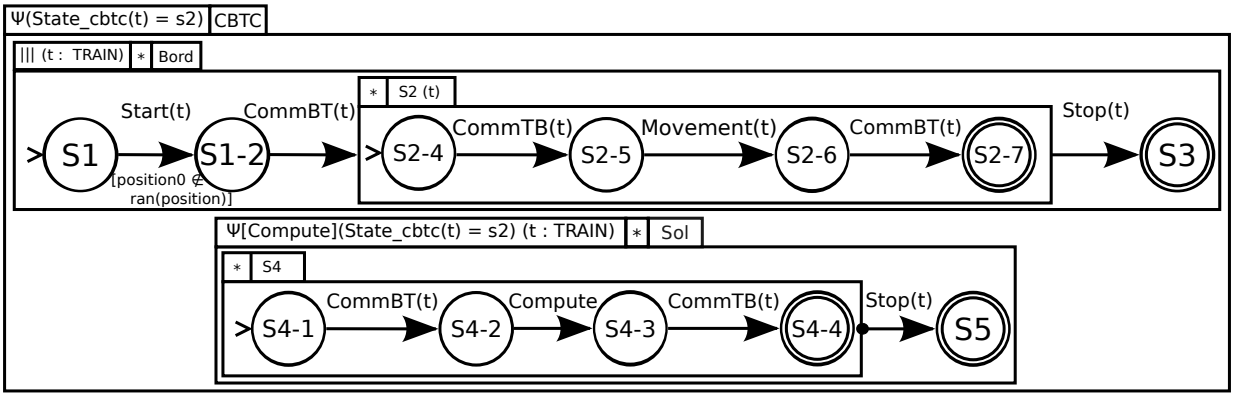

Figure 14: Train System ASTD Specification: Fifth Refinement

2.6 Fifth refinement

In this level the ASTD specification is decomposed into two subsystems. The ASTD specification can be seen in figure 14. Like the step between level 1 of specification and level 2 of specification (see section 2.2), this refinement does not totaly respect the refinement relation defined in [6]. The properties (4), (5) and (6) are proved, and $\alpha(A) = \alpha(C)$. The concrete specification is equivalent to the abstract one. Since the ASTD specifications are equivalent and contain the same operations, the Event-B specification is not modified.

References

[1] J. R. Abrial. *The Event-B Book*. 2007.

[2] Jean-Raymond Abrial. *The B-book: assigning programs to meanings*. Cambridge Univ. Press, Cambridge, 1996.

[3] M. Embe Jiague, M. Frappier, F. Gervais, P. Konopacki, J. Milhau, R. Laleau, and R. St-Denis. Model-driven engineering of functional security policies. In *International Conference on Enterprise Information Systems*, volume 3, pages 374–379, 2010.

[4] Marc Frappier, Frédéric Gervais, Régine Laleau, and Benoît Fraikin. Algebraic State Transition Diagrams. Technical report, Université de Sherbrook, 2008. http://www.dmi.usherb.ca/~frappier/Papers/astd.pdf.

[5] Marc Frappier, Frédéric Gervais, Régine Laleau, Benoît Fraikin, and Richard Saint-Denis. Extending Statecharts with Process Algebra Operators. *Inovation in System Software Engeneering*, Volume 4, Number 3:285–292, 2008.

[6] Marc Frappier, Frédéric Gervais, Régine Laleau, and Jérémy Milhau. Refinement patterns for ASTDs. *Formal Asp. Comput.*, 26(5):919–941, 2014.

[7] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.

[8] Alexei Iliasov, Elena Troubitsyna, Linas Laibinis, Alexander Romanovsky, Kimmo Varpaaniemi, Dubravka Ilic, and Timo Latvala. Supporting Reuse in Event B Development: Modularisation Approach. In *Abstract State Machines, Alloy, B and Z, Second International Conference, ABZ 2010*, volume 5977 of *LNCS*, pages 174–188. Springer-Verlag, February 22-25 2010.

[9] J. Milhau, A. Idani, R. Laleau, M. Labiadh, Y. Ledru, and M. Frappier. Combining UML, ASTD and B for the formal specification of an access control filter. *Innovations in Systems and Software Engineering*, 7(4):303–313, 2011.

[10] A. Pnueli. The Temporal Semantics of Concurrent Programs. *Theoretical Computer Science*, 13:45–60, 1981.