Domain Modeling with Event-B: An Experience with Transportation Domain

Atif Mashkoor, Jean-Pierre Jacquot, Jeanine Souquières

Domain Modeling with Event-B: An Experience with Transportation Domain [⋆]

Atif Mashkoor, Jean-Pierre Jacquot, Jeanine Souquières

LORIA – DEDALE Team – Nancy Université
Campus Scientifique
F-54506, Vandoeuvre-Lès-Nancy, France
{firstname.lastname}@loria.fr

Abstract. This paper presents preliminary results of utilization of Event-B for domain modeling. The development of new urban transport systems, based on autonomous vehicles, strongly requires the formal description of land transportation domain. Certification, safety, or security, for instance, crucially depend upon formal assessment that systems meet the required properties and constraints of the domain. Though Event-B has not been designed for domain modeling, yet, it was discovered that its notions of events and non determinism are well suited to formalize a domain with many autonomous agents. Refinements and systematically constructed proof obligations work well in this context, but we also need other operations in the modeling process, such as, "abstraction leaps" which have been introduced as a part of domain specification.

1 Introduction

Having good understanding of an application domain is a crucial prerequisite to develop software within that domain [1]. The understanding of domain is referred to as domain model. A domain model is a conceptual model of a system which describes various entities, phenomena and their inter-relationships, along with their important static and dynamic properties, related to a particular domain. The domain model may be expressed in the form of requirements, specifications, or architectural references. Generally, a domain model is used to verify and validate the understanding of a problem domain captured by different stakeholders of the domain [2] [3].

In order to be useful for software development and reusability, domain models should be captured and specified in some systematic way [4]. One systematic procedure to specify domain models is the use of formal methods. Though formal methods seem to be tiresome because of their time consumption and extensive manual efforts, yet, studies have proved their cost effectiveness as they minimize defects in earlier stages of software development [5].

According to [6], if domain models and requirements of software are not formally expressed, software correctness can not be meaningfully achieved. Safety is also one of

[⋆] This work has been partially supported by the French National Research Agency TACOS project, ANR-06-SETI-017 (http://tacos.loria.fr) and by the pôle de compétitivité Alsace/Franche-Comté CRISTAL project (http://www.projet-cristal.net).

the major factors which can not be overlooked while designing complex and critical systems. The development of correct and safe systems can be difficult and error prone with traditional software development methods. However, use of formal methods, in order to ensure their correctness and to structure their development from domain modeling to implementation, can significantly help system development.

B [7] is a formal method, having good tool support, which allows a stepwise development of modeling and reasoning about sequential programs. It has proved its strength in industry with the development of complex real-life applications in transportation domain, such as, the Roissy VAL [8]. Event-B [9] is an evolution of the B method, providing new mechanisms for developing large reactive and distributed systems. We already have an experience [10] of using Event-B for specification and development of situated Multi-Agents Systems and its application to platooning problem. Some other related work has been presented in [11,12] and [13] to demonstrate and refine the techniques of specification and safety analysis for the case study on automated freeways.

In this paper, we propose the use of Event-B for domain modeling, even though it has not been designed for this purpose, yet, we think, the technique of expression of abstract model based systems and their refinements using events is also suitable for domain modeling. We apply this technique to the transportation domain for the development of a new type of urban vehicle.

This paper is organized as follows. Section 2 provides an overview of the Event-B approach. Section 3 describes main entities necessary for transportation domain. Section 4 provides a stepwise Event-B specification of this domain. Section 5 presents the hierarchy of the Event-B model with three levels of abstraction. Finally, sect. 6 presents some preliminary lessons learned from this experience of formal domain modeling and suggests further directions for future work.

2 Basic Concepts of Event-B

Event-B [9] is a formal language for modeling and reasoning about systems. It is an evolution of classical B [7] for developing reactive and distributed systems. The Rodin platform [1] provides the tool support for Event-B.

Like classical B, the concept of refinement is the heart and soul of Event B. Systems are incrementally refined from abstract models to more concrete models while preserving correctness. The proof based development paradigm ensures the model correctness because each development includes proof obligations for invariants and refinement.

In an Event-B model, data is distributed between `contexts` and `machines` depending upon the nature of the data. A context holds the static data of the model. It is composed of `sets`, `constants`, `axioms`, and `theorems`. A machine holds the dynamic data of the model. In a machine `variables`, `invariants`, `variants`, `theorems`, and `events` are specified. Proof obligations are automatically generated to ensure the consistency of the model related to each context and machine.

The events in Event-B substitute the operations of classical B. Unlike operations, events are not called but observed. An event consists of `guards` and a `body`. When

[1] RODIN(Rigorous Open Development Environment for Complex Systems) is an EU funded research project IST511599 (http://rodin.cs.ncl.ac.uk)

the guards are evaluated to true, the event is triggered; it modifies the system state by executing actions on state variables. When the guards of several events are evaluated to true, choice of triggering event is non-deterministic. `Initialization` is an event that must appear into each Event-B machine. This event is used to initialize the state variables of the machine.

Both machines and contexts can be refined. A context can extend (refine) several contexts. Machines can see contexts, which allow events to refer to the static data of the model. A machine can refine only one other machine. An abstract event can be refined into several concrete ones; several abstract events can be merged into one concrete event.

The idea of decomposition allows the splitting of an Event-B model into smaller components to manage the increasing complexity of the design. The variables of the model are divided into external and internal variables. The internal variables only correspond to one component whereas external variables can be manipulated by all components. Events referring only to the internal variables of a component are placed in the same component. Other events which refer both internal and external variables appear in the component which reference the internal variables.

3 Transportation Domain Model

This work is part of the TACOS and the CRISTAL projects which are concerned about the development of a new type of urban transport systems based on autonomous vehicles, known as `CyCabs` [14].

The term "transportation" refers to the movement of people or goods by vehicles from locations to locations [6]. Many important concepts appear in this definition of transportation, which must be addressed during the transportation domain description, such as vehicles, locations, movement, and etc. Following are the intrinsic concepts of the transportation domain which relate to the movement of vehicle:

3.1 Locations

Each vehicle captures a location at any given moment. In this work, a location refers to a logical localization of a vehicle in contrast to its physical localization on geographical coordinates.

3.2 Nets, Hubs and Connections

We assume a transportation system which consists of several interconnected networks. One network is constituted of a set of hubs connected through connections. A hub is an abstraction of a place where a vehicle can stop such as stations, junctions, and etc. Connection is an abstraction of a directed road link between two hubs.

3.3 Junctions and Stations

We consider two kinds of hubs: junctions and stations. Junctions model road intersections and other crossings where safety requires special attention. Stations models places where passengers can come on and off vehicles, and where vehicles can be parked. Stations, and not junctions, are valid destinations of a travel.

3.4 Paths and Routes

Operative connections between hubs are built on top of a physical network. At a physical level, adjacent hubs are connected through paths, which are, in fact, directed edges connecting these vertices. A connection will then often be realized as a sequence of paths, which is called a route. Figure 1 shows a net with hubs and paths (directed connections).

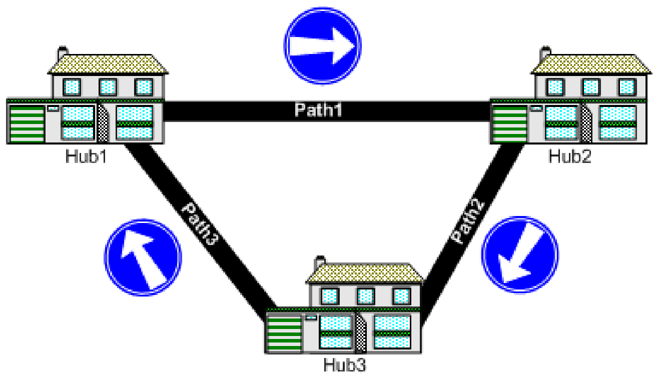

Fig. 1. A Net with Hubs and paths

3.5 Properties

Beside the general description of a road network, the desired properties of a transportation domain are also to be specified. In this paper, we consider the collision avoidance property. The term collision refers to the situation when two vehicles are at the same place at the same time. Until now, we have only considered collisions at intersections (which, by the way, represent the majority of collisions in real life). To express the non-collision property, we introduce the notion of capacity of a hub, which represents the maximum number of vehicles which can share the hub simultaneously. A collision occurs if load of the hub exceeds its capacity. This same notion of capacity also enables us to model the fact that station can only admit a finite number of vehicles.

4 Stepwise Development of the Model

Now the objective is to model the movement of a vehicle from one location (origin) to another (destination). The model is specified into three levels of abstraction. Level 1 specifies the traveling of a vehicle from one hub (origin) to another (destination). Level 2 states that, in order to travel from origin to destination, a vehicle must traverse all intermediate hubs and connections thus decomposing the travel event into `crossHub` and `traversePath` events. Level 3 specifies the `crossHub` event into more details and further decomposes it into `enterHub`, `leaveHub` and `wait` events. These levels of abstraction are shown by Fig. 2. Notice that `traversePath` event is not elaborated at this stage as only hub locations of vehicle are being considered for this work.

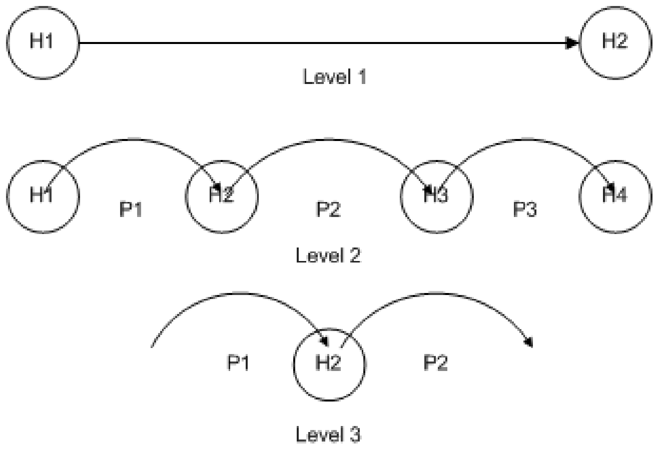

Fig. 2. Three Levels of abstraction

These three levels of abstraction have been specified using the stepwise development technique of Event-B. Each abstraction level may be comprised of one or more refinement steps. The concepts of abstraction and refinement should not be confused with each other. Refinement is a technical Event-B process in which each step adds further details to the model until it is meaningfully refined, however abstraction refers to the thought process in which initially a global view of some observable phenomenon is envisioned and later more details to the phenomenon are integrated with the passage of time. Following are the series of Event-B refinements.

4.1 Initial Model

The basic definition of movement is "change in position". Machine `Movement0`, shown by Fig. 3, specifies this fact in an abstract event, known as `travel`, in which a vehicle changes its current position to a new position. The position of a vehicle is specified with a variable `location`. An invariant is specified over this variable which states that a vehicle can hold any `GlobalLocation` over a net. As this is an abstract event, so, it does not specify any further information about the movement of a vehicle. This machine sees the context `StartState`, shown by Fig. 4, which initializes locations of vehicles.

```
MACHINE Movement0
SEES
StartState
VARIABLES
location
INVARIANTS
inv1 location ∈ Vehicles → GlobalLocations
EVENTS
INITIALISATION ≙
BEGIN
act1 location := startVehicleLocation
END
travel ≙
ANY
vehicle
newLocation
WHERE
grd1 vehicle ∈ Vehicles
grd2 newLocation ∈ GlobalLocations
grd3 newLocation ≠ location(vehicle)
THEN
act1 location(vehicle):=newLocation
END
END
```

Fig. 3. Initial machine

```
CONTEXT
StartState
EXTENDS
Location
CONSTANTS
startVehicleLocation
AXIOMS
type1 startVehicleLocation ∈
        Vehicles → GlobalLocations
END
```

Fig. 4. Context StartState

To specify the general transportation domain constructs, such as, nets, hubs, and connections, a context `Net`, shown by Fig. 5, is also modeled. Three types of axioms have been used in this context: typing axioms, which set the type of the constants, property axioms, which specify the constraints and properties of the constants, and technical axioms, which specify the technical axioms required for proof obligations.

A net consists of several hubs and connections. This fact is represented in the model for a hub with the help of a relation between a set of hubs and nets (type1) and for a connection with the help of a total function between a set of connections and nets (type2). A hub may be shared among several nets (prop7), however, a connection always belongs to only one net (prop6). `ConnectedHubs` is a constant which specifies those two hubs which are connected to each other by a particular connection (type3). This relation is further decomposed into two distinct relations i.e., `connectionOrigin` and `connectionDestination`. Former (type5) describes the origin hub for the connection and later (type6) refers to the destination of the connection.The constant `hubConnection` (type4) refers to the fact that each hub has one or more connections attached to it i.e., disconnected hubs are not possible in domain. It is also stated that both, hub and connection, linked to each other, should belong to the same net (prop3). Reflexive connections are not permitted in the domain so both the origin and destination hubs of the connections should be different (prop4). A hub might be shared between two or more nets (type7). Each net is consisted of at least two hubs and one connection (prop10).

4.2 First Refinement

The goal of this first refinement step is to add more constraints to the basic `travel` event: a movement is now seen as as a change in position from one hub to another. Since in our model only hubs can be the starting and ending points of the vehicles, so in our opinion, this is the next logical step we should take. Now, it is specified that the vehicle moves from one hub to another. The vehicle changes its location from origin hub to destination hub as a consequence of this event. Figure 6 shows the refined travel event.

4.3 Second Refinement

Vehicles in transportation domain follow a route to reach their destination from the origin. This fact is the basis to introduce this refinement step. In this refinement, it is assumed that starting position of the vehicle is somewhere on the origin of the route and destination of the vehicle is the last hub of the route. In other words, a vehicle is moving along a route in this refinement from origin to destination hub. Figure 7 shows the added refinement in the travel event.

To facilitate movement of vehicle anywhere on a net following a route, another context has been introduced at this level. This context `Net2` specifies the concept of paths and routes. Route, which is also a type of connection, is specified as a sequence of paths. All sequences of paths are not routes, but only those sequences whose both `connectionOrigin` and `connectionDestination` are stations and all intermediate hubs and connections can be traversed in order to reach destination.

4.4 Third Refinement

This refinement corresponds to the second level of abstraction of the model. Here, a model is witnessing an abstraction leap. Previous refinement stated that a vehicle travels from origin hub to destination hub following a route. Naturally, during a route, a vehicle must traverse many intermediate hubs and paths, in order, to reach its destination. Thus, modeling the phenomena of crossing hubs and traversing paths is the main reason for this abstraction leap because these phenomena add further more details to our thought process of how a vehicle travels.

In this refinement, three new events `crossHub`, `traversePath`, and `startTravel` are added. The rationale for introducing the events `traversePath` and `crossHub` is to facilitate the `travel` event in crossing hubs and traversing paths. `CrossHub` and `traversePath` are the convergent events and are decompositions of the `travel` event.

```
CONTEXT
Net
SETS
Nets
Hubes
Connections
CONSTANTS
obsNetHubs
obsNetConnections
connectedHubs
hubConnections
connectionOrigin
connectionDestination
areConnectedHubs
isSharedHub
AXIOMS
tech1   finite (Nets)
tech2   finite (Hubs)
tech3   finite (Connections)
type1  obsNetHubs ∈ Hubs ↔ Nets
type2  obsNetConnections ∈ Connections → Nets
prop1  dom(obsNetHubs) = Hubs ∧ ran(obsNetHubs) = Nets
prop2  dom(obsNetConnections) = Connections ∧ ran(obsNetConnections) = Nets
type3  connectedHubs ∈ Connections → Hubs × Hubs
type4  hubConnections ∈ Hubs → P1(Connections)
type5  connectionOrigin ∈ Connections → Hubs
type6  connectionDestination ∈ Connections → Hubs
prop3  ∀c · c ∈ Connections ⇒ obsNetConnections[{c}] ⊆ obsNetHubs[{connectionOrigin(c)}]
       ∧ obsNetConnections[{c}] ⊆ obsNetHubs[{connectionDestination(c)}]
prop4  ∀c · c ∈ Connections ⇒ connectionOrigin(c) ≠ connectionDestination(c)
prop5  ∀h,c · h ∈ Hubs ∧ c ∈ hubConnections(h) ⇒ obsNetConnections[{c}]⊆obsNetHubs[{h}]
prop6  ∀c· c ∈ Connections ⇒ card(obsNetConnections[{c}])=1
prop7  ∀h· h ∈ Hubs ⇒ card(obsNetHubs[{h}])≥1
type7  isSharedHub ∈ Hubs → 𝔹
prop8  ∀h· h ∈ Hubs ∧ card(obsNetHubs[{h}])>1 ⇒ isSharedHub(h) = TRUE
prop9  ∀h· h ∈ Hubs ∧ card(obsNetHubs[{h}])=1 ⇒ isSharedHub(h) = FALSE
tech4   finite (obsNetConnections)
tech5   finite (obsNetHubs)
prop10 ∀n· n ∈ Nets ⇒ card(obsNetHubs^{-1}[{n}])≥2 ∧ card(obsNetConnections^{-1}[{n}])≥1
type8  areConnectedHubs ∈ Hubs × Hubs → 𝔹
END
```

Fig. 5. Context Net

travel $\widehat{=}$
REFINES
travel
ANY
vehicle
newLocation
WHERE
grd1 vehicle $\in$ Vehicles
grd2 newLocation $\in$ hubLocations
grd3 newLocation $\neq$ location(vehicle)
grd4 areConnectedHubs(obsHubLocations^{-1}(location(vehicle))
    $\mapsto$obsHubLocations^{-1}(newLocation))=TRUE
THEN
act1 location(vehicle):=newLocation
END

Fig. 6. First refinement

position(vehicle) $\in$ obsHubLocations(connectionOrigin(r(1))) $\wedge$
newPosition $\in$ obsHubLocations(connectionDestination(r(card(r))))

Fig. 7. Second refinement

A variant `card(hubsToCross)+card(connectionsToTraverse)` is also specified to prevent these events from happening forever. Event `startTravel`, on the other hand, sets the starting conditions for travel.

The `traversePath` event, shown by Fig. 8, is expressed with the help of the following guards: The vehicle is currently positioned on the origin hub of a path and that hub has already been crossed. The destination hub of the path has not already been crossed. The path connecting both hubs is the member element of the route the vehicle needs to travel and has not already been traversed by the vehicle. If all of these guards are true then the position of the vehicle can be set as destination hub of the path and the path can be subtracted from the `connectionsToTraverse` variable.

The specification of the `crossHub` event is trivial. Following guards can be set in order to trigger this event: The vehicle should currently be positioned on the hub which it needs to cross and, moreover, that hub should not already be crossed by this vehicle. If these guards are true then this hub can be subtracted from the list of those hubs which need to be crossed by this vehicle. Figure 9 shows the event `crossHub`.

Notice that a variable `position` has replaced the variable `location` in these events. Though both variables represent the same phenomenon yet they are specified with different variables because they belong to different levels of abstraction. At abstract level `location` refers to the initial and final position of the vehicle and at lower level of abstraction the variable `position` is used to represent the intermediate position of the vehicle. This point will be discussed in more details in observation section.

```
traversePath ≙
ANY
vehicle
r
p
newPosition
WHERE
grd1 vehicle ∈ Vehicles
grd2 r ∈ routes
grd3 p ∈ paths ∧ p ∈ ran(r)
grd4 position(vehicle) ∈ obsHubLocations(connectionOrigin(p))
grd5 newPosition ∈ obsHubLocations(connectionDestination(p))
grd6 newPosition ≠ position(vehicle)
grd7 vehicle↦p ∈ connectionsToTraverse
grd8 vehicle↦connectionOrigin(p) /∈ hubsToCross
grd9 vehicle↦connectionDestination(p) ∈ hubsToCross
THEN
act1 position(vehicle) := newPosition
act2 connectionsToTraverse := connectionsToTraverse∖{vehicle↦p}
END
```

Fig. 8. Event traversePath

```
crossHub ≙
ANY
vehicle
hub
WHERE
grd1 vehicle ∈ Vehicles
grd2 hub ∈ Hubs
grd3 position(vehicle) ∈ obsHubLocations(hub)
grd4 vehicle↦hub ∈ hubsToCross
THEN
act1 hubsToCross := hubsToCross∖{vehicle↦hub}
END
```

Fig. 9. Event crossHub

4.5 Fourth Refinement

This fourth refinement corresponds to the third level of abstraction. The rationale for this abstraction leap is to specify the `crossHub` event at a fine-grained level. In fact, we want to express the collision avoidance property for vehicles and for this, we control the entrance of vehicles to the hub. In order to do so, the `crossHub` event is decomposed into three further events: `enterHub`, `leaveHub` and `wait`. To facilitate the specification of these events a new invariant is specified known as `hubLoad`. `HubLoad` indicates current load of vehicles on a hub. It is specified that each hub is capable to host a certain number of vehicles simultaneously which is given by a constant `hubCapacity`. The variant `hubLoad` guarantees that a vehicle would not enter into a hub, if a hub is already full.

An event `enterHub` (Fig. 10) is triggered when a vehicle enters into a hub whose `hubLoad` is lower than its capacity and is also not previously traversed by the vehicle. Once the event is triggered, the load of the hub is increased according to the number of vehicles. A vehicle must not enter into the hub if the load of the hub is equal to its capacity. In this scenario, it must wait for its turn. The event `wait` (Fig. 11) shows this case.

```
enterHub ≙
ANY
vehicle
hub
newPosition
WHERE
grd1 vehicle ∈ Vehicles
grd2 hub ∈ Hubs
grd3 position(vehicle)=obsHubLocations(hub)
grd4 vehicle ↦ hub ∈ hubsToCross
grd5 newPosition ∈ obsHubLocations(hub)
grd6 hubLoad(hub)<hubCapacity(hub)
THEN
act1 position(vehicle) := newPosition
act2 hubLoad(hub) := hubLoad(hub) + 1
END
```

Fig. 10. Event enterHub

```
Wait ≙
ANY
vehicle
hub

WHERE
grd1 vehicle ∈ Vehicles
grd2 hub ∈ Hubs
grd3 position(vehicle)=obsHubLocations(hub)
grd4 vehicle↦hub ∈ hubsToCross

grd6 hubLoad(hub)≥hubCapacity(hub)
THEN
act1 position(vehicle):=position(vehicle)

END
```

Fig. 11. Event wait

The event `leaveHub`, as shown by Fig. 12 triggers when a vehicle leaves a hub which is already crossed by it. Upon the triggering of this event, the load of the hub is decreased according to the number of vehicles which leave the hub.

```
leaveHub $\widehat{=}$
ANY
vehicle
hub
newPosition
WHERE
grd1 vehicle $\in$ Vehicles
grd2 hub $\in$ Hubs
grd3 position(vehicle) $\in$ obsHubLocations(hub)
grd4 vehicle$\mapsto$hub $\notin$ hubsToCross
grd5 newPosition $\notin$ obsHubLocations(hub)
grd6 position(vehicle)$\neq$newPosition
THEN
act1 position(vehicle) := newPosition
act2 hubLoad(hub) := hubLoad(hub)$-$1
END
```

Fig. 12. Event leaveHub

5 Event-B Hierarchy

Figure 13 presents an Event-B hierarchy of the domain model. It contains all the contexts and machines specified in the model. Important point to be noticed about the hierarchy is its three column format and indication of abstraction leaps.

The first column of the hierarchy shows the series of refinements of machines. These machines build the main construct of the model. Second column depicts the refinements of `StartState` contexts. These contexts contain the starting values of the variables used in the machines and hence are deployed for initialization purposes. Third column of the hierarchy is constituted of series of refinements of `Net` context. These contexts specify the supporting infrastructure for the `travel` event.

Figure 13 also indicates three different levels of abstraction. The first four rows of the hierarchy belongs to the first level of abstraction and the fifth and sixth rows belong to the second and third levels of abstraction respectively. The abstraction leaps are indicated with line separators. Whenever abstraction leap or refinement demands introduction of new supporting infrastructure, a new refinement of context `Net` and context `StartState` is added to the model.

Following is the list of proof obligations which were generated to validate and to ensure the correctness of the specification:

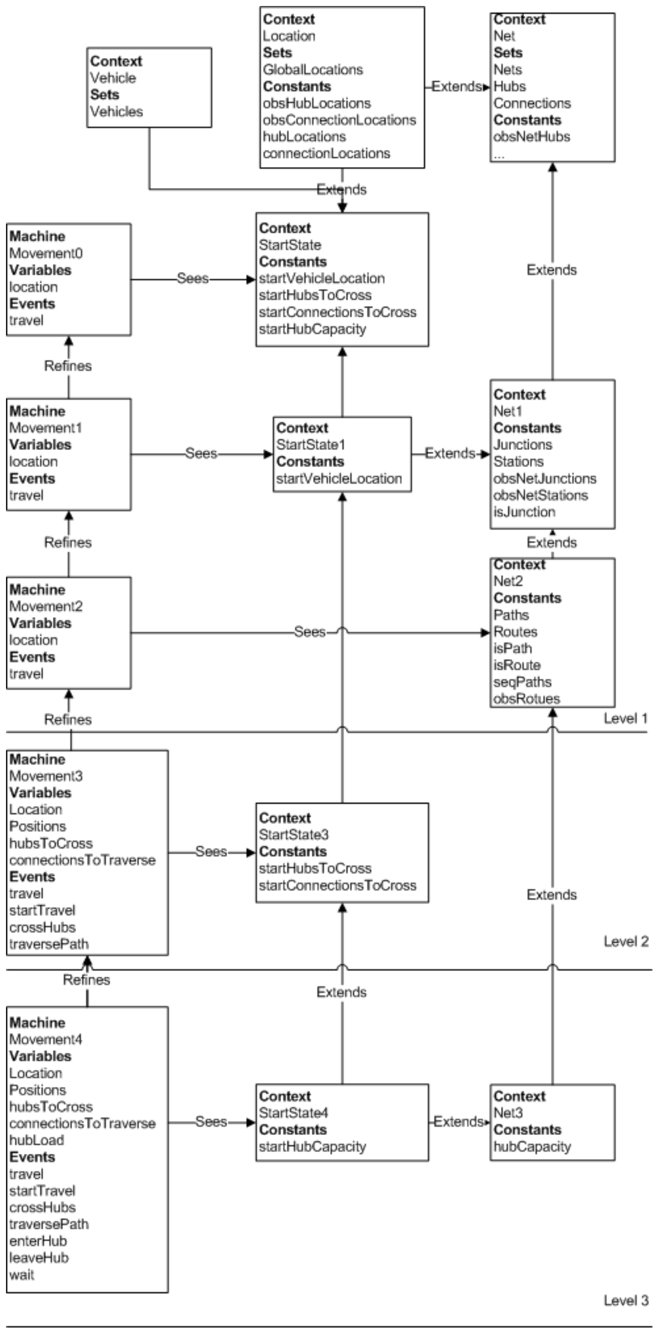

Fig. 13. The Event-B hierarchy

	Total	Automatic	Manual
Net	8	7	1
*Net*1	4	4	0
*Net*2	5	5	0
*Net*3	2	2	0
Vehicle	0	0	0
Location	0	0	0
StartState	3	3	0
*Machine*0	3	3	0
*Machine*1	1	1	0
*Machine*2	4	3	1
*Machine*3	15	14	1
*Machine*4	26	25	1

Naturally, the number of proof obligations increases when more details are added to models in refinement steps. The majority of proof obligations were automatically discharged by the prover and rest were discharged interactively.

6 Preliminary lessons

Domain modeling differs from system specification in several ways: we must express, straightforwardly, many properties which are part of the existing world, we refine less towards concrete implementation than towards finer analysis of properties, and we have a different criterion for stoping refinement.

Since Event-B and Rodin were not designed for domain modeling, so, we expect to confront with difficulties when using them for this task, hoping for revelation of interesting issues. Though, our work is still in its infancy stages, yet, some questions have already been raised regarding the tool, the language, and the formalization process.

Rodin has some minor annoyances which slow us down, for example, the number of straightforward proofs which require intervention from the user. Although they are discharged in just one or two clicks but this is distracting and time consuming. It also implies that after a crash (not frequent, but they happen) everything is to be typed again.

The major problem we have experienced with Rodin concerns inconsistent axioms. At present, Rodin does not warn when a model is logically inconsistent. We discover it when proofs become suspiciously easy to carry out. Although that may be difficult to implement, yet, we would appreciate if Rodin raises a flag when an inconsistent model appears. Our current solution to this problem is to include a theorem such as `TRUE = FALSE` at context levels. A successful proof is a sure indication of inconsistency[2]. The detection of inconsistencies is a crucial problem when modeling domains. Contexts are used to represent the fixed structures and properties of a real situation. So, they contain a rich but intricate set of logical formulas. As seen on Fig.5, the simple statement, that transportation networks are graphs with minimal properties, requires more than 20 axioms which raises the probability of introducing subtle inconsistencies.

We deal with some issues regarding the Event-B language as well. The presentation of axioms as a long list, in Event-B, is inconvenient. While this presentation makes sense at the mathematical level, it has drawbacks at the usability level. Looking at the axioms, we can see three categories: typing axioms, technical axioms (non emptiness,

[2] Note that the theorem should be eliminated from the context afterwards, otherwise, even if not proved, it will introduce an inconsistency in the machines that see it.

finiteness, definition of standard structures, etc.), and the real properties. Since the work of specifier is mainly about "fixing" the expression of the properties, so, it would be easier if the corresponding axioms could be visually isolated from others.

The lack of sequences, and, more generally, of the few usual algorithmic structures, implies they must be redefined each time they are required. This, in turn, makes the axioms list grow longer.

Our modeling process uses two different operations for specification evolution: the usual refinements and the abstraction leaps. The latter can be seen as the decomposition of a large event as a sequence of smaller ones. For instance, `travel` is decomposed as a sequence of `traversePath` an `crossHub`. Technically, there is no refinement relationship between these events, however conceptually, the abstraction leaps allow us to introduce a hierarchical structure into the specifications. We define which small events are decompositions of a large event. This hierarchical decomposition raises several issues:

– All the notions should be formally redefined at the new level, even if they are intuitively same. For instance, though the position of a vehicle on a network is the same notion at all three levels of our model, yet we need to introduce three different formal functions to model it having exactly same type. Although, the oblivion to redefine something will be caught through the impossibility to discharge a proof obligation, yet, it would be nice to have some support from the language or the tools to remind us.
– Determining the correctness of decomposition is difficult. For a decomposition to be correct, we need to be sure that a sequence of small events is equivalent to the large event in the sense that the latter is observed after a definite sequence of the former has been triggered. At present, we use the notions of variants and convergent events. They allow us to prove that no event will be observed indefinitely, but they do not allow us to prove that the certain event of the sequence will be observed. Also, it should be noted that we had to introduce non convergent events: `startTravel` and `wait`. The first event is an initialization, technically necessary for the expression of the guards and intuitively motivated by the modeling of the situation of "starting a journey". `Wait` is actually necessary to represent a reality of the domain: when two vehicles are in proximity of collision at an intersection, one has to stop. A formal property we may like to express is that the number of `wait` events is finite, which cannot be done in a simple way.
– Is it advisable to state all properties of a model as a provable logical formula? This question was raised by the finite waiting constraint we mentioned above. By introducing such a constraint in the domain at the development level we have reached, we may do much more than what we intend. In fact, we may imply the existence of a super-controller of the domain which has a knowledge of the global state of the domain. This is fine for domains such as rail transportation systems where a global supervisor is assumed from the beginning. This is fine also for domains such as air transportation systems where the vehicles have a very limited waiting capacity: a global control system is a logical necessity. This is not fine for a domain such as road transportation. There, each vehicle is assumed to act according to a limited perception of the local environment. The point is that we should be careful about

the nature of the properties we express. For some, the proof techniques of event-B are adequate; we should prove that the domain model is collision-free! For others, probabilistic or optimization techniques may be more appropriate.

Until now, the general philosophy of event-B has been well suited to our purpose. The notions of events and non determinism allow us easy modeling of independent vehicles without any assumption other than their common property: they move. The strong safety constraint we have considered is also easily modeled. This was done through standard refinement techniques. We are thus encouraged to proceed further.

As a future work, we intend to include several types of vehicles. In particular, we want to study how the platoons of vehicles impact our model. Another direction is the introduction of the dynamic properties of vehicles, such as, other kinds of collisions, oscillation, travel time, and etc.

References

1. Curtis, B., Krasner, H., Iscoe, N.: A field study of the software design process for large systems. Communications of the ACM **31**(11) (1988) 1268–1287
2. Neighbors, J.M.: Software construction using components. PhD thesis (1980)
3. Debaud, J.M., Rugaber, S.: A software re-engineering method using domain models. In: 11th IEEE International Conference on Software Maintenance (ICSM). (1995)
4. Arango, G.: A brief introduction to domain analysis. In: Proceedings of the ACM symposium on Applied computing (SAC). (1994) 42–46
5. Pressman, R.S.: Software Engineering: A Practitioner's Approach. McGraw-Hill Inc (2005)
6. Bjørner, D.: Development of transportation systems. In: International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA). (2007)
7. Abrial, J.R.: The B Book. Cambridge University Press (1996)
8. Badeau, F., Amelot, A.: Using B as a high level programming language in an industrial project: Roissy VAL. In: ZB 2005: Formal Specification and Development in Z and B, 4th International Conference of B and Z Users. Volume 3455 of LNCS., Springer-Verlag (2005) 334–354
9. Abrial, J.R., Hallerstede, S.: Refinement, decomposition, and instantiation of discrete models: Application to Event-B. Fundamenta Informaticae **77**(1-2) (2007) 1–28
10. Lanoix, A.: Event-B specification of a situated multi-agent system: Study of a platoon of vehicles. In: 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE), IEEE Computer Society (2008) 297–304
11. Hitchcock, A.: A Specification Of An Automated Freeway With Vehicle-Borne Intelligence. Research Report UCB-ITS-PRR-92-18, California Partners for Advanced Transit and Highways (PATH) (1992)
12. Hitchcock, A.: Methods Of Analysis Of IVHS Safety. Research Report UCB-ITS-PRR-92-14, California Partners for Advanced Transit and Highways (PATH) (1992)
13. Tsao, H.S.J., Hall, R.W., Shladover, S.E.: Design Options for Operating Automated Highway Systems. In: Proc. IEEE-IEE Vehicle Navigation & Information System Conference. (1993) 494–500
14. Baille, G., Garnier, P., Mathieu, H., Roger, P.G.: Le cycab de l'INRIA rhônes-alpes. Technical Report RT-0229, INRIA – Rhônes-Alpes (1999)