Transformation Heuristics for Formal Requirements Validation by Animation

Atif Mashkoor, Jean-Pierre Jacquot, Jeanine Souquières

Transformation Heuristics for Formal Requirements Validation by Animation ⋆

Atif Mashkoor[1] Jean-Pierre Jacquot[2] Jeanine Souquières[3]

LORIA – DEDALE Team – Nancy Université
Campus scientifique
F-54506, Vandoeuvre-Lès-Nancy, France

Abstract

Safety critical software systems are regulated by stringent certification requirements. The use of formal methods is the part of standard recommendations in particular for higher safety integrity levels. An important issue with formal methods is the problem of the validation of requirements: do they accurately capture the stakeholder needs? While proof tools guarantee the consistency of a specification, they are of little help to check if the specification models the desired behavior. This paper addresses the problem of the validation of Event-B specifications by animation. Once the specifications have been verified using the RODIN platform, they have to be transformed in order to be animated by the Brama animator. We propose transformation heuristics in order to produce a derived animatable specification which may be non-provable, but exhibiting the same behavior as the original specification.

Keywords: Event-B, Formal methods, Formal validation, Animation

1 Introduction

The use of the B method [1] for the development of industrial safety critical systems, involving higher safety integrity level certification and IEC 61508 [12] safety standard, is escalating over the period of time and specially in the transportation domain, such as [6] [4]. Studies have revealed that most of the anomalies, discovered in the development of safety critical systems, belong to requirement and specification phases [16] [11] [15] [9]. The role of validation and verification in the development of safety critical systems thus becomes very significant. While the use of provers guarantees the consistency of the model (verification), it is of little help to check if the specification models the desired behavior (validation).

The role of certification is to warrant that a system or component of a system conforms to its specified requirements and is acceptable for operational use. The certification of software items is notoriously difficult. Due to their immaterial nature, it is often impossible

⋆ This work has been partially supported by the ANR (National Research Agency) in the context of the TACOS project, whose reference number is ANR-06-SETI-017 (http://tacos.loria.fr), and by the Pôle de Compétitivité Alsace/Franche-Comté in the context of the CRISTAL project (http://www.projet-cristal.org).

[1] Email: Atif.Mashkoor@loria.fr

[2] Email: Jean-Pierre.Jacquot@loria.fr

[3] Email: Jeanine.Souquieres@loria.fr

to apply common certification techniques, such as measuring, stressing, or conforming to a standard. One of the common certification practices is to certify the development process. In this context, techniques such as the B method, which ensure the production of a code proven against its specification, are well suited. However, the completeness and quality of the model still remain important questions that can effectively be answered by model validation.

There are several ways to validate a specification: prototyping, structured walkthrough, transformation into a graphical language, animation, and others. All concur to the same goal: to evaluate a system to assess its conformance to its requirements, which later contributes in the demonstration that the system is operational. Each technique focuses on a particular point of view of the specification.

Animation focuses on the behavior of the system [23]. The principle is to simulate an executable version of the requirements model and to visualize the simulation in some form appealing to stakeholders. This point of view is well adapted to the paradigm of events as used in our specification language, Event-B, and to the kind of specified system: the domain model of land transportation. Animators use Finite State Machines to generate a simulation process which can be then observed with the help of UML diagrams, textual interfaces, or graphical animations [19]. With animation, we can check that events are fired in a sequence that follows the protocol we had in mind. When applicable, the computation of values can also be used to check that the state of the model evolves in a way consistent with the desired intentions.

Animation can be used early during the elaboration of the specification: there is no need to wait until it is finished. As a relatively low cost activity, animation can be frequently used during the process to validate important refinement steps. It then provides us with a validation tool consistent with the refinement structure of the specification process. As far as certification is concerned, this property may be of interest due to several reasons. One of them is the fact that problems are detected close to the point where their cause was introduced. This facilitates the understanding of the cause. Another reason is the fact that an unforseen behavior may be associated with a specific refinement. If we see a refinement as a formalization of a requirement, then we have an indication that some interactions between requirements need to be investigated.

The main objective of this paper is to show how to animate an Event-B specification with the animator Brama in order to validate its conformance to its requirements. The animation of the transport domain raised two major points: the specification must be transformed so that the tool can animate it, and the introduction of the values needed to run the animation is a technical, but serious problem as values are often complex, and hard to write in the required mathematical form. We have tackled the first point by defining transformation rules which produce a derived specification which may be non-provable, but which exhibits the same behavior as the initial specification.

Our process starts with a verified Event-B specification which is formally "downgraded" to be animated. This situation raises several questions to be answered:

- Can this be safely assumed that the observations during the animation process provide sound information about the specification?
- Why not design fully formal transformation rules?
- Why not produce at first a specification both verifiable and animatable?

This paper is organized as follows. Section 2 briefly introduces the basic concepts and tools used with the formal specification language, Event-B. Section 3 presents the

limitations of the Brama animator and proposes transformation heuristics which enable the animation of a proved specification, without modifying its behavior. This proposition is then illustrated in section 4 on a real case study, the transportation domain. Section 5 presents some related works. Section 6 concludes the paper with some lessons learned from this experience and some perspectives.

2 Terminology and prerequisites

Event-B [2] is a formal language for modeling and reasoning about large reactive and distributed systems. Event-B is provided with tool support in the form of a platform for writing and proving specifications called RODIN [4] .

2.1 The Event-B formal language

Abstract Specification

An Event-B abstract specification is encapsulated into a MODEL identified by a unique name. The system variables are given in the VARIABLES part. An INVARIANT defines the state space of the variables and their safety properties. Each event in the EVENTS part is a substitution statement. Their semantic is given by the weakest precondition calculus of Dijkstra [10]. An event consists of a guard and a body. When the guards of an event are true, the event can be enabled. When the guards of several events are true, the choice of the triggered event is non-deterministic. In addition, a CONTEXT can be defined to specify static data of sets, constants and their axioms. An Event-B model SEES at least one context. Proof obligations are generated to ensure the consistency of the model, i.e. the preservation of the invariant by the events.

Refinement

A refinement process is used to progress towards implementation. The abstract model is transformed into a more concrete and elaborated model. New variables can be introduced and the old variables can be refined to more concrete ones. This is reflected in the substitutions of the events as well. A WITH clause expresses the link between the parameters of an abstract event, (possibly removed in the refined event) and their concretization. New events may also be introduced in the refinements. These new events should not prevent forever the old ones from being triggered. A VARIANT can be introduced to ensure this property. It consists of a natural number which must decrease each time a new event is fired. Furthermore, one abstract event can be refined by several events, as well as several events can be merged into a single one. Proof obligations ensure that the refined model is consistent, i.e. its INVARIANT is preserved, and the VARIANT is decreased by the new events. Furthermore, they ensure that the refinement is correct, i.e. the refined events do not contradict their abstract counterpart.

2.2 The Brama animator

Brama [21] is an animator for Event-B specifications. It is an Eclipse based plug-in for the Event-B platform RODIN which can be used in two complementary modes. Either Brama can be manually launched from within the RODIN or it can also be connected to a

[4] http://rodin-b-sharp.sourceforge.net

Flash graphical animation through a communication server; it then acts as the engine which controls the graphical effects.

The figure 1 shows the "classic" interface of the animation. On the left hand side, the events of the animated machine appear. They are in one of two states: enabled or disabled, depending upon the evaluation of the guards `TRUE` and `FALSE` respectively. On the right hand side, the actual values of the machine variables are displayed. The buttons can be used to customize the display or to activate specialized value editors.

The basic user action is to click on an enabled event. This triggers three internal steps:

- to pick the values that make the guard true. When several values are possible, the choice is non deterministic;
- to compute the `ACTION` part of the event. Again, if several values are possible, the choice is non deterministic;
- to re-evaluate the guards of all events.

At all steps, Brama checks that the values, either provided by the user or computed by the events, do not break the invariants of the machine or the axioms in the contexts.

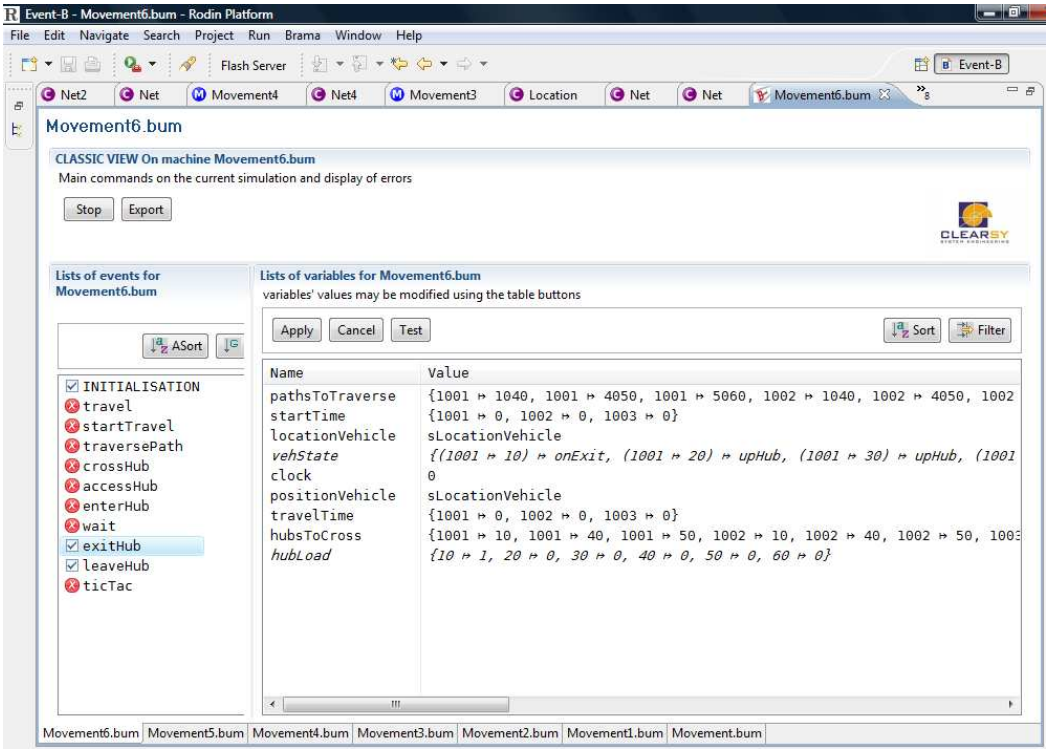

Figure 1. Basic interface of Brama animator for RODIN

In a specification which includes several refinements, each one can be animated independently. This feature has two consequences. The first one is that highly non-deterministic machines which are often found at the initial steps of the specification process may not be animatable, but this does not prevent the animation of further refinements where the non-determinism has been lowered. The second consequence is that the "refine" feature of all events must be turned to `false` even if they do not change [5] .

[5] This RODIN feature simulates a kind of inheritance from the refined machine when events are not modified in refinements.

An animation session begins by setting the values of the constants in the different contexts seen (either directly or transitively) by the animated machine. This is quite a tedious task: Brama accepts only numerical values, the values must be written in a pure mathematical language, and the value editor provided by Brama is a crude one line text field, without copy/paste. Adequate for specification with "small" contexts, these features soon reach their limits.

Then, the user must fire the `INITIALISATION` event, which is, at that time, the only enabled event. After this, the user will play the animation by firing the events until there is no more enabled event, or the system enters to a steady loop, or an error occurs (broken invariant or non computable action typically), or a deviation from the intended behavior is observed. In the last two cases, the specifier must go back to the specification in order to correct it.

3 Transformation heuristics

The animation of a specification has a natural limitation: every expression in the specification must be computable in a reasonable time. So, all quantified expressions should be equivalent to an enumeration over a set (infinity must be avoided, for instance.) Of course, this is exactly the kind of constraint that a specifier does not want to consider! Actually, such concerns should only be introduced when the development process reaches the implementation of the system.

The fact that a specification can be animated is distinct from the fact that the specification is formally correct. Brama does not use information from the proofs conducted to discharge the proof obligations. Like an incorrect program can be run, an incorrect specification can also be animated.

So, we have three propositions, which seem to prevent us from using animation to validate specifications:

(i) a correct specification may not be animatable

(ii) an incorrect specification may be animatable

(iii) most well written specifications are likely to be non animatable

To overcome this apparent contradiction, we propose a pragmatic approach based on controlled transformations of the specification. The transformations are designed with a strong constraint: to replace non computable expressions by computable expressions while guaranteing that the specification keeps the same behavior. We do not require the transformations to maintain the formal correctness of the specification: we do not mind if some proof obligations cannot be discharged.

Since our aim is to validate a specification, we insist that the starting point of the animation job must be a totally correct specification: there would be no point in validating an incorrect specification. The specification is formally "downgraded" to be animated. Incorrect behaviors detected during animation trigger the correction loop: modifications are made in the initial specification, the new or modified proof obligations must be discharged, and the transformations applied again. The whole process of validation with animation is depicted in the figure 2.

The conclusion of the process, i.e., that the specification is valid, depends crucially on the validity of the transformations that are applied. The following sections describe the ones we have found useful in our case study.

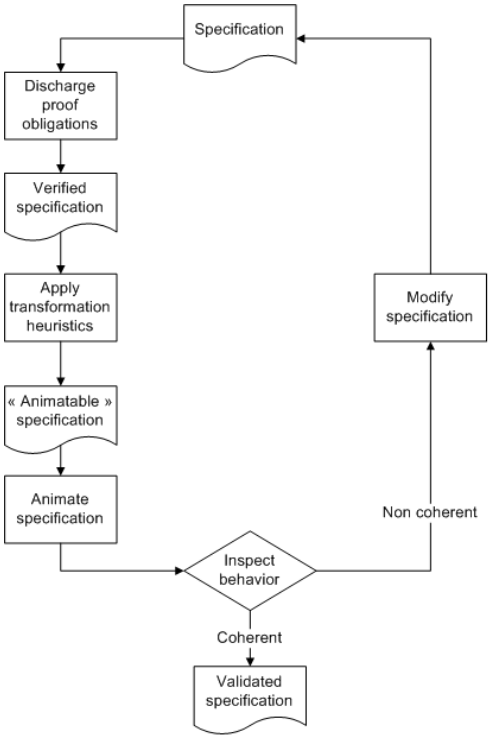

Figure 2. The process of validation by animation

3.1 Some identified limitations of Brama

When trying to animate specifications, we stumbled upon problems which could be arranged in a typology of six typical situations:

- Brama does not support the finite clause in axioms
- Brama must interpret quantifications as iterations
  - Case 1: Brama only operates on finite lists
  - Case 2: The finiteness of list does not assure its animation
  - Case 3: Brama explicitly requires typing information of all those sets over which iteration is performed in an axiom
- Brama has problems with dynamic bindings in substitutions
  - Case 1: Brama does not support dynamic mapping of variables in substitutions
  - Case 2: Brama does not support dynamic function computation in substitutions

For each situation, we have defined a “heuristic” which gives the symptoms and cause of the problem as an error message of Brama, a transformation pattern to apply to the specification, a cautionary explanation of the consequences of the transformation, and a rationale to justify why the behavior is kept same. They are presented in details below.

3.2 The heuristics

3.2.1 Rule 1: Remove the axiom "finite" from the specification

Symptom: Error message about keyword "finite" not being supported.

Pattern: Remove all the instances of axiom "finite" from the specification.

Caution: Removal of axioms "finite" invalidates many well-formedness proof-obligations.

Justification: Axioms like finiteness and non-emptiness can be considered as purely tech-

nical axioms [17] [18]. They do not bring much information into the specified system whose implementation will necessarily be finite, even if it could conceptually be infinite. These technical axioms are required by the inference rules used by the provers. Since all the sets of values will be defined by extension, the animation will work upon necessarily finite values. The behavior is trivially maintained.

3.2.2 Rule 2.1: Specify the finiteness of a quantified domain

Symptom: Error message about dependent variables which do not have an iterator.
Pattern: Limit the range of the list.

Original $n \cdot n \in N \Rightarrow expression(n)$
Transformed $n \cdot n \in min \mathinner{..} max \Rightarrow expression(n)$

Caution: The range must be wide enough so that the values computed during the animation never fall outside it. Some proof obligations may become impossible to discharge (e.g, $n{+}1 \in N$).
Justification: This heuristics is the opposite of Rule 1; the argumentation on the necessary finiteness of the values during animation holds. The major difference with Rule 1 is the necessity to check during the whole animation that the range is always wide enough. If this condition is ensured, then the behavior is the same.

3.2.3 Rule 2.2: Generalize expressions involving complex iterations

Symptom: Error message about the impossibility to build the iterators of the predicate.
Pattern: Take super-set of the expression.

Original $var = \{x \mid \exists n \cdot n \in N1 \land x \in 1 \mathinner{..} n \rightarrow y\}$
Transformed $var \in \mathbb{P}(N \nrightarrow y)$

Caution: Although there is an apparent similarity with the problem dealt with Rule 2.2, the situation is very different: the computation requires two levels of iterations. This transformation loosens the constraints on the values, some maybe essential to the behavior (for instance, the property that all integer between 1 and the length of the sequence belong to the range of the function). Brama cannot ensure anymore that the properties hold. The burden of the check is passed onto the input of the values.
Justification: Since the modified specification accepts more values than the initial specification, it has more behaviors. On the subset of values shared by the specification (that is, those values respecting the constraints left out by the generalization), both specifications must have the same behavior. Two cases must be considered:

- the value is a constant: it does not change during the animation and it keeps its properties,
- the value is a variable: at least one of the proof obligations in the initial specification deals with proving that the result of the computation belongs to the set. Since the initial specification is verified, the values in the modified specification have the same property.

3.2.4 Rule 2.3: Explicitly provide the typing information of all sets used in an axiom

Symptom: Error message about the impossibility to build the iterators of the predicate.
Pattern: Always provide the type of variables.

original $x \cdot expression(x)$
Trandformed $x \cdot x \in X \Rightarrow expression(x)$

Caution: The type provided must be consistent with the type inferred by the provers.

Justification: Brama does not use the information derived by the provers. The provided set is actually redundant. Brama needs it to set up the iteration process. Two cases must be considered:

- if the type is equal to a carrier set, or a subset, the modified expression is just a redundant form of the initial expression,
- if the type is an infinite set, such as $\mathbb{N}$, then Rule 2.1 should also be applied. The same caution and reasoning apply.

3.2.5 Rule 3.1: Avoid dynamic mapping of variables in substitutions

Symptom: Error message: "Default number can not be casted to IMapplet". Brama does not compute sets of tuples in substitutions.
Pattern: Rewrite the substitution to avoid mapping.

Original var := {x, y . x ∈ X ∧ y ∈ Y | x ↦ y}
Transformed var := ({x ∈ X | x} × {y ∈ Y | y})

Justification: The transformation is simply a rewriting of the initial expression as a formula in set algebra. While less readable, it has the same semantics.

3.2.6 Rule 3.2: Avoid dynamic function computation in substitutions

Symptom: Error message: "Related invariant is broken after executing the event". Brama cannot apply a function defined by its graph in a substitution.
Pattern: Rewrite the substitution to avoid function computation.

Original var := {x · x ∈ X | fun(x)}
Transformed var := {ran ({x · x ∈ X | x} ◁ fun)}

Justification: The transformation is simply a rewriting of the initial expression as a fomula in set algebra. While less readable, it has the same semantics.

4 Illustration on a case study

Our work is conducted within the CRISTAL and the TACOS projects. Both projects revolve around the design of a new urban transport system based on shared autonomous vehicles, called `CyCabs` [5]. Many issues must be addressed, such as usability, social acceptation, fleet management, traffic control, and so on. The most important issue, however, is the homologation or certification of such systems. None of the standards which already exist can be applied. A further difficulty comes from the fact that many components of such systems will be software components whose certification is still an open problem.

The case study presented below is based on the preliminary specification of the land transportation domain. As advocated in [13], we must have the best possible understanding of a domain to implement a successful system. The certification issue is also highly dependant on the quality of the domain description. If we think of a domain as a set of invariant properties and defined behaviors (protocols), then the proof that a component does not break the invariants and conforms to the protocol is an essential part of the certification process. Obviously, for the procedure to be correct, we need to ensure that the specification is an adequate model of the real domain.

By transportation systems, we mean the systems which cater the movement of people or goods by vehicles from locations to locations [8].

4.1 Presentation of the case study

In this article, a transportation network, modeled as a directed graph, consisting of vertices (hubs) and edges (paths) is considered. A hub may belong to one of two categories: junctions, which model the intersections or stations, which model the places where passengers and goods can enter and leave vehicles. A movement corresponds to an action by which a vehicle changes its location from one station to another. Since the movement is constrained by the topology of the network, it is thus necessary to introduce the concept of route which models its itinerary. The figure 3 shows a transportation network.

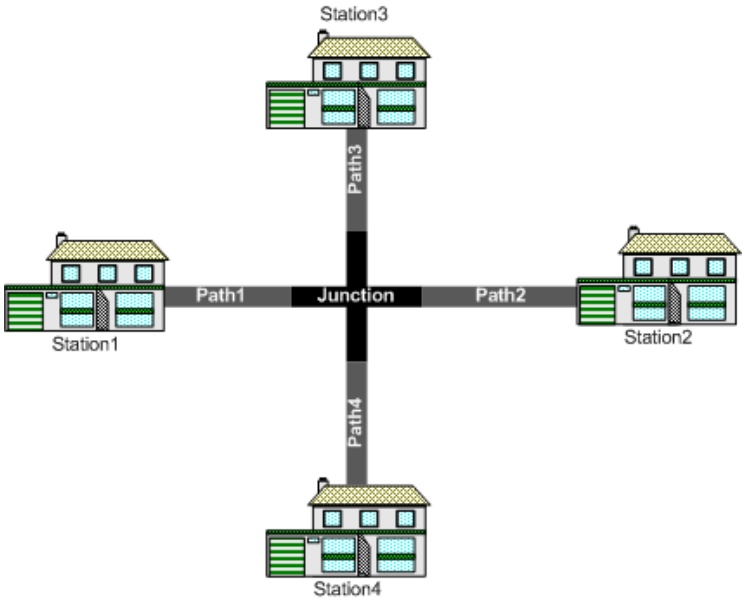

Figure 3. A transportation network

The apparent simplicity of the intuitive description of the domain should not mask the complexity which hides beneath it. In particular, several vehicles circulate simultaneously on a net and their interactions with each other constitute the properties of the global system. Some of the interesting properties of the system are collision avoidance on intersections and determination of the travel time.

To develop the specification, we followed the approach promoted by the B-method. We started from a very abstract definition of transportation and refined it to introduce more and more properties and behaviors. The specification animated here possesses eight machines. The hierarchy of the specification is demonstrated by the figure 4 which shows three level of abstractions and two steps of introduction of time in the model. A complete specification of the model is discussed in [18].

4.2 Stepwise animation of the specification

Each machine in the specification has been animated. We followed the refinement structure: a refinement was not animated until the refined machine was animated and validated. We thus ensured that the correction of an error trickles down the refinement chain.

4.2.1 Machine Movement:

This machine is the most abstract. It contains only one event, travel, which states that vehicles change their location in the network. The behavior is simplistic and does not

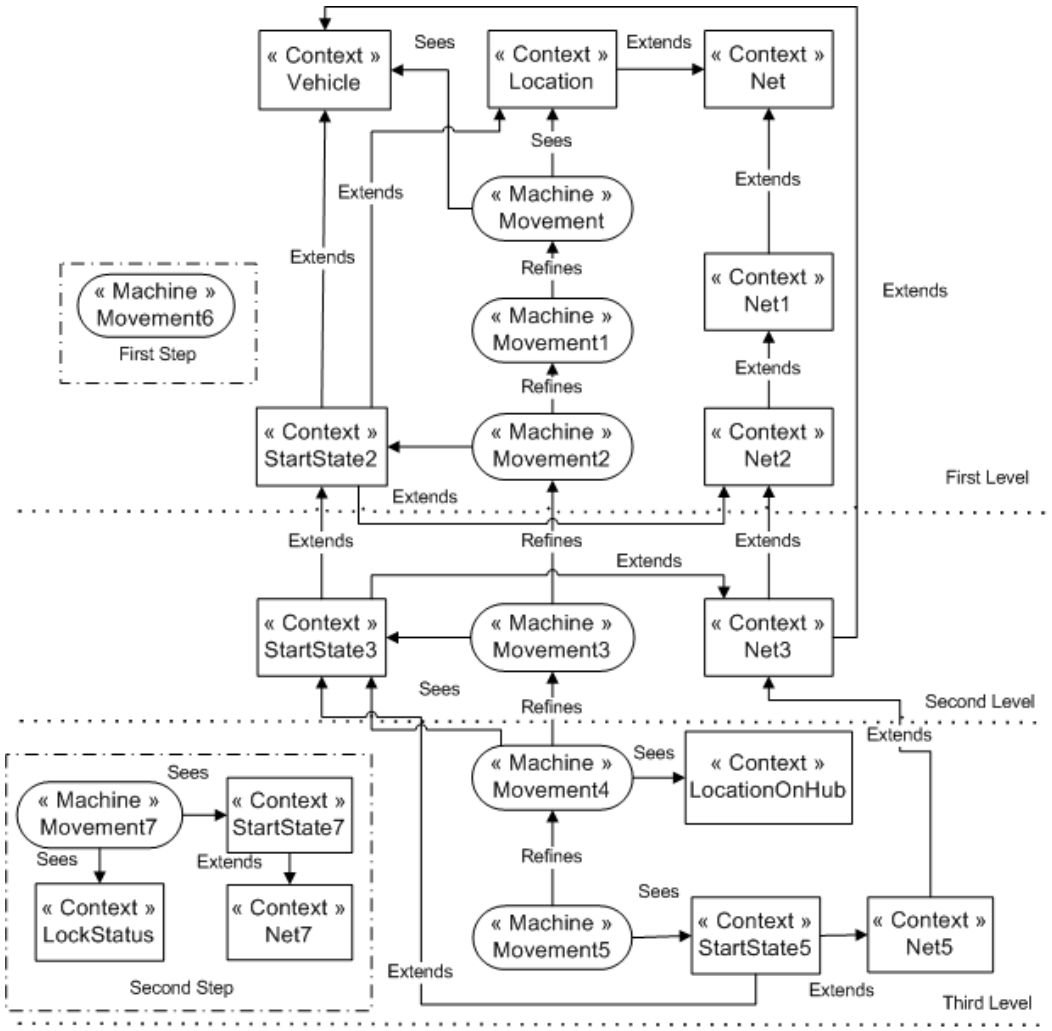

Figure 4. Hierarchy of an Event-B model for the transportation domain

present much interest *per se*. The interesting point lies in the definition of the values in the contexts. In particular, context Net specifies the topology of the network (a directed graph) and the basic properties which make it a transportation network. It is at this level that foundations of the model instance are built.

We had to apply Rule 1 (erasure of "finite" axioms) on Net to animate Movement. As a result, axioms such as:

$$\forall n \cdot n \in Nets \Rightarrow card(obsNetHubs^{-1}[\{n\}]) \geq 2 \wedge card(obsNetConnections^{-1}[\{n\}]) \geq 1$$

which expresses that a transportation network has at least two hubs and one connection, cannot be verified. Its well-formedness proof obligation requires obsNetHubs and obsNetConnections to be finite functions, hence their domains and codomains (Nets, Hubs, and Connections) be finite too.

As Nets, Hubs, and Connections are by essence finite, the justification of Rule 1 shows that the behavior of the transformed specification is same as the Movement.

4.2.2 Machine Movement1:

Movement1 refines the model by specifying that the locations of vehicles are Hubs at the beginning and the end of a travel event. Since this machine introduces only a small refinement in the model and sees the same contexts as seen by Movement, its animation is straightforward.

4.2.3 Machine Movement2:

The refinement Movement2 introduces the notion of routes. A route is a sequence of paths; a path is an edge in the graph between two hubs which are the vertices. The event travel is refined to state that locations are connected by a route.

The set of Routes is introduced in the context Net2. We had to use Rule 2.2 because we defined the notion of sequence as follows:

$seqPaths = \{seq \mid \exists n \cdot n \in N1 \land seq \in 1..n \rightarrowtail paths \land finite(seq) \land card(seq) = n\}$

This axiom was replaced by:

$seqPaths \in \mathbb{P}(N \nrightarrow paths)$

The most important effect of the application of the rule is to invalidate all the proofs, either in Net2 or in Movement2 and its subsequent refinements, which relied on the essential property of sequences:

$\forall s.\ s \in seqPaths \Rightarrow dom(s) = 1 \mathinner{..} card(s)$

As explained by the justification of Rule 2.2, if the values provided in Net2 conform to the essential property, then, the behavior is maintained. The input values in Net2 have been thoroughly checked.

4.2.4 Machine Movement3:

The refinement Movement3 introduces a decomposition of the notion of travel as a sequence of smaller movements: paths traversing and hubs crossing. The important behavior we need to validate is that a vehicle follows exactly the route that was assigned for its travel. Technically, this means that the events crossHub and traversePath must be fired in a strict order.

The ordering of the events is controlled by two variables, pathsToTraverse and hubsToCross, which models the movement of the vehicles along their routes. Their initialization in the startTravel event required us to use Rule 3.1 to transform

$pathsToTraverse := pathsToTraverse \lhd\!\!\!\!- \{path \cdot path \in ran(route) \mid vehicle \mapsto path\}$

into

$pathsToTraverse := pathsToTraverse \lhd\!\!\!\!- (\{vehicle\} \times \{path \cdot path \in ran(route) \mid path\})$

We had also to use Rule 3.2 to transform

$hubsToCross := hubsToCross \lhd\!\!\!\!- \{i \cdot i \in 1 \mathinner{..} card(route) \mid vehicle \mapsto connectionOrigin(route(i))\}$

into

$hubsToCross := hubsToCross \lhd\!\!\!\!- (\{vehicle\} \times ran\ (ran(\{i \cdot i \in 1..card(route) \mid i\} \lhd route) \lhd connectionOrigin))$

The generated proof obligations after the application of Rule 3.1 are easy to discharge since we do no more than rewriting. However the the proof of second transformation stumbles upon showing that route is finite, which can not be done (see Rule 1).

4.2.5 *Machine Movement4:*

Movement4 introduces the concept that the crossing of a hub follows a protocol. The refinement decomposes the crossHub event into four events, accessHub, enterHub, exitHub and leaveHub.

The protocol to cross a hub is now richer. The contexts in the specification are enriched by the definition of states of vehicles with respect to a hub.

An important invariant property which states that a vehicle can be on no more than one hub is introduced at this point:

$\forall\, v \cdot v \in Vehicles \Rightarrow card(\{hub \cdot vehState(v \mapsto h) = onHub \mid hub\}) \leq 1$

It required us to use Rule 2.3 to introduce the type of hub:

$\forall\, v \cdot v \in Vehicles \Rightarrow card(\{hub \cdot hub \in Hubs \wedge vehState(v \mapsto hub) = onHub \mid hub\}) \leq 1$

Since Hub is the infered type, we just introduced a redundant information.

Brama also helped us to find a specification error in one of the events of this machine. The event *exitHub* changes the state of the vehicle once a vehicle exits a hub. In our original specification, we forgot to specify the following guard:

$vehicle \in dom(hubsToCross)$

The error was detected because the omission made the event enabled before the startTravel event was fired. This violates the basic behavior of the system.

4.2.6 *Machine Movement5:*

The refinement Movement5 introduces the idea that only a definite number of vehicles can be simultaneously on a hub. This is an abstract definition of the property of non-collision at a hub. The contexts are enriched by the notion of hub capacity. A variable hubLoad is also added to Movement5.

The second error in our specification found by the animation was in the event *enterHub*. Its guard should state that it is true only if the load of the hub is strictly less than its capacity, but we had writen:

$hubCapacity(h) \geq hubLoad(h)$

Interestingly, this error was not caught during the verification: formally, the expression is correct. Actually, it specifies a behavior which could be admissible in another model.

The correction of the guard was obvious. The corrected specification could be proven without much problem, and the animation now shows the intended behavior: vehicles wait when a hub is full.

4.2.7 *Machine Movement6:*

The refinement Movement6 introduces the notion of time in the model which is then used to calculate the travel time of the vehicle. The animation of this machine was trivial as it does not introduce much new details in the model.

4.2.8 *Machine Movement7:*

Movement7 refines the notion of time in the model and relates it to the execution time of events for vehicles. In the context StartState7, we specify that each vehicle takes a certain amount of time to cross a hub:

$\forall\, v,\ h \cdot v \in Vehicles \wedge h \in Hubs \Rightarrow (\exists\, n \cdot n \in N1 \wedge timeToCross(v \mapsto h) = n)$

This expression required us to use Rule 2.1 to obtain:

$\forall v, h \cdot v \in Vehicles \land h \in Hubs \Rightarrow (\exists n \cdot n \in 1 \mathinner{..} 100 \land timeToCross(v \mapsto h) = n)$

The range of the interval $1 \mathinner{..} 100$ was chosen arbitrarily, but reasonable values with respect to the "unit" of time should be used elsewhere.

The event *wait* of this machine models the behavior where vehicles have to wait outside a hub when it is full. We had to use Rule 2.3 on the guard:

$\{v1 \cdot vehState(v1 \mapsto h) = onHub \land v1 \in dom(echeancier) \mid v1\} \neq \varnothing$

to fix the missing type:

$\{v1 \cdot v1 \in Vehicles \land vehState(v1 \mapsto h) = onHub \land v1 \in dom(echeancier) \mid v1\} \neq \varnothing$

As Vehicles is the inferred type, the transformation is just a redundancy.

The clock was introduced in the preceding machine under the guise of the event ticTac. It was the location of the third error found by the animation. The incorrect version of ticTac is shown by figure 5. The problem comes from the guard grd2, which was used to give a typing information, but which introduces a non-deterministic behavior for tic. It contradicts the guard grd3 which computes deterministically the value of tic. Thus ticTac was never enabled, which revealed an error. The correction was obvious: removal of grd2. Again, the verification process had been happy with the behaviorally inconsistent guards.

ticTac $\widehat{=}$
REFINES
ticTac
ANY
tic
WHERE
grd1 $ran(echeancier) \neq \varnothing \land clock < min(ran(echeancier))$
grd2 $tic \in N1$
grd3 $tic = min(ran(echeancier)) - clock$
THEN
act1 $clock := clock + tic$
END

Figure 5. Event ticTac (faulty version)

5 Related Work

In order to validate models specified in Event-B language, there are two main animators available other than Brama: ProB [7] and B2Express [3].

ProB is an animator and model checker for classical and Event-B specifications. The animation of an Event-B model requires its translation into classical B. To perform model checking and animation on specifications, contexts are supplied with finite values, then a transition system is built based on these values to be animated. The sequence of events, violating the invariants are produced by exploring states of the models helping in the visualization of incoherence of the models at each step.

B2Express is based on the translation of Event-B models into the data models specified in the Express formal data modeling technique [20]. The data models are then instantiated and animated. Following this approach, it becomes possible to trigger Event-B models, which ultimately trigger entity instantiation on the Express side. The Express models are checked, with the help of the ECCO Toolkit [22], by controlling data typing, the uniformity, and local and global constraints.

The advantage of Brama over these tools is that it is commercially available as Eclipse

based plug-in which directly animates the models from within the platform RODIN. The animation process is simple and does not involve complex tasks such as translation in other languages, thus minimizing the chances of errors introduced by translation process. Brama also supports integration of models with Flash tools to provide graphical animation of the models.

6 Conclusion

So far, our experience with animation as a tool to validate a complex specification has been very positive. The best practical indication is the fact that we discovered errors in our initial specification. Although these were minor errors, their existence confirms the point that verification of a specification does not ensure that it models the intended system.

In our approach, we start from a verified specification but we animate an unverified specification. This raises three questions: Can we safely assume that the observations during animation provide us with sound information about the specification? Why not design fully formal transformation rules? Why not produce at first a specification both verifiable and animatable?

Our answer to the first question lies in the rigorous definition of the transformation heuristics. Although they are not formal transformation rules in the strict sense, their minute definitions give the logical arguments for their validity. They can be scrutinized, criticized, and maybe invalidated for certain cases we oversaw. This gives them a strong basis to make us believe in their ability to keep behavioral properties unaltered.

The issue of designing fully automated and formal rules to transform the initial specification into a proven and animatable one is still open. Apart from the fact, we doubt the possibility to achieve such a goal, we do not think it is a good idea for two reasons. The first reason is practical. As Rule 2.1 shows, a good understanding of the domain is necessary to limit iterations to a practical range. Like tests for programs, the point is to be able to explore rapidly the domain to uncover problems so a slow animation would make the process unpractical. The second reason lies with the issue of correcting the errors. Rule 2.2 shows that transformations would entail drastic changes in the specification, maybe even in its structure. The problem of tracing a behavioral error back to the initial specification is likely to become very complex. Our pragmatic approach is consistent both with the concept of animation and the refinement structure of Event-B.

The answer to the third question is of a methodological nature. If we try to write an animatable specification, we must introduce very early in the design process some arbitrary constraints (Rule 2.1 for instance) and we must use convoluted expressions (Rule 3.2 for instance). In both situations, the specifier commits a sin: over specification and esoteric notations. The advice given for programming to keep things simple, general, and readable holds true for specification as well. We corrected more errors during the elaboration of the specification when discharging the proof obligations and careful cross-reading than during the animations. Of course, they were of different nature.

The lack of formal definitions for the transformation heuristics raises the point of the relation between the initial and transformed specifications. The case for the application of rules 3.1 and 3.2 is easy to make: it is a simple rewriting backed up by the definition of the operators. The introduction of a redundant typing information in rule 2.2 is also simple to prove: we can compute the type by using the typing inference rules of Event-B. The real issue arises when we extend (rules 1 and 2.2) or restrict (rule 2.1) the set of values acted upon by the events. Two situations must be considered. If the restriction or extension

concerns values that are defined in contexts and only used in events, the proof is simple. Contexts in Event-B define constants, so we have to prove first that the modified set includes or is included in the initial set, and second that the actual values provided for the animation belong to the initial set. If the modification concerns a set whose values are computed by events, it is not clear whether the related proof obligation can be designed. However a weaker technique than proof can be employed to detect problems, such as introduction of "Monitoring" events in the animatable specification. The guard of such an event would check that some value is outside its range or lack some structural property. The enabling of a monitoring event would signal an error.

The certification of a software item is concerned both by verification and validation activities. Ideally, the former should be fully formal, relying on proofs and formal analysis. The latter deals with an inherently informal element: the requirements. Our work, though rigorous, yet not strictly formal, is consistent with this element. The technique discussed here aims at improving the confidence in the software at earlier stages of development cycle.

We intend to enhance our transportation domain model by incorporating a notion of vehicles moving as a platoon. We have a preliminary specification of this moving mode [14]. An actual question concerns the certification of this mode in order to be operated in a general traffic network. No official procedure exists and it is not even clear what the requirements should be. Our simple domain already shows that there will be trouble at intersections. Current model states that any vehicle intending to enter an intersection may go on as soon as the intersection is free. A refinement of this protocol is needed in the platooning problem where the leader vehicle has to take additional constraints into the consideration, with respect to follower vehicles, while entering, to keep the platoon formation. Several strategies can be brainstormed to solve the problem, but some may have unwanted side effects. We expect animation to help us devise adequate requirements.

The work presented in this article is still in earlier stages and we are not yet able to animate all specifications, for example, the platoon specification mentioned above. This specification uses quite simple kinematics functions to compute a position given a speed and an acceleration. Since Event-B does not support the analytic definition of functions, they are defined by their graph. Although perfect for discharging proofs, yet this representation is inadequate for the animation. In particular, the values needed for the animation are intractably huge and cannot be put in the context. We believe that our approach can be extended to tackle such cases, although it will probably entail major transformations such as the replacement of contexts with "invisible" events used to compute the function during the animation. Future work is then already defined.

References

[1] Abrial, J.-R., "The B Book," Cambridge University Press, 1996.

[2] Abrial, J.-R. and S. Hallerstede, *Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B*, Fundamenta Informaticae **77** (2007), pp. 1–28.

[3] Ait-Sadoune, I. and Y. A. Ameur, *Animating Event-B models by formal data models*, in: *Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, ISoLA 2008*, 2008, pp. 37–55.

[4] Badeau, F. and A. Amelot, *Using B as a high level programming language in an industrial project: Roissy VAL*, in: *Formal Specification and Development in Z and B*, LNCS **3455** (2005), pp. 334–354.

[5] Baille, G., P. Garnier, H. Mathieu and P.-G. Roger, *Le cycab de l'INRIA rhônes-alpes*, Technical Report RT-0229, INRIA – Rhônes-Alpes (1999).

[6] Behm, P., P. Benoit and J. M. Meynadier, *METEOR: A successful application of B in a large project*, in: *Integrated Formal Methods*, LNCS **1708** (1999), pp. 369–387.

[7] Bendisposto, J., M. Leuschel, O. Ligot and M. Samia, *La validation de modèles Event-B avec le plug-in ProB pour Rodin*, Technique et Science Informatiques **27** (2008), pp. 1065–1084.

[8] Bjørner, D., *Development of transportation systems*, in: *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA)*, 2007.

[9] de Lemos, R., A. Saeed and T. Anderson, *On the integration of requirements analysis and safety analysis for safety-critical systems*, Technical report, University of Newcastle upon Tyne (1998).

[10] Dijkstra, E. W., "A Discipline of Programming," Prentice Hall, 1976.

[11] Ellis, A., *Achieving safety in complex control systems.*, in: *Safety Critical Systems Symposium*, 1995.

[12] IEC, *IEC functional safety and IEC 61508: Working draft on functional safety of electrical/electronic/programmable electronic safety-related systems* (2005).

[13] Jackson, M., "Software Requirements & Specifications," Addison-Wesley, 1995.

[14] Lanoix, A., *Event-B specification of a situated multi-agent system: Study of a platoon of vehicles*, in: *2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE)* (2008), pp. 297–304.

[15] Leveson, N., "Safeware: System Safety and Computers," Addison-Wesley Publishing Company, 1995.

[16] Lutz, R., *Analysing software requirements errors in safety-critical embedded systems*, in: *IEEE International Symposium on Requirements Engineering*, 1993.

[17] Mashkoor, A., J.-P. Jacquot and J. Souquières, *Domain Modeling with Event-B: An Experience with Transportation Domain*, Research Report INRIA-00326253, LORIA (2008), http://hal.inria.fr/inria-00326253/en.

[18] Mashkoor, A., J.-P. Jacquot and J. Souquières, *B Evénementiel pour la Modélisation du Domaine: Application au Transport*, in: *Approches Formelles dans l'Assistance au Développement de Logiciels - AFADL'09*, 2009.

[19] Ponsard, C., P. Massonet, J. Molderez and G. Dallons, *Formal requirements modelling and early verification and validation of critical systems*, in: *Approches Formelles dans l'Assistance au Développement de Logiciels - AFADL'07*, 2007.

[20] Schenck, D. A. and P. R. Wilson, "Information Modeling the EXPRESS Way," Oxford University Press, 1993.

[21] Servat, T., *BRAMA: A New Graphic Animation Tool for B Models*, in: *B 2007: Formal Specification and Development in B* (2006), pp. 274–276.

[22] Staub, M., G. Maier, "ECCO Toolkit, An environnement for the evaluation of EXPRESS models and the development of STEP based IT applications," (1997), user manual.

[23] Van, H. T., A. van Lamsweerde, P. Massonet and C. Ponsard, *Goal-oriented requirements animation*, in: *RE '04: Proceedings of the Requirements Engineering Conference, 12th IEEE International* (2004), pp. 218–228.