

Domain Engineering with Event-B: Some Lessons We Learned

Atif Mashkooor, Jean-Pierre Jacquot

► **To cite this version:**

Atif Mashkooor, Jean-Pierre Jacquot. Domain Engineering with Event-B: Some Lessons We Learned. 18th International Requirements Engineering Conference - RE'10, Sep 2010, Sydney, Australia. pp.252 - 261, 10.1109/RE.2010.37. inria-00431133v2

HAL Id: inria-00431133

<https://hal.inria.fr/inria-00431133v2>

Submitted on 19 Oct 2010

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

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

Domain Engineering with Event-B: Some Lessons We Learned

Atif Mashkoor, Jean-Pierre Jacquot
LORIA – Nancy Université
Vandœuvre lès Nancy, France
{firstname.lastname}@loria.fr

Abstract—

Well specified requirements are crucial for good software design and domain engineering helps better understanding and specification of requirements. Safety critical domains, such as transportation, exhibit interesting features, such as high levels of non-determinism, complex interactions, stringent safety properties, multifaceted timing attributes, etc. The formal representation of these features is a challenging task. This paper presents our experience of modeling land transportation domain in the formal framework of Event-B. We explore the possibility of using Event-B as a domain engineering tool. We discuss the problems posed by the introduction of time and how we tackle it. We design a technique based on animation to validate domain models.

Keywords-Domain engineering, Formal methods, Event-B, Animation, Brama

I. INTRODUCTION

Domain engineering is a methodology to document the facts of a particular domain. A domain model, which is the outcome of the domain engineering phase, defines the key concepts of a particular domain, such as majors entities, their inter-relationships, static and dynamic properties, functions, events, and behaviors. The main activities of the domain engineering phase are: domain analysis, domain design, and domain implementation. Where the domain analysis identifies and captures the domain facts, latter two concern about translation of these facts into system requirements.

The dogma of understanding the domain before the specification of requirements is crucial to software engineering. The idea of having enough details about the environment in which designed product is assumed to operate is already established in other engineering disciplines. All other engineers, such as aeronautical, electrical, chemical engineers know the domains of their respective fields. In software engineering, we can still develop software systems without knowing the particular domain, but the requirements of such systems may be flawed and their correctness is a big issue.

System engineering is a methodology to transform user requirements into a system which best satisfies them. There are numerous reasons to perform domain engineering prior to system engineering. For instance, it identifies, models, constructs, catalogs, and disseminates the system scope, it helps stakeholders understand better the system requirements, it can be effectively used to verify that the system meets essential properties, and so on. Furthermore, domain

engineering in a formal framework gives practitioners an effective grasp on notions such as correctness and validity of requirements.

We present here our preliminary experience with the engineering of a complex domain using Event-B. Event-B [1] is an evolution of classical B method [2] for system-level modeling and analysis of large reactive and distributed systems. We believe the use of Event-B is equally suitable for modeling environments and domains where such systems are assumed to work.

The domain under consideration for this work is land transportation. This domain presents a lot of interesting features to push the use of Event-B to some of its limits. For instance, we want to model vehicles moving independently, to understand their interaction (without formal communication), or to analyze situations where traffic jams occur.

We developed our model in the spirit embedded in Event-B. We liberally used refinements, both of machines and of contexts. We give a great deal of attention to proofs. Consequently, we now have a specification of the transport domain where all proof obligations have been discharged. We also had a special interest in the validation of the model which was achieved by our innovative use of animation of specifications.

During this modeling, we gathered many observations about the use of Event-B on several levels: language, tools, methods, and so on. This paper aims at sharing the salient points of our experience.

The presentation of the paper is organized as follows: next section presents the main motivation to write this paper followed by sections on related work and on language and tool we use. Then we present the domain description and its specification. Sections VI and VII describe the lessons which we learned while specifying and validating our domain model respectively. We conclude our paper in section VIII with some proposed future work.

II. MOTIVATION

Most of the customers express their requirements in natural language or in terms of scenarios. Most of the requirement engineering methodologies are therefore non-formal or semi formal. One of the problems with less formal techniques is that they may be ambiguous, which makes the

requirement engineering phase error prone. Formal requirement specifications, on the other hand, are precise enough to express the requirements but are difficult to understand for customers. The conventional issues of verification and validation may therefore impair the requirement engineering phase.

The earlier involvement of customers and use of formal techniques into software development may be a solution to this problem and domain model is the right artifact to start with. A formal domain model precisely specifies the domain requirements and with the help of techniques, such as animation, we can demonstrate the model to customers for their timely feedback. Our rigorous validation technique, discussed later in the paper, is based on animation and involves customers into the software development process right from the start; consequently errors can be detected right on the spot.

A formal domain model also proves to be very helpful at the time when the system requirements are being specified. In our case, our project associates use a goal oriented requirement engineering methodology KAOS (Knowledge Acquisition in autOated Specification) [3] for requirement elicitation and the goal model capturing user requirements can easily derive the conditions which system must meet from the specified domain model. The idea of gradual introduction of formalism into the development, i.e. requirement elicitation with (semi-formal) KAOS and their specification with (formal method) Event-B, is appealing and acceptable to both customers and developers. The fusion of KAOS with Event-B is recently gaining popularity, see for instance [4], [5], [6].

III. RELATED WORK

This research is closely related with Dines Bjørner's work [7], [8], [9]. In his work he uses RAISE Specification Language (RSL) [10] for the description of domains and concentrates towards the formalization of as much domain facts as possible. Our objective is slightly different. Although we head towards the enrichment of transportation domain model, our concerns are also to check the capability of Event-B as a domain engineering tool and to point out and (where possible) address the issues which we confront with during this exercise.

Previously, B has been employed for the development of transportation systems, see for instance [11], [12], [13], but most of the time the role of this language was limited to system modeling of a particular problem. Our work is different in a sense that we are modeling the domain, with Event-B specification language, where such systems are assumed to operate. The specifications of these aforementioned railway systems does contribute towards the completion of the land transport domain model, but as a part of the whole. Our models are more generalized and considering land

transportation domain in general, including road systems, railways, platooning systems, etc.

There has been some reflection within the B community regarding some of the issues raised in this paper. For example, [14] discusses the challenges of the expression of explicit timing properties elegantly in Event-B, a key element for utilization of formal methods into automotive sector. The expression of such properties is currently non standard. In another work [15], authors discuss the utility of Linear Temporal Logic (LTL) expressions in Event-B specifications. They remark about the inability of RODIN to prove the correctness of a system with respect to LTL expressions.

IV. LANGUAGE AND TOOLS

A. Event-B

Event-B 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 RODIN¹.

An Event-B model is composed of two constructs: MACHINE and CONTEXT. Machine, which defines the dynamic behavior of the model, contains the system variables, invariants which define the state space of the variables and their safety properties, theorems, variants, and events. Context, which defines the static behavior of the model, contains carrier sets, constants, axioms, and theorems.

The refinement process is used to progress from abstraction specifications to concrete and elaborated specifications. In refinements, new variables can be introduced and the old variables can also be refined to concrete ones. New events may also be introduced as long as they do not prevent forever the old ones from being triggered. Variants are explicitly introduced to ensure this property. Proof obligations are generated to ensure the consistency and correctness of both models: the abstract model and its refinement.

B. Brama

Brama is an animator for Event-B specifications. It is an Eclipse based plug-in for the Event-B platform RODIN. Brama can be used in two complementary modes. Either Brama can be manually controlled from within the RODIN interface or it can also be connected to a Flash² graphical animation through a communication server; it then acts as the engine which controls the graphical effects.

A typical animation session begins by setting the values of the constants in the different contexts seen (either directly or transitively) by the animated machine. 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

¹<http://rodin-b-sharp.sourceforge.net>

²Flash is a registered trademark of Adobe Systems Inc.

event, or the system enters to a steady loop, or an error occurs (broken invariant or non computable action typically).

A graphical interface can be connected to Brama in the form of a Flash application and events can be directly fired from there. A mechanism of observers is provided. Expressions and predicates can be individually monitored and their value is communicated to the Flash program each time it changes. Last, a scheduler mechanism allows for the automatic firing of events.

V. DOMAIN DESCRIPTION

A. Domain overview

Our work takes place within the framework of projects TACOS³ and CRISTAL⁴. These projects aim at studying new transportation systems using autonomous and self-service vehicles known as CyCabs [16]. One of the related concerns is the certification of such systems for which no standard exists.

The certification of a vehicle or a system is a process where it is verified that the vehicle meets some requirements. These requirements are derived from the expression and formalization of desirable properties that the whole transport system must incorporate. The issue for software controlled vehicles is to have an expression of those properties amenable to the use of formal verification. The model of the land transport domain is aimed at providing us with the formal expression of these properties.

The model has been defined with Event-B specification language, following the refinement principles advocated by the B method. We used the ability of Event-B to combine refinement and incremental enrichment of the specification. First, a general definition of transportation networks and the act of moving was given. Then, we introduced properties, one at a time.

Transportation is defined as the movement of people and goods from one location to another with the use of vehicles. We suppose the existence of a network composed of stations (places where vehicles can stop to be loaded and unloaded), intersections (places where roads join), and paths which connect stations and intersections together. Movements are constrained by the topology of the network: a vehicle must follow a sequence of adjacent paths to travel from its origin to its destination.

The general properties we want to express concerning transportation are safety and travel time. First is the idea that collision between vehicles must be avoided. Second is related to the fact that travel time is at the root of nearly all decisions made around transportation, either individually or socially.

³<http://tacos.loria.fr>

⁴<http://www.projet-cristal.org>

B. Event-B specification

Our current domain model contains one abstract machine and seven refinements. In parallel with the machines, two contexts are being refined. The first is the context **Net** which models static properties of the network (its topology, quantities associated to its elements, etc.). The second is the context **StartState** which helps to set and prove the **INITIALISATION** event of the machines.

It is easier to read and understand the specification when the refinements are grouped into what we call “observation levels.” A leap from one level to the next occurs when we decompose an abstract event into several ones, corresponding to a finer grain analysis. For instance, the decomposition of the most abstract `travel` event into a sequence of paths traversing and hubs crossing events correspond to a change of observation level. Figure 1 summarizes the four levels:

- First level of observation contains the definition of `travel` event and is specified by machines **Movement0**, **Movement1** and **Movement2**.
- Second level of observation decomposes `travel` event into `crossHub` and `traversePath` events and is specified by machine **Movement3**.
- Third level of observation decomposes `crossHub` event into `enterHub`, `leaveHub`, and `wait` events and is specified by machines **Movement4** and **Movement5**.
- Fourth level of observation decomposes `traversePath` event into `waitToEnterOnPath`, `leaveHub`, `moveOnPath` and `waitToMoveOnPath` events. It is specified in **Movement6** and **Movement7**.

New observation levels were introduced when a property could not be expressed within the existing levels.

The first level of observation is about setting up the main domain vocabulary and defining the basic properties of the domain. In the context **Net** and in its refinements we define the basic vocabulary of the transportation network, such as nets, hubs, stations, junctions, connections, paths, routes, etc. In the machine **Movement0** we abstractly define the `travel` event as relocation of a vehicle from one place to another. The further refinements at this level introduce a finer topology of the network (junctions, stations, paths, routes) and express the property that travel only occur between connected stations.

The second level of observation is about the property that a travel is constrained by the topology of the network. The abstract event is then decomposed into three events (`startTravel`, `crossHub` and `traversePath`) which must occur in a unique sequence to realize a travel.

The third level of observation is motivated by the introduction of the property of non collision at intersections. Such collisions are abstractly defined as the presence of too many vehicles on a hub at the same time. This lead

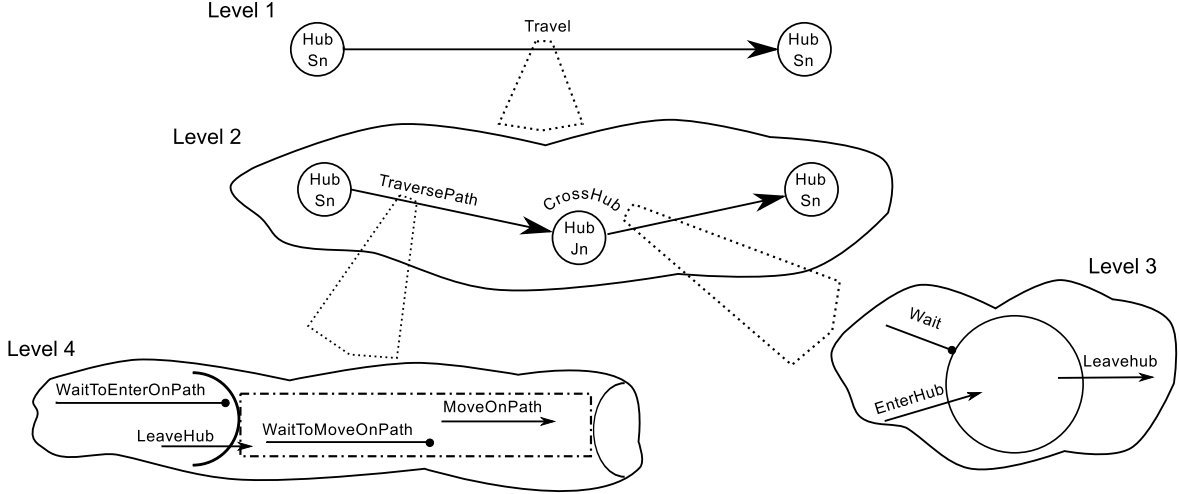


Figure 1. Levels of observations

us to decompose the `crossHub` event as a sequence of `wait`, `enterHub` and `leavehub` events. The choice between `wait` and `enterHub` is controlled by the notions of `hubLoad` (the number of vehicles present on the hub) and `hubCapacity` (the maximal number of vehicles that can be safely present on the hub). The second refinement at this level corresponds to the introduction of the notion of travel time which does not require a further observation leap.

Fourth level of observation is associated with the introduction of the property of non collision on paths (rear-end type of collision). The event `traversePath` is decomposed into a sequence of `waitToEnterOnPath`, `leaveHub`, `moveOnPath` and `waitToMoveOnPath` events. This models the abstract kinematics of the vehicles.

Following are two interesting properties of the domain which we model:

Collision Avoidance: In the real world, collisions are situations that must be avoided. We choose to model them as breach of an invariant. This way, it is easier to identify the conditions for a well behaving domain through the guards of events.

In real life, collisions can be classified in three types: front, rear and side. Front collisions are implicitly prevented by the topology of the network: paths are oriented and model one way lanes. Side collisions occur at intersections, rear collisions on paths. This prompted us to use two disjoint invariants. The events introduced at the second level made this separation easy to implement.

While a real collision happens when two vehicles are at the same place at the same time, we choose to model it more abstractly on the hubs. Our definition relies on the idea that a hub can carry only a fixed number of vehicles at a time. So, the invariant to maintain is easily written as:

$$\forall h. h \in \text{Hubs} \Rightarrow \text{hubLoad}(h) \leq \text{hubCapacity}(h)$$

where `hubLoad` is the actual number of vehicles on a

hub and `hubCapacity` is the maximum number of vehicles allowed on the hub. `hubLoad` is a function modified by the events, `hubCapacity` is a constant property for each hub. Interestingly, this definition does not require the introduction of time. It abstracts from the kinematics of the vehicles on the hub.

The specification of the absence of rear collision on paths is directly inspired from the natural definition. The corresponding invariant is:

$$\forall v1, v2. v1 \in \text{Vehicles} \wedge v2 \in \text{Vehicles} \wedge v1 \neq v2 \wedge v1 \in \text{dom}(\text{vehiclePosition}) \wedge v2 \in \text{dom}(\text{vehiclePosition}) \wedge \text{vehiclePath}(v1) = \text{vehiclePath}(v2) \Rightarrow \text{vehiclePosition}(v1) \neq \text{vehiclePosition}(v2)$$

where `vehiclePath` signifies the current path of the vehicle and `vehiclePosition` is a refinement of the notion of the location of a vehicle on the path.

In a further refinement, positions on paths are modeled as an interval on integers, starting at 0 and ending at `pathLen`. This allowed us to introduce the natural notion of safety distance (`criticalDistance`) that is used in the guards of the moving events. An instance of such a guard is:

$$\forall v. v \in \text{vehiclesOnPath} \wedge \text{vehiclePosition}(v) > \text{vehiclePosition}(\text{vehicle}) \Rightarrow \text{vehiclePosition}(v) - \text{vehiclePosition}(\text{vehicle}) > \text{criticalDistance}$$

Time: Time is a very important notion in the domain of transportation and our model needs to incorporate it. This notion is known to be tricky to define and to use. In fact, our domain suggests the existence of several flavors of time. One flavor is the travel time, where a clock is only observed at the beginning and at the end of a travel. Another flavor is the time used in modeling the kinematics where it controls the behavior of the vehicles.

Since Event-B lacks an explicit notion of time, we used the timing patterns for Event-B proposed by Cansell et al [17]. In this technique we use natural numbers to model time and a special `ticTac` event to make a global clock (*time*) advance.

The modeling of time was motivated by the introduction of the `wait` event on the third level. We proceeded in two steps. The first was the introduction of the notion of a clock and the notion of travel time as a difference between two readings of the clock. Although technically realized as a refinement of **Movement4**, this introduction is logically situated at the first observation level. The second step was the actual computation of the advance of the clock.

To do this, we modeled the technique used in simulating queue systems. We introduced a timed event queue (*activationTime*) which contains the time at which a moving vehicle must perform an event. The following invariants are introduced:

```
inv1 activationTime ∈ Vehicles → NAT
inv2 activationTime ≠ ∅ ⇒ time ≤ min(ran(activationTime))
```

A new guard is then introduced in the events concerned by time:

```
vehicle ∈ dom(activationTime) ∧ time=activationTime(vehicle)
```

The action part of the event modifies the event queue accordingly.

The timing pattern is specified by the event `ticTac` as shown by figure 2.

```
ticTac ≡
ANY
tic
WHERE
activationTime ≠ ∅ ∧ tic = min(ran(activationTime)) ∧ tic > time
THEN
time := tic
END
```

Figure 2. Event `ticTac`

A vehicle is introduced in the event queue by the `startTravel` event. It is removed from the queue when it reaches its destination.

Elements of an earlier version of this specification are discussed in [18]. A verified version of our specification is available at the following web address:

<http://dedale.loria.fr/?q=re-spec>

VI. LESSONS LEARNED: SPECIFICATION

A. Refinement and observation levels

Refinements and observation levels are distinct concepts. Refinements are the cornerstones of B method. They serve two purposes: methodologically, they allow specifiers to concretize the specification, and technically, they induce proof obligations which guarantee the correction of the development. They give the development a flat structure which may impair its readability.

Observation levels are a way to provide a specification with a super-structure which eases the understanding. They reflect either the “natural” structure of the objects or the

structure of the computation. For instance, the second observation level in the model reflects the static topology of a network, while the third level is more about the protocol to cross a hub.

The major advantage of thinking in terms of observation levels stands out when we introduce a new property. This structure provides us with a strong guideline. We experienced it with the introduction of time. The vocabulary and abstract constraints (time is ever increasing, for instance) were defined at the first level since this concerned only travels. Then we jumped directly to the third level to define the computation because durations could be associated to events at this level.

B. Parallel refinements

While the view of a development as a linear sequence of refinements makes sense in B where a system is developed, it is far less pertinent in Event-B where an environment is described. Properties are often independent, at least as far as their definition is concerned. We experienced this with time and collision avoidance. It would be even more important when we introduce properties such as energy consumption, for instance.

The problem with the linear sequence is that when we introduce a new property, we need to do this into a complex piece of text. Furthermore, most of the text is irrelevant to the property at hand. In some sense, this breaks the sound principle of separation of concerns.

In domain engineering the commonality/variability analysis and decomposition/recomposition of models has always been considered as integral features. Currently RODIN is missing such features, primarily parallel refinements. However, for its recent versions, few plugins have been proposed for composing Event-B models together: Feature Composition Plugin [19], Parallel Composition Plugin [20] and Shared Event Composition Plugin [21]. They are still prototypes and at early stages of development. We still need to investigate them in more details before recomposition of our models.

C. Protocols / ordering constraints in events

Once events are decomposed into smaller events, it is crucial that these events be fired in a strict order in order to maintain the cohesive behavior. For instance, the decomposition of the travel event is thought of as:

```
travel ≡ (startTravel; (crossHub; traversePath)+)
```

Unfortunately, Event-B does not provide us with traits to express this protocol. Instead, we must make explicit definition of the protocol with the help of control variables and guards in the events. This is complex and a source for errors.

This situation happened each time we introduced a new observation level. So, going from second to third level, we

decompose as follows:

$$\text{crossHub} \equiv (\text{wait}^*; \text{enterHub}; \text{leaveHub})$$

To go from third to fourth level, we decompose as follows:

$$\text{traversePath} \equiv (\text{waitToEnterOnPath}^*; \text{leaveHub}; \\ (\text{waitToMoveOnPath} | \text{moveOnPath})^*)$$

We use two basic techniques for controlling the protocols. The first is the introduction of control sets. We used it for the decomposition of travel. The control variables are the set of all hubs and paths the vehicle will have to pass. The next hub to cross or the next path to traverse is easily defined as the member of the control set which is related to vehicle's position. This technique has the advantage that a variant is quite easy to define but has the drawback to introduce complex computation of the sets. The second technique is the introduction of a notion of state markers, either through an explicit variable or a property, such as belonging to the domain of a relation. This can be seen as a form of coding a state machine. The advantage of using state markers is their easy definition but their drawback is the difficulty to set variants and generally to connect state markers to invariants.

Although without formal substance, the previous regular-expressions like formulas were of great help to set up the explicit control. It would be a welcome extension of Event-B or of its supporting tools if that kind of expression could be stated and be checked against the behavior of the events. Diagrammatic notations, such as the structure diagrams of Jackson System Development (JSD) [22] or formalism like Communicating Sequential Processes (CSP) [23] could be used.

D. Time modeling

Unsurprisingly, the modeling of time raised many questions. We used the timing patterns for Event-B proposed by Cansell et al [17] in our models. They assume a discrete time and in our model, travel time is of that kind. The "computation" of the clock with the timed event queue is cumbersome because it is explicit, but does not lead to specification difficulties. Indeed, a generic pattern emerged to write the refinement:

- 1) identify an event concerned by time;
- 2) introduce the standard guard (the same for all events);
- 3) introduce a substitution of the timed event queue; the actual value to substitute is of course dependent on the event.

Kinematics introduce a flavor of continuous time. This raises two questions: Is it legitimate to try to model this with the purely discrete means Event-B provides us? How will it merge with the previous definition of time? The answer to the first question is "Yes" if the model is to be the basis for a software implementation. By essence, computers are discrete machines. A fundamental parameter of any control software

for running machines is the frequency of their control loop. So, the actual time will be discrete. As the refinement of the specification to introduce sensible kinematics is not finished at the time of writing, we do not have a definitive answer to the second question. However, preliminary developments have shown that the event queue algorithm can be used. A kind of "fixed tick" can be introduced in the third step of the previous pattern to force events to be fired at a given frequency.

E. Safety and liveness properties

Safety: A safety property asserts that nothing bad happens [24]. Safety properties can be specified either as something that should never happen, or as some property that should always hold. We consider the safety property of collision avoidance. It is specified by the invariants of the model. All the invariant preservation proofs have been discharged. We are then assured that no event provokes a collision.

It should however be noted that the previous condition is necessary, but not sufficient to ensure safety in general. Although this does not yet happen in the current state of the specification, it will when kinematics will be fully specified. A moving vehicle should never be allowed to make a move which leads to a collision (i.e. no event should break the invariant), but it must also always be able to react (i.e. there should always be an enabled event). This last condition is similar to the liveness property discussed below.

Deadlock: A deadlock, in computation, is a state when some processes in a system are halted waiting for something to happen which can only be done by one of the halted processes. In transportation a similar phenomenon exists and is referred to as gridlock which describes an inability to move on a transport network (i.e. traffic jams). Both deadlock and gridlock are something that implementers must avoid. It is then important to characterize them at the level of the specification.

While deadlocks can be thought of as a situation in Event-B where no event is enabled, i.e., guards of all events are false, deadlock freeness would mean that some vehicle can always move i.e. at least one event is enabled all the time, such as stated with the following invariant:

$$G(E_1) \vee G(E_2) \vee \dots \vee G(E_n)$$

where $G(E_i)$ is the guard of event E_i .

In transportation domain, we can always experience the situation of traffic jams which may prevent all vehicles from moving. Since gridlock is a fact of life, we choose to allow them in the specification. At theoretical level, with the introduction of `wait`, we can say that a vehicle can wait in such situations, and at least this event can always be fired, but this is not an elegant solution. At specification level, RODIN does not do any deadlock freeness proof and it needs to be done by hand.

As an impact of the decision to allow gridlock in the model, later in the specification, the introduction of time forced the gridlock situations to “pop up” during some proof obligations. A solution was to introduce new events to model these gridlock situations. We have identified three such situations at present:

- first is when a vehicle needs to enter a station which is already full of parked cars. No vehicle will leave the hub and the moving vehicle is then “locked out”;
- second is when vehicle needs to enter a path which is full of other (stationary) vehicles. This vehicle is then “locked in”;
- third case is similar to the second case except the vehicle has already begun traversing of the path. It is then “locked on path.”

Modeling gridlock with special events has at least one advantage. The conditions of the blockage are clearly identified. Implementers who want a particular system to be jam free can derive their invariants from these conditions.

Liveness: Liveness property asserts that something good will happen “eventually” [24]. We have noted above that liveness can be a necessary condition to have safe systems. This notion can also be used for expressing non critical, but desirable properties. In our case, a desirable property is that a vehicle eventually reaches its destination and terminates its travel. This property cannot be formally expressed within Event-B framework because liveness properties involve the temporal concept “eventually” and until now there is no standard temporal constraint definitions for Event-B specifications. Even knowing that, due to traffic jams, the above liveness property is certainly not guaranteed, it would be very useful to be able to express it formally.

However, as proposed by [25], in order to prove liveness of our model we can prove that our system is non-divergent and enabledness preserving. By non-divergent we mean that newly introduced events do not take control forever and by enabledness preserving we mean that if an event is enabled at abstract level it is enabled at concrete level as well.

Non divergence is usually proven with the help of variants. We introduced the following variant at second level of observation:

$$\text{card}(\text{hubsToCross}) + \text{card}(\text{connectionsToTraverse})$$

it states that the newly introduced events `crossHub` and `traversePath` do not prevent the `travel` event to fire.

This notion of variant is useful to prove non divergence until the event `wait` is introduced at the third observation level. Since a vehicle can wait for indefinite period of time for its turn to enter a hub therefore our variant can not assure us that this event can not take control forever. This is a fact of life: land transportation domain is divergent.

We can prove enabledness preservation of the model by the standard consistency and refinement checking proofs which need to prove that the guards of one or more events in

the refinement are enabled under the assumption that guards of one or more events in the abstraction are also enabled.

This discussion on safety and liveness properties indicates that they are complex and tangled issues. It also shows that as far as domain models are concerned, there should not be only one rule like, say, no model shall deadlock, or models shall always be live. The point is that Event-B does not provide us the mean to express cleanly that kind of properties. We consider this as an important shortcoming.

F. Language and tools

Our unconventional use of Event-B and, consequently, of RODIN raised a few issues with the modeling language and the tool support. While the observations discussed below sound negative, we must emphasize the overall quality of the language and the tools: the major difficulties we encountered were caused by the complexity of the domain and by our own errors.

Considering the tool support, we have two observations:

- RODIN failed too often to discharge automatically obvious proofs. Worse, they were often discharged by a simple click. Although not really tiring, this “activity” becomes boring and, more importantly, very distracting. Many proofs require a lot of concentration; we expect tools to help rather than distract on this aspect.
- RODIN does not warn when axioms are inconsistent. The detection of contradicting axioms is hard. Now, we rely only on heuristic rules. We suspect a contradiction when we notice that proofs become mysteriously easy to discharge. Then, we introduce an axiom or a theorem such as `TRUE = FALSE`. Success in the proof signs a contradiction, failure provides us only with reasonable assurance. We know that proving the non contradiction of axioms is non decidable. However, the indication by RODIN that it has detected an inconsistency would be welcomed.

Our work prompted three remarks on the language:

- Refinement is the only structuring mechanism in Event-B. As discussed above (section VI-A), we would appreciate to group machines in other ways. This would not necessarily require a modification of the language, but could be achieved by the tools.
- The internal structure of Event-B machines and contexts is too flat. Again, a possibility to structure axioms or events into categories would improve greatly the readability. For instance, we classified our axioms in three categories (technical, typing, and property) and found this practice very helpful to maintain clean and readable specification.
- The feature of Event-B which we missed a lot was the notion of sequences. Currently we specify them by using the standard definition of sequences. We consider this only as a patch: it works but it brings clutter

to pieces of specifications that are already sufficiently complex.

VII. LESSONS LEARNED: ANIMATION

An important part of the specification of the transport domain amounts to model complex behaviors. Some are explicitly defined (e.g., the succession of `crossHub` and `traversePath` during a travel), some implicitly (e.g., the correct interaction of vehicles at intersections), and other unknowingly (e.g., only one vehicle at a time was allowed to travel in an early, erroneous in that case, specification). As a specifier, we confront with three questions: Do our specification models an actual behavior observed in the domain? Do our specification model the behavior we actually want to describe? How do we specify a certain behavior?

These questions correspond to well known software engineering concerns related to three different development activities. First question is about modeling “good” representations of the actual world. Second question concerns the validation of the formal expression against some already abstracted model. Third question is of a technical nature, related to the expressive power of the language.

We have found out that animation is a very valuable technique to help answer these three questions. While the observation of the animation (which does not need to have fancy graphics) gives a lot of information about the model and helps uncover errors, we also discovered that some activities around animation are also crucial. Activities such as setting up values for the animation (e.g., fixing a network’s actual topology) and inventing scenarios to act or observe provided us with a lot of insight about the specification text, about the model, and even about the traits of the reality we wanted to model.

Unfortunately, we soon discovered that not all specifications could be animated. Not only tools have their limitations, such as non supported features of the language for instance, but specification techniques, such as non constructive definitions, often prevent efficient computation of the values. To be useful, an animation needs to be reasonably fast.

We have then designed and described, as rigorously as possible, a set of rules which transform a non animatable specification into one that the animator Brama could animate. One can wonder why we do not produce an animatable specification at first. The reason is that our transformation rules “downgrade” the initial specification on two important counts: the specification becomes far less readable and, more importantly, may become unprovable. The transformation process tends to alter and suppress elements that are essential to discharge proof obligations.

Naturally, the relation between the behaviors seen during the animation and the ones described in the initial specification becomes a major issue. To solve the issue, we propose

a methodology to include the animation into a rigorous process, which is as follows:

- 1) start from a *fully* verified specification. This step is essential.
- 2) for each non animatable trait:
 - a) pick an appropriate rule
 - b) check that the applicability conditions hold
 - c) prove that the argument used in the justification part of the rule is valid
- 3) if an anomalous behavior is encountered, modify the initial specification, *prove* it to be correct, and restart from step one.

The proof in step 2(c) cannot be carried out in Event-B. Like mathematicians do, we use rigorous arguments, often relying on the fact that the initial specification has been proven correct, to assert that the initial and transformed specification share the same behaviors. In particular, those we want to observe.

The aforementioned animation issues, the rules, and the process are described in more details in [26] and [27].

It should be noted that our choice of tool, Brama, is contingent. At the time, it was the only one able to animate Event-B specifications. More recent tools such as AnimB⁵ and ProB [28], are now available and fully compatible with Event-B. While our rules should surely be adapted to these specific tools, we suspect that the general philosophy of animation we have adopted is still valid.

VIII. CONCLUSION

We find Event-B an adequate language for domain engineering, however there are still some important questions to address. They are about the language, the tools, and the use of domain models in requirement engineering.

About the language, the most limiting factor is the lack of expression of temporal or ordering constraints. We cannot straightforwardly state, and of course prove, properties such as liveness, deadlock freeness, fairness, and so on. Our domain exhibits many natural “protocols” and constraints; we do not think it is exceptional in this respect. Whether Event-B can be extended in this direction, or whether approaches based on mixing formalisms, such as advocated in [29] with CSP||B can be made practical, answers are beginning to appear. We just hope they can be used soon.

Tools are essential to formal methods. Without RODIN, the provers, and Brama, there is no way we could have reached the current state of the specification. However, they are still crude for an industrial usage. The tool we lacked the most was created by our needs with respect to animation. Applying the transformation rules, which we invented, requires some insight and intelligence (choice of the rule, check of the validity), but also tedious and boring work (text modification). We plan to implement the second

⁵<http://www.animb.org>

part as a form of plugin for RODIN. The boring parts of the transformation do not contain overly complex text manipulation.

We would appreciate also to see tools evolving in the direction of richer visualisation of the specifications. Our notes about observation levels, flat linear structures, parallel refinement, or composition of refinement can be seen through this light. We do not call for incorporating these into the language: it would be unwise to break something that works quite well! Instead, we think that tools based on a better understanding of the needs of the specifiers would be a more promising approach. There is clearly a need for research in this direction.

As we argued, knowledge of particular domain before prescription of requirements is a valuable asset. We have hinted at ideas, such as deriving invariants of a system from the properties expressed in the domain model. We now want to test this by studying the practical relation of our domain model with a separate specification, written also in Event-B, of a platooning system [30]. In particular, we would like to study how we can *immerse* the specification of a particular system into the domain model.

ACKNOWLEDGEMENTS

This work has been supported by ANR under ANR-06-SETI-017 TACOS project and Pôle de Compétitivité Alsace/Franche-Comté under CRISTAL project.

REFERENCES

- [1] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering*, Cambridge University Press, 2010.
- [2] —, *The B Book*, Cambridge University Press, 1996.
- [3] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.
- [4] A. Matoussi, F. Gervais, and R. Laleau, A First Attempt to Express KAOS Refinement Patterns with Event B, in *1st International Conference on Abstract State Machines (ASM), Alloy, B and Z (ABZ'08)*, Springer, London, UK, 2008.
- [5] B. Aziz, A. E. Arenas, J. Bicarregui, C. Ponsard, and P. Massonet, From Goal-Oriented Requirements to Event-B Specifications, in *First Nasa Formal Method Symposium (NFM'09)*, California, US, 2009.
- [6] A. Mashkoor and A. Matoussi, Towards Validation of Requirements Models, in *2nd International Conference on Abstract State Machines (ASM), Alloy, B and Z (ABZ'10)*, Springer, Orford, Canada, 2010.
- [7] D. Bjørner, Development of Transportation Systems, in *2nd International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA'07)*, Poitiers, France, 2007.
- [8] —, *Software Engineering 3: Domains, Requirements, and Software Design (Texts in Theoretical Computer Science, an EATCS Series)*. Springer, 2006.
- [9] —, *DOMAIN ENGINEERING: Technology Management, Research and Engineering*. JAIST, 2009.
- [10] H. Van, C. George, T. Janowski and R. Moore (Editors), *Specification Case Studies in RAISE*, Springer, 2002.
- [11] A. Papatsaras and B. Stoddart, Global and Communicating State Machine Models in Event Driven B: A Simple Railway Case Study, in *2nd International Conference of Z and B Users (ZB'02)*. Springer, Grenoble, France, 2002.
- [12] P. Behm, P. Benoit, and J. M. Meynadier, METEOR: A Successful Application of B in a Large Project, in *1st International Conference on Integrated Formal Methods (iFM'99)* Springer, York, UK, 1999.
- [13] F. Badeau and A. Amelot, Using B as a High Level Programming Language in an Industrial Project: Roissy VAL, in *4th International Conference of Z and B Users (ZB'05)*, Springer, Guildford, UK, 2005.
- [14] T. Lecomte, Deploy Deliverable D2, Technical Report, 2009. [Online]. Available: <http://www.deploy-project.eu/newsletter/deploy-newsletter-03.pdf>
- [15] M. Jastram, M. Leuschel, J. Bendisposto, and A. G. R. Jr, Mapping Requirements to B Models, Technical Report, 2009. [Online]. Available: <http://deploy-eprints.ecs.soton.ac.uk/104/>
- [16] G. Baille, P. Garnier, H. Mathieu, and P.-G. Roger, Le Cycab de l'INRIA Rhône-Alpes, INRIA Rhône-Alpes, Technical Report RT-0229, 1999.
- [17] D. Cansell, D. Mery, and J. Rehm, Time Constraint Patterns for Event B Development, in *7th International Conference of B Users*, Springer, Besançon, France, 2007.
- [18] A. Mashkoor, J.-P. Jacquot, and J. Souquières, B événementiel pour la modélisation du domaine: application au transport, in *Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'09)*, Toulouse, France, 2009.
- [19] A. Gondal, M. Poppleton, and C. Snook, Feature Composition - Towards Product Lines of Event-B models, in *1st International Workshop on Model-Driven Product Line Engineering*, Twente, The Netherlands, 2009.
- [20] M. Poppleton, The Composition of Event-B Models, in *1st International Conference on Abstract State Machines (ASM), Alloy, B and Z (ABZ'08)*, Springer, London, UK, 2008.
- [21] R. Silva, C. Pascal, T. Hoang, and M. Butler, Decomposition Tool for Event-B, in *Workshop on Tool Building in Formal Methods*, Orford, Canada, 2010.
- [22] M. A. Jackson, *System Development*, Prentice-Hall, 1983.
- [23] C. A. R. Hoare, Communicating Sequential Processes, *Commun. ACM*, vol. 21, no. 8, pp. 666–677, 1978.

- [24] L. Lamport, Proving the Correctness of Multiprocess Programs, *IEEE Transactions on Software Engineering*, vol. 3, no. 2, pp. 125–143, 1977.
- [25] D. Yadav and M. Butler, Verification of Liveness Properties in Distributed Systems, in *Second International Conference on Contemporary Computing (IC3'09)*, Noida, India, 2009.
- [26] A. Mashkoo, J.-P. Jacquot, and J. Souquière, Transformation Heuristics for Formal Requirements Validation by Animation, in *2nd International Workshop on the Certification of Safety-Critical Software Controlled Systems (SafeCert'09)*, York, UK, 2009.
- [27] A. Mashkoo and J.-P. Jacquot, Incorporating Animation in Stepwise Development of Formal Specification, Research Report INRIA-00392996, LORIA, Nancy, France, 2009, [Online]. Available: <http://hal.inria.fr/inria-00392996/en/>.
- [28] M. Leuschel and M. Butler, ProB: A Model Checker for B, in *12th International FME Symposium (FM'03)*, Springer, Pisa, Italy, 2003.
- [29] N. Evans and H. E. Treharne, Linking Semantic Models to Support CSP||B Consistency Checking, in *Fifth International Workshop on Automated Verification of Critical Systems (AV-OCS'05)*, Warwick, UK, 2005.
- [30] A. Lanoix, Event-B Specification of a Situated Multi-agent System: Study of a Platoon of Vehicles, in *2nd International Symposium on Theoretical Aspects of Software Engineering (TASE'08)*. IEEE, Nanjing, China, 2008.