Generic Expression in B of the Influence/Reaction Model: Specifying and Verifying Situated Multi-Agent Systems

Olivier Simonin, Arnaud Lanoix, Samuel Colin, Alexis Scheuer, François Charpillet

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Generic Expression in B of the Influence/Reaction Model: Specifying and Verifying Situated Multi-Agent Systems

Olivier Simonin — Arnaud Lanoix — Samuel Colin — Alexis Scheuer —

François Charpillet

N° 6304

September 2007

Thème COG

Rapport de recherche

ISSN 0249-6399

Generic Expression in B of the Influence/Reaction Model: Specifying and Verifying Situated Multi-Agent Systems[*]

Olivier Simonin[†] , Arnaud Lanoix[‡] , Samuel Colin[§] , Alexis Scheuer[†] , François Charpillet[¶]

Thème COG — Systèmes cognitifs
Projets MAIA et DEDALE

Rapport de recherche n° 6304 — September 2007 — 18 pages

Abstract: This paper addresses the formal specification and verification of situated multi-agent systems that can be formulated within the influence-reaction model as proposed in 1996 by Ferber & Muller. In this framework our objective is to prove the correctness of reactive multi-agent systems with respect to a certain formal specification or property, using formal methods. This is an important step to bring multi-agent systems to high quality standards as required for critical applications encountered in domains such as transport systems. A generic B writing of systems instantiating the influence reaction model is proposed, using patterns of specification. An illustration is then presented on the formal specification of a system operating electrical vehicles under precise automatic control at close spacings to form a platoon. The papers ends with considerations about further improvements of the framework, involving simulation and study of the properties of the system.

Key-words: Situated MAS, Influence/Reaction model, B method, Design patterns, Platooning

[*] This work was partially supported by the french national research agency ANR-06-SETI-017 (TACOS).
[†] Henri Poincaré University (Nancy 1, `http://uhp-nancy.fr`) and MAIA team (`http://maia.loria.fr`).
[‡] DEDALE team (`http://dedale.loria.fr`).
[§] Nancy 2 University (`http://univ-nancy2.fr`) and DEDALE team (`http://dedale.loria.fr`).
[¶] MAIA team (`http://maia.loria.fr`).

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Formulation générique en B du modèle influence/réaction pour spécifier et vérifier des systèmes multi-agents situés

Résumé : Cet article vise à spécifier et vérifier formellement les systèmes multi-agents situés qui peuvent être formulés dans le modèle influence-reaction proposé par Ferber & Muller en 1996. Dans ce cadre, notre objectif est de prouver, avec des outils formels, la cohérence de systèmes multi-agents réactifs par rapport à une specification ou une propriété donnée. Il s'agit d'un pas important pour parvenir à la définition de SMA respectant des normes de hautes qualités exigées par les applications critiques, comme par exemple dans le domaine des transports. Ainsi, nous proposons une expression en B du modèle influence-reaction, selon des patterns de spécification génériques. Nous illustrons cette proposition par la specification formelle d'un système de véhicules intelligents évoluant de façon autonome et à faible distance pour former un convoi (un platoon). Cette étude s'achève par une reflexion sur l'évolution du modèle proposé, en considérant en particulier les objectifs de simulation et d'étude des propriétés des systèmes.

Mots-clés : SMA situé, Modèle Influence/Réaction, Méthode B, Schemas de conceptions, Platooning

1 Introduction

We aim at defining a general approach to formally specify and study situated Multi-Agent Systems (MAS), i.e. systems composed of agents which evolve in a physical environment. We are especially interested in addressing critical decentralised systems in which autonomous components interact with each other to form a complex system. The MAS approach makes possible the modeling of such systems following a bottom-up methodology. The components (agents), their behaviours, the way they interact with each other or with the environment define the local level. It gives rise to the collective behaviours of a system (the global level) whose properties are not always predictable from the local level. This approach permits the definition of flexible decentralised systems able to (self)organise. Such systems are appealing with respect to their faculty of adaptation but they are difficult to study. Specifying and verifying properties of such systems remains an open issue, that we attack in this paper.

Our approach consists in choosing the framework Influence/Reaction [14], which is one of the few models expressing dynamics of situated MAS. It is also simple enough to make the challenge of verification reachable for simple properties.

In order to specify the influence/reaction model, we adopt the B method, used for modelling and reasoning about systems. The B method already demonstrated its ability to verify industrial-strength software for autonomous systems (as e.g. French automatic subway [7]). We aim here at exceeding the software framework so as to specify the physical part of situated MAS, as the overall dynamics of such systems depend on the interactions between the agents and their environment, i.e. a physical world. B only works at the level of integers, hence it seems not adapted to the modelling of MAS which are intrinsically continuous systems. However we circumvent this limitation by abstracting over the granularity of the model, i.e. the physical units of the variables.

In this paper we propose B design patterns able (i) to express the main parts of situated MAS and their dynamics following the Influence/Reaction definitions (ii) to help in verifying their local and global properties. Design patterns are a good way of communicating expertise by capturing the solutions to recurring design problems and re-using those solutions. We illustrate how our B patterns can be instantiated to study a specific MAS, by focusing on the platooning task.

The paper is organised as follows. Section 2 presents the Influence/Reaction model in order to clearly express dynamics of situated MAS. Then we propose in section 3 a generic expression in B for the I/R model using design patterns. Section 4 shows how patterns can be instantiated to study a real complex case: a reactive MAS model for coordination of several autonomous vehicles. Section 5 presents further improvements of the overall framework and section 6 discusses the related works. Eventually section 7 concludes and gives some perspectives.

2 The Influence/Reaction model

2.1 Principle of the model

The difficulty of designing and studying situated MAS comes from the autonomy of agents and their interactions within a common environment. They are highly distributed systems, where agents evolve in parallel, and more generally work in a dynamic environment. It is then difficult to formally express/simulate such systems and to predict their global behaviour [13, 15].

In particular, most MAS models do not allow to express simultaneous actions. They propose only a sequential representation of actions, which is not generally equivalent. For instance two robots situated on both sides of a door, and trying to open and close it simultaneously, should fail as their forces are balanced. In a sequential representation of actions, the door will sequentially be open and closed, or vice versa.

Ferber & Muller proposed in [14] the Influence/Reaction (I/R) model in order to express clearly the dynamics of such systems considering a discretization of time. In this model agents are described separately from the environment dynamics but connected to it by computing at each step which state they perceive (perceptions) and which influences they produce (reaction). This dichotomy allows mainly to compute the result of simultaneous actions performed by different agents at a given time. Indeed, the new state of the system is defined as the combination of the different influences produced by the agents. Considering the previous example, the robots will produce two influences that will be combined to define a null force. As a consequence the door will not move, which is the consistent behaviour of a real experiment.

The I/R model considers interactions between agents and the environment as a system composed of two dynamical sub-systems, which are coupled through agent perceptions of the environment and agent actions modifying the whole

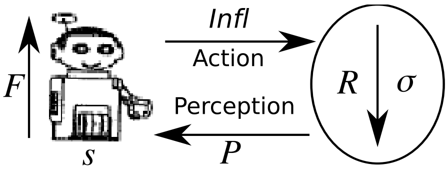


Figure 1: Interaction between an agent and its environment in the I/R model [13]

system (see fig. 1). The I/R model if formalised by

$$\left\{ \begin{array}{l} \left\{ \begin{array}{l} s_1(t+1) = F_1(s_1(t), P_1(\sigma(t))) \\ \dots \\ s_n(t+1) = F_n(s_n(t), P_n(\sigma(t))) \end{array} \right. \\ \\ \sigma(t+1) = R(\sigma(t), \prod_i Infl_i(s_i(t))) \end{array} \right.$$

where $s_i(t)$ is the internal state of the i^{th} agent at time t and $\sigma(t)$ the state of the environment at the same time t. P_i is a perception function from the environment to the internal state of the i^{th} agent. F_i is a behavioural function that computes the new internal state of the i^{th} agent from its perceptions and its previous state. $Infl_i$ is an action function that produces a set of influences on the environment, for the i^{th} agent. At last, R is the reaction function computing the new state of the environment from its current state and the combination of all the influences produced by agents. Influences are combined thanks to the $\prod$ operator. Details about definition of each function can be found in [13].

The evolution of the system is then expressed according to the following loop:

1. *at time t, each agent perceives, decides and produces influences on the environment ;*
2. *all the influences are combined to compute the new state of the whole system at time* $t+\Delta t$ *;*
3. $t \leftarrow t+\Delta t$ *, return to step 1.*

We can now specify the main advantages of using the I/R model to express a situated MAS:

- it allows the computation of simultaneous actions (or influences) thanks to the second operation of the loop: the R function;
- it ensures that agents decide at a time t according to perception of the environment at this date.
- it allows the computation of the new state of the whole system at a specific time $t+\Delta t$, while the time for computing this new state is independent of Δt duration (it corresponds to one iteration of the simulation loop);

2.2 Representation of the I/R model

We represent all the elements of the system by sets of variables. We clearly differentiate three kinds of variables: (i) external variables representing the state of the system, called *global variables*, (ii) internal variables of agents, called *local variables* and (iii) *influences* which are intended actions produced by agents.

- Global variables express $\sigma(t)$, defining objects and agents in the environment (e.g. position, speed, etc.), denoted $global_1, \dots, global_i, \dots, global_l$;
- Local variables express the internal state of agents $s(t)$, which are perceptions or knowledge (memory), denoted $local_1, \dots,\ local_j, \dots, local_m$;
- Influences express the actions produced by agents in order to change the system, denoted $influence_1, \dots, influence_k, \dots, influence_n$. Note that a communication is a particular influence produced by an agent and that can be perceived by one or several agents.

To reduce the complexity of B specification and property proofs we consider that agents are identical and defined by the same set of local variables and influences (its explains why we avoided mentioning agent indexes in previous sets of variables). However heterogeneity can be easily introduced by ignoring some variables in some agents. The types of the variables can be heterogeneous. Their definition will be explained along next sections.

We now express the four main functions involved in the Influence/Reaction model as a set of functions on variables.

- *Perceive (P)*: each local variable corresponding to a sensor is updated with *perceive* functions

$$local_j = perceive_j(global_1, \ldots, global_l)$$

- *Behave (F)*: the internal state of an agent can change using *behave* functions

$$local_k = behave_k(local_1, \ldots, local_m)$$

- *Infl*: an agent can produce a new influence with *infl* functions

$$influence_k = infl_k(local_1, \ldots, local_m)$$

- *React (R)*: the environment updates its global variables with *react* functions

$$global_i = react_i(global_1, \ldots, global_l, influence_1, \ldots, influence_n)$$

As mentioned by I/R authors [14] the react functions shall be realistic enough, i.e. able to express the desired properties of the system. For instance, it might not be necessary to include the altitude to model the position of robots moving on a plane ground.

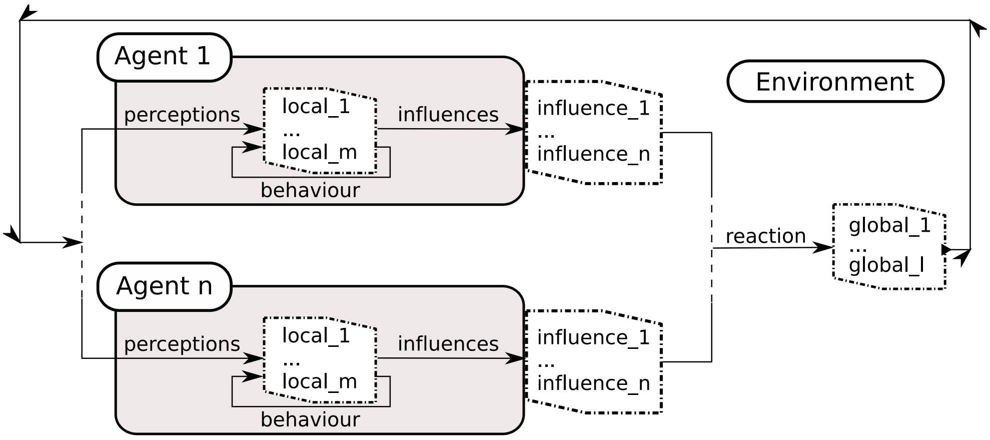


Figure 2: Representation of the I/R model

Figure 2 abstracts the whole I/R model, using sets of variables to express the states of the system. The evolution of the whole system is computed using the following *cycle* function: all perceptions → all behaviours → all influences → reaction → . . . We propose in the next section a generic B writing of this cycle function, for which a complete example is given in section 4.

3 Generic B patterns of I/R model

3.1 Specifying in B

The B method is a formal software development method used to model and reason about systems [2]. It is based on set theory and relations. Software development in B supports abstractly specifying the requirements of the systems and then refining these requirements through several steps to create a concrete description of the system which can be automatically translated to code. This incremental development process is called "refinement". It is a key feature for incrementally developing more and more detailed models, preserving correctness in each step. Each model consists in variables representing the state of the model, methods representing the possible evolutions of this state and an invariant specifying the safety requirements. Each method consists of a precondition part which is a predicate conditioning the activation of the method and a substitution part specifying the effects of the considered method.

The B method has been successfully applied in the development of several complex real-life applications, such as the Meteor project [7], the Roissy VAL [4], the Coppilot project [18] or the CaColac project [10]. It is one of the few formal methods which has robust and commercially available support tools for the entire development life-cycle, from specification down to code generation [8].

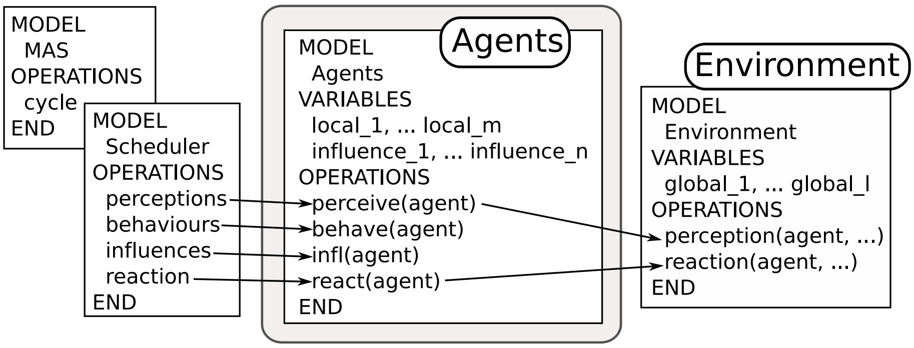


Figure 3: B patterns of the Influence/reaction model

Proofs of invariant consistency and refinement are part of each development. These *Proof Obligations* (POs) are generated from the model by applying certain semantic rules and automatically by support tools such as AtelierB [22], B4free [11] or the B-toolkit [3]. Then, they can be proven with B support tools or by any tool supporting first-order logic with set theory and axioms for the basic datatypes of B. Checking POs is an efficient and practical way to detect errors introduced during the development and to validate the correctness of the specified models.

3.2 An B patterns architecture of the I/R model

We propose a generic expression in B of the I/R model. We introduce formal B design patterns, following the shape of figure 2. The figure 3 shows a general rewriting of the I/R model using B concepts: (i) two generic B patterns express the *cycle* function of the I/R model, (ii) one pattern expresses the environment evolution, and (iii) one pattern expresses the agents behaviours. The next sections explore these various parts more precisely. Note that the given B models are not complete. They are only patterns and they have to be filled in for a specific system as it is illustrated in section 4.

The B architecture of the I/R patterns is given in figure 4, in terms of "inclusion" and "refinement" relations.

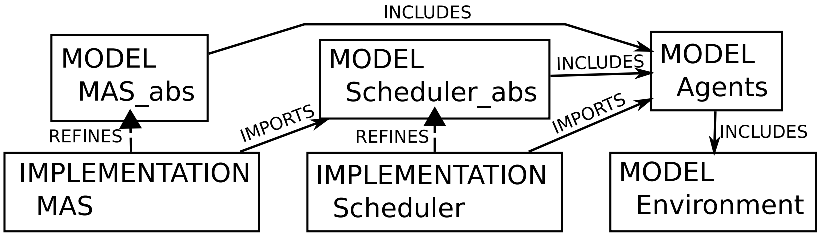


Figure 4: Architecture of the B patterns

3.3 The cycle Function of the I/R Model

To formalise the I/R model in B, we begin by expressing the cycle function, which is the loop on all the perceptions, all the decisions (behaviours and influences) and the global reaction. All these steps are characterised by a *step* variable valued by PERCEIVE, BEHAVE, INFL and REACT. We give a first B pattern called MAS that expresses the loop. In order to use a "concrete" loop construction (the WHILE loop), the B method requires two level of specifications, as shown in figure 5:

- at the abstract level, the operation cycle only states that the steps are done, without specifying *how*;
- at the implementation level, the loop is made explicit using a WHILE construction, that calls successively four "intermediate" methods perceptions, behaviours, influences and reaction, described in another B model and corresponding to the four global steps of the I/R model.

Figure 6 shows the abstract and the concrete views of the four methods perceptions, behaviours, influences and actions written into a second B pattern called Scheduler. The perceptions method expresses a loop on all the perceptions of all the agents. As previously mentioned, to give a WHILE construction, we must use two level of specifications. On the one hand, the abstract level expresses only that all the perceptions are done. On the other hand, the implementation level calls a

```
MODEL MAS_abs
INCLUDES Agents
OPERATIONS
cycle =
  PRE step = PERCEIVE
  THEN wholeSteps
  END
END
```

(a) abstract level

```
IMPLEMENTATION MAS
REFINES MAS_abs
IMPORTS Scheduler
OPERATIONS
cycle =
  BEGIN
    VAR i IN
      i := MAXINT;
      WHILE i > 0 DO
        perceptions ; next_step ;
        behaviours ; next_step ;
        influences ; next_step ;
        reaction ; next_step ;
        i := i − 1
      INVARIANT i ∈ NAT ∧ step = PERCEIVE
      VARIANT i
      END
    END
  END
END
```

(b) implementation level

Figure 5: The MAS B pattern

perceive method in turn for each agent of the system. The same goes for the behaviours, influences and reaction methods, by calling, respectively behave, infl and react, for each agent.

A last method next_step appears in Scheduler to implement the progress of step: from PERCEIVE to BEHAVE, from BEHAVE to INFL, etc.

The MAS and Scheduler B patterns we have presented so far are completely generic. They are parameterised only by the number of agents MAX_AGENTS and assume the existence of two other B models: Agents that specifies the behaviour of all the agents and Environment that gives a model of the laws of the world the agents evolve in.

The desired properties of the system will be specified in the environmental part of the model, and the proof of the B models will highlight what it is required of the behavioural part to be able to ensure those properties. Let us now describe more precisely how each of the environmental and behavioural part is modelled.

3.4 The Environment B pattern

The pattern shown in figure 7 gives a general B model of the environment. It contains all the variables $global_1, \ldots, global_l$ used to state the environment. Each $global_i$ variable can model:

- specific data of each agent represented by arrays associating the data to the agents (for instance, their location). These arrays $global_i$ are represented in B by total functions defined between the numbers of agents (the indices) 0..MAX_ AGENTS and the values contained thereinto;

- data not associated to any agent, but useful for modelling reason, as a global information necessary for all agents (e.g. location of other objects different from the agents). Note that the B pattern given in figure 7 focuses only on specific data of each agent.

The INVARIANT part of the generic model contains only typing predicates on the variables of the model. Safety properties can also be expressed here once the model has been instantiated for a specific problem.

The dynamics of the environment result from the "physical" interactions the agents have with it. These interactions consist of two methods for *sensing* the environment and for *acting* on it. These methods are called perception and reaction respectively in figure 7:

- perception is a method called by each agent to perceive its environment. It takes as parameters the old perceptions $local_j$ of the agent, and return the new ones, because B has no syntax for updating the value associated with an index through an operation call. We must express into perception the functions $local_j = perceive_j(global_1, \ldots, global_l)$ presented in section 2.2 that link together the perceptions $local_j$ and the environment variables[1]. Noise or errors of the sensors can be imitated here, for instance.

- reaction is a method called by each agent so that influences are combined in order to perform its actions. Then actuators change the global variables of the environment. This method takes as parameters the $influences_k$ influences decided by the considered agent and the others. The functions $global_i = react_k\ (global_1, \ldots, global_l, influence_1,$

[1] Example of the $\mathbin{<\!\!+}$ operator: $\{a \mapsto 1, b \mapsto 5\} \mathbin{<\!\!+} \{b \mapsto 2, c \mapsto 3\} = \{a \mapsto 1, b \mapsto 2, c \mapsto 3\}$.

```
MODEL Scheduler_abs
INCLUDES Agents
OPERATIONS
next_step = /* ... */
  BEGIN
    SELECT step = PERCEIVE
      THEN stepUpdate(BEHAVE)
    WHEN step = BEHAVE
      THEN stepUpdate(INFL)
    WHEN step = INFL
      THEN stepUpdate(REACT)
    WHEN step = REACT
      THEN stepUpdate(PERCEIVE)
    END
  END;
perceptions =
  PRE step = PERCEIVE
  THEN wholePerceive
  END ;
behaviours = /* ... */
  PRE step = BEHAVE
  THEN wholeBehave
  END ;
influences = /* ... */
  PRE step = INFL
  THEN wholeInfl
  END ;
reaction = /* ... */
  PRE step = REACT
  THEN wholeAct
  END
END
```

(a) abstract level

```
IMPLEMENTATION Scheduler
REFINES Scheduler_abs
IMPORTS Agents
OPERATIONS
next_step = /* ... */
  BEGIN
    IF step = PERCEIVE THEN stepUpdate(BEHAVE)
    ELSE
      IF step = BEHAVE THEN stepUpdate(INFL)
      ELSE
        IF step = INFL THEN stepUpdate(REACT)
        ELSE
          IF step = REACT THEN stepUpdate(PERCEIVE)
          ELSE skip
          END
        END
      END
    END
  END;
perceptions =
  BEGIN
    VAR agent IN
      agent := MAX_AGENTS + 1;
      WHILE agent > 0 DO
        agent := agent − 1;
        perceive(agent)
      INVARIANT agent ∈ 0..(MAX_AGENTS+1)
      VARIANT agent
      END
    END
  END ;
behaviours = /* ... */
  BEGIN
    VAR agent IN
      agent := MAX_AGENTS + 1;
      WHILE agent > 0 DO
        agent := agent − 1;
        behave(agent)
      INVARIANT agent ∈ 0..(MAX_AGENTS+1)
      VARIANT agent
      END
    END
  END ;
influences = /* ... */
  BEGIN
    VAR agent IN
      agent := MAX_AGENTS + 1;
      WHILE agent > 0 DO
        agent := agent − 1;
         infl (agent)
      INVARIANT agent ∈ 0..(MAX_AGENTS+1)
      VARIANT agent
      END
    END
  END ;
reaction = /* ... */
  BEGIN
    VAR agent IN
      agent := MAX_AGENTS + 1;
      WHILE agent > 0 DO
        agent := agent − 1;
        act(agent)
      INVARIANT agent ∈ 0..(MAX_AGENTS+1)
      VARIANT agent
      END
    END
  END
END
```

(b) implementation level

Figure 6: The Scheduler B pattern

```
MODEL Environment
VARIABLES
   global_1,  ...,  global_i ,  ...,  global_l
INVARIANT
  global_i ∈ 0..MAX_AGENTS → /* global_i type */
INITIALISATION
  global_i := /* global_i  init  */
OPERATIONS
new_local_1 ,..., new_local_m ⟵ perception(agent,...,local_j ,...)  =
  PRE
    agent ∈ 0..MAX_AGENTS ∧
  ∧ local_j ∈ 0..MAX_AGENTS → /* local_j type */
  THEN
    new_local_j :=  local_j ⩤ {agent ↦ /*perceive_j(global_1,...,global_l)  */ }
  END ;
reaction(agent ,..., influence_k ,...)  =
  PRE
    agent ∈ 0..MAX_AGENTS
  ∧ influence_k ∈ 0..MAX_AGENTS → /* influence_k type */
  THEN
     global_i  := global_i ⩤ {agent ↦ /* react_i(global_1,...,global_l ,
                                                   influence_1 ,..., influence_n)  */ }
  END ;
wholeReactions = /* ...  */
  BEGIN
    ANY ...,fresh_global_i ,...
    WHERE
      fresh_global_i ∈ 0..MAX_AGENTS → /* global_i type */
    THEN
      global_i  := fresh_global_i
    END
  END
END
```

Figure 7: The Environment B pattern

$\ldots, influence_n)$ linking environment data and influences have to be expressed here, by way of combining the influences. Noise or errors on the actuators can also be imitated here.

Moreover, the precondition part of both methods can be augmented to take into account safety properties, once the model has been instantiated for a specific problem.

A last method called wholeReactions can be automatically written in Environment. This method exists for proof reasons and express in an abstract manner that all the data are "fresh".

3.5 The Agents B pattern

Each agent is determined by its local variables, which characterise its perceptions and its memories, and by its influences. We specify all the agents in the same B pattern given in figure 8. The $local_j$ variables are arrays associating each agent to its perceptions (likewise for the $influence_k$ variables). In B, these arrays are expressed by total functions from the agents to the perceptions (respectively influences). As before, the INVARIANT part of Agents can also contain safety properties between the $local_j$ and $influence_k$ variables.

The perceive and react methods express the agent actions at the PERCEIVE and REACT steps of the I/R model. They are generic because their bodies correspond only to operations calls on the environment:

- perceive calls the perception method of Environment to perceive the environment data, and
- react calls the reaction method of Environment to propagate its influences into the environment

The two methods to be filled in are behave and infl. They express the behaviour functions $local_j = behave_j(local_1, \ldots, local_m)$ and the influence functions $influence_k = infl_k(local_1, \ldots, local_m)$ presented in section 2.2.

There are other methods appearing in the B model:

- stepUpdate is used by the scheduler to change the step in the IR cycle;
- wholePerceive, wholeBehave, wholeInfl, wholeReact and wholeSteps are here for proof reasons and express respectively that all the perceptions, all the decisions, all the acts, or all these steps are done. Only wholePerceive are shown, the others are similar.

```
MODEL Agents
INCLUDES Environment
CONCRETE_VARIABLES
  step
VARIABLES
  local_1,  ...,  local_j ,  ...,  local_m,
  influence_1,  ...,  influence_k,  ...,  influence_n
INVARIANT
  step ∈ STEP ∧
∧ local_j  ∈ 0..MAX_AGENTS → /* local_j type */
∧ influence_k ∈ 0..MAX_AGENTS → /* influence_k type */
INITIALISATION
  step := PERCEIVE
|| local_j   := /* local_j  init  */
|| influence_k  := /* influence_k  init  */
OPERATIONS
perceive(agent) =  /*  ...  */
  PRE
    agent ∈ 0..MAX_AGENTS ∧ step = PERCEIVE
  THEN
    local_1 ,  ...,  local_m ⟵ perception(agent, local_1,  ...,  local_m)
  END ;
behave(agent) =
  PRE
    agent ∈ 0..MAX_AGENTS ∧ step = BEHAVE
  THEN
     local_j  := local_j <+ {agent ↦ /* behave_j(local_1, ..., local_m) */ }
  END ;
 infl (agent) =
  PRE
    agent ∈ 0..MAX_AGENTS ∧ step = INFL
  THEN
    influence_k := influence_k <+ {agent ↦ /* infl_k(local_1,  ...,  local_m) */ }
  END ;
react(agent)= /*  ...  */
  PRE
   agent ∈ 0..MAX_AGENTS ∧ step = REACT
  THEN
    reaction(agent, influence_1,  ...,  influence_n)
  END ;
stepUpdate(new_step) =  /* ...  */
  PRE new_step ∈ STEP
  THEN step := new_step
  END ;
wholePerceive = /* ...  */
  PRE step = PERCEIVE
  THEN
    ANY ...,  fresh_local_j ,  ...
    WHERE
      fresh_local_j  ∈ 0..MAX_AGENTS → /* local_j type */
    THEN
      local_j  := fresh_local_j
    END
  END ;
wholeBehave = /* ...  */
wholeInfl  = /*  ...  */
wholeReact = /* ...  */
wholeSteps = /* ...  */
END
```

Figure 8: the Agents B pattern

4 Application to a platooning model

This section shows how the proposed B patterns can be instantiated for a specific MAS problem. The platooning task is defined as a set of autonomous vehicles which have to move in convoy, i.e. following the path of the leader (possibly driven by a human being) in a row (or platoon) and that should tend to an ideal distance between each other. We are considering platooning problems for which lateral control and longitudinal control are independent [12]. For sake of simplicity, we only present in this paper the longitudinal control.

4.1 A one dimensional platooning

We consider a set of vehicles moving in a one dimensional space, where the leader is quoted by number 0 and the last one by $MAX\_VEHICLES$. Physical position of the i^{th} vehicle is represented by a natural number $xpos_i$, while its dynamics is given as a natural number by its velocity $speed_i$. Please note that the physical units for length are left unspecified. Hence the distance can represent millimetres as well as meters.

Each vehicle has sensors to estimate its velocity p_speed_i, as well as the distance to the leading (i.e. previous) vehicle p_dist_i, and the velocity of this vehicle $p\_l\_speed_i$. Each vehicle selects its instantaneous acceleration $accel_i$ according to sensor values.

The perceptions p_speed_i, p_dist_i and $p\_l\_speed_i$ are supposed perfect:

$$\left\{\begin{array}{ll} p\_speed_i(t) = speed_i(t) & \\ p\_dist_i(t) = xpos_{i-1}(t) - xpos_i(t) & \text{, if } i > 0 \\ p\_l\_speed_i(t) = speed_{i-1}(t) & \text{, if } i > 0 \end{array}\right. \tag{1}$$

The acceleration $accel_i$ is decided using:

$$\left\{\begin{array}{ll} \text{if } i = 0, & accel_i(t+\Delta t) = \frac{IdealSpeed - p\_speed_i(t)}{\Delta t} \\ \text{otherwise,} & accel_i(t+\Delta t) = \frac{p\_dist_i(t) - IdealDist_i(t)}{2\Delta t^2} + \frac{p\_l\_speed_i(t) - p\_speed_i(t)}{\Delta t} \end{array}\right. \tag{2}$$

where $IdealDist$ is a function on agent speed or simply a constant given the ideal distance between two vehicles. This behaviour is not explained because is out of the scope of this paper.

Now, we express physical laws computing the dynamics of the vehicles. The reaction of the vehicles stems from their decision to accelerate or slow down in order to stay within an ideal distance from each other. It consists in updating the global variables $xpos_i$ and $speed_i$ according to their old values and the decided acceleration $accel_i$ using:

$$\begin{array}{ll} 1. & cons\_speed = speed_i(t) + accel_i.\Delta t \\ 2. & \left\{\begin{array}{lll} \text{if } cons\_speed > MaxSpeed, & \left\{\begin{array}{l} xpos_i(t+\Delta t) = xpos_i(t) + \Delta t.MaxSpeed \\ speed_i(t+\Delta t) = MaxSpeed \end{array}\right. & (3.1) \\ \text{if } cons\_speed < 0, & \left\{\begin{array}{l} xpos_i(t+\Delta t) = xpos_i(t) - \frac{speed_i(t)^2}{2.accel_i(t)} \\ speed_i(t+\Delta t) = 0 \end{array}\right. & (3.2) \\ \text{otherwise,} & \left\{\begin{array}{l} xpos_i(t+\Delta t) = \left(\begin{array}{l} xpos_i(t) + speed_i(t).\Delta t \\ + \frac{accel_i.\Delta t^2}{2} \end{array}\right) \\ speed_i(t+\Delta t) = cons\_speed \end{array}\right. & (3.3) \end{array}\right. \end{array}$$

where $MaxSpeed$ expresses the maximal velocity of a vehicle. Note that for this model we assume that actuators are perfect.

4.2 B specification of platooning

The architecture of the B models follows the patterns given in section 3, with a few renaming:

Environment becomes PhysicalVehicles

Agents becomes VehiclesControllers

Scheduler and the corresponding implementation are kept

MAS and the corresponding implementation are renamed Platooning

```
MODEL PhysicalVehicles
ABSTRACT_CONSTANTS
  new_xpos_when_max_speed,
  new_xpos_when_neg_speed,
  new_xpos_others,
PROPERTIES
  new_xpos_when_max_speed ∈ ℤ → ℤ
∧ new_xpos_when_max_speed = λ(xpos).(xpos ∈ ℕ | xpos + MAX_SPEED × TIME_STEP)
∧ new_xpos_when_neg_speed ∈ (ℤ × ℤ × ℤ) → ℤ
∧ new_xpos_when_neg_speed = λ(xpos,speed,accel).((xpos ∈ ℤ ∧ speed ∈ ℤ ∧ accel ∈ ℤ) | xpos − (speed∗speed) / (2∗accel))
∧ new_xpos_others ∈ (ℤ × ℤ × ℤ) → ℤ
∧ new_xpos_others = λ(xpos,speed,accel).((xpos ∈ ℤ ∧ speed ∈ ℤ ∧ accel ∈ ℤ) | xpos + speed × TIME_STEP + (accel × TIME_STEP × TIME_STEP) / 2)
VARIABLES
  speed,            /∗ real vehicles' speed ∗/
  xpos             /∗ real vehicles' position ∗/
INVARIANT
  speed ∈ 0..MAX_VEHICLES → 0..MAX_SPEED
∧ xpos ∈ 0..MAX_VEHICLES → ℕ
INITIALISATION
  speed := (0..MAX_VEHICLES) × {0} ||
  xpos := 0..MAX_VEHICLES ◁ positioner
OPERATIONS
new_perceived_speed,
new_perceived_distance,
new_perceived_front_speed
 ⟵ perception(i, old_perceived_speed, old_perceived_distance, old_perceived_front_speed) =
  PRE
    i ∈ 0..MAX_VEHICLES
  ∧ old_perceived_speed ∈ 0..MAX_VEHICLES → 0..MAX_SPEED
  ∧ old_perceived_distance ∈ 0..MAX_VEHICLES → ℤ
  ∧ old_perceived_front_speed ∈ 0..MAX_VEHICLES → 0..MAX_SPEED
  THEN
    new_perceived_speed := old_perceived_speed <+ { i ↦ speed(i)}
  || IF i = 0
    THEN
      new_perceived_distance := old_perceived_distance <+ { 0 ↦ 0 }
    || new_perceived_front_speed := old_perceived_front_speed <+ { 0 ↦ 0 }
    ELSE
      new_perceived_distance := old_perceived_distance <+ { i ↦ (xpos(i−1) − xpos(i)) }
    || new_perceived_front_speed := old_perceived_front_speed <+ { i ↦ speed(i−1) }
    END
  END;
wholeReaction = /∗ ... ∗/
reaction(i, acceleration) =
  PRE
    i ∈ 0..MAX_VEHICLES ∧
    acceleration ∈ 0..MAX_VEHICLES → MIN_ACCEL..MAX_ACCEL
  THEN
    ANY considered_speed
    WHERE considered_speed = speed(i) + acceleration(i) × TIME_STEP
    THEN
      IF (considered_speed > MAX_SPEED)
      THEN
        xpos(i) := new_xpos_when_max_speed(xpos(i)) ||
        speed(i)  := MAX_SPEED
      ELSE
        IF (considered_speed < 0)
        THEN
          xpos(i) := new_xpos_when_neg_speed(xpos(i),speed(i),acceleration(i)) ||
          speed(i)  := 0
        ELSE
          xpos(i) := new_xpos_others(xpos(i),speed(i),acceleration(i)) ||
          speed(i)  := considered_speed
        END
      END
    END
  END
END
```

Figure 9: The PhysicalVehicles model

```
MODEL VehiclesControllers
INCLUDES PhysicalVehicles
CONSTANTS
  compute_new_accel
PROPERTIES
  compute_new_accel ∈ (ℤ × ℤ × ℤ) → ℤ
∧ compute_new_accel = λ(d,vd,v).
  (
    (d ∈ ℤ ∧ vd ∈ ℤ ∧ v ∈ ℤ)
  |
    2 × (d − IDEAL_DISTANCE + (vd − v) × TIME_STEP)
    /  (TIME_STEP × TIME_STEP)
  )
CONCRETE_VARIABLES
  step
VARIABLES
  perceived_speed,
  perceived_distance,
  perceived_front_speed,
  accel_decision
INVARIANT
  step ∈ STEP
∧ perceived_speed ∈ 0..MAX_VEHICLES → 0..MAX_SPEED
∧ perceived_distance ∈ 0..MAX_VEHICLES → ℤ
∧ perceived_front_speed ∈ 0..MAX_VEHICLES → 0..MAX_SPEED
∧ accel_decision ∈ 0..MAX_VEHICLES → MIN_ACCEL..MAX_ACCEL
INITIALISATION
  step := PERCEIVE ||
  perceived_speed := (0..MAX_VEHICLES) × {0} ||
  perceived_distance := (0..MAX_VEHICLES) × {IDEAL_DISTANCE} ||
  perceived_front_speed := (0..MAX_VEHICLES) × {0} ||
  accel_decision := (0..MAX_VEHICLES) × {0}
OPERATIONS
stepUpdate(new_step) = /* ...  */
wholePerceive =   /*  ...   */
perceive(i) =
  PRE
     i ∈ 0..MAX_VEHICLES
   ∧ step = PERCEIVE
  THEN
     perceived_speed, perceived_distance, perceived_front_speed
       ←— perception(i,  perceived_speed, perceived_distance, perceived_front_speed)
  END;
wholeInfl =    /*  ...   */
 infl (i) =
  PRE
     i ∈ 0..MAX_VEHICLES
   ∧ step = INFL
  THEN
     IF i = 0
     THEN
        ANY new_accel
        WHERE new_accel = ( IDEAL_SPEED − perceived_speed(i) ) / TIME_STEP
        THEN
           accel_decision(i) := min({MAX_ACCEL, max({MIN_ACCEL, new_accel})} )
        END
     ELSE
        IF (perceived_distance(i) < ALERT_DISTANCE)
        THEN
           accel_decision(i) := MIN_ACCEL
        ELSE
           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
        END
     END
  END ;
wholeReact =   /*  ...   */
react(i) =
  PRE
     i ∈ 0..MAX_VEHICLES
   ∧ step = REACT
  THEN
     reaction(i, accel_decision)
  END;
wholeSteps =  /*  ...   */
END
```

Figure 10: The VehiclesControllers model

As can be seen, the new names of the Environment and Agents models reflect how the system is thought: PhysicalVehicles contains the modelling of the physical world and its laws. VehiclesControllers contains all the internal decision mechanism of the agents.

The properties we are interested in with this model are the expression of the position of the agents and their speeds, as stated in section 4.1, to establish that no collision can occur for instance. We instantiated the Environment pattern given in figure 7, by introducing two variables xpos and speed, as shown in figure 9.

The (speed) of agents is represented by an array associating a natural number between 0 and MAX_SPEED to each vehicles. Likewise the (xpos) of agents is represented by an array associating a natural number to the vehicles. The vehicles have an initial speed of 0. They are positioned so that there is a distance of IDEAL_DISTANCE between each other: this is the purpose of the positioner constant appearing in the initialisation.

The PhysicalVehicles model also implements the operations perception and reaction, as shown in figure 9. Perceptions evolve according the equation 1. The evolution of the environment is calculated by the reaction method:

- if the vehicle is supposed to travel over the maximum possible speed, MAX_ SPEED, then it is forcibly floored to this maximal speed. Its new position is updated accordingly by using the new_xpos_when_max_speed function (equation 3.1);
- if the vehicle is supposed to travel backwards, the model states that the vehicle should stop, hence it can not go backwards. The position xpos is updated accordingly with the new_xpos_when_neg_speed function (equation 3.2);
- in any other case, i.e. the vehicle will travel at a reasonable pace, the speed of the vehicle is updated with the new speed and its new position is calculated with the new_xpos_others function suited for the general case (equation 3.3).

We have also instantiated the Agents pattern given in figure 8 into the VehiclesControllers model containing the decision mechanisms. Figure 10 shows the local variables representing the perceptions (perceived_speed, perceived_distance and perceived_front_speed) and one influence (accel_decision), as already stated in section 4.1.

As established in section 3.5, only two methods have to be explicitly written: behave and infl. The others, perceive and react simply call their respective counterparts in the PhysicalVehicles model. As the example of platooning is a completely reactive MAS example, without any internal behavioural part in the agents, the method behave can be bypassed.

The infl method, shown in figure 10, produces only the accel_decision influence. This desired acceleration for an agent depends on whether it is the leader or not. It is computed by implementing the equation 2.

4.3 Soundness of the model, properties

As explained in section 3.1, the B method requires to prove formulas, called *Proof Obligations* (POs), so as to ensure the consistency of model. Table 1 shows how many POs are generated for the whole platooning model.

Component	Obvious	Proofs	Automatic	Interactive
Constants	1	0	0	0
PhysicalVehicles	15	16	11	5
Platooning_abs	3	0	0	0
Platooning	373	6	6	0
Scheduler_abs	14	0	0	0
Scheduler	90	59	59	0
VehiclesControllers	118	22	19	3
TOTAL	614	103	95	8

Table 1: Proof results for the B model of platooning

The *Obvious* column contains the figures for proofs that are immediately true, such as tautologies. The *Proofs* columns corresponds to more difficult formulas. Among these proofs, some can be automatically proven by the tool we used [11], some have to be proved interactively by giving hints to the prover as shown by the relevant columns of table 1.

An interactive proof means that the automatic part of the prover did not go far enough into the proof tree. As a consequence, most interactive proofs will be trivial. Actually most (but not all) proofs are done easily with pen and paper. A proof tool gives the means to ensure that no problem is overlooked.

As for our platooning model, the POs that had to be interactively proven correspond to the following properties:

- $min\{MAX\_ACCEL, max\{MIN\_ACCEL, new\_accel\}\}$ belongs to the MIN_ACCEL and MAX_ACCEL bounds. As stated above, this proof looks trivial but the prover did not look far enough into the proof tree. Interactively proving this property simply required hinting the prover to look further.
- The calculation of the updated position xpos in PhysicalVehicles is still a natural number, i.e. stays above 0. The formulas to be proved involve the computation of the new position w.r.t. the decided acceleration. For instance, when the resulting speed is negative, the formula is $xpos(i) - \frac{speed(i)^2}{2 \times acceleration(i)} \geq 0$ with i the considered vehicle. While seemingly trivial, one still has to hint the prover with the sign of each of the component of the formula so that the proof tool can deduce that the new calculated position is positive. Generally speaking, arithmetical statements are a hard point for theorem provers, but this tends to be less true over time.

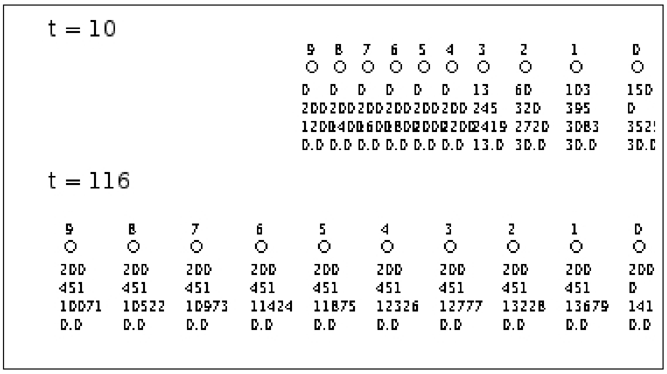
As table 1 states implicitly, the platooning model with simple typing constraints for the variables is fully proved. That means that if the B model is translated into programming code (such as C or Java) the variables will respect the bounds that have been specified for them.

Such a result was not achieved immediately though. The development process, through the generated proofs and the difficulties to prove some of them, pinpointed sometimes lacks of hypotheses about some constants of the system, such as the relationship between the bounds for the acceleration or between the acceleration and the maximal speed of the system. The next evolutions of this model are the study of collision avoidance and more complex phenomena such as oscillation in the platooning.

5 Further improvements of the framework

5.1 Simulation

Tests are an integral part of the formal approach. As an illustration, we have implemented the platooning model of section 4 into a simulator, by translating the B code into Java code required by the simulator. The code had no bug, because of the B modelling, and we were now able to explore the parameters of the physical domain, represented in the B model as constants with no explicit value.



```
t = 10
                              9   8   7   6   5   4   3    2    1     0
                              ○   ○   ○   ○   ○   ○   ○    ○    ○     ○
                              0   0   0   0   0   0   13   60   103   150
                              200200200200200200 245  320  395   0
                              120040060080000002002419 2720 3083  352
                              0.0 0.0 0.0 0.0 0.0 0.0 13.0 30.0 30.0  30.0
t = 116

 9       8       7       6       5       4       3       2       1       0
 ○       ○       ○       ○       ○       ○       ○       ○       ○       ○
 200     200     200     200     200     200     200     200     200     200
 451     451     451     451     451     451     451     451     451     0
 10071   10522   10973   11424   11875   12326   12777   13228   13679   141
 0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
```

Figure 11: Simulation of the platooning

This is illustrated by figure 11 where an instance of a platooning system with 10 agents is shown at timesteps 10 and 116. Each point represents a vehicle, the values beneath it being the speed, the distancy towards the leading vehicle, the absolute position and the acceleration in that order. We can see that at timestep $t = 116$, the vehicles have reached cruising speed while keeping an ideal distance between each other.

This simulation step can actually be included into the general development framework as it can help tuning the rules of the environment, as illustrated in figure 12. The B modelling provides means for studying the general properties of the system, the simulation provides a testbed for exhibiting good instances of the model or counterexamples.

5.2 Extracting new properties

Further evolutions of a model revolve around the extraction of its properties. To this end, one has to express explicitly into the invariant of either Environment or Agents the properties he thinks the model owns. Then, when attempting to check that these properties hold, two outcomes are possible:

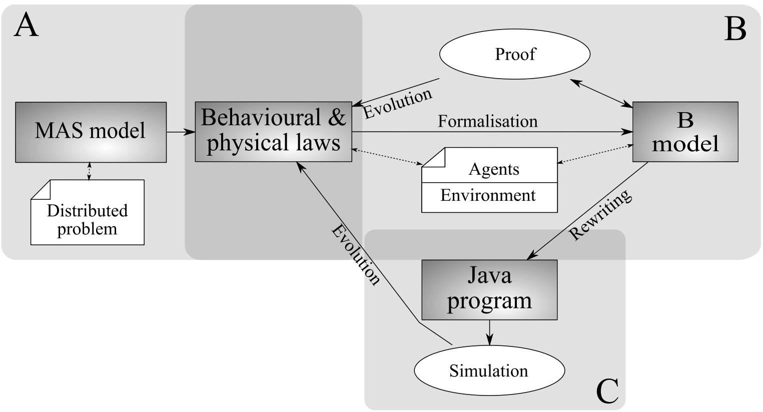


Figure 12: A general development processus

- The proof tool allows to prove, possibly with some interactive help, that the properties are indeed deduceable from the model. In that case all is well. This is what happens for instance with the saturation property mentioned in section 4.3 for the platooning model.
- The proof tool can not prove the properties. This can mean that either the property is false, or the model needs hinting, or the model lacks hypotheses.

As for any proof tool, showing that the property is false can be difficult, and in most cases, the best course action is to exhibit a counterexample. If the property does not look false, then the model must be completed. This process involves adding assertions at various points of the model, namely the preconditions of the various basic methods (perceive,. . .) and the invariants of Environment and Agents. These new assertions will finally help the prover reaching the full validation of the model with the property, or will pinpoint what hypotheses the model lacks w.r.t. the property.

6 Related work

Works on the specification and verification of MAS generally use as verification methods model-checking techniques, but they are most of the time tailored for specific types of MAS. They can be divided in two main categories focusing on logical agents (e.g. [17, 9] [21] [19]) and deliberative/communicating multi-agent organisations (e.g. [24]). However these models are not adapted to study situated MAS, due in part to the combinatorial explosion generated by their complexity. Indeed they are systems involving many (reactive) agents interacting within a dynamical environment. In this respect, theorem proving approaches abstract this explosive number of states into predicates describing their general properties, thus theorem proving helps dealing with systems having an infinite, or really big, number of states. The B method more particularly uses set theory, integer numbers and the theorems thereof to achieve this result.

We can nonetheless point out situated MAS designs based on model-checking. Hilaire et al [16] propose a general framework for modelling MAS that focuses on organisational aspects. They define OZS, a formal notation combining Object-Z and statecharts, in order to represent agents, their behaviours (using finite state automata) and their interactions. However this model does not address dynamical aspects of situated MAS, neither patterns for environment specification. Similarly, Regayeg et al [21] defined a new language based on the Z notation and linear temporal logic allowing specifications of the internal part of agents and the specification of the interaction protocol (communications) between the agents. They also based their proposed specifications on general patterns to be instantiated. The use of Z supporting tools allows them to model-check their specifications, but the proposed patterns/formalisms do not deal with dynamics of physical worlds. At last, [5] formalise situated MAS using coloured petri net. Once again this formalism is limited by the space explosion which requires some simplification of the model.

More closely to the B method, we can point out recent works [6] involving the use of Event-B (an adaptation of the B method for modelling reactive systems). This model focuses on the coordination between agents and only specifies the interaction protocol that structures the agent negotiation and decision making. Some patterns for the B specification of fault-tolerance protocols are proposed in the case of agent communication.

As far as we know, there is no formal framework expressing the whole I/R model and including verification tools. Nevertheless, we can point out a recent work [15] dealing with simulation of dynamic environments. This model employs notions of influences and reactions, but differs from the I/R model since its formalism employs a first-order representation of dynamism in the environment to express reactions. Moreover this model does not provide any tool or formal approach for verification and study of properties. However this work is an interesting reference for dealing with the implementation of the reaction function of the I/R model.

Otherwise, without considering situated MAS, there exists some models which handle the simultaneity of agent actions. For instance, Raimondi & al [20] propose an approach based on logical agents and the ISPL language allowing to express the combination of actions. A system called MCMAS (Model Checker for Multi-Agent Systems) is used to verify properties [19]. However, the ISPL language is not well suited for the description of situated MAS with complex environment.

We can mention as a concluding note a tool used in robotics to design mono-agent systems, ORCCAD [23], which is used for the specification and verification of such systems. In [1] ORCCAD is used to deal with the platooning of vehicles. However, this work focuses on the control of one vehicle and does not provide any generic pattern to deal with other applications.

7 Conclusion

We have proposed in this paper a formal pattern for the Influence/Reaction model proposed by Ferber & Muller [14]. This pattern is expressed with the B method, ensuring that a fully instantiated model fits the high quality standards of the domain of critical software. This pattern is also a generic expression of the Influence/Reaction model, allowing in particular to take into account simultaneous actions generated by agents. Indeed, any Multi-Agent System expressed through this model and defined over integer numbers, or that can be approximated with them, can be instantiated in this B pattern.

We furthermore illustrated the adequacy of our generic I/R formal model with the problem of platooning defined with a reactive MAS. We also obtained a platooning model that is sound up to typing constraints. We drew a general schema for adding and proving more general properties to an instantiated MAS and how the process unrolls when doing so.

The strength of our approach resides in the fact that we consider a formal model covering a whole system of agents. We are able to easily express the local properties of agents in the relevant part of the design pattern. We can easily match these local properties with desired global properties expressed in the environmental part of the design pattern. In other approaches, the design is focused on a single agent which is instantiated several times, and global properties are tentatively extracted, sometimes with difficulty, from the interaction of those agents. Our approach forces the developer to take into account at design time global properties, even if these are trivial. The global properties can be enriched later on without having to restart the design from the beginning.

Further evolutions of our proposition include elaborating on the expression of agents heterogeneity when considering one system. We also want to ease the specification and verification of global properties in the proposed framework. More generally, we started to develop an application that aims at automatically generating complete B patterns from agent behaviors and physical laws specified through an interface. We also intend to lift the limitation on integer numbers by defining the same kind of pattern for other formal methods supporting real numbers. Eventually, we plan to express in the platooning model more complex properties than presented, and to study other kinds of situated multi-agent systems.

References

[1] S. Abdou, M. Parent, and B. Espiau. Mission programming : Application to the distribution of empty vehicles in the praxitele project. In *4th IEEE Meditarrean Symposium on New Directions in Control and Automation (MCSA)*, Crete (GR), June 1996.

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

[3] B-Core(UK) Ltd. *B-Toolkit User's Manual, Release 3.2*, 1996.

[4] F. Badeau and A Amelot. 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 Int. Conf. of B and Z Users*, volume 3455 of *LNCS*, pages 334–354. Springer-Verlag, 2005.

[5] I. Bakam, F. Kordon, C Le Page, and F Bousquet. Formalization of a spatialized multi-agent system using coloured petri nets for the study of a hunting management system. In *FAABS 2000, Lecture Notes in Artificial Intelligence*, number 1871, pages 123–132, 2001.

[6] E. Ball and M. Butler. Event-b patterns for specifying fault-tolerance in multi-agent interaction. In *Workshop on Methods, Models and Tools for Fault Tolerance*, pages 4–13, 2007.

[7] P. Behm, P. Benoit, and J.M. Meynadier. METEOR: A Successful Application of B in a Large Project. In *Integrated Formal Methods, IFM'99*, volume 1708 of *LNCS*, pages 369–387. Springer Verlag, 1999.

[8] D. Bert, S. Boulmé, M.-L. Potet, A. Requet, and L. Voisin. Adaptable Translator of B Specifications to Embedded C Programs. In *Integrated Formal Method, IFM'03*, volume 2805 of *LNCS*, pages 94–113. Springer Verlag, 2003.

[9] F. Brazier, C. M. Jonker, and J Treur. Principles of component-based design of intelligent agents. *Data Knowledge Engineering*, 41(1):1–27, 2002.

[10] CACoLAC System - Automatic Gap Filler Control Device. http://www.fersil.fr/html/projets_cacolac_en.html.

[11] Clearsy. B4free. website, 2004. http://www.b4free.com.

[12] 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.

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

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

[15] A. Helleboogh, G. Vizzari, A. Uhrmacher, and F. Michel. Modeling dynamic environments in multi-agent simulation. *Autonomous Agents and Multi-Agent Systems*, 14(1):1–27, 2007.

[16] 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.

[17] C. M. Jonker and J Treur. Compositional verification of multi-agent systems: A formal analysis of pro-activeness and reactiveness. *Lecture Notes in Computer Science*, 1536:350–380, 1998.

[18] F. Patin, G. Pouzancre, and Servat T. A formal approach in the implementation of a safety system for automatic control of platform doors. In *4th AFIS Conference on System Engineering*, 2006.

[19] F. Raimondi and A. Lomuscio. Mcmas: a tool for verifying multi-agent systems. In *12th int. conf. on tools and algorithms for the construction and analysis of system (TACAS'06)*, volume 3920 of *LNCS*. Springer Verlag, 2006.

[20] F. Raimondi, C Pecheur, and A. Lomuscio. Applications of model checking for multi-agent systems: verification of diagnosability and recoverability. In *Concurrencey Specification and Programming (CSP 2005)*, 2005.

[21] 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.

[22] Steria – Technologies de l'information. *Proof Obligations: Reference Manual, version 3.0*, 1998.

[23] The ORCCAD Team. The ORCCAD architecture. *International Journal of Robotics Research*, 17(4):338–359, April 1998. Special issue on Integrated Architectures for Robot Control and Programming.

[24] M. Wooldridge, N. R. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. In *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 69–76. Seattle, WA, USA, 1999.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399