Event-B Specification of a Situated Multi-Agent System: Study of a Platoon of Vehicles

Arnaud Lanoix

Event-B Specification of a Situated Multi-Agent System: Study of a Platoon of Vehicles*

Arnaud Lanoix
LORIA, DEDALE Team – Campus scientifique
F-54506 Vandoeuvre-Lès-Nancy, France
arnaud.lanoix@loria.fr

Abstract

Situated Multi-Agents Systems (MAS), and other Agent-based systems, are often complex. Formal reasoning is needed to ensuring their correctness and structuring their development. Event-B is a formal method with tool support allowing a stepwise development of reactive distributed systems. MAS being a subclass of such systems, we propose using Event-B to helpful their specification and their safe development. In this article, we mainly report our experience with the Even-B stepwise development of a situated MAS which study the displacement of vehicles in a convoy. This article aims also at serving as a guide for the development of other MAS, taking agents-specific features into account.

1. Introduction

Multi-Agent Systems (MAS) are widely used for developing applications in the field of transportation, medical technologies or space exploration. The difficulty of designing and studying situated MAS comes from the autonomy of the agents and their interactions within a common environment. Agents are software entities that encapsulate their behaviour and can change, both pro-actively and reactively, their environment. These systems are highly distributed, where agents evolve in parallel, and more generally work in a dynamic environment. Individually, the agents may be very complex. Due to the contexts these systems are used in, i.e. critical contexts, the problem of ensuring their safety arises. The development of correct/safe situated MAS is difficult with traditional software development methods. Hence, formal methods are needed in order to ensure their correctness and structure their development from specification to implementation.

The B method [1] is a formal method provided with good tool support, but originally developed to model and reason about sequential programs. Event-B [2] is an evolution of the B method that is more suitable for developing large reactive and distributed systems. Software development in Event-B begins by abstractly specifying the requirements of the whole system and then refining them through several steps to reach a concrete description of the system which can be translated to code. Consistency of each model and each relationship between an abstract model and its refinements is obtained by proving it. Recently, tool support has been provided for Event-B specification and proof in the Rodin platform.

We are interested in the so-called platooning problem presented in Sect. 2, where the goal is to have several vehicles travelling in a convoy by defining simple rules for each of them. Moving in a convoy is thus their emergent behaviour. In a previous work, we have developed classical B models for this problem [13]. In this paper,we focus on an Event-B specification of the platooning problem and its application on the Rodin platform. The goal of this research is to demonstrate the use of formal development in the context of MAS oriented software. Section 3 presents Event-B. Next, we describe the Event-B model of the platooning problem we have realised in Sect. 4: we focus on the different steps of refinement necessary for establishing the whole model of the platoon. We also study the validation of the model by presenting the difficulties automatic prover had with some proof obligations and explaining how they were manually discharged. From this case study, we thus extract some generic guidelines for specifying situated MAS with Event-B, presented in Sect. 5. Section 6 presents some related works and Section 7 concludes this paper by giving some perspectives.

*This work has been partially supported by the ANR (National Research Agency) in the context of the ANR-06-SETI-017 TACOS project, and by the pôle de compétitivité Alsace/Franche-comté in the context of the CRISTAL project.

2. A Platoon of Vehicles

The CRISTAL project involves the development of a new type of urban vehicle with new functionalities and services. One of the major cornerstones of Cristal is the platooning problem.

A platoon is defined as a set of autonomous vehicles which have to move in a convoy, i.e. following the path of the leader (possibly driven by a human being) in a row (or a platoon). The control of a platoon involves the longitudinal control of the vehicles, i.e. maintaining a certain ideal distance between each other, and their lateral control, i.e. each vehicle should follow the track of its predecessor. Those controls can be studied independently [4]; we will only focus on the longitudinal control.

Through projects' collaboration with researchers of the MAIA team, we consider each vehicle as an agent. A vehicle's controller perceives informations about its environment before producing an instantaneous acceleration passed to the engine. In this context, the platooning problem can be considered as a situated multi-agent system (MAS) which evolves following a the Influence/Reaction model. This classical MAS model, proposed by Ferber & Muller [8, 7], organises the dynamics of situated MAS by synchronising the various evolution steps: **(i)** all the agents perceptions are done, **(ii)** all influences are decided, and **(iii)** the environment reacts by combining all the influences.

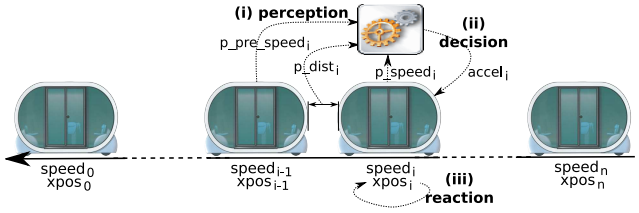

Figure 1. A platoon of vehicles

As we focus on the longitudinal control, the considered space is one-dimensional. Hence the position of the i^{th} vehicle is represented by a single variable $xpos_i$, its velocity by $speed_i$. The behaviour of the vehicle's controllers can be summarised as follows, see Fig. 1:

(i) perception step: each vehicle's controller uses sensors for estimating its velocity p_speed_i, the distance p_dist_i to its leading vehicle and the velocity $p\_pre\_speed_i$ of its leading vehicle. The sensors are supposed to be perfect. Of course, the leader does not need the last two pieces of information as it has no preceding vehicle:

$$\left.\begin{array}{l} p\_speed_i' = speed_i \\ p\_dist_i' = xpos_{i-1} - xpos_i \\ p\_pre\_speed_i' = speed_{i-1} \end{array}\right\} \quad \text{if } i > 1$$

(ii) decision step: each vehicle's controller can influence its speed by computing and passing to the engine an instantaneous acceleration $accel_i$. The acceleration can be negative, corresponding to the braking of the vehicle. $accel_i$ is defined according to the sensor values using mathematical laws, but which cannot be given here for confidentiality reasons.

(iii) reaction step: $xpos_i$ and $speed_i$ are updated, depending on the current speed $speed_i$ of the vehicle and a decided instantaneous acceleration $accel_i$ passed to the engine.

$$new\_speed = speed_i + accel_i$$

$$\left.\begin{array}{l} xpos_i' = \left(\begin{array}{l} xpos_i + Max\_Speed \\ -\frac{(Max\_Speed - speed_i)^2}{2.accel_i} \end{array}\right) \\ speed_i' = Max\_Speed \end{array}\right\} \text{if } \left(\begin{array}{l} new\_speed \\ > \\ Max\_speed \end{array}\right)$$

$$\left.\begin{array}{l} xpos_i' = xpos_i - \frac{speed_i^2}{2.accel_i} \\ speed_i' = 0 \end{array}\right\} \text{if } new\_speed < 0$$

$$\left.\begin{array}{l} xpos_i' = xpos_i + speed_i + \frac{accel_i}{2} \\ speed_i' = new\_speed \end{array}\right\} \text{otherwise}$$

These mathematical laws assume that the actuators of the engine are perfect. Three cases are distinguished, depending on the considered new speed.

Note. Our goal is to develop a formal framework in order to implement these laws and prove properties of the obtained model. The properties we are looking forward to in this model are among the following: (i) the model is sound bound-wise, i.e. none of the specified bounds are violated, (ii) no collision occurs between the vehicles, (iii) no unhooking occurs, i.e. the distance between vehicles cannot be infinitely long and (iv) no oscillation occurs, i.e. a phenomenon of a wave propagates from ahead of the platoon to its back, without never stabilising.

We focus on the soundness of the model and the absence of collision in the remainder of this document, but the reader must be aware that it is still an ongoing work.

3. Event-B

In order to be able to specify and verify correct situated MAS and other agent-based systems, we need to reason about these systems in a formal manner. Reasoning should be facilitated by adequate tool support. Event-B [2] is a formal language for modelling and reasoning about systems. It is an evolution of the classical B [1] for developing reactive and distributed systems. Event-B is provided with tool support currently in the form of a platform for specification and proof called Rodin[1].

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

Abstract Specifications An Event-B abstract specification of a system is encapsulated into a **MODEL** clause identified by an unique name. To each variable x in the **VARIABLES** clause is associated a domain of values. The data invariant I(x) in the **INVARIANT** clause defines the state space of the variables and their safety properties.

Each event in the **EVENTS** clause is a substitution statement. The semantic of these substitution statements are given by the weakest precondition calculus developed by Dijkstra [5]. Notice that a specific event called initialisation appears into each Event-B model. The state variables are initialised into this event. An event consists of a guard and a body. When the guard of an event is evaluated to true, the event can be enabled. When the guards of several events are true, the choice of the triggered event is non-deterministic.

In addition to Event-B models, a **CONTEXT** can be defined to specify static data of sets, constants and their axioms. A model **SEES** at least one context.

Proof obligations (POs) are generated to ensure the consistency of the model, i.e. the preservation of the invariant by the events.

Refinement A refinement process is used to progress towards implementation. An abstract model is transformed into a more concrete and elaborate model. A refined model specifies the abstract model it refines into a **REFINES** clause. New variables can be introduced and the old variables can be refined to more concrete ones. This is reflected in the substitutions of the events as well.

New events may also be introduced. These new events should not prevent forever the events already present from being triggered. A **VARIANT**, which is a natural number expression that the new events must decrease, is introduced for ensuring it. Furthermore, we can also merge several abstract events into one single event, as well as refine one abstract event by several events, as is permitted by an event-level **REFINES** clause. An event-level **WITH** clause expresses the link between the parameters of an abstract event, removed in the refined event, and their concretisation.

POs ensure that the refined model is consistent, i.e. its **INVARIANT** is preserved and that the **VARIANT** is decreased by the new events. Furthermore, they ensure that the refinement is correct, i.e. the refined events do not contradict their abstract counterpart. The abstract **INVARIANT** is also shown to be preserved.

Decomposition The approach of decomposition in Event-B is the inverse of the usual compositional approach in software design and programming. It allows the splitting of an Event-B model into smaller components for managing the increasing complexity of the design. Correspondingly, each proof task should be smaller, thus provable more automatically.

The variables of the initial model are divided into *external* and *internal* variables: the local variables are variables which concern only one component whereas external variables represent shared variables which can be modified by all the components; they are replicated in all the decomposed models.

The events referring only the local variables of one component only appear in the relevant component. The others which refer local and external variables appear in the component which reference the local variables. In addition, extra *external* events simulating their corresponding events using external variables only, must appear in the other components.

4. Event-B Specification of a Platoon

Figure 2 gives the various components of the Event-B specification of the platooning problem. The development is done following a stepwise refinement based on the I/R model. The I/R steps are gradually introduced:

1. platoon: it represents an overview of the platooning problem. The movement of the vehicles, corresponding to the reaction step, is globally viewed.

   The safety requirement of no-collision is expressed so far.

2. platoon_1: the global movement is split into each vehicle.

3. platoon_2: the speed of each vehicle is introduced. The movement of each vehicle is now defined as the application to the current speed of an acceleration passed to the engine (reaction laws).

4. platoon_3: the acceleration of each vehicle is now decided before the vehicles move. That corresponds to the specification of the decision step.

5. platoon_4: the controllers perceive (perception step), before they decide for an acceleration by applying the decision laws.

Furthermore, a context is required for defining the global constants and axioms of the case study. It is seen by all the other models.

The following sections detail the refinement steps, their Event-B specifications and their verification using Rodin.

4.1. Introducing the Reaction Step and Preventing Collisions: platoon

The Event-B model platoon represents the first specification of the platooning problem. Only the longitudinal po-

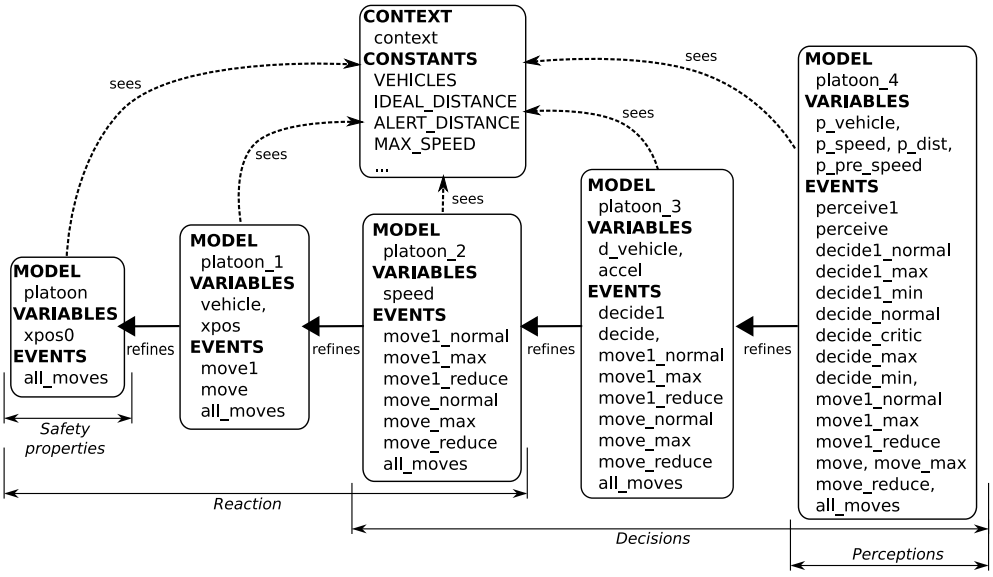

Figure 2. Even-B models of platooning

sitions of each vehicle are viewed. All the vehicles move in a simultaneous movement. We focus on the major safety property of the system: *no collision* must occur between a vehicle and its predecessor.

Event-B specification The positions are expressed by a functional variable $xpos0 \in 1..VEHICLES \rightarrow NAT$ which links together the index of a vehicle and its longitudinal position. The leader vehicle is indexed by 1. The no-collision property is expressed by the invariant:

$$\forall v.(v \in 2..VEHICLES \Rightarrow (xpos0(v-1) - xpos0(v)) > CRITICAL\_DISTANCE)$$

A single event all_moves models the simultaneous movement of all the vehicles. New positions for the vehicles are "magically" chosen with respect to the safety property and are affected to the previous positions:

all_moves $\widehat{=}$
ANY magic_xpos **WHERE**
$magic\_xpos \in 1..VEHICLES \rightarrow \mathbb{N} \wedge$
$\forall v.\ (v \in 2..VEHICLES \Rightarrow$
$(magic\_xpos(v-1) - magic\_xpos(v)) > CRITICAL\_DISTANCE)$
THEN
$xpos0 := magic\_xpos$
END

	Total	Automatic		Manual	Reviewed	Unproved
platoon	6	5	83%	1	0	0

Verification by proof Rodin generates the necessary POs for validating the preservation of the invariant by the model. All the POs are discharged automatically by the prover. The only manual PO involves the initialisation of the position of each vehicle:

$$initial\_xpos(v-1) - initial\_xpos(v) > CRITICAL\_DISTANCE$$

This proof has been done with Rodin as follows:
The hypothesis $\forall v.(v \in 1..VEHICLES \Rightarrow initial\_xpos(v) = (VEHICLES-v)*IDEAL\_DISTANCE)$ is instantiated with v: $initial\_xpos(v) = (VEHICLES-v)*IDEAL\_DISTANCE$
It is also instantiated with $v-1$: $initial\_xpos(v-1) = (VEHICLES-v+1)*IDEAL\_DISTANCE$
We substitute the previous results into the goal, and simplify the formula that becomes $IDEAL\_DISTANCE > CRITICAL\_DISTANCE$: the prover shows the goal is verified.

4.2. Splitting the Reaction Step: platoon_1

The simultaneous movement of all the vehicles is decomposed, i.e. the movement of each vehicle is viewed one after the other, starting from the leader.

Event-B specification In order to identify the current vehicle which has to move, a variable $vehicle \in 1..VEHICLES+1$ is introduced. Another new variable $xpos \in 1..VEHICLES \rightarrow NAT$ is introduced for modelling the position of each vehicle during the movement. The safety property has to be strengthened: now, we ensure that after each single movement, no collision has occured up to the vehicle which has to move:

$$\forall v.(v \in 2..vehicle-1 \Rightarrow (xpos(v-1) - xpos(v)) > CRITICAL\_DISTANCE)$$

Note that this safety property implies the previous one when vehicle=VEHICLES+1.

Because the leader vehicle has a specific behaviour, two new events are introduced for modelling the single movement of a vehicle:

- move1 models the movement of the leader vehicle. A new position magic_xpos_vehicle is chosen with the only constraint that the vehicle moves forward.
- move models the movement of each following vehicle, one after the other. A new position magic_xpos_vehicle is chosen such that the vehicle moves without colliding into its preceding vehicle. Moreover, it must move forward: the new chosen position is greater than the previous one.

```
move ≙
ANY magic_xpos_vehicle WHERE
  vehicle ∈ 2..VEHICLES ∧
  magic_xpos_vehicle ∈ ℕ ∧
  magic_xpos_vehicle ≥ xpos(vehicle) ∧
  xpos(vehicle−1) − magic_xpos_vehicle > CRITICAL_DISTANCE
THEN
  vehicle := vehicle+1 ‖ xpos(vehicle) := magic_xpos_vehicle
END
```

We must express a variant to ensure that these new events do not take the control for ever. Because vehicle is increased after each move, the variant we consider is:

```
(VEHICLES+1) − vehicle
```

When vehicle=VEHICLES+1, all the movements are done. We refine the abstract event all_moves by choosing the new variable xpos as a correct value (a witness) of the previous parameter magic_xpos.

```
all_moves ≙
REFINES all_moves
WHEN
  vehicle = VEHICLES + 1
WITH magic_xpos = xpos
THEN
  xpos0 := xpos ‖ vehicle := 1
END
```

	Total	Automatic		Manual	Reviewed	Unproved
platoon_1	22	19	86%	3	0	0

Verification by proof Among the manual proofs that remain to do after discharging the POs, one is most interesting: we have to show that the invariant of no-collision is preserved when a vehicle moves, i.e. a new position is chosen.

```
(xpos⩤{vehicle ↦ magic_xpos_vehicle})(v − 1)
 − (xpos⩤{vehicle ↦ magic_xpos_vehicle})(v)>CRITICAL_DISTANCE
```

We have the following hypotheses:

```
  v ∈ 2..vehicle
∧ ∀ v . (v∈2..(vehicle−1) ⇒ xpos(v−1) − xpos(v) > CRITICAL_DISTANCE)
∧ xpos(vehicle−1) − magic_xpos_vehicle > CRITICAL_DISTANCE
```

Into Rodin, we separate the case of the updated vehicle from the other vehicles by rewriting the operator ⩤ to obtain:

```
  (({vehicle} ⩤ xpos) U {vehicle ↦ magic_xpos_vehicle})(v−1)
 − (({vehicle} ⩤ xpos) U {vehicle ↦ magic_xpos_vehicle})(v)
> CRITICAL_DISTANCE
```

We add a new sub-goal v ∈ 2..(vehicle−1) ∨ v=vehicle and run a proof by cases.

1. v ∈ 2..(vehicle−1)

   Hypothesis (b) is instantiated with v:
   xpos(v−1) − xpos(v) > CRITICAL_DISTANCE

   We add some new sub-goals, until we re-obtain the goal:

   - xpos(v−1)=({vehicle} ⩤ xpos)(v−1),
   - xpos(v)=({vehicle} ⩤ xpos)(v),
   - xpos(v−1)=(({vehicle} ⩤ xpos) U {vehicle ↦ magic_xpos_vehicle})(v−1),
   - xpos(v)=(({vehicle} ⩤ xpos) U {vehicle ↦ magic_xpos_vehicle})(v),

2. v=vehicle

   By simplification rewriting, the goal becomes:
   (({vehicle} ⩤ xpos) U {vehicle ↦ magic_xpos_vehicle})(vehicle−1) − magic_xpos_vehicle > CRITICAL_DISTANCE

   We add some new sub-goals:

   - xpos(vehicle−1)=({vehicle} ⩤ xpos)(vehicle−1),
   - xpos(vehicle−1)=(({vehicle} ⩤ xpos) U {vehicle ↦ magic_xpos_vehicle})(vehicle−1).

All the introduced sub-goals are proved into Rodin, hence this PO is verified. The other manual POs are also verified and we are confident that

- platoon_1 is consistent, i.e. its invariant is preserved,
- platoon_1 is a correct refinement of platoon, i.e. the no-collision property verified by platoon is preserved by platoon_1, and
- the new events introduced into platoon_1 do not take the control forever.

Remark Notice that a lot of manual POs follow the same scheme of proof: (i) rewrite of the operator ⩤ and (ii) proof by cases on v ∈ 2..(vehicle−1) ∨ v=vehicle.

4.3. Implementing the Reaction Laws: platoon_2

The movement of each vehicle is now defined as a reaction to an instantaneous acceleration passed to the engine. The speed of the vehicle must now be considered in order to apply the acceleration and compute the new position.

Event-B specification To model the instantaneous speed of each vehicle, a variable speed $\in$ 1..VEHICLES $\rightarrow$ 0..MAX_SPEED is introduced. The events move1 and move are refined by considering an acceleration parameter magic_accel and computing the new speed nspeed and position nxpos resulting from the application of this acceleration. These reaction laws are given in Sect. 2. As three cases have to be distinguished depending on the new computing speed, the event move is refined as follows:

1. move_normal: the vehicle travels within the acceptable speed limits (nspeed $\in$ 0..MAX_SPEED);
2. move_max: the vehicle violates the maximum possible/allowed speed (nspeed $>$ MAX_SPEED);

move_max $\widehat{=}$
REFINES move
ANY magic_accel, nxpos **WHERE**
vehicle $\in$ 2..VEHICLES $\wedge$
magic_accel $\in$ MIN_ACCEL..MAX_ACCEL $\wedge$
speed(vehicle) + magic_accel $>$ MAX_SPEED $\wedge$
nxpos = xpos(vehicle) + MAX_SPEED $-$ ((MAX_SPEED $-$ speed(vehicle)) $\times$ (MAX_SPEED $-$ speed(vehicle))) / (2 $\times$ magic_accel) $\wedge$
xpos(vehicle$-$1) $-$ nxpos $>$ CRITICAL_DISTANCE
WITH magic_xpos_vehicle = nxpos
THEN
vehicle := vehicle+1 $\|$ xpos(vehicle) := nxpos $\|$
speed(vehicle) := MAX_SPEED
END

3. move_reduce: the vehicle goes backwards (nspeed $<$ 0).

The same refinement occurs for event move1. All these events take the new computed position nxpos as a correct value of the previous abstract parameter magic_xpos_vehicle.

	Total	Automatic		Manual	Reviewed	Unproved
platoon_2	43	27	63%	12	4	0

Verification by proof We mainly have to prove the consistency of the computation of the new position when applying the acceleration. All these POs are difficult to handle for the prover, due to arithmetics, in particular with the division operator, and the rewriting of (in)equalities that guide the proof.

- Four POs have to be proved to ensure the soundness of the computation of the new position, in the normal case:

  xpos(vehicle) + speed(vehicle) + magic_accel/2 $\in \mathbb{N}$

  We have speed(vehicle) + magic_accel $\geq$ 0.

  The proof is done by adding and proving successively these "necessary" sub-goals into Rodin:

  - magic_accel $\geq -$speed(vehicle)
  - magic_accel $\geq -$speed(vehicle) $\times$ 2
  - magic_accel/2 $\geq$ ($-$speed(vehicle) $\times$ 2)/2
  - speed(vehicle)+magic_accel/2 $\geq$ 0
  - xpos(vehicle) $\geq$ 0
  - xpos(vehicle)+speed(vehicle)+magic_accel/2 $\geq$ 0

  The other POs can be done easily just by cutting and pasting the previous proof tree.

- Four POs have to be proved for ensuring that the new position is consistent when the vehicle attempts to go backwards. These POs are discharged by decomposing the goal into some sub-goals before proving them with Rodin as previously.

- Four POs have to be proved for ensuring the soundness of the new position when the vehicle violates the maximum possible speed. This PO cannot be discharged with Rodin, but can be done by pen and paper, since Rodin is unable to prove a rewriting of the goal.

As all the POs are verified, then we can conclude that the computations are consistent and that platoon_2 refines platoon_1 and the no-collision property is preserved.

4.4. Introducing the Decision Step: platoon_3

In the previous model, instantaneous accelerations are passed to the engine to move the vehicles. In this refinement step, we introduce the vehicle's controllers that decide of the accelerations.

Event-B specification A variable accel $\in$ 1..VEHICLES $\rightarrow$ MIN_ACCEL..MAX_ACCEL is introduced to save the decided acceleration until the movement happens. Another new variable d_vehicle $\in$ 1..VEHICLES+1 has to be introduced for identifying the current vehicle's controller which has to decide for an acceleration.

Two new events decide1 and decide are introduced for modelling the decision step. A correct acceleration is chosen magically and saved into accel. The event decide1 is enabled only once, when d_vehicle=1. decide is enabled until d_vehicle=VEHICLES+1.

```
decide ≙
ANY magic_accel WHERE
  vehicle = 1 ∧
  d_vehicle ∈ 2..VEHICLES ∧
  magic_accel ∈ MIN_ACCEL..MAX_ACCEL
THEN
  d_vehicle := d_vehicle + 1 ‖ accel(d_vehicle) := magic_accel
END
```

As new events are introduced, a variant must be defined to prove that these new events do not take control for ever. The variable d_vehicle is increased by each new events, then a correct variant can be

```
(VEHICLES+1) − d_vehicle
```

The guard of the old events are strengthened by d_vehicle=VEHICLES+1 to ensure that all the controller's decisions are taken before the vehicles move. The events corresponding to move are refined by taking into account the decided accel instead of magic_accel.

```
move_max ≙
REFINES move_max
ANY nxpos WHERE
  vehicle ∈ 2..VEHICLES ∧
  speed(vehicle) + accel(vehicle) > MAX_SPEED ∧
  nxpos = xpos(vehicle) + MAX_SPEED − ((MAX_SPEED − speed(vehicle))
                × (MAX_SPEED − speed(vehicle))) / (2 × accel(vehicle) ∧
  xpos(vehicle−1) − nxpos > CRITICAL_DISTANCE ∧
  d_vehicle = VEHICLES + 1
WITH magic_accel = accel(vehicle)
THEN
  vehicle := vehicle+1 ‖ xpos(vehicle) := nxpos ‖ speed(vehicle) := MAX_SPEED
END
```

	Total	Automatic		Manual	Reviewed	Unproved
platoon_3	62	56	90%	6	0	0

Corrections guided by the proof Often, unproved POs indicate failures or mistakes into the model. The next PO has been generated:

```
xpos(d_vehicle−1) − (xpos(d_vehicle) + speed(d_vehicle) + magic_accel/2) >
                                                        CRITICAL_DISTANCE
```

We are unable to discharge this PO. It shows that we forgot into the event decide a precondition linking magic_accel and CRITICAL_DISTANCE. By adding the correct precondition, the PO becomes automatically proved.

Verification by proof Rodin generates a lot of POs to validate the preservation of the invariant and the refinement. The majority of them is discharged automatically by the prover, only six are to be proved interactively. One of them concerns the preservation of the no-collision property by the decision:

```
xpos(v−1) − (xpos(v) + speed(v) + (accel⩤{d_vehicle ↦ magic_accel})(v) / 2) >
                                                        CRITICAL_DISTANCE
```

This PO is discharged following the same reasoning as in Subsect. 4.2, by rewriting the operator $\lhd\!\!+$ and making a proof by case on v ∈ 2..(d_vehicle−1) ∨ v=d_vehicle.

4.5. Introducing the Perception Step and Implementing the Decision Laws: platoon_4

In this step, we model the perceptions that the vehicle's controllers have on their environment, i.e. the controlled vehicle and the leading one. The decision laws can then be derived from the perceptions.

Event-B specification The perceptions of a controller about its environment are:

- its velocity p_speed ∈ 1..VEHICLES → 0..MAX_SPEED,
- the distance p_dist ∈ 2..VEHICLES → ℤ to its leading vehicle, and
- the velocity p_pre_speed ∈ 2..VEHICLES → 0..MAX_SPEED of its leading vehicle.

A variable p_vehicle ∈ 1..VEHICLES+1 has been introduced for identifying the current vehicle's controller that has to be perceived. To ensure that all the perceptions are done before deciding and moving, we strengthen the guard of the old events by p_vehicle = VEHICLES + 1.

Two new events perceive1 and perceive are introduced for modelling the perception step. The leader vehicle has no predecessor, so it is a specific case.

```
perceive ≙
WHEN
  vehicle = 1 ∧
  d_vehicle = 1 ∧
  p_vehicle ∈ 2..VEHICLES ∧
THEN
 p_vehicle := p_vehicle + 1 ‖ p_speed(p_vehicle) := speed(p_vehicle) ‖
 p_dist(p_vehicle) := xpos(p_vehicle−1) − xpos(p_vehicle) ‖
 p_pre_speed(p_vehicle) := speed(p_vehicle−1)
END
```

As aforementioned, a variant has to be defined. p_vehicle is used to know which vehicle has to perceive, then a variant can be

```
(VEHICLES+1) − p_vehicle
```

The decision of the acceleration is computed from the perceptions. The events decide1 and decide are refined by seven events implementing the decision laws and depending on whether

- the vehicle is the leader or not,
- the perceived distance is less than an alert distance value,
- the decided acceleration is between MIN_ACCEL and MAX_ACCEL, or not.

```
decide_min ≙
REFINES decide
ANY naccel WHERE
  vehicle = 1 ∧
  d_vehicle ∈ 2..VEHICLES ∧
  p_vehicle = VEHICLES+1 ∧
  p_dist(d_vehicle) ≥ ALERT_DISTANCE ∧
  naccel = 2 × (p_dist(d_vehicle) − IDEAL_DISTANCE
            + p_pre_speed(d_vehicle) − p_speed(d_vehicle)) ∧
  naccel < MIN_ACCEL
WITH magic_accel = MIN_ACCEL
THEN
  d_vehicle := d_vehicle+1 ‖ accel(d_vehicle) := MIN_ACCEL
END
```

	Total	Automatic		Manual	Reviewed	Unproved
platoon_4	80	70	87%	6	3	1

Corrections guided by the proof An unproved PO indicates a mistake into the model. First the perceived distance p_dist was defined as a function from 2..VEHICLES to $\mathbb{N}$. A PO was generated about the consistency of p_dist.

$$p\_dist \lhd\!\!+ \{p\_vehicle \mapsto xpos(p\_vehicle - 1) - xpos(p\_vehicle)\} \in 2 \;..\; VEHICLES \to \mathbb{N}$$

Rodin is unable to prove that $xpos(p\_vehicle-1) - xpos(p\_vehicle) \geq 0$. We have no hypothesis that can help the proof, then the PO indicates a mistake. When changing the type of p_dist to $\mathbb{Z}$, the corresponding PO becomes automatically discharged.

Verification by proof Most of the POs are automatically discharged by the prover, save for only 10 unproved POs. 6 of them can be proved manually by rewriting the $\lhd\!\!+$ operator and doing a proof by case on $v \in 2..(p\_vehicle-1) \vee v = p\_vehicle$ as explained in Subsect. 4.2.

Three POs involve the correctness of the decision for an acceleration in the normal case (i.e. the perceived distance is more than ALERT_DISTANCE). These POs cannot be proved with Rodin for arithmetical reasons: the prover is unable to validate the suggested rewriting which is necessary to achieve the proofs, but they are done by hand.

Unprovable PO At the moment of writing, the remaining unproved PO concerns the decision for the acceleration when the perceived distance between the vehicles is less than *ALERT_DISTANCE*:

$$xpos(d\_vehicle-1) - (xpos(d\_vehicle)+speed(d\_vehicle)+MIN\_ACCEL/2) > CRITICAL\_DISTANCE$$

This PO seems to be unprovable under the current hypotheses. It indicates a mistake or a failure in the current model. We are discussing with experts of the domain for making the current model evolve.

4.6. Decomposing the Model between Vehicles and Controllers

The controllers of the vehicles must be separated from the environment. The model platoon_4 can be decomposed in two parts: vehicles and controllers as shown Fig. 3.

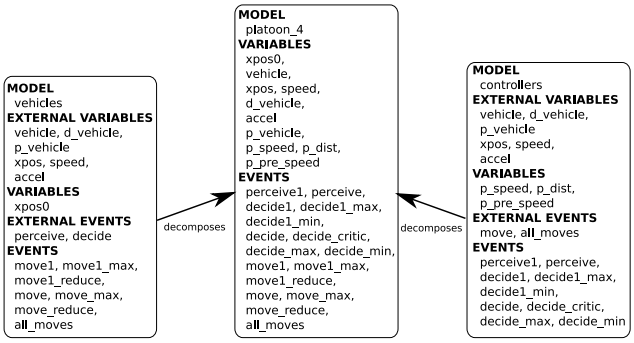

Figure 3. Decomposition of *platoon_4*

Event-B specification The variables are split between *external* and *internal* variables. The local internal variables of the controllers model are the perceptions that the controllers have from their associated vehicles and the predecessor. Considering the vehicles model, the only local variable is the position xpos0 of the vehicles after all the movement. All the other variables are external. This includes the counters vehicle, d_vehicle and p_vehicle, the variables of the environment xpos and speed which are perceived and the decided influence accel.

The events of the model platoon_4 are also split among the decomposed models. The *internal* events of vehicles involve the movement of the vehicles whereas internal events of controllers are dedicated to perceptions and decisions. Extra external events must be added to the decomposed models in order to "simulate" (using the external variables only) the corresponding events of the full model. The events perceive and decide of vehicles abstract the behaviours of perceive1, perceive and decide1_n∨mal, decide1_max, decide1_min, decide_n∨mal, decide_critic, decide_max, decide_min respectively. The same goes for the move events into controllers.

Remark Notice that this last step cannot be verified by Rodin because the decomposition mechanism is currently supported neither by the platform nor by any other tool.

5. Generalisation to Specification of Other Multi-Agents Systems

The platooning case study presented before is a good example for the Event-B specifications and verification by proof of other situated MAS. Its development provides us some guidelines we summarised in this section.

To specify the perceptions and the influences of the agents and the environment parts specific to each agent, the use of functional variables is a convenient manner of modelling them with Event-B. Other variables can be added to the environment to represent independant parts of agents.

The way of specifying the synchronisation proposed by the I/R model can be generalised. Three variables r_agent, d_agent and p_agent (defined on 1..MAX_AGENTS) are counters which indicate the step in the I/R cycle.

The various events are introduced by refinement, following the I/R model. Events are gradually introduced, starting by the last, i.e. the environment reaction, to finish by the first step, i.e. the agents perceptions (See Fig. 2). The guards of the old events are strengthened to ensure that the new events are done before the old ones are enabled and a variant is dedicated to preventing the new events from taking the control forever.

Reaction step Three levels of refinement are dedicated to the specification of the reaction. Safety properties of the environment are expressed at the beginning. They must be preserved by the agents: the refinement also ensures that they are preserved throughout the evolution of the model.

Currently, the splitting of a global reaction into a reaction expressed agent by agent specified into platoon_1 cannot be done. The majority of MAS has a global reaction that only expresses all the influences in a global manner. Other levels of refinement might be necessary when the splitting is possible, though.

Influences are taken into account as an event's parameter into platoon_2. If the computation of the reaction is complex, more than one refinement can be necessary to help the prover.

Decision step This step is progressively introduced. Influences are applied to reaction into platoon_2. Events corresponding to the decision step are introduced in platoon_3. The "real" computation of influences taking into account the perceptions is done into platoon_4.

Perception step It is the last part of a MAS we model. Perceptions are introduced into platoon_4 and directly linked with influences. In some more complex cases, it seems easier to first introduce the perception step into a first level of refinement before linking perceptions and influences into a second one.

Remark Some MAS have a local behaviour between the perception and decision step. This local behaviour modifies some local variables as historic variables, which are considered for taking the decision. This kind of agents is called *hysteretic* contrary to the *tropistic* agents which have no internal behaviour. Local behaviour can be specified by inserting some refinement levels between the decision level and the perception level.

About decomposition The decomposition step sketched in Subsect. 4.6 can also be generalised to obtain two Event-B components Environment and Agents. The following table summarises how the various parts of the initial model are assigned to the components:

	Environment	Agents
r_agent, d_agent and p_agent	external	external
Perceived physical variables	external	external
Unperceived physical variables	internal	-
Perceptions variables	-	internal
Influences variables	external	external
Perception events	external	internal
Decision events	external	internal
Reaction events	internal	external

6. Related Works

Hilaire et al [9] propose a general framework for modelling MAS based on Object-Z and statecharts. This framework focuses on organisational aspects in order to represent agents and their roles. Similarly, Regayeg et al [11] combine Z notations and linear temporal logic to specify the internal part of agents and the specification of the communication protocols between agents. They propose general patterns and the use of Z support tools to model-check their specifications. It is to be noticed that the proposed patterns do not deal with dynamics of physical world.

Inverno and Saunders [10] have developed a multi-agent approach for simulating the behaviour of stem cells. Their aim is to highlight which properties are required on components in order to maintain general properties. Their formal models, written in Z, are based on a layered technique in which physical, biological and chemical environment are considered separately.

We can also point out a work [6] involving the use of classical B to model agents roles and interactions. [3] focuses too on the interaction protocol between agents using Event-B. Some patterns for the B specification of fault-tolerance protocols are proposed in the case of agent communication.

Schneider et al [12] apply their framework based on CSP and B as a starting point for the simulation of a biomedicine MAS. They only focus on the clotting behaviour of artificial platelets.

7. Conclusion

The models presented in Sect. 4 are completely specified into Rodin, the platform which supports the Event-B formalism. The generated POs are validated to ensure the correction of the specification. Results regarding POs are given Fig. 4. As expected, the number of POs increases with each refinement step. The majority of them are automatically discharged by the prover and the others are done interactively : difficulties come mainly from arithmetics.

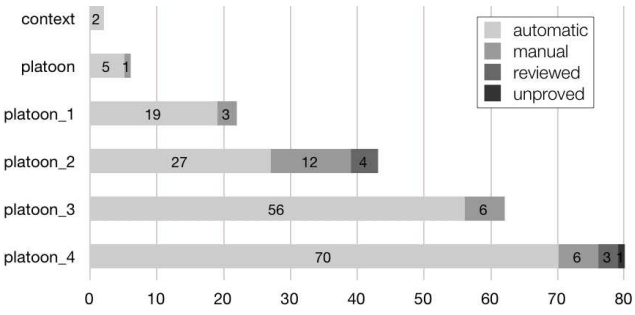

Figure 4. Proof obligations results

The Event-B specification of the platooning problem presented here shows some advantages of using a formal method to model MAS: the formalisation focuses on understanding the MAS. The proof process helps identifying mistakes in the model and pinpointing weaknesses in the assessment of the hypotheses. Not surprisingly from a software engineering point of view, knowledge of the experts of the domain was required for completing the hypotheses of the system.

From the case study of the platooning, we thus extract some generic guidelines for the development with Event-B of other situated MAS taking agents-specific features into account. We plan to study other situated MAS following the same guidelines.

Further work includes the study of the same platooning problem with related formalisms such as CSP||B [12] to focus on the interaction protocol between the agents. It also includes the specification and verification of other properties, such as unhooking or oscillation. We also plan to make the model evolve in order to take into account the lateral control or perturbations like pedestrians or other vehicles in the environment.

Acknowledgements We address our many thanks for common efforts and fruitful discussions to Olivier Simonin, Alexis Scheuer and François Charpillet, from the MAIA team of the LORIA, and to Samuel Colin and Jeanine Souquières, from the DEDALE team.

References

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

[2] J.-R. Abrial and S. Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to Event-B. *Fundamenta Informaticae*, 77(1-2):1–28, 2007. Special issue on ASM'05.

[3] E. Ball and M. Butler. Event-B patterns for specifying fault-tolerance in Multi-Agent interaction. In *Proceedings of Methods, Models and Tools for Fault Tolerance*, Oxford, UK, July 2007.

[4] P. Daviet and M. Parent. Longitudinal and lateral servoing of vehicles in a platoon. In *Proceeding of the IEEE Intelligent Vehicles Symposium*, pages 41–46, 1996.

[5] E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[6] H. Fadil and J. Koning. A formal approach to model multiagent interactions using the B formal method. In *Fourth IEEE International Symposium on Advanced Distributed Systems (ISADS 2005)*, volume 3563 of *LNCS*, pages 516–528. Springer, 2005.

[7] J. Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-wesley Professional, 1999.

[8] J. Ferber and J. P. Muller. Influences and reaction : a model of situated multiagent systems. In *2nd Int. Conf. on Multiagent Systems*, pages 72–79, 1996.

[9] V. Hilaire, P. Gruer, A. Koukam, and O. Simonin. Formal specification approach of role dynamics in agent organisations: Application to the Satisfaction-Altruism Model. In *Int. Jour. of Software Engineering and Knowledge Engineering (IJSEKE)*. in press, 2006.

[10] M. Inverno and R. Saunders. Agent-based modelling of Stem Cell organisation in a Niche. In S. A. Brueckner, G. Di Marzo, S. A. Karageorgos, and R. Nagpal, editors, *Engineering Self-Organising Systems : Methodologies and Applications*, LNAI. Springer-Verlag, 2005.

[11] A. Regayeg, A. H. Kacem, and M. Jmaiel. Specification and verification of multi-agent applications using temporal z. In *Intelligent Agent Technology Conf. (IAT'04)*, pages 260–266. IEEE Computer Society, 2004.

[12] S. Schneider, A. Cavalcanti, H. Treharne, and J. Woodcock. A layered behavioural model of platelets. In *11th IEEE International Conference on Engieerging of Complex Computer Systems, ICECCS*, 2006.

[13] O. Simonin, A. Lanoix, S. Colin, A. Scheuer, and F. Charpillet. Generic Expression in B of the Influence/Reaction Model: Specifying and Verifying Situated Multi-Agent Systems. INRIA Research Report 6304, INRIA, Sept. 2007.