Scaling Up with Event-B: A Case Study

Faqing yang, Jean-Pierre Jacquot

Scaling up with Event-B: A Case Study ⋆

Faqing Yang and Jean-Pierre Jacquot

LORIA – DEDALE Team – Nancy Université
Vandoeuvre-Lès-Nancy, France
{firstname.lastname}@loria.fr

Abstract. Ability to scale up from toy examples to real life problems is a crucial issue for formal methods. Formalizing a algorithm used in vehicle automation (platooning control) in a certification perspective, we had the opportunity to study the scaling up when going from a (toy) model in 1D to a (more realistic) model in 2D. The formalism, Event-B, belongs to the family of mathematical state based methods. Increase was quantitative: 3 times more events and 4 times more proofs; and qualitative: trigonometric functions and integrals are used. Edition and verification of the specification scale up well. The crucial part of the work was the adaptation of the mathematical and physical model through standard heuristics. The validation of temporal properties and behaviors do not scale up so well. Analysis of the difficulties suggests improvements in both tool support and formalism.

1 Introduction

This paper relates our experience with the specification of a realistic algorithm for the control of autonomous vehicles. The problem to solve has interesting characteristics:

- the development should lead to a certified product (a component for a car moving in the public space),
- the physical and mathematical model uses common mathematical notions such as trigonometric, kinematics, integrals, and so on,
- there is an existing empirical solution.

The problem is known as *platooning:* autonomous vehicles that move as virtual trains. All vehicles follow the virtual track defined by the first vehicle while keeping to a minimum the distance between them. The issue is then to guarantee that the control algorithm is safe, i.e., that vehicles can never collide.

We chose Event-B because the concepts of proof obligation and formal refinement are well fitted for the task of guaranteeing an implementation. However, several points needed to be assessed. Could a non-functional property such as non-collision be specified in Event-B? Is the support environment, particularly the provers, strong enough for such a problem? Can we model adequately a system which contains continuous functions, real numbers, or geometric relationships, in a framework which is based on discrete sets and integers?

⋆ Work partially supported by ANR under project ANR-06-SETI-017 TACOS (http://tacos.loria.fr), and by Pôle de Compétitivité Alsace/Franche-Comté under CRISTAL project (http://www.projet-cristal.net).

A first specification was written on a simplified version of the problem. The model considered only a linear track (1D) and the control was only aimed at keeping some ideal distance while avoiding collisions. While simplistic, this model was important on three respects: it allowed us to identified the "hard" parts, it prototyped the properties of interest, and it provided us with a neat structure for the development.

The next specification considered the platooning problem from a realistic point of view. Vehicles are now moving on a plane, the leader of the platoon is not constrained to a predefined track, the properties of interest are now the non-collision and the distance form the virtual track drawn by the leading vehicle.

The changes between a model in 1D to a model in 2D do not seem that big. Instead of one value, the control law must now compute two values: linear acceleration and derivative of the curvature. Furthermore, the system is assumed to stay within the boundaries which guarantee that the lateral and longitudinal controls can be modeled and computed independently.

However, working on a plane introduces notions such as trigonometric functions, curvature, and so on. While this means a modest increase in complexity for a mathematically literate person, those new concepts introduce genuine difficulties for the specifier; e.g, how to model a sine function when co-domains are restricted to integers?

The paper discusses some scaling-up issues when going from a 1D model to a 2D model. We could solve some, most importantly the consistency proofs and the adaptation of the mathematical model. Other issues can be solved but at a high cost, proving global temporal properties is among them. Last, some are yet beyond our reach because the tools cannot deal with the complexity; animation falls in this case for instance.

The paper is structured as follows: Section 2 introduces the notation and semantics for Event-B; Section 3 describes the platoon problem and the model used; Sections 4, 5, 6, 7 and 8 discuss the different aspects of scaling up with Event-B: mathematics, specification structure, temporal properties, tools and process; finally, Section 9 concludes.

2 Event-B Language

Event-B [18,2] is an evolution of the classic B method [1]. Designed for modeling the environment where a piece of software developed with the B-method must execute, Event-B proved to be a good formalism for specifying and reasoning about systems such as concurrent systems, reactive systems, or complex algorithms. Event-B is a state based specification technique. It embodies a process: formal refinement.

Formal Model. A formal model consists of a *state* and *events*. A state is a set of variables constrained by invariants. Values associated to variables are either symbols, integers, or set-theoretic constructions upon those (powersets, relations, functions, etc.). Invariants are expressed as formulae in first-order predicate calculus. Events are guarded generalized substitutions. Guards are formulae on the state and substitutions apply simultaneously on a subset of state's variables.

The semantics of a model is given by a few rules. Substitutions use the weakest precondition calculus of Dijkstra [8]; events must keep the invariant; when several guards are true, the choice of the event to fire is non-deterministic; there must exist a computable initial state. The intuitive behavior of a specification is easy to explain: first,

the *INITIALISATION* event is fired, then a cycle begins where: all guards are evaluated, one event is picked among those with a "true" guard, and its substitutions are executed. The cycle ends when no guard is true, which means either that the system has reached a terminal state, or that the system is deadlocked. Infinite cycles are also possible, which could be the correct behavior or indicate a reachability problem for a terminal state.

The formal semantic rules are implemented as *proof obligations.* To *verify* a model, that is, to assess its consistency, we need to discharge all the proof obligations.

Refinement. Event-B allows one to express that a model is a refinement of another, more abstract model. Refinement consists in introducing new variables. An abstraction invariant relates the new variables to the abstract variables. Events from the abstract model can be kept untouched in the refinement, or can be rewritten using the new variables. New events can be introduced too. In practice, it is often useful to think in term of *reification* of variables and of *decomposition* of an event into several smaller ones.

The semantics of refinement is given by proof obligations. Proving a refinement correct amounts to prove that concrete events maintain the invariant of the abstract model, the abstraction invariant, and do not prevent abstract events to be triggered.

The syntactic structure of the language was designed so that the proof obligations can be easily generated and broken into small formulae. The Rodin platform [20] provides the practical framework to carry out modeling in Event-B. It seamlessly integrates modeling and proving, and provides mechanisms for extension and configuration so that it can be tailored to different application domains or development methods.

3 The Platooning Problem

3.1 Platoons

Research on urban mobility systems based on fleets of small electric vehicles stresses the importance of a new moving mode: platooning. A platoon is defined as a convoy of autonomous vehicles which follow exactly the same path and which are spaced at very close distance one from the other.

In this work, we consider platoons formed by a *leader* vehicle and *followers*. Leaders and followers have different control laws. We specify only the follower control law. Its aim is to keep as close as possible to the preceding vehicle while following a virtual ideal track without colliding. We use a model of vehicle where the control can be decomposed into longitudinal (distance with preceding vehicle) and lateral laws. We assume operating conditions such that the two controls can be set independently [7].

There are numerous strategies to form and maintain platoons, characterized by their degree of centralization and the volume of communication. We specify a minimal strategy: no central control and no communication between vehicles other than perception, i.e., a vehicle can sense a few information from the preceding vehicle (distance, speed, etc.). The virtual track is set by the leader. The control is local to each vehicle, based on current state and perceptions. This strategy may not be the most efficient but it is very robust. In particular, it can be used as a fall-back in case of failure in a system using more sophisticated algorithms. Hence the need to guarantee its correctness.

Within this problem setting, platoons can be considered as situated multi-agent systems (MAS) which evolve following the Influence/Reaction model [10,9]. Development

of the specification follows a stepwise refinement process based on this model: (i) driving systems perceive, (ii) decisions are taken, and (iii) physical vehicles move.

3.2 Research Goal and System Hypotheses

We aim at modeling formally a pragmatic strategy known as *Daviet-Parent algorithm* [7] in order to prove that implementations enjoy certain properties [12] such as: (i) the model is sound bound-wise, (ii) no collision occurs between the vehicles, (iii) no unhooking occurs, and (iv) no oscillation occurs.

Presently, we focus on two essential safety properties: no collision within a platoon occurs[1] and the soundness is maintained.

Our model is based on the following system hypotheses:

- we consider a set of $N(\geq 2)$ vehicles forming a linear platoon,
- motion of vehicles is limited by fixed bounds on velocity, acceleration, curvature and derivative of curvature,
- we consider forward-only motions on a non self-intersecting track,
- we suppose that the frequency of the control algorithm is the same for all vehicles, so they can be modeled as synchronized,
- sensors are perfect and their accuracy is such that the velocity of the previous vehicle can be precisely known,
- actuators of the engine are perfect.

The hypotheses are strong but not that far from reality considering (1) we are not modeling fault, fault-tolerance, or such matters, (2) near perfect abstract sensors or actuators can be built from merging results of several concrete ones.

3.3 State of Vehicles

In the 1D model, vehicles move on a linear track, equivalent to a rail. The state of the i^{th} vehicle at time t is the pair $(xpos_i(t), speed_i(t))$, where $xpos_i$ represents position on the track and $speed_i$ represents the velocity. Control consists in setting of an acceleration to modulate speed. The behavior law is represented by (1) extracted from [24], where $MaxSpeed$ is the maximum velocity, $accel_i$ is the acceleration, and Δt is the time increment:

$$\begin{cases} n\_speed = speed_i(t) + accel_i(t).\Delta t & \\ xpos_i(t+\Delta t) = \begin{cases} xpos_i(t) + MaxSpeed.\Delta t & \text{if } n\_speed > MaxSpeed \\ xpos_i(t) - \frac{speed_i(t)^2}{2.accel_i(t)} & \text{if } n\_speed < 0 \\ \left(\begin{array}{l} xpos_i(t) + speed_i(t).\Delta t \\ + \frac{accel_i(t).\Delta t^2}{2} \end{array} \right) & \text{otherwise} \end{cases} & \\ speed_i(t+\Delta t) = \begin{cases} MaxSpeed & \text{if } n\_speed > MaxSpeed \\ 0 & \text{if } n\_speed < 0 \\ n\_speed & \text{otherwise .} \end{cases} & \end{cases} \tag{1}$$

[1] Collisions between platoons or between a vehicle and an obstacle should of course be considered in a real system. First kind should be taken care by the control law of leaders, second kind is dealt with by lower level emergency systems. Both are outside the scope of this work.

The acceleration $accel_i$ is chosen according to the current state of the i^{th} vehicle and the values sensed on the preceding vehicle.

In the 2D model, vehicles move on a plane. The vehicle state η must model its position, represented by cartesian coordinates (x,y), and its attitude, represented by the orientation θ of vehicle's axis with respect to x-axis. The behavior law now contains a velocity v and a trajectory's curvature κ which are controlled by application of a linear acceleration a and a derivative of the curvature χ. When a control (a,χ) is applied to a state $(x_0,y_0,\theta_0,v_0,\kappa_0)$ at time t for a period Δt, the new state at time $t+\Delta t$ becomes:

$$\begin{cases} x = x_0 + \cos\theta_0 F_C(\Delta t, v_0, \kappa_0, a, \chi) - \sin\theta_0 F_S(\Delta t, v_0, \kappa_0, a, \chi) \\ y = y_0 + \cos\theta_0 F_S(\Delta t, v_0, \kappa_0, a, \chi) - \sin\theta_0 F_C(\Delta t, v_0, \kappa_0, a, \chi) \\ \theta = \theta_0 + v_0\kappa_0\Delta t + (a\kappa_0 + v_0\chi)\frac{\Delta t^2}{2} + a\chi\frac{\Delta t^3}{3} \\ v = v_0 + a\Delta t \\ \kappa = \kappa_0 + \chi\Delta t \end{cases} \tag{2}$$

where $F_C(\Delta t, v_0, \kappa_0, a, \chi) = \int_0^{\Delta t}(v_0+at)\cos(v_0\kappa_0 t + (a\kappa_0+v_0\chi)t^2/2 + a\chi t^3/3)dt$ and $F_S(\Delta t, v_0, \kappa_0, a, \chi) = \int_0^{\Delta t}(v_0+at)\sin(v_0\kappa_0 t + (a\kappa_0+v_0\chi)t^2/2 + a\chi t^3/3)dt$.

At this mathematical level, the increase in complexity is noticeable but not dramatic: most of it boils down to the expansion of standard geometric formulae.

4 Scaling up with Event-B: Mathematics

4.1 1D Model Adaptation

The formulae in the 1D model are basic arithmetic expressions; they contain no special mathematical functions. The values of $xpos_i$, $speed_i$ and $accel_i$ can easily be modeled as integer numbers. It suffices to choose a system of units small enough to reach the accuracy needed in practice. Hence, they can be expressed straight away in Event-B.

4.2 2D Model Adaptation

By contrast, a simple look at the 2D model shows that we need to transform the model so it can be expressed in Event-B. The most obvious "problems" are the sine and cosine functions (meaningless on Integers) and the integral in F_C and F_S expressions.

The Discretization Issue. The heart of the difficulty lies in the discretization of continuous kinematic values such as position, speed or acceleration. The question is then: Why not use continuous values? We are not ready to answer positively for two reasons.

First reason is practical. Current provers within the B world consider only integer numbers. Even with these "simple" numbers, proofs are often complex and intricate. It is not clear that provers doing a good job with real numbers will be available soon.

Second reason is deeper. Software systems are inherently discrete. Because of numerous latencies in the autonomous car (sensing data, computing controls, driving actuators), the control system will operate at a rather slow frequency. So, the actual system will run as if time is discrete.

The B formal method aims at producing code which is proven to maintain functional invariants. So, we need to introduce the discretization at some point. Our position is that we must introduce this fundamental feature early in the models: as soon as we need "continuous" values in the specification. Of course, we must then develop techniques and strategies to take care of this feature.

Although reasonably simple from a mathematician point of view, the 2D model cannot be translated directly in Event-B. We need to "refine" it. We used three heuristics.

(1) Free Physical Units. In Event-B, the easiest representation of continuous kinematic values such as position, velocity, acceleration is integer numbers. By keeping the physical units unspecified but homogeneous (e.g., the unit of velocity is equal to unit of distance divided by unit of time.), we can adapt the representation to the desired accuracy of the computations. Distances can be millimeters as well as meters, and times can be milliseconds as well as seconds.

(2) Approximate Mathematical Functions. The restriction to Integers of the ranges of sine or cosine is a three value set: not very interesting. To solve this problem, we introduce a special dimensionless constant μ and we consider $\mu\cos\theta$ and $\mu\sin\theta$ instead of $\cos\theta$ and $\sin\theta$. We do the same with F_C and F_S and consider μF_C and μF_S. By choosing a μ with a big value, expressions can be reasonably coded with integers.

Event-B provers know about standard rules of arithmetic but ignore trigonometric or general calculus rules. In order to use the provers, we use Taylor series and identities to transform expressions into arithmetic approximations.

Last, the vehicle state $\eta = (x, y, \theta, v, \kappa)$ is represented by a 6-tuple $(x, y, \gamma^\theta, \sigma^\theta, v, \kappa)$. The values of x, y, v and κ are integers; the units must be taken small enough to obtain a good accuracy. The values of γ^θ and σ^θ are also integers which respectively represent $\mu\cos\theta$ and $\mu\sin\theta$. We define a carrier set POINT to denote the set of all possible vehicle states. The approximate, but accurate, model of 2D platooning is then:

$$\begin{cases} x = x_0 + (\gamma_0^\theta \tilde{F_C}(\Delta t, v_0, \kappa_0, a, \chi) - \sigma_0^\theta \tilde{F_S}(\Delta t, v_0, \kappa_0, a, \chi))/\mu^2 \\ y = y_0 + (\gamma_0^\theta \tilde{F_S}(\Delta t, v_0, \kappa_0, a, \chi) + \sigma_0^\theta \tilde{F_C}(\Delta t, v_0, \kappa_0, a, \chi))/\mu^2 \\ \gamma^\theta = (\mu_C \gamma_0^\theta - \mu_S \sigma_0^\theta)/\mu \\ \sigma^\theta = (\mu_C \sigma_0^\theta - \mu_S \gamma_0^\theta)/\mu \\ v = v_0 + a\Delta t \\ \kappa = \kappa_0 + \chi\Delta t \end{cases} \tag{3}$$

where $\mu_C = \mu - \beta^2/(2\mu)$, $\mu_S = \beta - \beta^3/(6\mu^2)$ with $\beta = v_0\kappa_0\Delta t + (a\kappa_0 + v_0\chi)\Delta t^2/2 + a\chi\Delta t^3/3$, $\tilde{F_C}(\Delta t, v_0, \kappa_0, a, \chi)$ and $\tilde{F_S}(\Delta t, v_0, \kappa_0, a, \chi)$ expanded as Taylor series.

Event-B translation is straightforward but yields overly long expressions.

(3) Check and Rewrite Mathematical Formulae for Provability. Many properties of formulae on real numbers can be safely assumed when we restrict their use to integer numbers, but not all. Consider the true equality with real numbers: $a*(b/c) = (a*b)/c$. Its equivalent with natural numbers is $a*(b\div c) = (a*b)\div c$. Unfortunately, this

equality is not true anymore (hint: $\div$ denotes the integer quotient). So any proof which relies on the equality cannot be discharged anymore.

In the initial 1D platooning model, we found two reviewed goals that were instances of the above example. The "obvious" formula below, straight translation of the 1D mathematical model, introduces non provable proof obligations.

```
xpos0+MAX_SPEED−(((MAX_SPEED−speed0)∗(MAX_SPEED−speed0))/(2∗accel0))
```

We proved the model by avoiding the problem with the equivalent expression:

```
xpos0 + ((MAX_SPEED−speed0)∗(MAX_SPEED−speed0))/(2∗accel0) +
speed0∗((MAX_SPEED−speed0)/accel0) +
MAX_SPEED∗(1−(MAX_SPEED−speed0)/accel0)
```

5 Scaling up with Event-B: Specification Structure

An important question when we started the 2D modeling was: can we keep the same development structure as for 1D modeling? We had two reasons. First, a great deal of effort had been put into it so it is intelligible, consistent with the general MAS model, then easy to validate. Second, we can expect proofs structures (and even whole proofs) too to be similar if developments are similar.

We have been able to keep the exact same structure of the development. The same refinements with the same rationales are present in both specifications. In fact, we used the development of 1D specification as a "road-map" for development of the new model.

The structure of the specification [12] consists of an abstract machine Platoon and four refinements. Each development introduces a clearly identified concept. Platoon sets the "vocabulary" and the safety property of interest. Platoon_1 splits the platoon's movement into each vehicles' movements. Platoon_2 implements the physical reaction laws. Platoon_3 introduces the decision step. Platoon_4 introduces the perception step and implements the decision laws.

5.1 Decomposition of Events

During the refinement, the number of events increased much more steeply. A simple pattern explains this explosion.

Both specifications implement the reaction laws in machine Platoon_2 and the decision laws in machine Platoon_4. We need to decompose the abstract events move1 and move in machine Platoon_1, and the abstract events decide1 and decide in machine Platoon_3 into more concrete ones. Let us consider the decomposition of event move.

The mathematical model (1) in 1D indicates that three cases must be considered when computing a new state. This is due to the fact that *speed* is bounded. In Event-B, conditional definitions are expressed by the use of guards. This means that the move event must be decomposed into three events, one for each situation (*speed* reaching lower bound, reaching upper bound, or within bounds.)

All events where *speed* is a parameter are decomposed following the analysis exemplified in Table 1.

Table 1. Decomposition of move event in the 1D model

n_speed_i	< 0	$\in 0..MAX\_SPEED$	$> MAX\_SPEED$
move	*move_reduce*	*move_normal*	*move_max*

In the 2D model, we have to consider two bounded parameters: speed and curvature.

$$\begin{cases} n\_speed_i = speed_i + accel_i.\Delta t \\ n\_\kappa_i \quad\quad = \kappa_i + \chi_i.\Delta t \quad . \end{cases} \tag{4}$$

The analysis must then take into account the combination of three cases for n_speed_i and three cases for n_κ_i. So, events are refined into nine following the pattern of Table 2.

Table 2. Decomposition of move event in the 2D model

$n\_speed_i \setminus n\_\kappa_i$	$< -MAX\_\kappa$	$\in -MAX\_\kappa..MAX\_\kappa$	$> MAX\_\kappa$
< 0	*move_vmin_κmin*	*move_vmin_κ*	*move_vmin_κmax*
$\in 0..MAX\_SPEED$	*move_v_κmin*	*move_v_κ*	*move_v_κmax*
$> MAX\_SPEED$	*move_vmax_κmin*	*move_vmax_κ*	*move_vmax_κmax*

5.2 Statistics of the Specifications

The multiplication of events depicted above happened a few times. It should be noted that other refinement strategies for the abstract event move could have been chosen. For instance, we could have kept the refined event unique, but at the expense of very complex guards. Our trade-off lengthens the specification text and increases the number of proof obligations but each proof is much simpler.

Table 3 shows the increase in complexity when passing from the 1D initial model, to the 1D revised model using the technique presented in Sect. 4.2 and augmented with deadlock-freeness, to the 2D model.

Introducing a safety property such as deadlock-freeness has little impact on complexity. While the number of variables roughly doubled when going 2D, all other measures varied by a four-fold increase. Interestingly, the ratio between manually and automatically discharged proof obligations increases just a little: most of the new proof obligations are simple ones. The most important increase is the number of theorems. They are used to ease the proofs by introducing only once standard mathematical properties. This is a consequence of the introduction of more complex arithmetic expressions in the 2D model.

Table 3. Statistics of the specifications

	1D initial model	1D revised model	2D model
Sets	0	0	1
Constants	15	15	50
Axioms	27	27	86
Variables (last refinement)	10	10	16
Invariants	16	17	29
Events (last refinement)	15	15	39
Guards (last refinement)	81	81	354
Theorems	1	4	46
Variants	3	3	3
Automatic POs	187	196	743
Manual POs	29	36	177
Reviewed POs	4	1	0
Undischarged POs	0	0	0
Total POs	220	233	920

6 Scaling up with Event-B: Temporal Properties

Temporal properties can be classified into two categories. Safety properties specify that nothing bad will happen. Liveness properties specify that something good will eventually happen [11]. Our model must guarantee a safety property: deadlock-freeness.

This safety property was introduced after a long period of perplexity where we were baffled by observing collisions in our programmed simulations of the exact same model that was verified, i.e., where we had proven that firing any event kept a strictly positive distance between vehicles as invariant [12]. The mystery was lifted when we realized that if a moving vehicle cannot react, i.e., its control system is deadlocked, then it will likely collide with something.

Deadlock-freeness is not well integrated into the Event-B framework. As for many other temporal properties, we can use "tricks". For deadlock-freeness, we build the disjunction of the guards of all events other than *INITIALISATION* and we prove it is a theorem. So, we can be sure that one event at least can always be fired.

The trick works well on small models but does not scale up. In the 1D specification, the deadlock-freeness theorem in machine Platoon_2 is a formula with around 42 lines (7 events times 6 lines per guard). It would be around 390 lines long in the last refinement of the 2D specification (39 events times 10 lines per guard). Two problems arise then. One concerns the management of a proof of a disjunction of 39 cases. Rodin proof explorer is well thought out and we are quite confident that, with time and patience, we could discharge the proof. The second concerns the construction of the formula. Right now, we must rely on a manual cut and paste procedure. Needless to say, the probability of introducing a non obviously detectable error is too high for the result to be trusted. Clearly, we need a tool to build automatically this formula.

7 Scaling up with Event-B: Tools

Formal methods depend heavily on automated support. They require long, intricate, and generally tedious chains of reasoning to discharge or establish properties, even trivial ones. This is the nature of formal proof systems. Effective tools are not a "nice addition" to a formal method, but a key factor for its deployment. Event-B is supported by Rodin, a framework which integrates gracefully tools to edit, verify and validate models.

7.1 Edition and Verification

The increase in size shown in Table 3 did not pose any problems to Rodin editors, either the native structural editor or pluggable text-editors such as Camille [5]. Likewise, visualization tools such as the LaTeX generator or the pretty-printer were up to the task.

Provers were also able to deal with the increase in complexity. In fact, the general strategy of breaking a verification proof into several smaller proof obligations as used by B spreads the increase in complexity on much more proof obligations, but each one remains reasonably simple. Some proofs were more complex because the model uses more complex formulae, not because it is bigger.

7.2 Animation Plug-in for Validation

Animation is a technique to *execute* specifications. Thus, we can play, experiment and observe the behavior of models. Several tools support the technique [4,13,26,21,23]. The principle of animating an Event-B specification is a simple three-step process:

1. the user gives values to the constants and carrier sets in the contexts,
2. the *INITIALISATION* event is fired to set the system in its initial state,
3. the animator enters a loop:
   (a) compute the guard of all events, enable those for which the guard is true. When events are parameterized, pick one value, if any, which makes the guard true,
   (b) the specifier fires one of the enabled event; the substitutions are computed,
   (c) check if the invariants still hold [optional].

The computation of the invariant is superfluous when the animated specification has been fully proven. However, it is a very valuable feature when animation is used on unproven specifications, in particular to check potential candidate for invariant formulae.

It may sound strange to use anecdotal observations in a context where mathematical proofs are pivotal to the method. However, we firmly believe that such semi-formal activities are useful, and even sometimes necessary, for three reasons.

The first reason relates to the notion of *validation*. Proofs show that a particular text is logically consistent and that the last model in a sequence of refinements is a correct concretization of the initial model. However, proofs do not tell if a model is an adequate description of the desired behavior of the system. Animation exhibits behavior.

The second reason concerns temporal properties. Not all properties can be expressed in Event-B. Animation can then be used to "test" the specification for certain properties.

We can set up scenarios and look if the system goes only through safe states. Like tests, animation does not prove correctness but shows errors.

The last reason is that animation is a good, practical, tool to get deep insights on complex specifications. Actually, animation was mainly invented for this reason [3].

We used intensively animation to understand the collision problem in the 1D platooning specification. In particular, animation helped us to understand which values lead to deadlocks. From those, we could abstract to general configurations, and then relate to parts of the deadlock-freeness theorem.

We also found out, in another work [16], that animation could be a reasonably cheap way to get a correct refinement of an abstract fact into a complex behavior. Animation helped to define the guards and the explicit coding of causal order required by Event-B.

7.3 Breaking Animators

The positive experience with 1D induced us to use animation early in the 2D model development. Unfortunately, animators failed us even on the first refinement.

We tried with two different animators, Brama[2] and ProB [14]. In both cases, the notion of POINT (3) was the first visible obstruction. We needed to code it, either crudely as integers with Brama or more abstractedly as symbols with ProB. Either way, a list should be provided by hand. This is not realistic and even meaningful.

One way to get rid of POINTs is to refine them as their coordinates. Each coordinate is an integer function which could be individually managed but six of them create too complex a space. Brama seems to enumerate values, ProB uses more sophisticated constraint solving techniques; both strategies fail on the 6-dimensions space.

This can be explained by two important features of Event-B: non-determinism and definition of values by their properties. Animators are then more oriented toward "picking" values rather than "computing" values. This orientation is fine most of the time, but it should not be exclusive. Sometimes, even in abstract specifications, we know some expressions are deterministic computations: kinematic functions are a good example. In those cases, we would appreciate to be able to tell this to the animator.

In the transformations we developed to make specifications "animatable" [17], we have some ad-hoc heuristics which force a computation. Their major drawback is to make the text of the guards and the substitutions much more complex. While in principle they are applicable, we have not yet tried them in the 2D specification. We lack the automated editors required by the number of expressions to transform.

8 Scaling up with Event-B: Process

Like for any complex artifact, we need a precise and definite process to build *good* formal specifications. A *good* formal specification should have the following properties: (a) it is logically consistent, (b) it has proven functional properties, (c) it meets non-functional properties, and (d) it is a reasonable model of the problem.

[2] http://www.brama.fr

Formal refinement is the keystone around which the B-method is designed. Its embodiment into the language and the support tools allows one to develop pieces of software where an implementation is proven against its specification. Refinements break down the verification process into discharging many, but small, proof obligations. So issues (a) and (b) are well taken care of. Event-B uses the same strategy. Here, models are complexified while retaining the same functional properties.

To deal with the issues (c) and (d), i.e., the validation of the specification, we have defined an extended refinement process depicted in Fig. 1.

The idea is to associate validation activities to formal refinement steps. A development step is then composed of four activities:

1. refinement of the physical/mathematical model. The mathematical expressions are refined so that they can be translated into Event-B and lead to provable properties. Discretization is studied at this stage.
2. formal refinement of the Event-B specification with all proof obligations discharged.
3. animation of the specification. The specification is transformed in order to be animatable [17], scenarios are elaborated, and behavior is observed with the animator.
4. analysis and refinement of temporal properties. This activity leads to the introduction of variants and theorems for temporal properties and their proof.

The order of the activities is important. A mis-adapted physical model may lead to non provable, although correct, formulae. There is no point to validate an unproven specification as it may be inconsistent. Animation can help get better insights on the temporal constraints which need to be formalized.

The process was successfully tested with the 1D model. Actually, it was through replaying the construction of the 1D model with the full process that we were able to identify the flaws in the initial model and to produce a revised, correct, version. As can be inferred from the above discussion on tools, we could not fully validate the 2D specification due to the lack of automated tool support. However, the process helped us the ask the pertinent questions when validating through "walkthrough" of the specification. Furthermore, it gave us ideas on improvements of the environment.

9 Conclusion and Future Works

Our experience with a real-world model is both reassuring and worrying. This is consistent with our findings on using Event-B for the modeling of transportation domain [15]. On the very positive side: we could model a reasonably complex algorithm and prove its correctness. *Daviet-Parent algorithm*is a good representative of a class of problems of great practical importance: problems for which we have empirical solutions, prototype implementation, and a strong need to certify it before we can use it in practice.

Event-B is a good candidate for formal modeling and development of real systems. First, the language has sufficient power to express complex mathematical models and algorithms. Second, the formalism embodies a sound, effective, and easy to use refinement based process. Last, the tool support for edition and verification is up to the task. Of course, stronger provers or syntactic sugar-coating to help navigate long texts would be welcome improvements but are not mandatory to make the method usable.

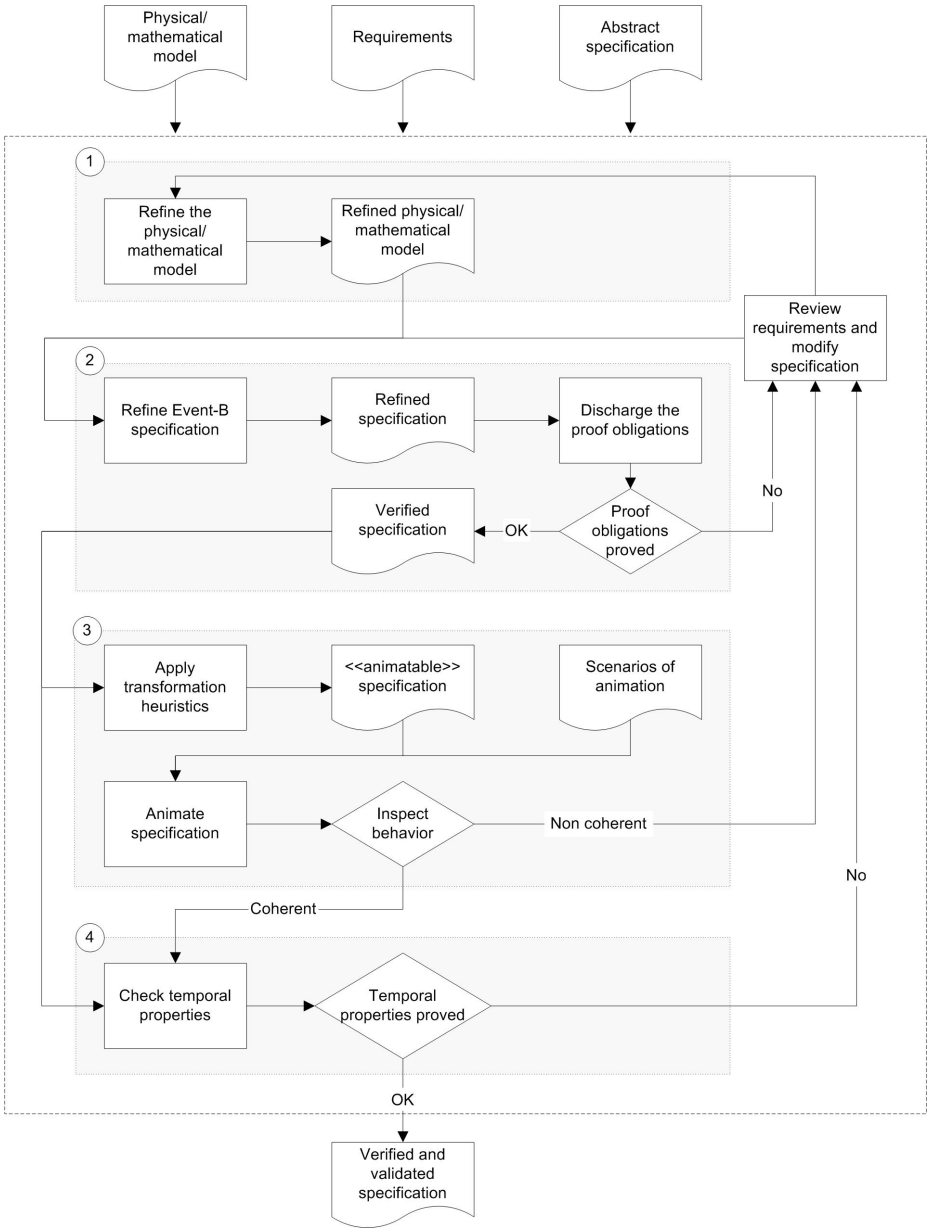

Fig. 1. A step of development process

A second reason to be optimistic was our ability to deal with a physical and mathematical model which incorporates complex functions and continuous time. This has required some sweat and efforts, but was never a blocking factor. We think that there are a few "conditioning techniques" that can be used at the mathematical level to put continuous model in a form suitable to Event-B. We have identified some of them.

Whether Event-B should support continuous functions or real numbers is an interesting question. The answer may not be clear-cut. In our system, real numbers would have eased the writing and maybe some consistency proofs. However, they would not help much improving an implementation because actual vehicles will operate on a discrete time. The control software of actual prototype vehicles operates around 20 Hz. That makes for quite a discrete time.

The worrying issue lies with the checking of temporal properties, either formally through proofs, or pragmatically through animation.

On the formal side, the current situation is not adequate. Only "coarse" properties can be expressed and even then, awkwardly. This is clearly an area where research is needed. A way to overcome this limitation is to associate B or Event-B with another formalism which supports temporal modelling. CSP||B proposed in [22,25] is a candidate. Experiments [6,19] conducted on the specification of the platoon problem indicate a good potential. However, two big issues are still opened: the automation of the proof of consistency between CSP and B parts, and the refinement divergence between CSP and B. Whether CSP||B will scale up soon to realistic models is not yet clear.

On the pragmatic side, we need better animators. Such tools have a very important property: they act on the specification itself. We can be confident that observing animations is observing the model's behavior. It is not clear that the failures to use animators that we have identified can be overcome soon. We are currently working on the idea of translating the specification into an executable language like C or MATLAB. Such translators could be used as a "fall-back" when standard animation fails. They will not exhibit the real model's behavior but a reasonably close version.

Acknowledgments We are indebted to our colleagues from MAIA and TRIO groups at LORIA with special thanks to A. Scheuer for his development of the 2D model.

References

1. Abrial, J.R.: The B Book. Cambridge University Press (1996)
2. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
3. Balzer, R.M., Goldman, N.M., Wile, D.S.: Operational specification as the basis for rapid prototyping. SIGSOFT Softw. Eng. Notes 7(5), 3–16 (1982)
4. Bendisposto, J., Leuschel, M., Ligot, O., Samia, M.: La validation de modèles Event-B avec le plug-in ProB pour RODIN. Technique et Science Informatiques 27(8), 1065–1084 (2008)
5. Bendisposto, J., Fritz, F., Leuschel, M.: Developing Camille, a Text Editor for Rodin. In: Proc. Workshop on Tool Building in Formal Methods. colocated with ABZ Conference – Orford – Canada (2010)
6. Colin, S., Lanoix, A., Kouchnarenko, O., Souquières, J.: Using CSP||B Components: Application to a Platoon of Vehicles. In: 13th International ERCIM Wokshop on Formal Methods for Industrial Critical Systems (FMICS 2008). LNCS, Springer-Verlag (Sep 2008)

7. Daviet, P., Parent, M.: Longitudinal and Lateral Servoing of Vehicles in a Platoon. In: Proceeding of the IEEE Intelligent Vehicles Symposium. pp. 41–46 (1996)
8. Dijkstra, E.W.: A Discipline of Programming. Prentice Hall (1976)
9. Ferber, J.: Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence. Addison-Wesley Professional (1999)
10. Ferber, J., Muller, J.P.: Influences and Reaction : a Model of Situated Multiagent Systems. In: 2nd Int. Conf. on Multi-agent Systems. pp. 72–79 (1996)
11. Lamport, L.: Proving the Correctness of Multiprocess Programs. IEEE Transactions on Software Engineering 3(2), 125–143 (1977)
12. 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). pp. 297–304. IEEE Computer Society (2008)
13. Leuschel, M., Adhianto, L., Butler, M., Ferreira, C., Mikhailov, L.: Animation and Model Checking of CSP and B using Prolog Technology. In: Proceedings of the ACM Sigplan Workshop on Verification and Computational Logic VCL'2001. pp. 97–109 (2001)
14. Leuschel, M., Butler, M.: ProB: An Automated Analysis Toolset for the B Method. STTT 10(2), 185–203 (2008)
15. Mashkoor, A., Jacquot, J.P.: Domain Engineering with Event-B: Some Lessons We Learned. In: 18th International Requirements Engineering Conference - RE'10. pp. 252–261. IEEE, Sydney Australie (2010)
16. Mashkoor, A., Jacquot, J.P., Souquières, J.: B événementiel pour la modélisation du domaine: application au transport. In: Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'2009). p. 19. France Toulouse (2009), http://hal.inria.fr/inria-00326355/en/
17. Mashkoor, A., Jacquot, J.P., Souquières, J.: Transformation Heuristics for Formal Requirements Validation by Animation. In: 2nd International Workshop on the Certification of Safety-Critical Software Controlled Systems - SafeCert 2009. Royaume-Uni York (2009), http://hal.inria.fr/inria-00374082/en/
18. Metayer, C., Voisin, L.: The Event-B Mathematical Language (Oct 2007)
19. Nguyen, H.N., Jacquot, J.P.: A Tool for Checking CSP||B Specifications. In: Proc. Workshop on Tool Building in Formal Methods. colocated with ABZ Conference – Orford – Canada (2010)
20. RODIN: Rigorous Open Development Environment for Complex Systems. website (Aug 2007), http://rodin-b-sharp.sourceforge.net
21. Schmid, R., Ryser, J., Berner, S., Glinz, M., Reutemann, R., Fahr, E.: A Survey of Simulation Tools for Requirements Engineering. Tech. Rep. 2000.06, University of Zurich (2000)
22. Schneider, S., Treharne, H.: Communicating B Machines. In: ZB '02: Proceedings of the 2nd International Conference of B and Z Users on Formal Specification and Development in Z and B. pp. 416–435. Springer-Verlag, London, UK (2002)
23. Siddiqi, J.I., Morrey, I.C., Roast, C.R., Ozcan, M.B.: Towards quality requirements via animated formal specifications. Ann. Softw. Eng. 3, 131–155 (1997)
24. Simonin, O., Lanoix, A., Colin, S., Scheuer, A., Charpillet, F.: Generic Expression in B of the Influence/Reaction Model: Specifying and Verifying Situated Multi-Agent Systems. INRIA Research Report 6304, INRIA (Sep 2007), http://hal.inria.fr/inria-00173876/en/
25. Treharne, H.: Combining Control Executives and Software Specifications. Ph.D. thesis, University of London (2000)
26. Van, H.T., van Lamsweerde, A., Massonet, P., Ponsard, C.: Goal-Oriented Requirements Animation. In: RE '04: Proceedings of the Requirements Engineering Conference, 12th IEEE International. pp. 218–228. IEEE Computer Society, Washington, DC, USA (2004)