Guidelines for Formal Domain Modeling in Event-B

Atif Mashkoor, Jean-Pierre Jacquot

Guidelines for Formal Domain Modeling in Event-B

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

***Abstract*—In this paper, we explore the possibility to use Event-B as a formal domain modeling tool. We identify the areas where domain modelers can struggle and present some guidelines to avoid these pitfalls. We mainly address three questions about domain modeling: what to specify, how to refine, and how to verify. We discuss the strategy to express domain assumptions, protocols, time, and temporal properties. We also analyze the refinement and proof system of Event-B in this realm. We advocate small incremental steps and alternative refinement mechanisms, such as "observation levels." We find animation a very helpful activity to complement the verification process.**

***Keywords*-Formal methods, Domain modeling, Event-B**

I. Introduction

Domain is a universe of discourse, an area of human and societal activity for which some support in the form of computing may be desired [1]. Examples of domains are transport, finance, health care, etc. Domain modeling is the process to document the key concepts of a particular domain, such as entities, their inter-relationships, static and dynamic properties, operations, events, and behaviors. System modeling, on the other hand, is the process in which different models, such as domain analysis, are used to design and construct systems.

A well-founded domain model provides a clear picture of the problem domain to all stakeholders. On the one hand, it helps drawing the underlaying systems' boundaries and on the other hand, it provides a vocabulary which keeps the meaning of the domain's concepts harmonious to everyone.

We can model a domain in a number of ways, either textually, graphically, or formally. While modeling high-assurance domains, we prefer the formal approach. Though this choice requires advanced skills with some formal notation, yet it yields several notable advantages. First, a formal specification of the model is amenable to verification and testing which should result into a trustworthy software. Second, it helps to engineer the right software. The systematic derivation of requirements from formal domain models ensures that the resulting software is exactly what the customer want; ambiguities are eliminated right from the start.

The appeal of model-based, i.e., graphical, method is mainly due to an apparent ease of use of the tools and documents. In contrast, formal techniques are often caricatured as ascetic and out-of-reach of average developers. However this is just a perception, not a reality. Thanks to sophisticated and easy-to-use tools, many complexities tied to the application of formal methods can be abstracted away behind convivial interfaces. Execution techniques of specifications, like animation, can also involve customers into the development at earlier stages. Specification errors do not trickle down the development chain in this fashion. Specification techniques based on mixing formalism, such as [2], [3], also help requirements engineering. Providing developers with practical guidelines for developing formal models is also an effective mean of promoting their use.

We present here our experience of domain modeling with the formal method Event-B [4]. Event-B is an evolution of the B method [5] for system-level modeling and analysis of large reactive and distributed systems. We believe that using Event-B is equally suitable for modeling environments and domains where such systems are assumed to operate.

We have gathered our observations while modeling a high assurance domain: land transportation. Our work took place within the framework of the projects TACOS[1] and CRISTAL[2]. These projects aim at studying new transportation systems using autonomous and self-service vehicles known as CyCabs [6]. CyCabs are small computer-controlled electric cars. They can move in three modes: driven by a human, driven by their inboard computer, or within a *platoon.* In the last mode, several CyCabs assemble as a train without material connections between the cars.

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

We modeled two general properties of transportation, safety and travel-time. The first is the idea that collisions between vehicles must be avoided. The second reflects the fact that travel time is at the root of nearly all decisions made about transportation, either individually or socially.

In this paper, based on our experience, we present some

[1] http://tacos.loria.fr
[2] http://www.projet-cristal.org

guidelines for formal domain modeling with Event-B. Section II presents a brief overview of the employed formal method. In Section III, we discuss that what a domain model should be comprised of. Sections IV and V analyze the refinement and verification process of domain modeling in Event-B respectively. Section VI presents some related work. Finally, Section VII concludes the paper and indicates some future work directions. The *fully-verified* version of the transport domain model which is used as the reference specification is available at the following web address: `http://dedale.loria.fr/?q=transport-domain`.

II. Event-B

Event-B is a formal language for modeling and reasoning about large reactive and distributed systems. Event-B is based on set theory and standard first-order predicate logic. Event-B is provided with Rodin[3], a platform which supports the writing and proving of specifications.

A. Structuring

An Event-B model is composed of two constructs: machines and contexts. Machines, which define the dynamic behavior of the model, contain the system variables, invariants, variants, and events. Variables are typed; their values may be integers, sets, relations, functions or any other set-theoretical construct. Invariants define the state space of the variables and their safety properties. Variants are related to the correction of refinements.

An event defines transitions from one state to another. It is composed of guards and actions. A guard is a predicate and an action is an assignment statement to a state variable and is achieved by a generalized substitution. Events can be triggered when their guard is true; the choice of the event to fire is non-deterministic. The actions of a particular event are executed simultaneously. Contexts, which define the static elements of the model, contain carrier sets, constants, axioms, and theorems. The last two are predicates expressed within the notation of first-order logic and set theory.

B. Refinement

Event-B embeds the concept of refinement which is then the basic element of the specification development process. A refinement consists of introducing new variables and events. Abstract events are refined by strengthening their guards and adding actions to the new variables.

Variants can be introduced to guarantee that some new events (called convergent) do not prevent older ones from triggering. They are either a natural number expression which must be decreased or a set which must be made smaller, by each convergent event and not be increased by each anticipated (new but not convergent) event.

A machine can be refined into another machine which then contains a more detailed or concrete description of the model. A context can be refined (extended) into one or more contexts. It contains the static pieces of information of a model associated to the refinement. A machine can see several contexts, that is, use the names and properties it contains. A context can be seen by several machines.

[3]http://rodin-b-sharp.sf.net

C. Proofs

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, maintain the abstraction invariant, and, when appropriate, decrease variants monotonically.

Proofs also ensure that specifications meet essential system properties, such as well-definedness and invariant-preservation. The proof obligations generated by the tool are required to be discharged using provers, either automatically or interactively.

III. What to specify?

A domain specification is an abstract description of a domain. This document intends to specify the domain's intrinsic and extrinsic assumptions, protocols, properties, etc. In this section, we discuss what a typical domain model should be comprised of and how to model it in Event-B.

A. Model assumptions

A domain model is the composition of different assumptions about the particular domain. These assumptions can be classified into three categories: structural facts, behavioral laws, and enforceable properties. While always expressed as predicates on the state, these assumptions can be written in the specification either as invariants in machines, axioms in contexts, or guards in events.

Structural facts are the constants of a domain. So, they naturally fit into Event-B contexts. We use axioms to express the structural properties of the model. For instance, in transportation domain, contexts are used to define facts, such as "vehicles have bounded speed and acceleration," "a route is an acyclic sequence of adjacent paths," or $position(t+dt) = accel(t)dt^2/2 + speed(t)dt + position(t)$.

Behavioral laws are described by events and particularly by their guards. For example, as shown in Figure 1, the law of movement as displacement of vehicles to new locations is modeled with the event `travel`.

EVENT travel $\widehat{=}$
ANY vehicle, newLocation
WHERE vehicle$\in$Vehicles $\wedge$ newLocation$\in$GlobalLocations $\wedge$
newLocation$\neq$location(vehicle)
THEN location(vehicle):=newLocation
END

Figure 1. Event `travel`

Further precisions, such that departure and arrival must be stations and be connected by a route are straight refinements of the event.

Enforceable properties are properties which are necessary for a domain to be well-behaved. They are expressed by invariants. We express the absence of collision this way. Of course, the expression depends on the level of abstraction and is subjected to refinement. For instance, we define a collision on a hub as "too many vehicles on a hub at the same time." When finer movements on a path of the network are described, it is refined to the intuitive definition of "no two vehicles at the same place at the same time."

Whether a particular domain assumption should be expressed as a behavioral law or as an enforceable property is a difficult choice to make. For example, consider the absence of collisions which has been modeled by invariants. We could get the same by modeling this assumption as a special `collide` event. Formally, there is a strong relationship between the two descriptions: the guard of the hypothetical `collide` event is the negation of the invariant.

$$\neg Guard(Collide) \Rightarrow Invariant(NoCollision)$$

The choice between the two expressions depends on the kind of system one has to develop. For instance, developers of a road traffic monitoring system will likely prefer `collide` events since their system will have to deal with such situations. Developers of a traffic light control system will likely prefer invariant expressions as it is one of the goals of their system.

B. Define protocols

A domain exhibits several protocols. For instance, `travel` must be realized in a specific sequence of crossing hubs and traversing paths.

$$travel \equiv (startTravel;\ (crossHub;\ traversePath)+)$$

This reflects a protocol induced by the network topology and an implicit constraint of land transport. The protocol for crossing a hub must itself be decomposed for collision free traveling:

$$crossHub \equiv (wait*;\ enterHub;\ leaveHub)$$

So as the protocol of traversing a path which can be defined as follows:

$$traversePath \equiv (waitToEnterOnPath*;\ leaveHub;\ (waitToMoveOnPath|\ moveOnPath)*)$$

Unfortunately, Event-B does not provide us with an easy-to-use notation to model the protocols defined above with regular-expression-like formulas. We need to use the standard mechanisms of events and guards. In practice, this mean that we need explicit control predicates over these events so that they can be orderly fired with certain parameters to realize the respective protocols correctly.

Current research on CSP||Event-B [7] or atomicity decomposition [8] is promising and will lead to elegant solutions to this issue in the future. Meanwhile, we must resort to hand-coded causal orders.

We use two basic techniques: control sets and state markers. For instance, the correct sequence of `crossHub` and `traversePath` to decompose a `travel` is controlled by the set of `hubs` and `paths` that still remain to cross and traverse respectively: each event removes one easily characterized element from the set. Crossing hubs is controlled by a discrete marker associated to the vehicles with respect to hubs, where each event sets the marker:

$$vehicleState \in Vehicles \times Hubs \rightarrow \{\text{initial}, \text{entering}, \text{onHub}, \text{leaving}, \text{crossed}\}$$

Both techniques have some advantages and disadvantages. In the first approach the variant is quite easy to define but at the expense of complex computations (or definitions) of the sets. In the second technique, state markers are easy to use in guards and they make the sequence of events easy to understand. However, it is difficult to set up the variants and, generally, to connect state markers to invariants. Our longest proofs were associated with state markers.

C. Specify time

Time is an important property in high assurance domains; transportation is no exception. A difficulty with time is that it comes in several flavors, most notably discrete and continuous; transportation exhibits both. For instance, modeling travel-time requires discrete time (two readings of a clock), whereas modeling the movement on a path requires continuous time for the kinematics of vehicles. Event-B has no built-in notion of time.

In our models, we use the technique inspired by [9]. It is an instance of event-queue simulation techniques. It uses a global clock which can go only forward (see figure 2 and 3 for the abstract and refined versions), and an event-queue, `activationTime`. The introduction of time follows a regular pattern:

1) pick an event to "time"
2) add the guard
   vehicle∈dom(activationTime) ∧ time=activationTime(vehicle)
3) add the action
   activationTime := activationTime ⩤ {vehicle ↦ time + timeInc}.
   timeInc is, of course, dependent on the particular event. It can be a constant, an arbitrary value or a computation on the event queue.

EVENT ticTac ≙
ANY tic
WHERE tic∈N ∧ tic>time
THEN time:=tic
END

Figure 2. `ticTac`

EVENT ticTac **REFINES** ticTac ≙
ANY tic
WHERE activationTime≠∅ ∧
tic=min(ran(activationTime)) ∧ tic>time
THEN time:=tic
END

Figure 3. Refined `ticTac`

The technique works well both for the discrete time which we use for the abstract protocol of crossing hubs and the continuous time which we use for the kinematics. Events are

now synchronized and can only be fired at their appropriate time. Of course, we do not use real continuous time, but a discrete approximation through the choice of a small fixed tick for *timeInc*.

D. Express temporal properties

Temporal properties, such as safety and liveness, are essential to safety critical and high assurance systems. A safety property states that something bad never happens, whereas a liveness property states that something good eventually happens [10]. We need to model temporal properties in both domains and systems, but they play different roles.

A safety property is easily stated in Event-B, as long as it can be expressed as an invariant. No-collision is a typical safety property. While modeling domains, this is adequate: the proof that all events maintain the invariant is sufficient to establish that the domain is well-behaved. Of course, it would not suffice for a system: we must ensure that it is free of deadlocks and divergences.

In fact, we know that the real domain of land transportation is subjected to gridlocks (traffic jams), which are a form of local deadlocks, and, hence to a form of divergence: a vehicle can wait infinitely. So, a good model of the domain must include these facts.

In our transportation model, the gridlocks are actually modeled as special events (`lockXXX`). The guards of these events state the conditions for systems implemented to work within the domain to keep them free-flowing. Interestingly, the `lockXXX` events popped out naturally when we introduced the time: some proof obligations could not be discharged without introducing the negation of the gridlock condition.

It is to be noted that the usual techniques to guarantee deadlock-freeness and non-divergence in Event-B specifications [11] are not useful in our case. Introducing the `wait` event (`wait`'s action is `SKIP`) is essential for ensuring no-collision. This ensures that the disjunction of the guards of all events is true but makes `variant` clauses impossible. Since a vehicle can wait an indefinite period of time, we cannot prove that the `wait` event does not prevent other events from happening.

Unfortunately, it is not always possible to express finer liveness properties in Event-B. Presently, we do not know how to state that a vehicle which has started a travel will eventually reach its destination. Though not really critical for land transport, it is for air transport. For such properties, we need to resort to model checking or animation.

IV. How to refine?

As any text, a formal domain model is only the end-product of a complex thought-process. Its role is to present most of the knowledge necessary to construct software pieces which operate harmoniously and safely. However, readers trying to extract the knowledge out of this formal piece of text are often confronted with a very difficult task. Sometimes, a domain fact hides under cryptic notations, sometimes it is spread over several parts of the model, or sometimes, it is implicit. Understanding a formal text of some complexity is often equivalent to re-inventing the process of its construction and figuring out the rationale of the development steps.

Event-B, like B, embodies a process idea: formal refinement. Refinements have two roles. On the formal hand, they help master the proof of implementation correctness by breaking a big intractable proof into many manageable small ones. On the methodological hand, they help structure and organize the development process. Linking both roles together is adequate in B where the aim of the development is to produce a working code from a given specification. A refinement in B is a process step where we transform an abstract representation of a piece of information into a concrete and computable representation.

When modeling a domain, we prefer to see refinements as process steps where a new piece of information is added to the model. We find useful to classify refinements in two different categories:

1) state enrichment: a new concept or a new constraint on the state is added. Contexts are extended and the events concerned by the novelty are refined (guards and actions are changed). The structure of the specification is not changed.
2) event decomposition: a "large" event is decomposed into several "smaller" ones. The structure of the model is altered.

The second kind of refinement corresponds to a change of "observation level." Observation levels [12] are a way to provide a specification with a super-structure which eases its understanding. They reflect either the "natural" structure of the objects or the structure of the behavior. For instance, decomposition of a protocol into a sub-protocol refers to a change of observation levels. It provides us with an easy way to introduce a new property into the model. This notion is important at the methodological level.

A. Refine slowly

We advise to use small incremental steps while developing domain models. Ideally, only one new fact should be introduced per refinement.

Recording rationales for refinements is essential for later understanding of the formal text. Currently, Rodin provides only minimal abilities in this domain: simple comments. They are not well fitted for long explanations. We make use of this small-step approach to introduce the vocabulary of transportation network, for instance.

More importantly, while some facts will appear at a unique place like a new invariant or a couple of axioms, many will be scattered over the text. Behavioral laws are typical of this. For instance, understanding the protocols which prevent collisions requires to consider the guards of

two different sets of events. Expressing each type of collision by one specific refinement eases the work.

Last, expressing only one feature of the domain at a time helped us finding patterns, such as timing (cf. Section III-C), in the refinements of events. Again, this kind of regularity in the expression makes later analysis of the text easier. We should also note that small refinement steps are well supported by Rodin. They do not cost much.

B. Refine unconventionally

The observation levels should be used to organize the development rather than the implicit linear view of refinements.

Event-B has inherited from B the view that a development is a sequence of refinements. This conception is adequate in B, less so in Event-B. 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. For instance, if we wanted to introduce the notion of energy consumption, we would like to start the new feature analysis as depicted in Figure 4. From this, we could refine the notion along the observation levels and merge the resulting model with the current specification.

INVARIANT
  meter $\in$ Vehicles $\rightarrow$ int // energy meter
  energyConsumed$\in$ Vehicles $\rightarrow$ int
EVENT travel **REFINES** travel $\widehat{=}$
ANY
  vehicle, newLocation, meterReadingAtStart
WHERE
  vehicle$\in$ Vehicles $\wedge$ newLocation$\in$ GlobalLocations $\wedge$
  newLocation $\neq$ location(vehicle) $\wedge$
  meterReadingAtStart $\leq$ meter(vehicle)
THEN
  location(vehicle) := newLocation
  energyConsumed := meter(vehicle) $-$ meterReadingAtStart
END

Figure 4. Introduction of energy consumption: what we want

Instead, Event-B's flat refinement structure would force us to write the `travel` event as illustrated by figure 5 and to introduce in all other events a dummy action of the form:

$$\text{meter} \in| \text{ meter'(n)} \geq \text{meter(n)}$$

This action simply states that `meter` is susceptible to be modified by future refinements. Even if the addition of such an action does not pose any problem, it tends to clutter the text and causes distraction.

Organizing the introduction of a new feature along the observation levels (at least, one refinement per level) has several advantages:

- we make "small" steps, focusing on a small and specific set of events,
- we can relate more easily failures during the proofs to incompatibilities between the feature's definition and the existing model,

INVARIANT
  meter $\in$ Vehicles $\rightarrow$ int // energy meter
  energyConsumed$\in$ Vehicles $\rightarrow$ int
EVENT travel **REFINES** travel $\widehat{=}$
ANY
  vehicle, newLocation, r, origin, destination,
  meterReadingAtStart
WHERE
  r $\in$ routes $\wedge$
  // ...
  // 14 lines of guards
  // ...
  meterReadingAtStart $\leq$ meter(vehicle)
THEN
  location(vehicle):= newLocation
  travelTime(vehicle) := time $-$ startTime(vehicle)
  activationTime := {vehicle} $\lhd$ activationTime
  speed(vehicle) := 0
  acceleration(vehicle) := 0
  energyConsumed := meter(vehicle) $-$ meterReadingAtStart
END

Figure 5. Introduction of energy consumption: what we have

- the levels point "naturally" to the feature's facets we need to analyse.

We used this guideline when introducing the notion of time. On the first level, time is only travel-time, something which just needs a readable clock and an event to make it tick. On the second level, time is the same and nothing new is needed (no refinement). On the third level, time becomes a computable quantity and the event-queue technique is introduced. On the fourth level, time becomes continuous and needs to be discretized because of the absence of the abilities to model $\mathbb{R}$ in Rodin. The path was easy to follow but is not reflected in the sequence of refinements.

V. How to verify?

Verification of a domain amounts to assert that the specified facts, laws and properties about the particular domain are consistent, checkable and provable. A verified domain model is, therefore, considered as a consistent set of assumptions about the domain. In fact, this hypothesis does make sense as an unprovable model, of course, can not be trusted.

Refinements serve different purposes in system development than in domain modeling. In the former case, the refinement is a more concrete description of the same model. We need to prove that the new description enjoys the same functional properties. In the latter case, the refinement is an enrichment of the model. We need to show that the new feature is consistent with the previous ones. Although based on the same set of proof obligations, the proving process needs to be observed from another point of view.

A. Beware of easy proofs!

Asserting the consistency of an assumption expressed as an invariant is relatively easy. Either the invariant related proof obligations can be discharged or not. This is safe. However, when an assumption is expressed as an axiom, it is hard to prove its consistency. Proof obligations assert

well-formedness and well-typing, but not consistency. We are then always at a risk to introduce a fact which is contradicting with the rest of the model. Although hard, the ability to detect contradiction among axioms is crucial for the correctness of the model.

Unfortunately, Rodin does not warn us when axioms are inconsistent. One should always keep a skeptical eye on the proofs. If proofs become mysteriously easy to discharge, beware! The introduction of an axiom or a theorem, such as TRUE = FALSE is a useful heuristic. Success in the proof signs a contradiction, failure provides us only with reasonable assurance.

B. Beware of obvious truth!

Discharging a proof obligation using a tool may not always be possible. A proof which cannot be carried out by a prover is not always the sign of an error in the specification. In such cases, a pen-and-paper proof may work.

In Rodin, we can make "approximations" by declaring the goals as "reviewed." While legitimate in certain situations, in particular due to shortcomings of current provers, reviewing an "obvious" goal may lead to a surprise. For instance, assuming $x(y/z) = (xy)/z$ seems natural, except that this is true in $\mathbb{R}$, not in $\mathbb{N}$. Though numerically correct as the difference becomes actually negligible when numerators are much bigger than denominators, this approximation is yet formally incorrect.

One should always be careful while making approximations and going outside the tool to prove an hypothesis. Investing time in proving "the obvious truth" is worth it. Dynamic testing techniques, such as animation with realistic values, can give insight on the validity of the approximations and on the solidity of the model.

C. Use animation to complement provers

Animation allows specifiers to check the behavior of the specification by observing its execution. Non-provable safety and liveness properties can be assessed by analyzing state values and event-enabledness status. We can make solid observations on the behavior of a model by a trace analysis of the simulated scenarios.

Animation cannot show that a property always hold, but it may help to generate counter examples which show that the model is partly incorrect. Animation and model-checking play a similar role, in fact, ProB [13] offers both functions.

A very useful side-effect of animation is to help get better insights on the model. Implicit properties and unexpected behavior, either good or bad, will become apparent. We did even use it as a prototyping tool to fix the expression of tricky protocols. The execution of the specification (without translation into code) identifies right away any undesired transitions in the model's behavior.

Nevertheless, the most important role of animation is to participate in the validation of the model, i.e., to assess that the model is an adequate abstract mathematical representation of the real domain. Ideally, animation should be used after each refinement, at least, after each introduction of a new observation level. The catch is that well-written specifications often contain traits that can not be dealt with by animators. We have then developed behavior-preserving transformations [14] for some of these traits which are cost-effective enough to be used intensively.

VI. Related work

A. Domain modeling in RAISE

In recent times, Dines Bjørner's work [15], [16] is most notable in the field of formal domain modeling. He uses RAISE Specification Language (RSL) [17] for the description of domains and concentrates towards the formalization of as many domain facts as possible.

Our research differs from his work on two main fronts. First, we head towards the enrichment of domain models while paying as much attention to verification and validation as specification. Second, our concerns are also to check the capability of Event-B as a domain modeling tool and to point out and address (where possible) the issues with which we confront during this exercise. The choice of formal method can be another difference. A brief comparison of both languages, Event-B and RAISE [18], is available in [19].

B. Modeling of the transportation domain in Event-B

Previously, Event-B has been used for the development of transportation systems, see for instance [20], [21], [22], but most of the time the role of this language was limited to system modeling of a particular component. Our work is different in a sense that we are modeling the domain, where such systems are assumed to operate. The specifications of these aforementioned railway systems do contribute towards the completion of the land transport domain model, but as a part of the whole. Our model is more general and could be used for different kind of transportation systems, such as road, railways, conveyors, etc.

C. Alternate refinement mechanisms in Event-B

Linear refinements are the de facto standard of specification development in Event-B. Their counterparts, other than observation levels, are Retrenchments [23], Feature development [24] and Parallel refinements [25].

Retrenchment was proposed as a liberalization of the notion of refinement to capture more informal aspects of development within a formal framework. A retrenchment step from an abstract to a concrete level of abstraction allows strengthening of the precondition, weakening of the postcondition, and mixing of state and I/O information between the levels of abstraction by mediation of two extra predicates per retrenchment. In particular, it allows non-refinement-like behavior to be expressed via the weakened postcondition. This allows the specification of low level

details of the model without cluttering up the formal text by unnecessary code which is mandatory to discharge proofs.

Feature-oriented specification development is a mechanism to specify the behavioral variability of the model. A feature, in this case, is a simple B machine which is atomic with respect to other functionalities. The composition of several machines, each highlighting distinctive features, forms the final model. The composition of these features is still a difficult issue. The development is supported by the Rodin plugin [26].

Parallel refinements is another idea of model decomposition to handle the complexity introduced by linear refinements. In this technique, the large model is cut into several smaller components. The model decomposition is either based on shared variables [25] or on shared events [8]. The tool support is presented by [27].

Our view on refinement of Event-B models by grouping them into observation levels is different from all of the aforementioned methods. They require intricate proof obligations to measure the correctness of the model and a change in the language. Our approach can be adopted without touching the semantics of the formalism and by just providing a visual modification at the level of tool.

D. Timing and temporal properties in Event-B

The specification of timing and temporal properties in Event-B is known to be a challenging task. The expression of these properties, a key element for use of formal methods in the automotive sector, is currently non-standard in Event-B. Correctness of such specifications thus becomes an issue.

Like us, [28] reuses and adapts the timing pattern of Event-B proposed by [9]. Similarly, the pattern of Joochim et al. [29] proposes the use of global time and also interacts with a number of active times. This pattern formalizes the Timing Diagram of UML and does not address timing properties in general. In addition, its usage is recommended at abstract stages rather than in later refinements.

Abrial et al. [30] propose the use of temporal properties to model the dynamic behavior in Event-B specifications. In its continuation, [31] proposes the use of Linear Temporal Logic (LTL) for such modeling. Like others, [32] also proposes an extension of Event-B to incorporate three LTL operators: *next*, *eventually*, and *bounded eventually*. In this work, standard Event-B structures, WHEN, THEN and END are modified to represent these operators. Such models deviate from the standard Event-B notations and their verification and validation are a major challenge. The point is that expressing temporal properties is still a challenge in Event-B. Approaches based on mixing formalism, such as fusion of KAOS with Event-B [3] seems a promising solution.

VII. Conclusion and future work

In this paper, we have proposed some guidelines to use Event-B as an effective tool for formal domain modeling. Induced from our experience with transportation, they should, of course, be tested and validated on other domains. Nonetheless, we think that the notion of observation levels, the alternate viewpoint on refinement, the analysis of the role of proofs, and the importance of animation are good premises for a method of domain modeling with Event-B.

Though we have found Event-B an adequate language for domain modeling, yet there are few questions which must be answered. They are mainly about the expression of temporal properties, the tools and the validation of models.

In Event-B, we cannot straightforwardly express and prove properties such as deadlock freeness, liveness, fairness, etc. We consider this an important shortcoming of the language. At the level of tool, Rodin is still in development stages and still needs to be matured for industrial use.

The validation of models is as important as their verification. As formal refinements structure the development and verification process, we believe they can also help structure the validation process. Not all specifications are by default animatable; some needs to be transformed. Work presented in [14], [33] is a preliminary exploration of this idea.

The unprecedented level of complexity of the domain has made it quite difficult to ensure its resilience feature, the ability of the system working within to stay dependable while facing changes. Absence of these properties may lead to devastating accidents unless they are explicitly specified and proved. In future, we are contemplating to model the resilience properties of the domain.

Another challenging task is to observe how different transportation applications can be derived from this complex domain model.

References

[1] D. Bjørner, "Domain engineering," in *In BCS FACS Seminars, Lecture Notes in Computer Science, the BCS FAC Series (eds. Paul Boca and Jonathan Bowen*. Springer, 2008, pp. 1–42.

[2] A. Idani and Y. Ledru, "Dynamic Graphical UML Views from Formal B Specifications," *International Journal of Information and Software Technology*, vol. 48, no. 3, pp. 154–169, 2006, elsevier.

[3] 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)*, Orford, Canada, 2010.

[4] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[5] ——, *The B Book*. Cambridge University Press, 1996.

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

[7] S. Schneider, H. Treharne, and H. Wehrheim, "A CSP Approach to Control in Event-B," in *Integrated Formal Methods - IFM 2010 Integrated Formal Methods - 8th International Conference, IFM 2010*, ser. LNCS, vol. 6396. Nancy France: Springer Berlin / Heidelberg, 2010, pp. 260–274.

[8] M. Butler, "Decomposition Structures for Event-B," in *Proceedings of the 7th International Conference on Integrated Formal Methods*, ser. IFM '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 20–38.

[9] D. Cansell, D. Mery, and J. Rehm, "Time Constraint Patterns for Event B Development," in *7th International Conference of B Users*, ser. LNCS, J. Julliand and O. Kouchnarenko, Eds., vol. 4355. Springer Verlag, 2007, pp. 140–154.

[10] L. Lamport, "Proving the Correctness of Multiprocess Programs," *IEEE Transactions on Software Engineering*, vol. 3, no. 2, pp. 125–143, 1977.

[11] D. Yadav and M. Butler, "Verification of Liveness Properties in Distributed Systems," in *Second International Conference on Contemporary Computing (IC3'09)*. Noida, India, 2009, pp. 625–636.

[12] A. Mashkoor and J.-P. Jacquot, "Domain Engineering with Event-B: Some Lessons We Learned," in *18th International Requirements Engineering Conference (RE'10)*, Sydney, Australia, 2010.

[13] M. Leuschel and M. Butler, "ProB: A Model Checker for B," in *FME 2003: Formal Methods*, ser. LNCS 2805, K. Araki, S. Gnesi, and D. Mandrioli, Eds. Springer-Verlag, 2003, pp. 855–874.

[14] A. Mashkoor, "Formal Domain Engineering: From Specification to Validation," Ph.D. dissertation, Université Nancy II, Jul. 2011. [Online]. Available: http://tel.archives-ouvertes.fr/tel-00614269/en/

[15] D. Bjørner, "Development of Transportation Systems," in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA)*, 2007.

[16] ——, "Train: The railway domain - a "grand challenge" for computing science & transportation engineering," in *Building the Information Society, IFIP 18th World Computer Congress, Topical Sessions, Toulouse, France*, 2004, pp. 607–612.

[17] *Specification Case Studies in RAISE*. London, UK: Springer-Verlag, 2002.

[18] T. R. L. Group, *The RAISE specification language*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

[19] A. Mashkoor and J.-P. Jacquot, "Utilizing Event-B for Domain Engineering: A Critical Analysis," *Requirements Engineering*, vol. 16, no. 3, pp. 191–207, 2011.

[20] A. Papatsaras and B. Stoddart, "Global and Communicating State Machine Models in Event Driven B: A Simple Railway Case Study," in *ZB '02: Proceedings of the 2nd International Conference of B and Z Users on Formal Specification and Development in Z and B*. London, UK: Springer-Verlag, 2002, pp. 458–476.

[21] M. Butler, "A System-Based Approach to the Formal Development of Embedded Controllers for a Railway," *Design Automation for Embedded Systems*, vol. 6, pp. 355–366, 2002.

[22] D. Essamé, "Handling Safety Critical Requirements in System Engineering Using the B Formal Method," in *Computer Safety, Reliability, and Security*, ser. Lecture Notes in Computer Science, M. Heisel, P. Liggesmeyer, and S. Wittmann, Eds. Springer Berlin / Heidelberg, 2004, vol. 3219, pp. 115–115.

[23] R. Banach and M. Poppleton, "Retrenchment: An Engineering Variation on Refinement," in *Proceedings of the Second International B Conference on Recent Advances in the Development and Use of the B Method*. London, UK: Springer-Verlag, 1998, pp. 129–147.

[24] M. R. Poppleton, "Towards Feature-Oriented Specification and Development with Event-B," in *Proceedings of the 13th international working conference on Requirements engineering: foundation for software quality*, ser. REFSQ'07, 2007, pp. 367–381.

[25] J.-R. Abrial and S. Hallerstede, "Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B," *Fundamenta Informaticae*, vol. 77, no. 1-2, pp. 1–28, 2007.

[26] 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, pp. 18–25.

[27] 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.

[28] J. W. Bryans, J. S. Fitzgerald, A. Romanovsky, and A. Roth, "Patterns for Modelling Time and Consistency in Business Information Systems," in *Proceedings of the 2010 15th IEEE International Conference on Engineering of Complex Computer Systems*, ser. ICECCS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 105–114.

[29] T. Joochim, C. Snook, M. Poppleton, and A. Gravell, "Timing Diagrams Requirements Modeling Using Event-B Formal Methods," in *IASTED International Conference on Software Engineering (SE2010)*, Innsbruck, Austria, 2010.

[30] J.-R. Abrial and L. Mussat, "Introducing Dynamic Constraints in B," in *B '98: Proceedings of the Second International B Conference on Recent Advances in the Development and Use of the B Method*. London, UK: Springer-Verlag, 1998, pp. 83–128.

[31] J. Groslambert, "Verification of LTL on B Event Systems," in *B 2007: Formal Specification and Development in B*, ser. Lecture Notes in Computer Science, J. Julliand and O. Kouchnarenko, Eds. Springer Berlin / Heidelberg, 2006, vol. 4355, pp. 109–124.

[32] J. Bicarregui, A. Arenas, B. Aziz, P. Massonet, and C. Ponsard, "Towards Modelling Obligations in Event-B," in *Proceedings of the 1st international conference on Abstract State Machines, B and Z*, ser. ABZ '08. Springer-Verlag, 2008, pp. 181–194.

[33] A. Mashkoor, J.-P. Jacquot, and J. Souquières, "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.