An Experience with a Formal Modelling of a Multi-Agent System: the Platooning Problem

Samuel Colin, Arnaud Lanoix

An Experience with a Formal Modelling of a Multi-Agent System: the Platooning Problem*

Samuel Colin and Arnaud Lanoix

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

Abstract. Multi-Agent systems (MASs) are used in more and more critical domains such as transportation or medical technology, although their use is limited by the lack of validation and adequate software engineering support. Formal methods have proved their strength in that regard. We propose the use of the B formal method as a sound software engineering approach for specifying and validating MASs. We illustrate this approach on the basis of the agent-based platooning problem, i.e. vehicles moving in a convoy. We hereby aim at demonstrating the solid support provided by formal methods to such systems and how their use help the extraction of implicit knowledge from the models for MASs engineers.

1 Introduction

Software agents and Multi-Agent Systems (MAS) are widely used to develop applications in the field of transportation, medical technologies or space exploration. The most limiting factor in industry is the difficulty of designing and studying complex situated MAS. This difficulty comes from the lack of adequate software engineering methods able to manage the autonomy of the agents evolving in parallel. Their interactions between each others and with a common environment leads to the problem of preservation or the occurrence of emergent properties at the global level of the system.

Due to the contexts these systems are used in, i.e. critical contexts, the problem of ensuring their safety arises. To that end, use of formal methods is needed, which has begun to receive a substantial amount of interest. The questions linked to this formalisation are such as: can a MAS be formalised ? If not, can I characterise a smaller part of it be formalised ? If on the contrary the system can be formalised, are the desired properties expressible ? If they are, can they be verified ? In case of success, what new knowledge can be tracted from the formalisation ? Can the model be adapted to other similar MAS ?

The first questions have begun to be answered. For instance, Hilaire et al [1] propose a general framework for modelling MAS based on Object-Z and statecharts. This

* This work has been partially supported by the ANR (National Research Agency) in the context of the TACOS project, whose reference number is ANR-06-SETI-017 (http://tacos.loria.fr), and by the pôle de compétitivité Alsace/Franche-comté in the context of the CRISTAL project (http://ww.projet-cristal.org).

framework focuses on organisational aspects in order to represent agents and their roles. Similarly, Regayeg et al [2] 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 worlds.

Inverno and Saunders [3] 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.

Schneider et al [4] apply their framework based on CSP and B as a starting point for the simulation of a biomedicine MAS. They focus on the clotting behaviour of artificial platelets. We can also point out a recent work [5] involving the use of Event-B. It focuses on the coordination between agents and only specifies the interaction protocol. Some patterns for the B specification of fault-tolerance protocols are proposed in the case of agent communication.

We ourselves are interested in the so-called platooning problem presented in Sect. 2, where the goal is to have several vehicles travel in a convoy by defining simple rules for each of them. Moving in a convoy is thus their emergent behaviour. Thanks to collaboration with the MAIA team, we developped B models to answer to this problem. Section 3 gives an overview of the B method. We present in Sect. 4 the B modelisation of the agent-based platooning. We more particularly were interested in the problems or advantages linked with the choice of the B method to formalise this MAS. We also focus on which informations can be extracted to MAS engineers. Section 5 concludes this paper and gives some perspectives.

2 An Example of MAS : 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.

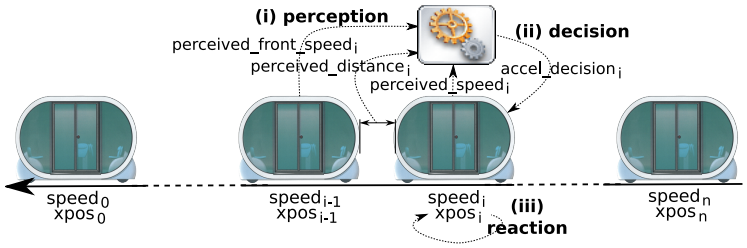

Fig. 1. A platoon of vehicles

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 [6]; 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 instantanous acceleration passed to the engine. In this context, the platooning problem can be considered as a situated MAS which evolves following the Influence/Reaction model, proposed by Ferber & Muller [7]. In this model, agents are described separately from the environment. The link is done by computing, at each step Δt, which environment state each agent perceives and which influences it produces. The new state of the environment is defined as the combination of the various influences produced by the agents. I/R organises the dynamics of situated MAS by synchronising the different evolution steps following a cycle, as shown Fig. 1: **(i)** all the agents perceptions are done, **(ii)** all influences are decided, and **(iii)** the environment reacts by combining all the influences.

As we focus only on the longitudinal control of the platoon, 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 uses sensors for estimating its velocity $perceived\_speed_i$, the distance $perceived\_distance_i$ to its leading vehicle and the velocity $perceived\_front\_speed_i$ of its leading vehicle. The sensors are supposed to be perfect.

$$\begin{aligned} perceived\_speed_i(t+\Delta t) &= speed_i(t) \\ perceived\_distance_i(t+\Delta t) &= xpos_{i-1}(t) - xpos_i(t), \text{ if } i > 0 \\ perceived\_front\_speed_i(t+\Delta t) &= speed_{i-1}(t), \text{ if } i > 0 \end{aligned} \tag{1}$$

(ii) decision step: each vehicle can influence its speed and position by computing and passing to the engine an instantaneous acceleration $accel\_decision_i$. The acceleration can be negative, corresponding to the braking of the vehicle. $accel\_decision_i$ 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 $accel\_decision_i$ to the engine.

$$new\_speed = speed_i(t) + accel\_decision_i(t+\Delta t).\Delta t$$

$$\text{if } new\_speed > Max\_speed, \begin{cases} xpos_i(t+\Delta t) = xpos_i(t) + \Delta t.Max\_Speed \\ speed_i(t+\Delta t) = Max\_Speed \end{cases} \tag{2a}$$

$$\text{if } new\_speed < 0, \begin{cases} xpos_i(t+\Delta t) = xpos_i(t) - \frac{speed_i(t)^2}{2.accel\_decision_i(t+\Delta t)} \\ speed_i(t+\Delta t) = 0 \end{cases} \quad (2b)$$

$$\text{otherwise}, \begin{cases} xpos_i(t+\Delta t) = \begin{pmatrix} xpos_i(t) + speed_i(t).\Delta t \\ + \frac{accel\_decision_i(t+\Delta t).\Delta t^2}{2} \end{pmatrix} \\ speed_i(t+\Delta t) = new\_speed \end{cases} \quad (2c)$$

These mathematical laws assume that the actuators of the engine are perfect.

The laws come from experts of MAS. Our goal is to develop a formal framework in order to implement them and prove properties of the obtained model. The properties we are looking forward to in this model are among the following:

- The model is sound bound-wise, i.e. none of the specified bounds are violated.
- No collision occurs between the vehicles.
- No unhooking occurs, i.e. the distance between vehicles cannot be infinitely long.
- 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 The B Method

B is a formal software development method used to model and reason about systems [8]. It is based on set theory and relations. Its key features are:

- First-order *set-theoretic* foundations, which are a well-understood domain of mathematics
- *Modularity* for helping with the specification of big systems by means of separated development
- *Refinement* for a detail-wise incremental development and code generation

The B method has proved its strength in the industry in complex real-life applications such as the Roissy VAL [9]. The B method is also supported by academic [1] and commercial [2] support tools.

Before going into the chosen case study, we present firstly how a B model is built and verified so as to help the reader get the intuition behind the B method.

Building a B model The principle behind building a B model is to express what properties of the system are always true after one evolution step of the model. Verifying that the model is correct is thus akin to verifying that the properties are preserved no matter which step of evolution the system takes.

[1] B4free: http://www.b4free.com/

[2] AtelierB: http://www.atelierb.eu/, B-toolkit: http://www.b-core.com/btoolkit.html

```
MODEL example
SEES seen_model
INCLUDES included_model
VARIABLES var1, var2
INVARIANT inv
INITIALISATION init
OPERATIONS
 out1, out2 ⟵ method1(in1, in2) ≙
 PRE pre1
 THEN body1
 END
END
```

Fig. 2. A generic B model

In B terms, the properties are specified into the **INVARIANT** clause of the model and its evolution is specified into several operations specified in the **OPERATIONS** clause. Consider the generic model given Fig. 2. It consists of variables var1 and var2 whose properties are specified in the invariant inv. It also consists of an operation method1 which states how var1 and var2 can evolve, possibly in function of some additional parameters. method1 can also return some values. The precondition pre1 of method1 helps specifying the types of the parameters or constrains. The body body1 is specified in terms of the B language of substitutions, which is very similar to the Dijkstra's weakest precondition calculus [10].

The other clauses are related to modularity: following the **SEES** clause, seen_model is a B model whose state is visible but can not be changed. Following the **INCLUDES** clause, included_model is another B model whose state is visible and whose evolution can be controlled by way of calling its operations.

Verifying a B model As stated earlier, verifying a B model means verifying that each operation maintains the properties of the model. It means that, assuming the model starts in a state respecting the invariant and the precondition of the chosen operation holds, then the resulting state also respects the invariant. Hence for operation method1 it means verifying that inv $\wedge$ pre1 $\Rightarrow$ [body1]inv.

The [body1]inv notation means "the weakest hypotheses required for body1 to establish inv". Obtaining these hypotheses involves their calculation. It suffices to say here that this calculus is largely based upon Dijkstra's calculus, already mentioned above. In the end, knowing that weakest_hypotheses $\Rightarrow$ [body1]inv, verifying the B model for operation method1 means proving the inv $\wedge$ pre1 $\Rightarrow$ weakest_hypotheses formula. This step is usually realized with the help of a theorem prover.

Refining a B model A strength of the B method is its stepwise refinement feature: it is possible to specify the general mathematical properties of a model at an abstract level, and refine it later on. The **REFINEMENT** of a model makes it less indeterministic and more precise, until the code of the operations can actually be implemented in a programming language. The last level of refinement before code generation is called an **IMPLEMENTATION**.

A refinement is specified through the **REFINES** clause that states which model or refinement it is based upon. Building the rest of the refinement is similar to building an abstract B model. Verifying a refinement also involves proving formulas, although there is difference w.r.t. verifying abstract models. Instead of proving that the invariant is maintained, the verification is actually about proving that the refined operations do not contradict their abstract counterparts. We will not present refinement further as it is little used in our case study, presented below in next Section.

4 A B Model for the Platooning Problem

We present a B model of the platooning problem and difficulties of the proof of this model. After presenting its overall architecture, we dwell into more details of the elaboration of each B model and how proving them influenced their evolution. We conclude the section with general remarks pertaining to the verification of the whole platooning model.

4.1 Organisation of the B Model for the Platooning Problem

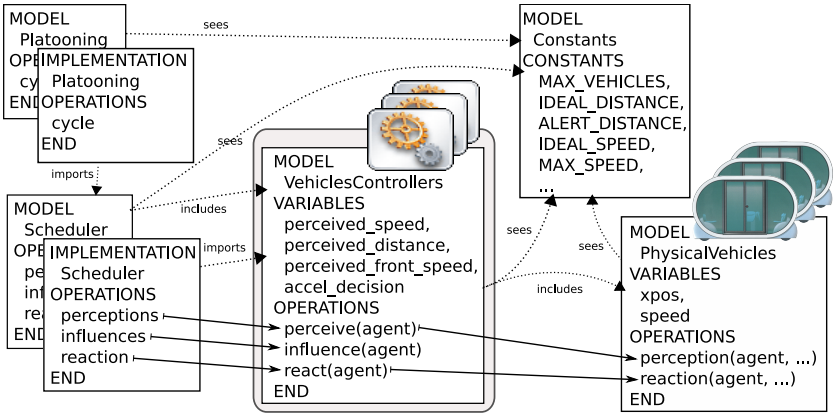

Fig. 3. Organisation of the platooning B model

The components of our model and their organisation is depicted Fig. 3, where:

Constants holds the constants used throughout the model and the hypotheses on these constants.

PhysicalVehicles models the interactions between vehicles and the environment, i.e. their perceptions and their actions.

VehiclesControllers models the steps a vehicle can be in: a vehicle can perceive, decide for an acceleration or attempt to move.

Scheduler schedules the steps all the vehicles go through: all vehicles do their perceptions, then they all do their decisions and finally their actions. Due to B expression, Scheduler is a so-called *implementation* model because loops can only be expressed in them. Hence Scheduler refines the abstract expression of scheduling, a model called Scheduler_abs.

Platooning schedules each global step expressed in Scheduler, i.e. it schedules in a loop all the perceptions, all the decisions and all the actions. For the same reason as above, Platooning is an implementation which refines Platooning_abs.

Note that we have only one B model for n vehicles: we have modelled the variables of each vehicles into corresponding arrays indexed by the unique identifiers of the vehicles. For instance, the variable $speed_i$ becomes the the value at the i^{th} index of the speed array.

A known limitation of the model is that it is based on integer numbers, while the mathematical model presented Sect. 2 relies on the continuous domain of real numbers. We actally assume that the units of the models are precise enough to "absorb" the errors introduced by the use of integers (by using millimetres instead of meters, for instance). Moreover, using floating-point numbers in the implementation is another instance of the same problem.

4.2 B Model for the Environment

The environment, modelled by PhysicalVehicles, holds the laws that involve the perceptions and the reaction, each one found in its respective method as follows.

The perception method models (1), as illustrated by the following snippet of B code:

```
  new_perceived_distance := old_perceived_distance <+ { i ↦ (xpos(i−1) − xpos(i)) }
|| new_perceived_front_speed := old_perceived_front_speed <+ { i ↦ speed(i−1) }
```

The updated perceptions for vehicle i are obtained by overriding the old perceptions with the difference between the involved vehicle i and its leading vehicle.

The reaction method models the update of the environment w.r.t. the acceleration decided by a given vehicle, as expressed by (2). The following snippet depicts the update of the position and speed of a vehicle when it travels at full speed:

```
IF (considered_speed > MAX_SPEED)
THEN
  xpos(i) := new_xpos_when_max_speed(xpos(i)) ||
  speed(i) := MAX_SPEED
```

Formulas for calculating the new position are enclosed into functions for readability purposes. Equation (2a) in charge of calculating the new position when the considered speed is over the maximum speed is defined as follows:

```
  new_xpos_when_max_speed ∈ ℤ → ℤ
∧ new_xpos_when_max_speed = %(xpos).(xpos ∈ ℕ | xpos + MAX_SPEED × TIME_STEP)
```

Properties are specified in the invariant. For instance, the following predicate expresses that vehicles never go backwards, don't violate the maximum speed bound and that there is always a minimal distance between the vehicles, i.e. no-collision:

```
  speed ∈ 0..MAX_VEHICLES → 0..MAX_SPEED ∧ xpos ∈ 0..MAX_VEHICLES → ℕ
∧ ∀i.( i ∈ dom(xpos) − {0} ⇒ xpos(i−1) − xpos(i) ≥ CRITICAL_DISTANCE)
```

Proof and its impact on the model Properties expressed in the invariant of PhysicalVehicles have various influences on its methods. For instance, the reaction method did not initially ensure the safety property of minimal distance between vehicles. We added in the precondition the requirement that vehicles are far enough from each other so as not to violate the safety property. The introduction of this safety distance, which we called ALERT_DISTANCE, highlighted at the proof step that the model missed hypotheses between the constants representing the key distances. We thus added these hypotheses, for instance the relationship between ALERT_DISTANCE and CRITICAL_DISTANCE:

```
ALERT_DISTANCE > MAX_SPEED × MAX_SPEED / (−MIN_ACCEL)
                 + MAX_SPEED × TIME_STEP + CRITICAL_DISTANCE
```

4.3 B Model for the Vehicle Controllers

The vehicles interact with their environment by perceiving it and by applying their decisions. The methods perceive and react are not shown here as they are just wrappers calling the relevant methods of PhysicalVehicles.

The vehicles must also decide which acceleration they apply based on their perceptions, as reflected by the influence method. The specific case of the leader is taken into account. For readability purpose, the computation of the decision in the non-leader case has been defined into a specific compute_new_accel function which is not shown here for confidentiality reasons. Moreover, the resulting acceleration is filtered so as to stay between the set bounds of the system. The following snippet of code is for the non-leader case:

```
ANY new_accel
WHERE new_accel = compute_new_accel(perceived_distance(i),
                                    perceived_front_speed(i),
                                    perceived_speed(i))
THEN
  accel_decision(i) := min( {MAX_ACCEL, max({MIN_ACCEL, new_accel})} )
END
```

Proof and its impact on the model The proofs for VehiclesControllers involve proving the soundness of the computation, i.e. the filtering of the acceleration mentioned above, and the fact that the position of two consecutive vehicles are distant of more than ALERT_DISTANCE. This last part is not proved yet because of difficulties for the tool to prove arithmetically-heavy formulas.

Experiments also showed us how expressing new properties in the model influenced its shape. Any global property, such as the absence of collision, affects indeed the various parts of the model in a specific way. As stated in Sect. 4.2, the absence of collision made us add a new requirement in the precondition of the reaction method. This new requirement will have to be met by the decision made by the vehicles, which might also require enriching the B model of the vehicles. As the decision depends on the perceptions of the agents, which depend on the value of the environment, the modification gets back to the absence of collision.

In short, any newly expressed global property of the model will have to imply itself through one evolution step of the whole model. In a way, the proof of the B model implicitly mimics the self-referential nature of the mathematical model. As a consequence, it is crucial to think thoroughly each newly expressed property so as it is not too strong nor too weak.

4.4 B Models for the Global Control Loop

The self-referential nature of the model can be made explicit by the use of B loops, even though the B models presented so far are enough for describing the whole model. As depicted in the general architecture of the model presented Fig. 3, Scheduler is composed of three loops realising the corresponding steps (perceptions, decisions and actions). Similarly, the global loop expressed in Platooning makes the perceptions, influences, reaction sequence explicit.

4.5 General remarks about the verification of the whole model

If we characterize the verification in the context of the whole model when interactive proving is required, the formulas tend to be divided between simple and complex statements. By "simple statements", we mean formulas such as the following predicate, easily done by pen and paper:

$$
\begin{array}{l}
\quad \mathsf{i} \in \mathbb{Z} \land \mathsf{j} \in \mathbb{Z} \land \mathsf{k} \in \mathbb{Z} \\
\land\ \mathsf{i} \leq \mathsf{j} \\
\Rightarrow \\
\quad \mathsf{min}(\{\mathsf{j}, \mathsf{max}(\{\mathsf{i},\mathsf{k}\})\}) \in \mathsf{i} \mathinner{..} \mathsf{j}
\end{array}
$$

By "complex statements", we mean arithmetically-heavy formulas or formulas for which it is difficult to see if the hypotheses are sufficient. The following formula, where irrelevant have been left out, is an example of arithmetically-heavy formula:

$$
\begin{array}{l}
\quad \ldots \\
\land\ 0 \leq -1 - \mathsf{speed(i)} - \mathsf{acceleration(i)} \\
\land\ -1 - \mathsf{MAX\_SPEED} + \mathsf{speed(i)} + \mathsf{acceleration(i)} + 1 \leq 0 \\
\Rightarrow \\
\quad \mathsf{xpos(i)} - \mathsf{speed(i)} * \mathsf{speed(i)} / (2 * \mathsf{acceleration(i)}) \in \mathbb{N}
\end{array}
$$

This formula corresponds to case of the reaction where the vehicle wants to go backwards. For proving it, we have to infer positivity or negativity of the sub-terms and provide the intermediate deduction steps to the prover.

Formulas involving the absence of collision are part of those formulas for which it is difficult to know if the hypotheses are sufficient. The following formula corresponds to the absence of collision when the vehicle is going at full speed, with the irrelevant hypotheses left out:

$$\begin{array}{l} \forall i \ .(\ i \in dom(xpos) \wedge \ \neg(i = 0) \Rightarrow CRITICAL\_DISTANCE \leq xpos(i-1) - xpos(i) \ ) \\ \wedge \ ... \\ \wedge \ 0 \leq -1 - MAX\_SPEED + speed(i) + acceleration(i) \\ \Rightarrow \\ 0 \leq -MAX\_SPEED - CRITICAL\_DISTANCE + xpos(i-1) - xpos(i) \end{array}$$

This formula can not be proved as is: the missing hypothesis is that the vehicle is already far enough of its leading vehicle so as not to collide it if it stops abruptly. Hence, as stated in Sect. 4.2 we enriched the precondition of the reaction method with the following hypothesis:

$$(\ i \neq 0 \Rightarrow xpos(i-1) - xpos(i) \geq ALERT\_DISTANCE)$$

Once the relationship between ALERT_DISTANCE and CRITICAL_DISTANCE was stated, the absence of collision at full speed could be proved.

Component	Total	Automatic	Interactive
Constants	0	0	0
PhysicalVehicles	16	11	5
Platooning_abs	0	0	0
Platooning	6	6	0
Scheduler_abs	0	0	0
Scheduler	59	59	0
VehiclesControllers	22	19	3
TOTAL	103	95	8

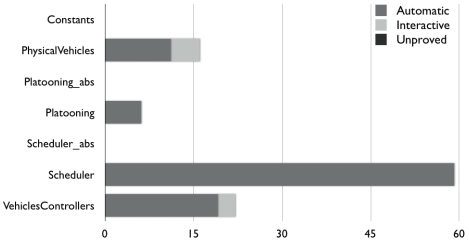

Fig. 4. Proof results: Platooning with soundness property

If we consider the whole platooning model with the sole soundness property, the generated proof obligations consisted of 614 obvious proof obligations and 103 non-obvious proof obligations. The verification of the non-obvious formulas was automatic for 95 of them, leaving 8 formulas to be proved interactively, even if for some of them it is still an overstatement, as there are "simple statements" among them. The precise distribution of proof obligations is given Fig. 4.

Component	Total	Automatic	Interactive	Unproved
Constants	0	0	0	0
PhysicalVehicles	20	11	6	3
Platooning_abs	0	0	0	0
Platooning	7	7	0	0
Scheduler_abs	0	0	0	0
Scheduler	60	60	0	0
VehiclesControllers	23	19	3	1
TOTAL	110	97	9	4

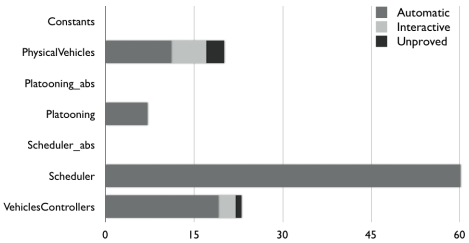

Fig. 5. Proof results: Platooning with no-collision property

As a concluding remark for this section, we can note that when the properties to be proved are linked to the model itself (as the absence of collisions), domain knowledge becomes mandatory, hence a dialogue has to be kept with the experts. For reference, the *current* version of the model with the absence of collision requires the verification of 110 proof obligations, 97 of which are proved automatically, 9 interactively and 4 still to be proved as shown in figure 5.

5 Conclusion

We have proposed a B model for the platooning problem whose soundness has been verified with the help of B proof tools. Expressing additional properties into the model helped us to quickly 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. The advantage of using a formal method here was to avoid resorting to lengthy experimentations in order to understand where the model had flaws: the proof process helped identifying them easily.

The evolution itself of the model is also of interest. While its very first version was a single B model, the successive versions introduced the various layers presented earlier: the focus on longitudinal control, the splitting into an environmental part and an agent part, the three main steps of decision, perception and action, the separation of the constants from the model and the global loops of the model made explicit.

Hence the evolution of this model led us naturally to abstract the architecture from the problem itself. The obtained patterns matched very closely the well-known Influence/Reaction model of the Multi-Agent community [7]. We completed this small gap by proposing generic B patterns [11] suitable for instantiating any MAS expressed with the I/R model. An overview of these patterns is given Fig. 6. With the use of a formal method, the originality of these patterns is to provide a framework for expressing soundness of the instantiated MAS and specifying additional emerging properties in a tool-supported environment. The expectable advantages of this approach were already stated for the particular model of the platooning problem, as an instantiation of the patterns.

Further work includes the study of the same platooning problem with related formalisms such as CSP||B [4] and Event-B [12]. It also includes understanding better the self-referential nature of some emerging properties in the platooning problem: absence of collision, unhooking, oscillation. The goal there would be to see if there are evolution and proof patterns linked to the expression of additional properties in the model.

Acknowledgement. We address our many thanks to Olivier Simonin, Alexis Scheuer and François Charpillet, from the MAIA team of the LORIA, for common efforts and fruitful discussions in the context of the TACOS and CRISTAL projects.

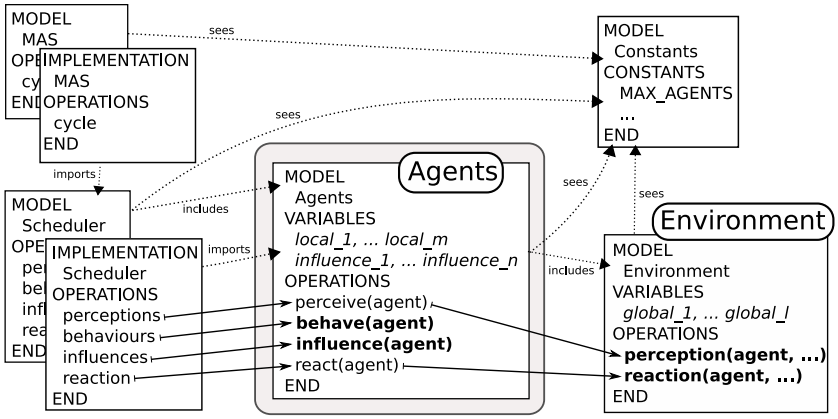

Fig. 6. B patterns of the I/R model

References

1. Hilaire, V., Gruer, P., Koukam, A., Simonin, O.: 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)
2. Regayeg, A., Kacem, A.H., Jmaiel, M.: Specification and verification of multi-agent applications using temporal z. In: Intelligent Agent Technology Conf. (IAT'04), IEEE Computer Society (2004) 260–266
3. Inverno, M., Saunders, R.: Agent-based modelling of Stem Cell organisation in a Niche. In Brueckner, S.A., Di Marzo, G., Karageorgos, S.A., Nagpal, R., eds.: Engineering Self-Organising Systems : Methodologies and Applications. LNAI, Springer-Verlag (2005)
4. Schneider, S., Cavalcanti, A., Treharne, H., Woodcock, J.: A layered behavioural model of platelets. In: 11th IEEE International Conference on Engieerging of Complex Computer Systems, ICECCS. (2006)
5. Ball, E., Butler, M.: Event-B patterns for specifying fault-tolerance in Multi-Agent interaction. In: Proceedings of Methods, Models and Tools for Fault Tolerance, Oxford, UK (2007)
6. Daviet, P., Parent, M.: Longitudinal and lateral servoing of vehicles in a platoon. In: Proceeding of the IEEE Intelligent Vehicles Symposium. (1996) 41–46
7. Ferber, J., Muller, J.P.: Influences and reaction : a model of situated multiagent systems. In: 2nd Int. Conf. on Multi-agent Systems. (1996) 72–79
8. Abrial, J.R.: The B Book. Cambridge University Press (1996)
9. 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
10. Dijkstra, E.W.: A Discipline of Programming. Prentice Hall (1976)
11. Simonin, O., Lanoix, A., Colin, S., Scheuer, A., Charpillet, F.: Generic Expression in B of the Influence/Reaction Model: Specifying and Verifying Situated Multi-Agent Systems. INRIA Research Report 6304, INRIA (2007)
12. Abrial, J.R., Hallerstede, S.: Refinement, decomposition, and instantiation of discrete models: Application to Event-B. Fundamenta Informaticae **77**(1-2) (2007) 1–28 Special issue on ASM'05.