Engineering Applications of Artificial Intelligence 26 (2013) 211–226

Contents lists available at SciVerse ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Formalizing the construction of populations in multi-agent simulations

Benoit Lacroix [a,b,*], Philippe Mathieu [b], Andras Kemeny [a]

[a] Renault, Technical Center for Simulation, 1 avenue du Golf, 78288 Guyancourt Cedex, France
[b] University of Lille, Computer Science Department, LIFL, 59655 Villeneuve d'Ascq Cedex, France

ARTICLE INFO

Article history:
Received 12 September 2011
Received in revised form
17 February 2012
Accepted 15 May 2012
Available online 15 June 2012

Keywords:
Agent
Agent-based simulation
Behavior
Variety
Consistency
Road traffic

ABSTRACT

In individual-centered simulations, the variety and consistency of agents' behaviors reinforce the realism and validity of the simulation. Variety increases the diversity of behaviors that users meet during the simulation. Consistency ensures that these behaviors improve the users' feeling of immersion. In this work, we address the issue of the simultaneous influence of these two elements. We propose a formalization of the construction of populations for agent-based simulations, which provides the basis for a generic and non-intrusive tool allowing an out-of-the-agent design. First, the model uses behavioral patterns to describe standards of behaviors for the agents. They provide a behavioral archetype during agents' creation, and are also a compliance reference, that allows to detect deviant behaviors and address them. Then, a specific process instantiates the agents by using the specification provided by the patterns. Finally, inference enables to automate behavioral patterns configuration from real or simulated data. This formalization allows for the easy introduction of variety in agents' behaviors, while controlling the conformity to specifications. We applied the model to traffic simulation, in order to introduce driving styles specified using behavioral patterns (e.g. cautious or aggressive drivers). The behavioral realism of the traffic was therefore improved, and the experimentations we conducted show how the model contributes to increase the variety and the representativeness of the behaviors.

1. Introduction

Variety and consistency are two keys that foster the realism in individual-centered simulations (Maim et al., 2009; Wright et al., 2002). Indeed, repetitive or inconsistent behaviors highly disturb users' immersion. In this work, we propose a solution that enables to improve the agents' behavioral realism during multi-agent simulations. For this purpose, we adopt an approach that explicitly takes into account the variety and consistency of the behaviors.

In this paper, we propose a behavioral differentiation model, which provides the basis for a generic and non-intrusive tool, allowing an out-of-the-agent design. It describes the agents' behavior using behavioral patterns, which describe standards of behaviors, external to the simulation model, that are used to create agents of specific categories. At runtime, the behaviors conformity is checked using the specifications provided by the patterns. The agents are created using a specific process that allows to introduce behavioral irregularities. This process is inspired by an approach proposed by Reynolds (1999) for fuzzy path following. We have generalized and extended it to generate values in a constrained space, and to control the randomness of the process. Finally, the behavioral patterns can be inferred from simulations records or real-world situations. Based on an unsupervised learning technique, the self-organizing maps proposed by Kohonen (1995), inference enables to automatize the model configuration.

The main contribution of this work is a formalization of the construction of populations for agent-based simulations, both ab initio and as generalization from sample data. Moreover, it describes a set of tools that not only enhances the immersion of the final user during the simulation – thanks to the improvement of the realism – but also assists the simulation designer in building scenarios in a more automated way.

We applied this model to traffic simulation in SCANER^{TM}, the application developed and used for driving simulation by the French car manufacturer Renault (Reymond and Kemeny, 2000; Oktal, 2012). The software modules we developed introduce various driving styles in the traffic (e.g. aggressive, cautious drivers). These driving styles are specified using behavioral patterns. They allow for the easy population of the database in an automated way, and the update of pre-existing scenarios. These developments have already been included in the commercial version of the software. Beyond the subjective improvement in the

* Corresponding author at: University of Lille, Computer Science Department, LIFL, 59655 Villeneuve d'Ascq Cedex, France.
E-mail addresses: lacroix.benoit@gmail.com (B. Lacroix), philippe.mathieu@univ-lille1.fr (P. Mathieu), andras.kemeny@renault.com (A. Kemeny).

0952-1976/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.engappai.2012.05.011

traffic realism, our experimentations show how the proposed tool increases the variety and representativeness of the behaviors.

This paper is organized as follows. In Section 2, we present related works. Sections 3 and 4 present the grounds for this work, and an overview of the proposed approach. We then describe our model: the data model (Section 5), the generation method and algorithms (Section 7), and the inference techniques used to automate the simulation configuration (Section 8). Section 9 shows how the model addresses our issue: increasing variety in simulation, while insuring the behaviors consistency. Section 10 presents the application of the model to traffic simulation in driving simulators, and how it improves the realism in an industrial application. Section 11 introduces another example, to illustrate the genericity of the approach. Finally, the model is discussed in Section 12, and future works are presented in Section 13.

2. Related works

Our work focuses on how we can articulate both behavioral variety and consistency in simulations. These two notions are considered as central issues in various works.

For instance, in crowd simulations, the realism is crucial to improve the users' immersion during the simulation, and to improve the results validity. The realism is directly related to the variety of the agents' looks. Techniques increasing this diversity have thus been developed: for instance, Maim et al. (2009) automatically change virtual humans' appearance by using diverse colors, accessories and shape parameters. Variety in the simulation is increased, which improves users' immersion. However, this approach is based on hard coded parameters in the graphics models, which limits its genericity and flexibility.

In videogames, more and more attention is focused on the improvement of the gameplay. Moreover, in that domain, the non-player agents' behaviors are often implemented as scripts, defined by long and complex list of rules (Tozour, 2002). However, such scripts can contain weaknesses that decrease the player's immersion in the virtual environment. Focusing in particular on the variety and consistency of the behaviors, Spronk et al. (2006) have proposed to dynamically improve agents' behaviors using online learning. The agents' behaviors being defined by sets of rules associated to activation weights, those weights are then dynamically adapted during the game to improve the behaviors. However, during the agent's creation, these weights are randomly selected: the approach does not specifically address this initial instantiation. It could therefore benefit from an automated mechanism introducing behavioral profiles at this level.

In the computer graphics and virtual reality field, various approaches focus on easing the configuration of the virtual environments. For instance, Ulicny et al. (2004) have proposed dedicated tools for the designers, based on a *painting* metaphor. Using a *brush*, the designer can *paint* new pedestrians in a simulation, or visual and behavioral characteristics on existing ones. This approach enables to easily create agents and to increase the behavioral variety, but remains focused on the graphical part of the simulation. In other works, predefined behavioral patterns are proposed to the user (Pellens et al., 2009). After having selected a pattern, the user has to adapt the pattern parameters to the values he desires. However, this approach does not take into account the possibility to automate the construction of whole populations.

Finally, in driving simulation, Wright et al. (2002) created virtual drivers' characters to reinforce the users' immersion during the simulation. The model combines the drivers' speed choices with behavioral parameters (sex, age, aggressiveness, alcoholic intoxication and fatigue). When implemented in a driving simulator, the model improves the realism as perceived by the users. However, this approach lacks flexibility: the modifications have to be integrated within the core of the traffic model. Such changes are not easy to introduce in commercial softwares, due to back-compatibility issues or even because no modification is allowed on these core components.

3. Motivation of the work

Our initial objective was to enhance the simulation realism. In this section, we discuss how this objective led us to a second one, which is to help the designer elaborate simulation scenarios. Then, we describe the constraints introduced by the requirement to integrate the proposed approach in a commercial software, and the design choices needed to meet these requirements.

3.1. Objectives

The main objective of this work was to improve the realism of the agents' behavior in multi-agent simulations. To do so, we focused on the variety and consistency of the behaviors. Indeed, simultaneously taking both these dimensions into account is important for the users' immersion into a simulation. In traffic simulation, the observation of different kind of behaviors (e.g. aggressive or cautious drivers) contributes to the realism feeling (Wright et al., 2002). However, an aggressive driver is characterized not only by a high speed, but also by short security distances and a tendency to disregard speed limits. Consistency among all these properties – and not only a subset of them – characterizes the "aggressive driver" category. To handle these dependencies, we, therefore, proposed to explicitly take both variety and consistency into account.

Furthermore, in most commercial simulation tools, a scenario is designed specifically for one experiment. A designer is in charge of creating and implementing these scenarios. Since the design of the scenario highly influences the simulation realism, he plays a crucial role in the simulation outcomes. For instance, in SCANER^{TM}, the simulation tool used at Renault for driving simulation, each vehicle had to be manually added in the scenario, and each vehicle parameter had to be manually modified: reproducing the variety usually met in a real traffic was thus a long and repetitive task. Moreover, if the parameter values were not carefully chosen, unrealistic behaviors could appear during the simulation. The consequences were that scenarios were usually created with a small number of vehicles, and that these vehicles kept their (similar) default parameters.

To successfully increase the realism during a simulation, we had thus not only to improve the agents' behaviors themselves, but also to help the scenario designer to introduce these realistic behaviors into the simulation.

3.2. Constraints

Another dimension we had to take into account when designing the model is that it had to be integrated into a commercial software. This introduced two specific constraints: first, the proposed model had to provide a running mode where non-regression with the previous version would be insured; second, the proposed model had to be totally non-intrusive, as no modification of the original source code of the simulation software was allowed by the editor.

To guaranty non-regression, we proposed to use a two-level control mechanism in the agents' creation process. The first level provides an easy way to switch off the improvements introduced by the model, and to return to the previous simulation running

mode. The other one allows to finely configure the properties of the creation process.

To guaranty non-intrusiveness, we proposed a model that can use only external parameters of the simulation, provided for instance through an API, and run in a specific process outside the simulation software.

These objectives – increase the realism for the final user while considering the designer crucial role – and constraints – guaranty non-regression and provide a non-intrusive model – drove the design of the proposed model.

4. Overview of the approach

To address these needs, we propose a formalization of the construction of populations for multi-agent simulations.

4.1. Presentation of the approach

This formalization is structured around the notion of *behavioral pattern*, which describes both a cluster of similar behaviors, and a standard of behavior. For instance, in traffic simulation, it might be aggressive or cautious drivers "categories": aggressive drivers will have similar, but not identical, behaviors. This standard of behavior might also be a description they do not share: a driver can behave aggressively while considering himself normal, or even cautious.

Moreover, in our case, the non-intrusiveness constraint means that the simulation and the proposed model have to be dissociated. The simulation software, including the agents' decision model, cannot be modified, and has to be used as provided by the simulator editor. Therefore, we cannot suppose that the decision model of the agents in the simulation handles the notion of *behavioral pattern*. This has one major implication, which is also one of the limits of the approach: the agents cannot reason on their behavior. This means that the agents cannot explicitly choose to change their *pattern*. An agent only changes it during the course of its actions, depending on its decision model, and without knowing it.

Finally, we chose to base our approach on a parameter description of the behavior, as this representation is common in simulations and can therefore be used ad hoc. If more complex representation are used, they usually can be reduced to a parameter based description. For instance, in the software used at Renault, drivers' behaviors are described through parameters like their maximal speed, their percentage of respect of the speed limits, or their security distance with other vehicles. In this paper, we therefore restrict our approach to simulations where the behaviors can be expressed through a set of parameter values.

Note that if the non-intrusiveness constraint does not have to be respected – for instance if the source code of the simulation can be freely modified – the simulation agents could be made able to reason about their own behaviors. To do so, the simulation model has to be altered, to allow the agents to own a representation of their current *behavioral pattern*. Then, in specific situations, an agent can choose to change its *pattern*, temporarily or permanently. This change is achieved through a request for a new set of parameter values from the behavioral differentiation model that will replace its current values.

4.2. Global presentation of the model

In our approach, behavioral patterns are used to provide a description of agents' behaviors, based on their configuration parameters (Lacroix et al., 2008b), and to check the conformity at runtime.

The model's mechanism can be described as follows (Fig. 1). During the conception of a simulation scenario, the designer specifies *Behavioral Patterns* and *Parameters*, based on his knowledge of the behavioral parameters used by the simulation agents. *Model Agents* encapsulate these elements, and represent a transition structure between *Behavioral Patterns* and simulation agents. Hence, model and simulation can be implemented in distinct computing processes, which allow a non-intrusive use of the model. Finally, at runtime, the *Model Agents* are used to control the simulation agents' parameter values, and assess behavioral irregularities. Such irregularities can be authorized, limited, or forbidden.

Variety is introduced through the various aspects of the patterns definition, as well as irregular behaviors. Consistency and conformity are ensured by the possibility to detect irregularities and control the respect of the specification at runtime.

Example 1. We introduce here the example used in the following sections. The context is a crowd simulation populated with virtual humans, for which we suppose we have no access to the simulation decision model. Each pedestrian of the crowd uses three behavioral parameters:

- its maximal speed $v_{\max} \in [0, 15]$ km/h,
- its desired speed $v_d \in [0, v_{\max}]$,
- and its personal space $s \in \{big, normal, small\}$, representing the private area each individual aims at preserving around itself.

5. Presentation of the data model

In this section, we present the definitions of the proposed model. We consider a multi-agent system with a set of agents $\mathcal{A}$ defined in an environment $\mathcal{E}$. The agents $a_i \in \mathcal{A}$ are characterized by various parameters $\mathcal{P}^{a_i}$.

In the following, we consider the use of the model only with external parameters of the simulation, i.e. the parameters accessible from outside the simulation, by opposition to internal parameters, which are accessible only from inside the simulation. Indeed, the proposed model is not dependent on the simulation engine when it applies to external parameter.

It is to be noted that the model can only be used on a subset of agents using the same parameters. For instance, if the simulation includes pedestrians and vehicles, the model has to be applied independently to each agent's type.

5.1. Parameters

We first introduce the notion of *Parameter*. In the model, a *Parameter* is designed to match one of the agents' behavioral parameters in the simulation. The behavioral parameters of the

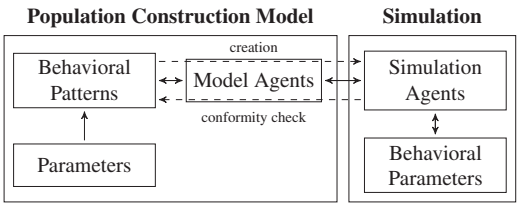

Fig. 1. The designer specifies *Behavioral Patterns*, which are used to instantiate the *Model Agents*. These *Model Agents* enable to assign parameter values to simulation agents and control the conformity to the *Behavioral Pattern*. They allow to fully separate model and simulation, enabling their implementation in distinct processes.

simulation may have several corresponding *Parameters* in the model, to answer different needs.

A *Parameter* is a tuple including different elements. A *Parameter* holds a reference on another *Parameter* p_{ref}, called the "reference parameter". p_{ref} is either another *Parameter* or null. Moreover, if $p_{\text{ref}}(p) = null$, then p is called a "root parameter".

A *Parameter* includes two elements relative to its definition space. First, the definition domain $\mathcal{D}_p$ defines the *Parameter* type and possible values. A *Parameter* might be a real number, an integer, a string, or even an action rule. The domain can be continuous, discrete, etc. It is to be noted that in the applications presented, the *Parameters* are mostly real numbers defined over discrete or continuous domains. The second element is the default value $v_{d_p} \in \mathcal{D}_p$. By default, with real numbers, on a discrete or discontinuous domain this value is the closest number of the domain mean value; on continuous domains it is the mean value.

The two last elements of a *Parameter* are used for agent creation and conformity check. The probability distribution g_p is defined on the distribution domain $\mathcal{D}_p$. This distribution is used to generate values for the *Parameter*, using an algorithm described in Section 7. It contributes to the creation of variety among agents' behaviors. By default, g_p is the uniform distribution over $\mathcal{D}_p$. Finally, the distance function f_p allows to compute the gap between a *Parameter* value and its definition domain. This function is used to quantify the irregularities, as described in Section 6. By default, on a discrete or discontinuous domain, this function returns 0 over the domain, and 1 outside. On a continuous domain, the default function is

$$\forall x \in \mathcal{D}_{p_{\text{ref}}}, \quad f_p(x) = \begin{cases} 0 & \text{if } x \in \mathcal{D}_p \\ \dfrac{|x - mean(\mathcal{D}_p)|}{\max(\mathcal{D}_{p_{\text{ref}}}) - \min(\mathcal{D}_{p_{\text{ref}}})} & \text{else} \end{cases}$$

Without loss of generality (the bounds may be chosen as wide as necessary), the definition domains have to be finite, to allow the max and min function to be defined.

Finally, we note $\mathcal{P}$ the set of all the *Parameters* defined in the model.

Definition 1. A *Parameter* p is a tuple $(l_p, p_{\text{ref}}, \mathcal{D}_p, v_{d_p}, g_p, f_p)$ defined by

- l_p a unique label,
- $p_{\text{ref}}(p)$ a reference parameter,
- $\mathcal{D}_p$ a finite definition domain, with if $p_{\text{ref}} \neq null$, then $\mathcal{D}_p \subseteq \mathcal{D}_{p_{\text{ref}}}$,
- $v_{d_p} \in \mathcal{D}_p$ a default value,
- g_p a probability distribution over $\mathcal{D}_p$,
- $f_p : \mathcal{D}_{p_{\text{ref}}} \mapsto [0, 1]$ a distance function, with if $p_{\text{ref}}(p) = null$, then $f_p : \mathcal{D}_p \mapsto [0, 1]$.

Example 2. Based on Example 1, we define two "root parameters", the maximal speed v_{max} and the personal space s, as well as two other *Parameters* based on these "root parameters", the normal maximal speed v_{normal} and the normal personal space s_{normal}

- the maximal speed v_{max} is defined by $p_{\text{ref}} = null$, $\mathcal{D}_p = [0, 15]$ km/h, $v_{d_p} = 5$ km/h, g_p and f_p the default functions,
- the personal space s is defined by $p_{\text{ref}} = null$, $\mathcal{D}_p = \{big, normal, small\}$, $v_{d_p} = normal$, g_p and f_p the default functions,
- the normal maximal speed v_{normal} is defined by $p_{\text{ref}} = v_{max}$, $\mathcal{D}_p = [4, 6]$ km/h, $v_{d_p} = 5$ km/h, g_p the normal distribution described by a mean value $\mu = v_{d_p}$ and variance $\sigma^2 = 1$ truncated at $\mathcal{D}_p$ bounds, and f_p the default distance function,
- the normal personal space s_{normal} is defined by $p_{\text{ref}} = s$, $\mathcal{D}_p = \{normal\}$, $v_{d_p} = normal$, g_p and f_p the default functions.

We have $\mathcal{P} = \{v_{max}, s, v_{max,normal}, s_{normal}\}$. Fig. 2 represents v_{max} and v_{normal}, as well as three other *Parameters* that could be created, to illustrate the variety of possible configurations.

5.2. Behavioral patterns

We then define the *Root Behavioral Pattern*. The *Root Behavioral Pattern* is used to list all the "root parameters" defined in a simulation. As will be seen in Section 9, it is used to ensure agents' conformity with their specification.

All *Parameters* in the *Root Behavioral Pattern* are "root parameters", i.e. their "reference parameter" is null, and all "root parameters" belong to the *Root Behavioral Pattern*.

Definition 2. A *Root Behavioral Pattern* B_{root} is a set $\{\mathcal{P}_{B_{\text{root}}}\}$, with $\mathcal{P}_{B_{\text{root}}}$ a finite set of parameters p, which verifies

$$\forall p \in \mathcal{P}, \quad (p_{\text{ref}}(p) = null) \Leftrightarrow (p \in \mathcal{P}_{B_{\text{root}}})$$

By definition, B_{root} is unique.

Example 3. In our example, B_{root} is defined by $\mathcal{P}_{B_{\text{root}}} = \{v_{max}, s\}$.

We then introduce the notion of *Behavioral Pattern*. *Behavioral Patterns* are used to specify the agents' behaviors. During agents creation, they provide a behavioral archetype for the agents. Then, at runtime, they are used to check the conformity with the specifications.

A *Behavioral Pattern* is a set including different elements. First, it holds a "reference pattern" $B_{\text{ref}}(B)$, different from null. The induced hierarchy is used to introduce common characteristics for sets of *Behavioral Patterns*. A *Behavioral Pattern* also holds a finite set $\mathcal{P}_B$ of *Parameters* p. Each of these *Parameters* must have a not null "reference parameter", i.e. no "root parameter" can be included in a *Behavioral Pattern*. Moreover, each *Parameter* p must have a "parent" *Parameter* in the "reference pattern" $B_{\text{ref}}(B)$ (i.e. $p_{ref}(p) \in \mathcal{P}_{B_{ref}(B)}$), and each *Parameter* of the "reference pattern" has to be included at most once. A *Behavioral Pattern* also holds a set of properties $\mathcal{Q}_B$. These properties allow to reference an application context, like a geographic area of validity.

Finally, two elements relative to deviations from the *Behavioral Pattern* are included. The first one is the deviation rate τ_B, used during the agents' creation. The deviation rate is a probability that describes if, for an agent instantiated from this *pattern*, the agent

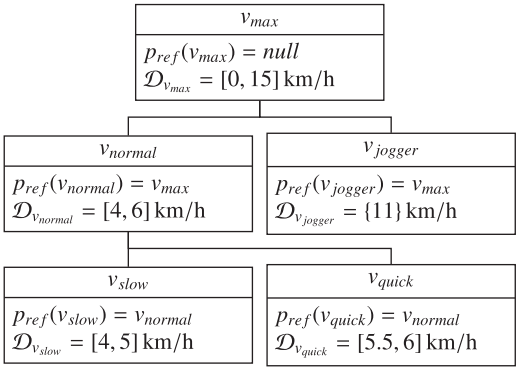

Fig. 2. Each parameter definition domain is a subset of the definition domain of its reference parameter, which induces a *Parameters* hierarchy. v_{normal} and v_{jogger} have v_{max} as reference parameter, but v_{jogger} is very specific: its domain is a singleton. As for v_{slow} and v_{quick}, they use v_{normal} as reference parameter, and define more specific domains.

will be created as deviant. The second one is the maximal gap to the pattern $\delta_{\max_B}$, which specifies the tolerance toward the amplitude of the behavioral irregularities. These two elements allow the user to have a close control on the irregularities, both in proportion and in amplitude: a certain percentage τ_B of the population instantiated from a *Behavioral Pattern* can be created in deviation from the *pattern* in a certain limit specified by $\delta_{\max_B}$.

A *Behavioral Pattern* thus defines a set of *Parameters* ($\mathcal{P}_B$) in a specific context ($\mathcal{Q}_B$), and provides regulative elements relative to potential irregularities (τ_B and $\delta_{\max_B}$).

Definition 3. A *Behavioral Pattern* B is a set $\{l_B, B_{\text{ref}}, \mathcal{P}_B, \mathcal{Q}_B, \tau_B, \delta_{\max_B}\}$ with

- l_B a unique label,
- $B_{\text{ref}}(B)$ a reference pattern, with $B_{\text{ref}}(B) \neq null$,
- $\mathcal{P}_B$ a finite set of parameters p, which verifies
$$\begin{cases} \forall p \in \mathcal{P}_B, & p_{\text{ref}}(p) \neq null \\ \forall p_1 \in \mathcal{P}_B, & \exists p_2 \in \mathcal{P}_{B_{\text{ref}}}, p_{\text{ref}}(p_1) = p_2 \\ \forall p_1 \in \mathcal{P}_B, & \forall p_2 \in \mathcal{P}_B - \{p_1\},\ p_{\text{ref}}(p_1) \neq p_{\text{ref}}(p_2) \end{cases}$$
- $\mathcal{Q}_B$ a set of properties,
- τ_B a deviation rate, with $\tau_B \in [0,1]$,
- $\delta_{\max_B}$ a maximal gap to the pattern, with $\delta_{\max_B} \in [0,1]$.

Example 4. Our first objective is to create pedestrians having a normal behavior in the simulation. However, we would like to obtain some agents adopting a slightly different behavior, to increase the variety and thus the simulation realism. To answer that need, we only have to define the *Behavioral Pattern* B_{normal} using the following elements: $B_{\text{ref}} = B_{\text{root}}$, $\mathcal{P}_B = \{v_{max,normal}, s_{normal}\}$, $\mathcal{Q}_B = \emptyset$, $\tau_{B_{\text{normal}}} = 0.05$, and $\delta_{\max_{B_{\text{normal}}}} = 0.1$. This way, most of the agents instantiated from this pattern will behave normally, and there is a 5% chance that they will adopt a deviant behavior at their creation. The maximal gap to the pattern can reach 10%. Fig. 3 represents B_{root} and B_{normal} as well as two other patterns that could be defined similarly.

The *Behavioral Patterns* present two interesting properties. The first one is that for each *Parameter* of a *Behavioral Pattern*, the definition domain of this *Parameter* is necessarily included in the domain of the corresponding *Parameter* in the "reference pattern". The definition domains are thus more and more restricted when we go down the *Behavioral Patterns* hierarchy.

Property 1. $\forall B \neq B_{\text{root}}, \forall p \in \mathcal{P}_B, p_{\text{ref}}(p) \in \mathcal{P}_{B_{\text{ref}}(B)}$ *and* $\mathcal{D}_p \subseteq \mathcal{D}_{p_{\text{ref}}(p)}$.

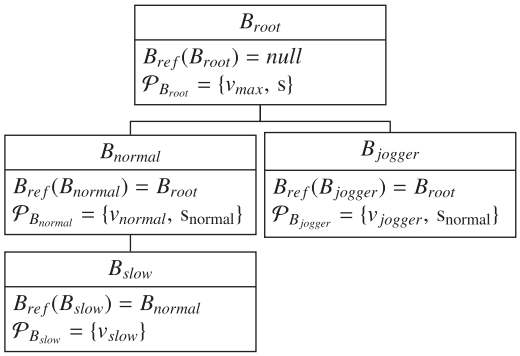

Fig. 3. The *Root Behavioral Pattern* only includes *Root Parameters*. A *Behavioral Pattern* includes only *Parameters* which reference parameter is included in its reference pattern. However, all *Parameters* do not have to be included: B_{slow} only includes the maximal speed (v_{slow}), and not the personal space.

Proof. Let $B_1 \neq B_{\text{root}}$ and $p_1 \in \mathcal{P}_{B_1}$. By Definition 3, $\exists p_2 \in \mathcal{P}_{B_{\text{ref}}(B_1)}$ such that $p_{\text{ref}}(p_1) = p_2$. Moreover, $p_1 \in \mathcal{P}_{B_1}$, thus $p_{\text{ref}}(p_1) \neq 0$. Therefore $p_2 \neq 0$. By Definition 1, we have $(p_2 \neq 0) \Rightarrow (\mathcal{D}_{p_1} \subseteq \mathcal{D}_{p_2})$. □

A second property is that no *Behavioral Pattern* can hold a *Parameter* having a definition domain wider than the "reference parameter" in B_{root}. B_{root} defines the whole set of *Parameters* which might be used in the *Behavioral Patterns*, as well as the wider definition domains that can be used.

Property 2. $\forall B \neq B_{\text{root}}, \forall p \in \mathcal{P}_B, \exists! p_r \in \mathcal{P}_{B_{\text{root}}}, \mathcal{D}_p \subseteq \mathcal{D}_{p_r}$.

Proof. Let $B_1 \neq B_{\text{root}}$ and $p_1 \in \mathcal{P}_{B_1}$. With Definition 3, $\exists p_2 \in \mathcal{P}_{B_{\text{ref}}(B_1)}$ such that $p_{\text{ref}}(p_1) = p_2$. Moreover, p_2 is unique: with Definition 1 each parameter holds one and only one reference parameter. If $B_{\text{ref}}(B_1) = B_{\text{root}}$, p_2 suits our need and we choose $p_r = p_2$. If not, we try again the process until we reach B_{root}. The hierarchy being finite, and each behavioral pattern holding one and only one reference pattern, the process is finite and has a solution. □

5.3. Model agents

We then introduce the notion of *Model Agent*. A *Model Agent* is a middleman between a *Behavioral Pattern* and a simulation agent: a *Model Agent* encapsulates the behavioral parameters of a simulation agent, and associates a *Behavioral Pattern* to these parameters. It allows to implement the model and the simulation in different computing processes.

A *Model Agent* a_m holds a reference *Behavioral Pattern* B_{a_m} and a set of pairs of *Parameters* and associated values (p, v_p). The *Parameters* come from the reference pattern B_{a_m}, and all the *Parameters* of this pattern are included. For each *Parameter*, the value v_p has to be inside the domain $\mathcal{D}_{p_{\text{ref}}}$, but not necessarily inside $\mathcal{D}_p$.

A *Model Agent* is the instantiation of a *Behavioral Pattern*: it includes all its *Parameters*, each of one associated to a value. The instantiation process is described in Section 7.

Definition 4. A *Model Agent* a_m is a set $\{l_{a_m}, B_{a_m}, \mathcal{C}_{a_m}\}$ with

- l_{a_m} a unique label,
- B_{a_m} a reference behavioral pattern,
- $\mathcal{C}_{a_m} = \{(p, v_p),\ p \in \mathcal{P}_{B_{a_m}}\}$ a set of pairs of parameters labels and associated values, which verifies
$$\begin{cases} \forall p \in \mathcal{P}_{B_{a_m}}, & \exists c \in \mathcal{C}_{a_m},\ c = (p, v_p) \\ \forall c = (p, v_p) \in \mathcal{C}_{a_m}, & \exists p_1 \in \mathcal{P}_{B_{a_m}},\ p_1 = p \\ \forall c = (p, v_p) \in \mathcal{C}_{a_m}, & v_p \in \mathcal{D}_{p_{\text{ref}}}(p) \end{cases}$$

The conditions on the set $\mathcal{C}_{a_m}$ enable to ensure that each *Parameter* of the *Model Agent*'s reference pattern B_{a_m} is included in the *Model Agent*; all *Parameters* included in the *Model Agent* belong to the reference pattern B_{a_m}; and all the values included in the *Model Agent* belong to the definition domain of the "reference parameter" of the *Parameter* included in the *Model Agent*'s reference pattern.

Example 5. A *Model Agent* a_{m_1} instantiated from the behavioral pattern B_{normal} might have the following characteristics: $B_{a_{m_1}} = B_{\text{normal}}$ and $\mathcal{C}_{a_{m_1}} = \{(v_{max,normal}, 5.2), (s_{normal}, normal)\}$. The *Model Agent* associates a value to each of the parameters included in its reference pattern.

5.4. Settings

Finally, a *Setting* is used to define the context of a simulation, i.e. the available *Behavioral Patterns* and global model parameters.

The *Setting* includes all or part of the *Behavioral Patterns* defined in the environment. A real value σ_S specifies the global determinism of the simulation. This value is used during the

generation of the *Model Agents*, according to the process presented in Section 7.

Definition 5. An *Setting S* is a set $\{l_S, \mathcal{B}_S, \sigma_S\}$ with

- l_S a unique label,
- $\mathcal{B}_S$ a finite set of behavioral patterns,
- σ_S the global determinism criteria of the simulation, with $\sigma_S \in [0,1]$.

Example 6. The *Setting* S_{town}, defined by $\{\{B_{\text{normal}}\}, 0.01\}$ includes the pattern "normal pedestrian", with a 1 % global determinism criteria.

6. Behavioral irregularities

After having defined the semantics of our approach, we now consider the question of the behavioral irregularities. In our approach, irregularities from the behavioral patterns are used to increase the behaviors variety. Moreover, their quantification enables to control the conformity to the specification.

6.1. Definition

Irregularities are defined as follows: a *Model Agent* presents a behavioral irregularity if and only if at least one of the values associated to one of its *Parameters* does not belong to this *Parameter* domain in the *Behavioral Pattern*.

Definition 6. A *Model Agent* a_m presents a behavioral irregularity from its reference behavioral pattern B_{a_m} if and only if $\exists c = (p, v_p) \in \mathcal{C}_{a_m}$ such that $v_p \notin \mathcal{D}_p$.

Example 7. Let a_{m_2} be a *Model Agent* with $B_{a_{m_2}} = B_{\text{normal}}$ and $\mathcal{C}_{a_{m_2}} = \{(v_{max,normal}, 7), (s_{normal}, normal)\}$. The reference pattern specifies $\mathcal{D}_{v_{max,normal}} = [4,6]$ km/h. We have $7 \notin \mathcal{D}_{v_{max,normal}} : (a_{m_2}$ deviates from its reference behavioral pattern.

An agent deviates occasionally from its reference pattern, when its parameter values cross the bounds of the definition domains during the simulation. An agent can also present a behavioral irregularity at its creation, before eventually going back to a conformal state. Irregularities increase the variety of the produced behaviors, allowing unspecified behaviors to appear. However, to be able to ensure the conformity to the specifications, these irregularities have to be quantified.

Definition 7. Let q_{a_m} be defined by

$$q_{a_m} = \frac{\sum_{c \in \mathcal{C}_{a_m}} f_p(v_p)}{Card(\mathcal{P}_{N_{a_m}})}$$

q_{a_m} represents the quantification of agent's a_m deviation. The higher the number of deviant *Parameters*, and the higher the gap with the *Parameter* definition domain, the greater the q_{a_m} value is. The quantification is based on the function f_p to allow users to customize it if needed.

Example 8. With *Model Agent* a_{m_2} from Example 7

$$q_{a_{m_2}} = \frac{f_{v_{max,normal}}(v_{max,normal}(a_{m_2})) + f_{s_{normal}}(s_{normal}(a_{m_2}))}{Card(\{v_{max,normal}, s_{normal}\})} = \frac{\frac{7-5}{20-0} + 0}{2} = 5\%$$

Property 3. $\forall a_m \in \mathcal{A}_m$, $q_{a_m} \in [0,1]$.

Proof. Let $c = (p, v_p) \in \mathcal{C}_{a_m}$. Then $v_p \in \mathcal{D}_{p_{\text{ref}}}$ (Definition 4), therefore $f_p(v_p) \in [0,1]$ (Definition 1). Furthermore, $Card(\mathcal{C}_{a_m}) = Card(\mathcal{P}_{N_{a_m}})$ (Definition 4), therefore $q_{a_m} \in [0,1]$. □

When an irregularity is detected, the model can either allow or forbid it, depending on the value of the maximal gap to the pattern $\delta_{\max_{B_{a_m}}}$. At each time step of the simulation, the population construction model executes the Algorithm 1 to control agents' deviation. If $q_{a_m} \leq \delta_{max_{B_{a_m}}}$, the agent is in the authorized limits, and the new parameter values are kept. If $q_{a_m} > \delta_{max_{B_{a_m}}}$, the deviation is too high, and is therefore forbidden: the *Parameter* values previously saved are restored and copied in the simulation agent, erasing the new values. The agent therefore remains in the authorized limits.

Algorithm 1. Reaction to behavioral irregularities.

Require: $\mathcal{A}_m$ the set of *Model Agents*, a the simulation agent represented by a_m

```
for all a_m ∈ A_m do
  a_temp ← a_m {temporary copy of a_m}
  for all p ∈ P_{a_m} do
    v_p(a_m) ← v_p(a) {copy a's parameters values in a_m}
  end for
  if q_{a_m} > δ_{max_{B_{a_m}}} then {irregularities forbidden}
    a_m ← a_temp {restore a_m's previous values}
    for all p ∈ P_{a_m} do
      v_p(a) ← v_p(a_m) {force a to its previous state}
    end for
  end if
end for
```

6.2. Deviant behaviors

Different kinds of *Model Agents* may be produced, depending if their behavior is defined by static parameters only, or also by dynamic ones. A dynamic parameter can be modified directly by the simulation decision model, at runtime. In contrast, a static parameter can only change value if it undergoes an external action (a user manual modification for instance). For an agent $a_i \in \mathcal{A}$, we have $\mathcal{P}_d^{a_i}$ the set of dynamic parameters, and $\mathcal{P}_s^{a_i}$ the set of static ones ($\mathcal{P}^{a_i} = \mathcal{P}_d^{a_i} \cup \mathcal{P}_s^{a_i}$).

First, a *Model Agent* can be *conformal*. Such an agent defines only static parameters, and each of those parameter takes its value in the parameters definition domain. These parameters being static, their values will never change, and the agent will always conform to the behavioral pattern.

Definition 8. A *Model Agent* a_m of reference pattern B_{a_m} is *conformal* if and only if $\forall p \in \mathcal{P}_{B_{a_m}}$, $p \in \mathcal{P}_s^{a_m}$ and $v_p(a_m) \in \mathcal{D}_p$.

Example 9. In the behavioral pattern B_{normal} defined in Example 4, we now specify $\delta_{max_{B_{normal}}} = 0$. v_{max} and s_{normal} being static, they do not change value during the simulation: each *Model Agent* instantiated from B_{normal} is *conformal*.

A *Model Agent* can also be *always deviant*. Such an agent defines at least one static parameter that was instantiated with a value outside its definition domain. The agent presents therefore a behavioral irregularity from its reference pattern. The parameter being static, it is never modified: the agent remains in deviation during the whole simulation.

Definition 9. A *Model Agent* a_m of reference pattern B_{a_m} is *always deviant* if and only if $\exists p \in \mathcal{P}_{B_{a_m}}$ such that $p \in \mathcal{P}_s^{a_m}$ and $v_p(a_m) \notin \mathcal{D}_p$.

Example 10. The agent a_{m_2} defined in Example 7 is *always deviant*: v_{normal} is a static parameter, and the value 7 does not

belong to $\mathcal{D}_{v_{normal}}$. v_{normal} does not change during the simulation, therefore a_{m_2} always remains in deviation.

Finally, a *Model Agent* can be *deviant*. A *deviant* agent defines at least one dynamic parameter. At its creation or during the simulation, this parameter may take a value outside its definition domain. If this happens, the agent presents therefore a behavioral irregularity from its reference pattern. However, the parameter being dynamic, this value may change again and return to the parameter definition domain, and the agent to a conformal state. A *deviant* agent may never reach the state of behavioral irregularity, if the specification of σ or $\delta_{max_{B_a}}$ does not allow it.

Definition 10. A *Model Agent* a_m of reference pattern B_{a_m} is *deviant* if and only if $\exists p \in \mathcal{P}_{B_{a_m}}$, $p \in \mathcal{P}_d^{a_m}$.

Example 11. Suppose that a dynamic parameter $v_{desired}$ and a behavioral pattern $B_{desired}$ including this parameter have been defined. Then any *Model Agent* instantiating $N_{desired}$ is *deviant*.

7. Parameters generation

Our approach describes agents' behaviors using behavioral patterns. However, the creation of agents' parameters requires the instantiation of a *Behavioral Pattern* into a *Model Agent*. In this section, we describe an algorithm designed to control the randomness introduced in an instantiation process, and its application to the case of behavioral patterns instantiation (Lacroix et al., 2008a).

7.1. Motivation of the algorithm design

The proposed algorithm was inspired by an approach proposed by Reynolds (1999) for path following. In that approach, an agent is enabled to steer along a predetermined path, represented for instance by a spline or a poly-line, instead of being rigidly constrained to follow it. The idea is to constrain the agent's position inside a "tube", defined as a circle of a specified radius swept along the path. To do so, the – predicted – future position of the agent is projected on the path. If the distance between the projection and the predicted position is greater than the specified radius, the agent has to steer back to the path. Otherwise, it is considered as being correctly following the path.

Similar to this approach, our objective is to generate values in a determined space, and if the generated values is predicted to be outside the domain (i.e. in deviation), we want to be able to react (usually refusing or accepting the value). Instead of constraining an agent's position in a "tube", we want to constrain a parameter value within a definition domain. However, in our case, the algorithm has to handle many different parameters, possibly of different types. Inspired by Reynolds approach, we thus propose an algorithm able to cope with any number of dimensions and any type of parameters.

Moreover, we added the possibility to control the randomness of the process. Indeed, the objective was to provide a tool that enhances the simulation realism and eases the scenario design. To answer these needs, the algorithm had to provide a precise control mechanism. This control has to be assured on the global determinism of the process, to allow reproducibility, and on the determinism of each object instantiation, to allow a very fine level of specification. We therefore introduced a two level control: the higher level controls the global determinism of the algorithm, and the lower level controls the determinism on an object-level basis.

This necessity to control irregularities is required for industrial needs. In traffic simulation for instance, you may need to allow them to broaden the range of simulation cases tested, when in exploitation of immersive simulators you have to guaranty no irregularity can ever appear. Moreover, it addresses the non-regression constraint introduced in Section 3.2.

7.2. Algorithm description

The objective is to introduce variety among agents, while producing consistent behaviors. To do so, the creation of the agents is based on their reference *Behavioral Pattern*. Each of these pattern is associated to a *Parameters* set, and these *Parameters* specify a definition domain and a probability distribution on that domain. These characteristics provide all the elements needed to create the agents. Moreover, to increase the variety, we wish to be able to allow the creation of deviant agents. $\delta_{\max_B}$ is used to control this deviation.

Algorithm 2. Generation of the parameter values.

Require: a behavioral pattern $B = \{B, B_{\text{ref}}, \mathcal{P}_B, \mathcal{Q}_B, \tau_B, \delta_{\max_B}\}$, σ the global determinism criteria.
Ensure a_m a Model Agent. B its reference pattern.

```
repeat
  α ← uniform_random([0,1]) ; k = k+1
  if α < σ then
    for all p ∈ P_B do
      β ← uniform_random([0,1])
      if β < τ_B then
        v_p(a_m) ← g_{p_ref} {potential deviation, v_p ∈ D_{p_ref}}
      else
        v_p(a_m) ← g_p {no deviation, v_p ∈ D_p}
      end if
    end for
  else
    for all p ∈ P_B do
      v_p(a_m) ← g_p {no deviation, v_p ∈ D_p}
    end for
  end if
until q_(a_m) < δ_{max_B} or k > max_try
```

The algorithm proceeds as following (Algorithm 2). First, a random number α is generated using a uniform function over $[0,1]$, and compared to the global determinism criteria σ. This provides the first (high) level of control. If $\alpha < \sigma$, irregularities are allowed, and we proceed to the second level of control. The second level controls the irregularities at the *Parameter* level. For each *Parameter* of the *Model Agent*, a random number β is generated using a uniform function over $[0,1]$. If $\tau_N > \beta$, irregularities are allowed for this *Parameter* p, and the *Parameter* value is computed using the $g_{p_{\text{ref}}}$ function, taking value in $\mathcal{D}_{p_{\text{ref}}}$, and not $\mathcal{D}_p$. As $\mathcal{D}_p \subset \mathcal{D}_{p_{\text{ref}}}$, the value might be outside $\mathcal{D}_p$: irregularities are possible. If $\tau_N \leq \beta$, irregularities are forbidden. The value is computed using g_p, and thus $v_p \in \mathcal{D}_p$: this parameter will not be in deviation. Finally, if irregularities have been globally forbidden, all *Parameters* p are generated in their definition domain $\mathcal{D}_p$.

This procedure also evaluates the conformity to the reference *Behavioral Pattern*. If the irregularity exceeds the maximal gap to the pattern $\delta_{\max_B}$, the generated values are rejected, and the procedure is run again. It is to be noted that to ensure the end of the procedure, a conformal agent is automatically generated after a configurable number of steps (empirically fixed at 10). This pragmatic solution introduces a bias in the generation process, but guaranties its efficiency.

Example 12. The behavioral pattern B_{normal} from Example 1 allows to generate agents having a 4–6 km/h maximal speed, and "*normal*" as personal space value.

The proposed algorithm creates the *Parameter* values of the *Model Agent* by instantiating its reference *Behavioral Pattern*. Furthermore, the controlled randomness introduced in the process enables the creation of a great variety of agents. Executed at the beginning of the simulation, this algorithm creates the initial agents population. During the simulation, it can be used to dynamically instantiate the agents.

8. Automated configuration of the model

The last part of the proposed model aims at providing the model with one important capability: the automation of behavioral patterns configuration (Lacroix et al., 2009). The objective is to assist the user in the configuration of the model to ease its implementation in existing simulation.

8.1. Behavioral patterns inference

Our objective was to a use a method which would: not require any domain specific configuration to preserve the model genericity; be fully automated to limit supervision needs; and be robust to errors to allow the use of low quality data.

To answer these issues, we chose to use unsupervised classification techniques. Indeed, unsupervised algorithms do not require user feedback during classification: they automatically build clusters of similar data. Moreover, these algorithms are generic: using inputs built with agents' parameters values and outputs matching the *behavioral patterns* structure, the algorithm can be used on a large variety of applications. Finally, these algorithms are robust, as noisy inputs are categorized in existing clusters without triggering errors.

A wide variety of unsupervised algorithms have been developed (Gallant, 1993), and various well-tried techniques are available (Anderberg, 1973; Carpenter and Grossberg, 1987). In our particular case, we selected Kohonen neural networks (Kohonen, 1995). In Kohonen networks, also called self-organizing maps, during the training phase, for each example input vector, the closest neuron to this vector is declared the winner, and the weights of this neurons and its neighbors are adjusted toward the input vector. The network self-organizes, and progressively builds a topological map of the inputs. One of the properties of this map is that the vectors which are close in the data space match close neurons in the map space. Therefore, they provide a spatial representation of the data that allows the final users to visualize them more easily, as a low dimensional view of high dimensional data. This user friendly visualization is the first reason why we chose to use Kohonen Networks. The second reason is that in self-organizing maps, the distance function can be easily adapted to match user needs.

8.2. Algorithm description

Our method is based on a Kohonen network having a rectangular topology, using $(k+1)\cdot(k+1)$ neurons. The network is dynamically built using the size k of the inputs, for genericity purpose. Inputs are vectors holding the values of the agents' parameters.

Algorithm 3 presents the procedure used. During initialization, a Kohonen network is built, and a *Behavioral Pattern* is created for each of the network neurons. The network is then trained using the input set. The second step automatically builds the elements of the *Behavioral Patterns*. For each neuron $u_{i,j}$, a *Behavioral Pattern* $B_{i,j}$ and the vector $W_{i,j}$ of weights computed during training are available. Because of their design, each component $w_{i,j,h}$ of the vector $W_{i,j}$ matches a *Parameter* p_h of $B_{i,j}$. Thus, we assign the default value $w_{i,j,h}$ to this *Parameter*. It uses the default probability distribution and distance function. Finally, during the third step, the definition domains of the *Parameters* are computed. The whole set of inputs is used in classification mode (instead of training). The extreme values of the inputs provide the definition domain bounds.

After these training steps, a full set of *Behavioral Patterns* has been created.

Algorithm 3. Automated creation of behavioral patterns.

Require: a simulation which external parameters are $\mathcal{P}_e=\{p_i, i\in[1,n]\}$, a set of inputs $\mathcal{E}=\{E_l\}$ with $\forall l, E_l=(v_k)$, $k\leq n$, where the v_k match the simulation parameters

1: **network initialization**: creation of a rectangular Kohonen network $\mathcal{K}$ of size $(k+1)\cdot(k+1)$, neurons $u_{i,j}$ and weights $W_{i,j}=(w_{i,j,h})$, $h\leq k$
2: **behavioral patterns initialization**: creation of $(k+1)^2$ *Behavioral Patterns* $B_{i,j}$, holding the k *Parameters* p_k which labels matches the inputs, with $Q_{B_{i,j}}=\emptyset$, $\tau_{B_{i,j}}=0$ and $\delta_{\max_{B_{i,j}}}=1$. Each $B_{i,j}$ uses B_{root} as reference pattern.
3: **training** of $\mathcal{K}$ with the set of examples $\mathcal{E}$
4: **for all** $i,j\leq k+1$ **do** {*Behavioral Patterns* creation}
5: **for all** $h\leq k$ **do**
6: save the weight value $w_{i,j,h}$ of the current neuron $u_{i,j}$ as the default value of the matching *Parameter* p_h in the *Behavioral Pattern* $B_{i,j}$: $v_{d_{B_{i,j}}}(p_h)\leftarrow w_{i,j,h}$
7: if $w_{i,j,h}$ is greater than the maximum, or lower than the minimum of $\mathcal{D}_{p_{\text{ref}}(p_h)}$, update the corresponding bound of the domain
8: **end for**
9: **end for**
10: **for all** $E_l\in\mathcal{E}$ **do** {*Parameters* definition domains creation}
11: classify the example E_l using the network $\mathcal{K}$. Let $u_{i,j}$ be the triggered neuron
12: **for all** $h\leq k$ **do**
13: if $w_{i,j,h}$ is greater than the maximum or lower than the minimum of $\mathcal{D}_{p_h}$, update the corresponding bound of the domain
14: **end for**
15: **end for**

8.3. Automated configuration using real or simulated data

This procedure enables to automatically produce a set of *Behavioral Patterns* matching real or simulated data. To do so, model *Parameters* matching the parameters available in the data set are created. Using the data as inputs, the Algorithm 3 infers a set of behavioral patterns that represent the recorded observations.

This method provides an easy way to parameterize a model, and allows to reproduce observed situations. In simulations representing observable situations, for instance those involving pedestrians or vehicles, data can be recorded from the real world. These data can then be used to infer a set of behavioral patterns. Another option is to use this method to produce patterns using a set of simulated data recorded from a previous simulation. The objective is then usually to reproduce a specific setting.

It is to be noted that we reach here one of the limits of the approach: considering that agents' behaviors are fully defined by their parameters. Indeed, the data are recorded at a specific time step, when a behavior is characterized over time. Therefore, the approach does not reproduce exactly the agents' real behavior, but creates similar initial conditions for the simulation. However,

this remains a promising first step toward a full automation of the approach.

9. Back to our issue: variety and consistency

In this section, we show how the model presented provides a solution to our issue: the introduction of variety in the agents' behaviors, and the conformity control of their parameters with the specifications. We finally present how the model can be introduced in existing simulations.

9.1. Introduction of variety in agents' behaviors

Behavioral Patterns provide a flexible tool to create various behaviors. In our approach, this variety is introduced through the *Behavioral Patterns* definitions, and through the behavioral irregularities.

Based only on the definitions of the *Behavioral Patterns*, the ability to create as many *Parameters* and as many *Behavioral Patterns* as desired offers wide variety possibilities. Indeed, the definition domains of the *Parameters* can introduce specific characteristics, and a *Behavioral Pattern* can include any *Parameter*. For example, if a *Behavioral Pattern* only specifies a single *Parameter*, whose definition domain is a singleton, the *Behavioral Pattern* will always produce the same value for this *Parameter*. On the contrary, if a *Behavioral Pattern* is based on all simulation *Parameters*, each of them having a wide definition domain, then an important variety of agents' parameter values, and thus behaviors, will be produced.

Example 13. In order to study the influence of joggers in a simulated pedestrian flow, we introduce the *Behavioral Pattern* B_{jogger}, with the *Parameter* v_{jogger} of reference parameter v_{max}, such that $v_{d_{jogger}} = 11$ km/h and $\mathcal{D}_{v_{jogger}} = \{11\}$ km/h. The generation of a population of 99% of the agents instantiating B_{normal} and 1% instantiating B_{jogger} allows to introduce easily joggers in the simulation. Moreover, deactivating that *Behavioral Pattern* is enough to prevent the appearance of any jogger in the simulation.

Variety is also introduced between the agents belonging to the same *Behavioral Pattern*. Indeed, during the instantiation of the values by the generation algorithm, the probability distribution naturally introduces variety: the *Parameters* in the *Model Agent* take various values in their definition domain. The flexibility of the approach is increased by the possibility of adapting the distribution to users' needs. The default function is a uniform distribution, but any kind of function can be used, like Gaussian ones.

Example 14. Two agents instantiated from the same *Behavioral Pattern* B_{normal} will get two different sets of values from the definition domain. For instance, two agents a_1 and a_2 instantiated from B_{normal} could be a_1 ($v_{max}(a_1) = 4.7$ km/h, $s(a_1) = normal$), and a_2 ($v_{max}(a_2) = 5.2$ km/h, $s(a_2) = normal$).

Another element used to increase variety in agents' behavior is the behavioral irregularities. Irregularities allow the occurrence of unspecified behaviors in the simulation. For instance, in driving simulators used to evaluate drivers' aid systems, unusual behaviors might unsettle the drivers. This widens the test range, and thus increases the system robustness. In the model, three parameters specify the irregularities. The deviation rate τ_N defines the proportion of allowed irregularities. The maximal gap to the pattern $\delta_{\max_B}$ avoids *Parameter* values to be "too far" from the behavioral pattern. Finally σ centralizes the activation of deviation use at the global level (if $\sigma = 0$, no irregularity can appear, whatever values taken by τ_B).

Example 15. We define the *Behavioral Pattern* $B_{\text{normal}''}$, similar to B_{normal}, but with a deviation rate $\tau_{B_{\text{normal}''}} = 0.01$. $\sigma = 1$ to allow deviations. The *Behavioral Pattern* $B_{\text{normal}''}$ produces 1% of deviating agents, which speed is generated from $\mathcal{D}_{v_{\max}} = [0, 15]$ km/h instead of $\mathcal{D}_{v_{\text{normal}}} = [4, 6]$ km/h. For instance, agent a_3 ($v_{max}(a_3) = 12.4$ km/h and $s(a_3) = small$) and a_4 ($v_{max}(a_4) = 1.4$ km/h and $s(a_4) = normal$) are two deviating agents created from the *Behavioral Pattern* $B_{\text{normal}''}$: $v_{max}(a_3) \notin \mathcal{D}_{v_{\text{normal}}}$ and $v_{max}(a_4) \notin \mathcal{D}_{v_{\text{normal}}}$. Allowing irregularities produces pedestrians moving quickly as well as slow ones, others accepting a small personal space, which increases the variety in the simulation.

Fig. 4 illustrates the different possibilities presented in this section. We represent *Behavioral Patterns* involving two parameters as two dimension figures. It is to be noted that for better visualization, the personal space parameter is represented here as a continuous parameter between -1 and 1. With $-1 = small$, $0 = normal$ and $1 = big$, rounding the values to the nearest integer brings us back to the case of Example 1. Various possibilities are illustrated in Fig. 4: a *generic* behavioral pattern, having wide definition domains; the *normal* pattern from Example 4; a *shopping* pattern involving slower agents accepting only small personal spaces; or a *in a hurry* pattern with quick agents having small personal spaces. The *jogger* behavioral pattern presented in Example 13 is represented by a singleton. Agents a_1 and a_2, from Example 14, belong to the *normal* pattern, but adopt distinct behaviors. Finally, agents presenting irregularities regarding the *normal* pattern will be generated using any value in the "root behavioral pattern", creating agents like a_3 and a_4. Here, these agents are *always deviant*, but *deviant* ones could also be created depending on the model specification.

9.2. Parameters conformity regarding the specifications

The model enables to introduce easily variety among the agents' behaviors. It allows also to control the conformity of agents' parameter values at their creation and during the simulation. To do so, we compare the values of agents' parameters and the specification provided by their reference behavioral patterns, and react according to the criteria defined by the user. Presented in Algorithm 4, the mechanism is an extension of Algorithm 1. It controls the conformity using two elements: the *Root Behavioral Pattern* B_{root}, and the maximal gap to the pattern $\delta_{\max_B}$. This control is done using the *Model Agent*, which provides a reference

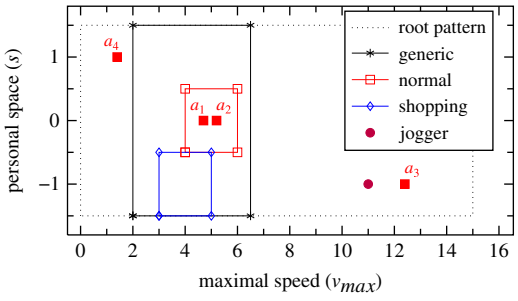

Fig. 4. Behavioral pattern definitions offer various possibilities to create variety. For instance: a *generic* pattern with a wide definition domains, a *normal* pattern, a *shopping* pattern, or a very specialized *jogger* pattern. Agents a_1, a_2, a_3 and a_4 have been instantiated from the *normal* pattern. They all have different behavioral parameters: agents a_1 and a_2 are *conformal* agents, when a_3 and a_4, as *always deviant* agents, presenting behavioral irregularities regarding both their maximal speed and personal space.

between the simulation agents and the *Behavioral Patterns*. The *Model Agent* enables to refer to the adequate "root parameters", *Root Behavioral Pattern*, or *Parameters* definition domains.

First, by implementation, no behavioral irregularity from the *Root Behavioral Pattern* B_{root} is allowed. According to Property 2, this means that if one of the parameters of a simulation agent takes a value outside the domain of the corresponding "root parameter", this change is forbidden, and the value is forced at the domain bound. Therefore, the definition domains specified for the "root parameters" define the absolute limits of the parameter values in the simulation. The *Root Behavioral Pattern* B_{root} provides a frame for the model execution: the system guaranties that no parameter will ever go beyond the definition domains defined in that *Root Behavioral Pattern*. This property is particularly interesting for the final users, who can easily understand this mechanism.

Second, the maximal gap to the pattern $\delta_{\max_B}$ constrains the behaviors at runtime. This parameter defines the maximal allowed value for q_{a_m}. If $\delta_{\max_B}=0$, no deviation is allowed. If $\delta_{\max_B}=1$, the *Parameter* values can change freely, and behavioral irregularities are possible (within B_{root} limits). Between these two values, $\delta_{\max_B}$ defines the degree of conformity to the specification, allowing more or less important irregularities to appear.

Algorithm 4. Conformity control of agents' parameter values.

Require: $\mathcal{A}_m$ the set of Model Agents, a the simulation agent represented by a_m, b=*false* a boolean

for all $a_m \in \mathcal{A}_m$ **do**
  $a_{temp} \leftarrow a_m$ {temporary copy of a_m}
  for all $p \in \mathcal{P}_{a_m}$ **do** {copy a's parameter values in a_m}
    $\nu_p(a_m) \leftarrow \nu_p(a)$
  end for
  for all $p \in \mathcal{P}_{a_m}$ **do** {check of root behavioral pattern}
    if $\nu_p(a_m) \notin \mathcal{D}_{p_{ref}(p)}(B_{\text{root}})$ **then**
      $b \leftarrow$ *true* {change forbidden}
    end if
  end for
  if $q_{a_m} > \delta_{\max_{B_{a_m}}}$ **then** {quantification of the irregularities}
    $b \leftarrow$ *true* {change forbidden}
  end if
  if b **then** {change forbidden, return to previous values}
    $a_m \leftarrow a_{temp}$
    for all $p \in \mathcal{P}_{a_m}$ **do**
      $\nu_p(a) \leftarrow \nu_p(a_m)$
    end for
  end if
end for

That flexibility enables a precise control over the simulation: in experimentations where the behaviors have to remain under control, irregularities will be limited or forbidden. In simulations where unusual or unspecified behaviors are interesting, they will be authorized. We point out that even if irregularities are authorized, they do not necessarily appear. For instance, if the definition domain of a *Parameter* is close to these of its reference parameter, they will seldom happen.

Finally, it is to be noted that the behaviors instantiated by the model will be consistent only if the *Behavioral Pattern* specification is itself consistent. Indeed, nothing guaranties the consistency of the behaviors defined using the *Behavioral Patterns*. On the contrary, users might wish to experiment the influence of inconsistent behaviors, and introduce them using patterns. The model only allows to control the conformity of agents' parameter values with the specification provided in the *Behavioral Pattern*.

9.3. Introducing the model in an existing simulation

The model can be used as a tool to introduce diversity in the simulation and control the conformity of the elements produced. Indeed, by design, the tool integration does not require any modification of the simulation. The model interacts with the simulation using its public configuration parameters. The simulation runs using its own decision model, and sends a request to the model in specific situations, like for parameters creation.

To use the tool, a user only has to specify the *Parameters*, their default values, their definition domains, and one *Behavioral Pattern*. All other elements required by the model are automatically completed using default values.

These default values are created as following. For each *Parameter p* provided by the user, a "root parameter" is automatically created, using the same values, and set as p reference parameter. The probability distribution g_p and distance function f_p use their default values, and a *Root Behavioral Pattern* B_{root} including all the "root parameters" is created. As for user defined *Behavioral Patterns*, their reference pattern is B_{root}, none of them hold properties ($\mathcal{Q}_B=\emptyset$), no deviation is allowed ($\tau_B=0$) and deviations remain unconstrained ($\delta_{\max_B}=1$). Finally, a *Setting* is automatically created with all the patterns and $\sigma=0$.

Communication between model and simulation can rest on pre-existing simulation functions. It might for instance be network messages, like in the application presented in Section 10. If such a function does not exist, the input point has to be created, for instance with specific developments or by adding the model as an internal module. The non-intrusive property is then partially lost, but using the model remains interesting due to its modularity and the out-of-the-agent design it introduces.

Finally, the complexity of the configuration process can be illustrated through the number of parameters that have to be manually defined to produce a fixed variability. In the case of a population of n agents using p behavioral parameters, if no automated process is used, p^n parameters have to be defined, and the consistency of the behaviors has to be manually assessed. Using the proposed model, for each *Behavioral Pattern B*, at most p *Parameters* have to be defined, each one involving the definition of five numerical values, one probability distribution, and one distance function. If we use normal distributions and the default distance function, for a setting involving n_B patterns, $7 \cdot p \cdot n_B$ values have to be set, and the consistency of the produced behaviors is guaranteed.

The proposed approach allows to produce various and consistent behaviors while defining only a limited number of parameters.

10. Application to traffic simulation

In this paper, our main application is the simulation of road traffic in driving simulators. In this section, we present our objectives in that particular case, the implementation of the model in a commercial software, and experimental results showing how it improved the existing simulation.

10.1. Traffic simulation in driving simulators

Traffic and drivers' behavior is a widely studied field, from a psychological (Salvucci et al., 2001; Summala, 2005) as well as engineering perspective (Bazzan, 2005; Dresner and Stone, 2008). One of the tools involved in these researches is driving simulators. In such simulators, a real driver is introduced in the simulation, and interacts with autonomous vehicles handled by the simulation (Fig. 5). Driving simulators are used in the automotive

industry for various purposes, like the development of driving aid systems or vehicles design (Reymond et al., 2001; Toffin et al., 2007). The traffic has thus to be as realistic as possible, to induce a mental state similar to a drive in the real world and improve the validity of the experimentation results. The objective in driving simulators is to immerse the driver of the simulator in the simulated traffic, and produce specific traffic situations (Olstam and Espié , 2007). In contrast to traffic simulation tools like AIMSUN (Barcelò and Casas, 2002) or VISSIM (Fellendorf and Vortisch, 2001), the reproduction of real traffic flows is therefore not the most important criteria used to evaluate the validity of the results.

Various driving simulation software are available (Cremer et al., 1995; Olstam, 2005; Doniec et al., 2008). Among them, SCANER^{TM} was initially developed by Renault, and is now co-developed and distributed by Oktal (2012). SCANER^{TM} is based on a distributed architecture, and modules communicate using the network. Communications are based on a common protocol, available through an API. This API was used to implement the developments presented here.

In SCANER^{TM}, the traffic module is based on a multi-agent system architecture (Champion et al., 1999). The traffic decision model uses different pseudo-psychological parameters, to incorporate the psychological factors involved in drivers' behaviors (Dewar, 2002). Presented in Table 1, these parameters are taken into account during the decision phase of the model. They allow to manage the drivers' behaviors, in particular those related to the risk taking and the respect of traffic law.

10.2. Objective: introducing a controlled variety in the traffic

The major issue at stake in SCANER^{TM} was that the behaviors of the autonomous vehicles were based on parameters defined during scenario creation. As presented in Section 3, each vehicle had to be manually added in the scenario, and was created with the same behavioral parameter values. The design of a scenario involving a high number of vehicles was thus a long and repetitive work, and scenario was therefore usually created with as few vehicles as possible, and these vehicles used their (similar) default parameters. These elements highly diminished the immersion of the users in the simulation, and two main elements were thus targeted in our work: introduce variety to increase the users' immersion, and automate the process to ease the work of the scenario designers.

To achieve these goals, we introduced driving styles in the traffic. Indeed, driving styles reflect the fact that drivers can adopt various behaviors (aggressive, cautious, etc). Moreover, it has been shown that introducing such styles in a simulated traffic increases the users' immersion in driving simulators (Wright et al., 2002).

Driving styles represent "categories" of drivers having similar personality traits. They can thus be represented using *Behavioral Patterns*. Moreover, styles imply consistency between parameters: an aggressive driver not only drives quickly, but takes also more risks. *Behavioral Patterns* allow taking these constraints into account.

Driving styles being described as *Behavioral Patterns*, the behavioral differentiation model allows to automatically and easily generate agents belonging to the desired *Behavioral Pattern*, i.e. drivers reproducing the specified driving style.

10.3. Implementation of the model in the existing simulation

The model enables the introduction of driving styles in the simulation. Wright et al. (2002) showed that driving simulator users only distinguish a limited set of driving styles: aggressive, normal, and cautious. In SCANER^{TM}, we chose to provide these three driving styles as the default configuration of the software. However, parameters can be manually changed to adapt to the simulation context: for instance, an aggressive behavior in France may not be perceived as such in Italy.

We also chose to use only the existing parameters of the traffic decision model: the maximal speed v_{max}, the safety time t_s, the overtaking risk r_o, the speed limit risk r_s, the observe priority o_p and the observe signalization o_s parameters. This way, no modification of the simulation is needed, and the introduction of the model is fully non-intrusive. As underlined in Section 3, this was a major issue for the integration in the commercial version of the software.

The definition domains were empirically defined, using the expertise of scenario designers. We defined three *Behavioral Patterns*, matching the three targeted driving styles: B_{cautious}, B_{normal} and $B_{\text{aggressive}}$. For these three *Behavioral Patterns*, $\tau_{B_i} = 0$, $\delta_{\max_{B_i}} = 1$, and $\mathcal{Q}_{B_i} = \{France, highway\}$. Table 2 presents the *Behavioral Patterns* and *Parameters* used in the implementation. For all the real numbers, the probability function g_p is defined as a

Table 1
The pseudo-psychological parameters of the agents are taken into account during the decision phase of the traffic decision model, and influence the virtual drivers' behaviors.

Parameter	Possible values	Default
Maximal speed v_{max}	[0,∞] km/h	130 km/h
Security distance t_s	[0,∞] s	2 s
Risk taking to overtake r_o	[−1, 2]	0
Speed limit respect r_s	[0,∞]	1
Priorities respect o_p	{*true*,*false*}	*True*
Traffic signs respect o_s	{*true*,*false*}	*True*

Fig. 5. Left, one of Renault's driving simulators: the user, wearing a virtual reality helmet, drives in the simulation using a real car's cockpit. Right, a screenshot of SCANER^{TM}, the application used at Renault, where the scenario involves confronting the driver to a risky situation while driving on snow, with limited car adherence.

Table 2
Definition of the *Parameters* of the *Behavioral Patterns*. From up to bottom: the "root parameters" of the *Root Behavioral Pattern* B_{root}, and the *Parameters* of B_{cautious}, B_{normal}, and finally $B_{\text{aggressive}}$. Each of these three behavioral patterns enables the creation of agents reproducing a specific driving style.

Behavioral pattern	Parameter label	p_{ref}	$\mathcal{D}_p$	v_{d_p}	g_p	f_p
B_{root}	v_{max} (maximal speed)	0	[0, 300] km/h	130 km/h	$\mu = 130$, $\sigma = 10$	Default
	t_s (safety time)	0	[0, 10] s	2 s	$\mu = 2$, $\sigma = 1$	Default
	r_o (overtaking risk)	0	[−1, 2]	0	$\mu = 0$, $\sigma = 1$	Default
	r_s (speed limit risk)	0	[0, 2]	1	$\mu = 1$, $\sigma = 1$	Default
	o_p (observe priority)	0	{*true*, *false*}	*True*	Default	Default
	o_s (observe signalization)	0	{*true*, *false*}	*True*	Default	Default
B_{cautious}	$v_{max,cautious}$	v_{max}	[90, 110] km/h	100 km/h	$\mu = 100$, $\sigma = 10$	Default
	$t_{s,cautious}$	t_s	[1.5, 2.5] s	2.0 s	$\mu = 2.0$, $\sigma = 0.25$	Default
	$r_{o,cautious}$	r_o	[−1.0, 0.0]	−0.5	$\mu = -0.5$, $\sigma = 0.25$	Default
	$r_{s,cautious}$	r_s	[0.8, 1.0]	0.9	$\mu = 0.9$, $\sigma = 0.025$	Default
	$o_{p,cautious}$	o_p	{*true*}	*True*	Default	Default
	$o_{s,cautious}$	o_s	{*true*}	*True*	Default	Default
B_{normal}	$v_{max,normal}$	v_{max}	[110, 130] km/h	120 km/h	$\mu = 120$, $\sigma = 10$	Default
	$t_{s,normal}$	t_s	[1.0, 2.0] s	1.5 s	$\mu = 1.5$, $\sigma = 0.25$	Default
	$r_{o,normal}$	r_o	[0.0, 1.0]	0.5	$\mu = 0.5$, $\sigma = 0.25$	Default
	$r_{s,normal}$	r_s	[0.9, 1.1]	1.0	$\mu = 1.0$, $\sigma = 0.025$	Default
	$o_{p,normal}$	o_p	{*true*}	*True*	Default	Default
	$o_{s,normal}$	o_s	{*true*}	*True*	Default	Default
$B_{\text{aggressive}}$	$v_{max,aggressive}$	v_{max}	[130, 150] km/h	140 km/h	$\mu = 140$, $\sigma = 10$	Default
	$t_{s,aggressive}$	t_s	[0.5, 1.5] s	1.0 s	$\mu = 1.0$, $\sigma = 0.25$	Default
	$r_{o,aggressive}$	r_o	[1.0, 2.0]	1.5	$\mu = 1.5$, $\sigma = 0.25$	Default
	$r_{s,aggressive}$	r_s	[1.0, 1.2]	1.1	$\mu = 1.1$, $\sigma = 0.025$	Default
	$o_{p,aggressive}$	o_p	{*true*, *false*}	*True*	Default	Default
	$o_{s,aggressive}$	o_s	{*true*, *false*}	*True*	Default	Default

normal distribution of mean value μ and variance σ, truncated at $\mathcal{D}_p$ bounds. Finally, the *Root Behavioral Pattern* B_{root} is defined by $\mathcal{P}_{B_{\text{root}}} = \{v_{max}, t_s, r_o, r_s, o_p, o_s\}$.

The developments have been implemented in different modules, and interfaced with the simulation using the SCANER^{TM} API. The first module is used to edit pre-existing scenarios. It introduces driving styles during scenario design, and allows to apply driving styles on groups of vehicles, using a statistical repartition defined by the user (for instance, 10% aggressive and 90% normal drivers). The second module dynamically creates new vehicles during the simulation, and controls the vehicles parameters. It is used to easily populate a database using traffic generators, and create an "ambient traffic". The model is applied during the vehicles creation. The module also provides additional functionalities, such as traffic statistic recording. Finally, a third module allows to infer behavioral patterns from simulation records.

These three modules provide functionalities at the different simulation steps: during scenario creation, to easily introduce variety in the traffic; at runtime, to create new vehicles of various behaviors and control their parameters; and after the simulation, to analyze its results.

10.4. Experimental results

We evaluated the model contributions through experimentations on the traffic. We present here the protocol used, as well as experimental result on variety, behaviors representativeness, and behavioral patterns inference.

10.4.1. Experimental protocol

The simulations took place on a database representing a highway, on a 11 km long section. There are no entries or exit lanes on the section, to ensure a constant vehicles flow. Three traffic detectors recorded vehicles speed, lane, and initial behavioral patterns, at kilometers 2.2, 6 and 10.8. Traffic generators created a traffic demand of 3000 vehicles/h at the beginning of the section, which represents a dense flow. After the initial creation of the vehicles, the SCANER^{TM} traffic module was used to compute their displacements. Each simulation lasted 2h30 – due to the commercial software design, time could not be accelerated – and the data presented below are average values of five distinct simulation runs. The p-values of Welch's t tests computed between each series of measured parameters values were never lower than 0.49 (on average 0.69 for the speed and 0.66 for the safety times, with a minimal value of respectively 0.55 and 0.49).

For this evaluation, we defined three different *Settings*. To avoid the influence of behavioral irregularities on the results, we set their global determinism criteria to zero ($\sigma = 0$). The first *Setting*, S_{default}, includes only one restrictive *Behavioral Pattern* B_{default}. In B_{default}, every *Parameter* is defined as a singleton, and irregularities are forbidden. The *Parameters* use the default values attributed by the scenario editor: $v_{max} = 130$ km/h, $t_s = 2$ s, $r_o = 0$, $r_s = 1$, $o_p = true$, $o_s = true$. Using such a behavioral pattern has the same effect as deactivating the model: each vehicle is created with the same parameter values. This *Setting* is our reference in that evaluation: the parameter values resulting from this configuration are similar to former hand-designed scenarios.

The second *Setting*, $S_{\text{normal\_only}}$, also includes only one *Behavioral Pattern*. This *Behavioral Pattern*, $B_{\text{normal\_only}}$, is an adaptation of the pattern B_{normal} presented in Section 10.3, where the maximal speed definition domain is [100, 140] km/h instead of [110, 130] km/h. All other *Parameters* remain unchanged. During the simulations, the traffic generators create 100% of the vehicles using the *Behavioral Pattern* $B_{\text{normal\_only}}$.

Finally, the third *Setting*, $S_{\text{all\_patterns}}$, is based on the three *Behavioral Patterns* B_{cautious}, B_{normal} and $B_{\text{aggressive}}$ defined in Section 10.3. During the simulation, the traffic generators create 10% of the vehicles using the *Behavioral Pattern* B_{cautious}, 80% using B_{normal} and 10% using $B_{\text{aggressive}}$.

In this work, we chose to observe different quantitative criteria to assess the validity of the approach. Indeed, the usual approach, i.e. leading a user study using questionnaires, is highly subjective

and sensible to the context. In our case, the software and the proposed model are used in various countries, from China to England. Instead of proving the approach on a specific setting that would not be extensible to other contexts, we therefore chose to propose a first set of quantitative criteria to assess the validity of a setting. However, as we state in Section 12, this work would certainly benefit from a complementary user study.

10.4.2. Evaluation of the behaviors variety

We first evaluated the influence of the behavioral patterns on the behaviors variety. The Fig. 6 represents the vehicles speed distribution recorded by the detector 2. The data recorded by the other detectors are similar. When no pattern is used (setting S_{default}), the recorded speeds are either low, between 70 and 90 km/h (46% of the vehicles), or high, around 130 km/h (40% of the vehicles): the curve presents a "camel back" shape. Vehicle behaviors are too similar to adapt to small variations in the traffic flow: the left lane remains slow, the right one quick. Very few vehicles change lane or overtake, which explains the shape of the distribution.

When a behavioral pattern is introduced, using the setting $S_{\text{normal\_only}}$, the speed distribution is more balanced: 60% of the vehicles adopt a speed between 90 and 115 km/h, and 30% between 115 and 140 km/h. The curve shape is similar to a Gaussian centered on 110 km/h. Vehicles behavior is more dynamic, involving lane changes and overtakings. Moreover, the vehicles average speed reaches 100.4 km/h, against 91.5 km/h for S_{default}. That increase shows that the behaviors variety improves vehicles use of the road network.

In the third case, with the setting $S_{\text{all\_patterns}}$, the speed distribution also presents a Gaussian shape. Vehicles average speed increases to reach 103.7 km/h: again, an increased behavioral variety improves the vehicles speeds. The introduction of additional behavioral pattern produces two "bounces" in the curve, at 70 and 150 km/h, showing that more extreme and less predictable behaviors have been successfully introduced. These elements illustrate how behavioral patterns allow to increase the heterogeneity of the created population, as characteristics of the subpopulations – here the cautious and aggressive drivers – appear in the global population.

Finally, we analyzed the vehicles travel time through the whole section. The introduction of a behavioral pattern reduces that time by 11.6%, from 5 min 35 s to 4 min 56 s. This matches our observations on the speed: more variety allows vehicles to better use the road network and travel faster. However, if more extreme behaviors are introduced using $S_{\text{all\_patterns}}$, the travel time does not decrease, but increases: +6.1%, from 4 min 56 s with $S_{\text{normal\_only}}$ to 5 min 14 s with $S_{\text{all\_patterns}}$. Simultaneously, the average speed increases: +3.3% at detector 2, from 100.4 km/h to 103.7 km/h. This counter-intuitive result can be explained by the presence of slow vehicles, which limit the average progression of the traffic, even if some vehicles drive at higher speeds.

10.4.3. Representativeness of the behaviors

Another element we aimed to evaluate was the representativeness of the behaviors produced by the model. Fig. 7 presents the vehicles repartition between the left and right lanes of the highway, depending on their reference behavioral pattern. Data have been recorded at detector 2, with the setting $S_{\text{all\_patterns}}$. Most of the aggressive drivers are on the left lane (on average, 71%), when most of the cautious ones stay on the right lane (82%). Normal drivers repartition is more balanced, with 72% of drivers on the right lane, and 28% on the left one.

According to their behavioral pattern, aggressive drivers adopt high speeds and small security margins. They often overtake, and we naturally observe them on the left lane. On the contrary, cautious drivers stay on the right lane, as their behavioral characteristics drive them to do. The model introduces the behaviors we desired to observe.

10.4.4. Behavioral patterns inference

Finally, to evaluate the monitoring part of the model, we based our experimentations on data recorded from a simulation. Experimentations with real world data will be considered in the future works.

The data were recorded at detector 2 during 2h30, using the setting $S_{\text{all\_patterns}}$. About 3159 different vehicle recordings were produced. Using the method presented in Section 8, we inferred a set of nine behavioral patterns involving two behavioral parameters, the maximal speed and the security distance, based on the recorded vehicles speeds and safety times.

Fig. 8 represents the sample data set, as well as the inferred behavioral patterns. Each *Behavioral Pattern* is represented by a diamond-shape drew the coordinates of the default values of its *Parameters* (i.e. $(v_{d_{v_{max}}}, v_{d_{t_s}})$). Then, for each behavioral pattern, a rectangle represents the definition domain of each parameters of this behavioral pattern (i.e. $\mathcal{D}_{v_{max}}$ on the x-axis, $\mathcal{D}_{t_s}$ on the y-axis).

Each behavioral pattern is also associated to a probability factor that represents the probability that it will be used to generate a vehicle. Here, this probability is built using the number of vehicles belonging to the behavioral pattern (i.e. matching the corresponding neuron in the Kohonen network) over the total number of recorded vehicles. Table 3 presents the values obtained for the three behavioral patterns of higher occurrence probability.

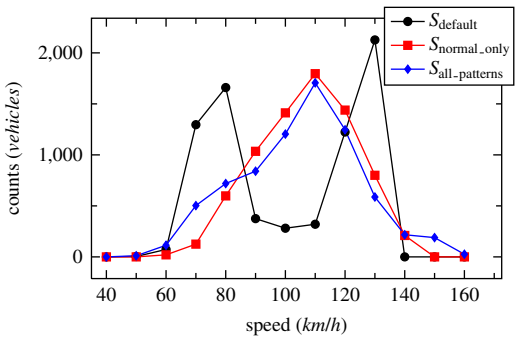

Fig. 6. Speed distributions depending on the *Behavioral Pattern* set used in the simulation. The *Setting* S_{default} induces a "camel back" curve shape, denoting the limited dynamics of the traffic, when $S_{\text{normal\_only}}$ and $S_{\text{all\_patterns}}$ present more reality-like Gaussian shapes.

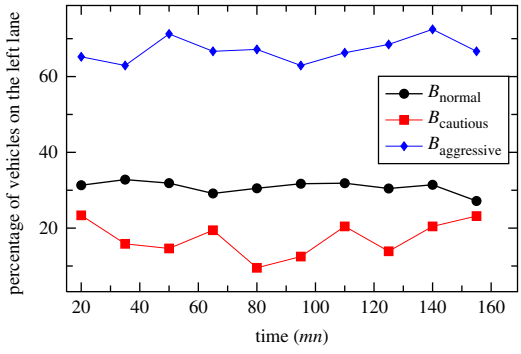

Fig. 7. Vehicles repartition between left and right lanes on the highway, depending on their behavioral pattern (setting $S_{\text{all\_patterns}}$). Aggressive drivers are mostly observed on the left lane (71%), cautious and normal ones on the right lane (only 18% and 28% of them respectively on the left lane).

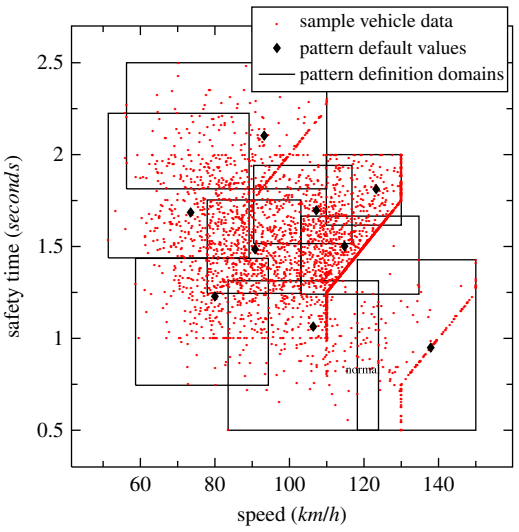

Fig. 8. Recorded safety times over the maximal speeds of the vehicles, and the behavioral patterns inferred from these sample data. The diamond-shape represent the default values of the behavioral pattern parameters, and the rectangles their definitions domains. Using the behavioral pattern occurrence probability, a population statistically close to the recorded one can then be created with the generation algorithm.

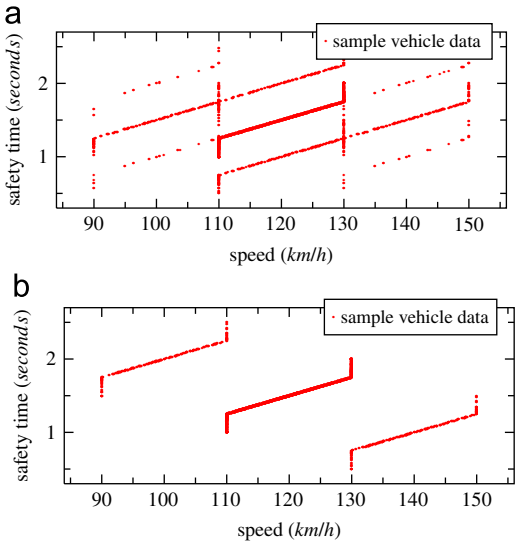

Fig. 9. First, the results obtained with the intermediate approach, and second those obtained using *Behavioral Patterns* and the setting $S_{\text{all\_patterns}}$. The intermediate approach creates vehicles presenting inconsistent behaviors regarding the specifications – high speeds and high safety time for instance – when the *Behavioral Patterns* only produce consistent behaviors.

Table 3
Parameter values of three of the nine inferred behavioral patterns (the three presented behavioral patterns are those of higher probability).

Pattern	$(\nu_{d_{\nu_{max}}}, \nu_{d_{t_s}})$	$\mathcal{D}_{\nu_{max}}$	$\mathcal{D}_{t_s}$	p
1	(103.1, 134.7)	[114.8, 1.2]	[1.7, 1.5]	0.22
2	(77.9, 103.1)	[90.8, 1.2]	[1.7, 1.5]	0.14
3	(83.5, 123.9)	[106.3, 0.5]	[1.3, 1.1]	0.13

Using these inferred behavioral pattern parameters and the generation algorithm, a population statistically close to the recorded one is produced by using each behavioral pattern function of its occurrence probability. The inferred behavioral patterns represent accurately the space of recorded data, which is promising for the use of this technique with real data sets.

10.4.5. Comparison with an intermediate solution

To further illustrate the interest of the approach, we compared the results obtained using *Behavioral Patterns* with an intermediate solution. Indeed, another solution to introduce variety in the simulation could be to select the parameters we want the population to differ in, before applying simple statistical distributions to these parameters.

We created a population using this method and the same distributions and proportions as those used in the setting $S_{\text{all\_patterns}}$. The maximal speed parameter of the vehicles were therefore generated using either the probability distribution of $\nu_{max,normal}$ – a normal distribution of mean value $\mu = 120$ and variance $\sigma = 10$, see Table 2 – $\nu_{max,cautious}$ or $\nu_{max,aggressive}$, used with a respectively 80%, 10% and 10% probability. Similarly, the safety time of the vehicles were generated using $t_{s,normal}$, $t_{s,cautious}$ and $t_{s,aggressive}$, used with the same 80%, 10% and 10% respective probabilities.

Fig. 9 presents the obtained results. Although the population created with the intermediate solution presents a satisfying variety of behaviors, some of those behaviors do not respect our consistency criteria. Indeed, we observe vehicles having a high maximal speed and high safety time, or low maximal speed and low safety time. On the contrary, the use of *Behavioral Patterns* guaranties that only behaviors consistent with the specified behaviors appear: here, cautious drivers – low maximal speed and high safety time –, normal and aggressive ones.

The consistency criteria, which was one of the objectives of the proposed approach, can therefore not be tackled easily with a simplified approach, when *Behavioral Patterns* successfully take this issue into account.

10.5. Conclusion on the application to traffic simulation

Using behavioral patterns in SCANER^{TM} successfully introduces driving styles in the simulation. The behavioral variety is increased, and representative behaviors are created, which contributes to the users' immersion.

Furthermore, the designers' work is simplified by the introduction of the model. Vehicles parameter values are automatically created, and definition domains can be easily adapted to the context. The manual modification of vehicle parameters is therefore reduced, and the automation of scenario design increased. Due to these contributions, the model has already been introduced in the commercial version of the application.

However, this evaluation would benefit from a complementary study in urban environments. Introducing deviant behaviors in this context remains an open issue, as complex intersections already require specific traffic models (Doniec et al., 2008). The SCANER^{TM} traffic model is currently under improvement to be able to successfully tackle this issue.

Considering future works on this application to traffic in driving simulators, a very promizing perspective is the integration of this approach with dedicated traffic simulation softwares. Punzo and Ciuffo (2011) for instance associated SCANER^{TM} with a traffic simulator to produce results that would be valid at the macroscopic level, while allowing realistic vehicles behaviors in the vicinity of the human driver. It would be very interesting to evaluate if similar results could be obtained using the population construction model, using for instance behavioral patterns learned from real data or a traffic simulation software.

11. Illustration of the genericity of the approach

The proposed tool can be used for different applications. For instance, the introduction of variety in crowds simulation presented in Section 2 is based on the variation of texture colors, accessories and body shapes. In each of those cases, the method rests on the random selection of values in constrained definition domains. Using our approach, all those criteria can be included in *Behavioral Patterns*, which would improve maintenance and flexibility of the solution.

All we have to do is to describe the pedestrians' parameters using the proposed semantics, and define associated behavioral patterns. For instance, lets create *Parameters* for the color elements (for example the shade *Parameter shade*, with $\mathcal{D}_{shade} = [20, 250]$ and $v_{d_{shade}} = 135$), *Parameters* for each accessory category (for example the "men's bag" *Parameter* b_{men}, with $\mathcal{D}_{b_{\text{men}}} = \{none, briefcase\}$ and $v_{d_{b_{\text{men}}}} = none$), and a *Parameter* for the body shape (*shape*, with $\mathcal{D}_{shape} = [30, 130]$ and $v_{d_{shape}} = 50$). Then, we define a *Behavioral Pattern* B_{man}, with $\mathcal{P}_{B_{\text{man}}} = \{shade, b_{\text{men}}, shape\}$. The generation algorithm naturally introduces variety using these values. Furthermore, using this out-of-the-agent design, the configuration is available to users who do not have any graphical expertise, when these elements were previously hard-coded in the graphic models.

12. General discussion

The objective of this work was to improve the agents' behavioral realism in the simulation. The proposed approach focuses on the behavioral variety and consistency, and is based on the use of the agents' configuration parameters. Whether or not these parameters can be considered as representing the agents' behavior determines the relevance of the approach. As our work shows, the parameters provide an easy and intuitive way to influence the agents' behavior, which addresses our issues. Moreover, using these parameters allow the final users to intuitively implement the model contributions. We think this is one of the key strength of the model, as it facilitates its use in industrial applications.

Another element to be considered is the validation of the model contributions. Section 10 shows how the model introduces variety in the simulation, and that the generated behaviors are conform to the users' expectations. However, the current evaluation should be completed with a simulators user study. This study would prove the validity of the approach on a limited set of settings, and allow to improve the automated validation with complementary informations on the most significant parameters. We note that it remains difficult, though, to demonstrate the model contributions without any application frame: it would be very interesting to introduce criteria enabling such a quantification.

Finally, behavioral patterns inference let appear one of the limits of the approach. Punctual data that are used to infer behavioral patterns do not accurately capture behaviors, because they do not include a temporal dimension. The inferred behavioral patterns only enable to reproduce the agents' behaviors at the time of recording, but not their behaviors over the time. These behavioral patterns thus reflect the situation at a specific time step, but the situation will probably quickly adopt a different course. In order to address this issue, we plan to add a time-dependent analysis of the behaviors. For instance, the agents' behaviors could be recorded continuously, and these profiles used to infer the behavioral patterns.

13. Future works

Some of the functionalities provided by the model have not been fully used in the presented applications. Among them, behavioral pattern properties and behavioral irregularities present fruitful perspectives. Behavioral pattern properties enable the introduction of context-specific patterns, which could be activated and deactivated depending on the situation. For instance, a specific set of behavioral patterns could be used for urban traffic, and another one for highway traffic. The model would then switch automatically at runtime depending on the vehicle position. As for behavioral irregularities, they can be used to introduce unspecified behaviors in the simulation. These different possibilities have to be explored in future applications.

Finally, a very promising development is the automation of behavioral patterns creation. Indeed, one of the current limits of the approach is the behaviors description, which is only based on the agents' parameters. Extending the model by automatically creating a higher level system would provide interesting properties. The objective is to induce the behavioral patterns formal definitions from the pattern implementation, with a bottom-up approach. Indeed, our users know their application field: they would define parameters and definition domains as they are now used to, and, from these elements, patterns would be automatically created. This work represents the next step in the automation and simplification of the process, which follows the automation of patterns configuration presented in Section 8.

14. Conclusion

In this work, we proposed a formalization of the construction of population for agent-based simulations. Our objective was to propose a solution improving the simulation realism, by simultaneously taking into account the variety and consistency of the agents' behaviors, and to provide a tool for scenario designers, to ease the conception tasks. The proposed model involves three main dimensions. The behaviors are described using behavioral patterns, based on the agents' behavioral parameters. This allows to specify behaviors during the conception of simulation scenario, and to control this specification at runtime. A specific algorithm then instantiates the parameter values from the behavioral patterns. This algorithm provides two levels of randomness control: a global criteria and a parameter based criteria. Finally, the third part of the model presents an automated method for the construction of behavioral patterns from sample data, based on automatic learning techniques.

These elements introduce variety in the simulation, using the behavioral patterns definition capabilities, the properties of the generation process, and the behavioral irregularities. Behaviors consistency is ensured by the enforcement of a root behavioral pattern and user defined parameters characterizing behaviors variability. The model provides a generic approach and non-intrusive solution, and allows an out-of-the-agent design.

We applied the model to traffic simulation in driving simulators, using the software used at Renault, SCANeR™. The objective was to introduce the model contributions in the simulation, and to ease scenario designers works. Experimental results showed that the model increased the variety and produced representative behaviors. These contributions as well as the flexibility of the approach led to the integration of the model in the commercial version of the application.

Finally, future works will address one of the limits of that approach, the choice of a parameter based control of the behaviors. We will also pursue promising perspectives leading to fully automate the model.

References

Anderberg, M.R., 1973. Cluster Analysis for Applications. Academic Press, New York.

Barcelò, J., Casas, J., 2002. Dynamic network simulation with AIMSUN. In: International Symposium on Transport Simulation. Yokohama, Japan.

Bazzan, A.L.C., 2005. A distributed approach for coordination of traffic signal agents. Autonomous Agents Multi-Agents Syst. 10 (2), 131–164.
Carpenter, G., Grossberg, S., 1987. A massively parallel architecture for a self-organizing neural pattern recognition machine. Comput. Vision Graph. Image Process. 37, 54–115.
Champion, A., Mandiau, R., Kolski, C., Heidet, A., Kemeny, A., 1999. Traffic generation with the SCANeR^{TM} simulator: towards a multi-agent architecture. In: Driving Simulation Conference. Paris, France. pp. 311–324.
Cremer, J., Kearney, J., Papelis, Y., 1995. HCSM: a framework for behavior and scenario control in virtual environments. ACM Trans. Modeling Comput. Simulation 5 (3), 242–267.
Dewar, R.E., 2002. Individual differences. In: Dewar, R., Olson, P. (Eds.), Human Factors in Traffic Safety. Lawyers & Judges, pp. 111–142.
Doniec, A., Mandiau, R., Piechowiak, S., Espié, S., 2008. A behavioral multi-agent model for road traffic simulation. Eng. Appl. Artif. Intell. 21, 1443–1454.
Dresner, K., Stone, P., 2008. A multiagent approach to autonomous intersection management. J. Artif. Intell. Res. 31, 591–656.
Fellendorf, M., Vortisch, P., 2001. Validation of the microscopic traffic flow model VISSIM in different real-world situations. In: Transportation Research Board. Washington, DC, USA.
Gallant, S.I., 1993. Neural Network Learning and Expert Systems. MIT Press.
Kohonen, T., 1995. Self-Organizing Maps. Springer.
Lacroix, B., Mathieu, P., Kemeny, A., 2008a. A normative model for behavioral differentiation. In: IEEE/WIC/ACM International Conference on Intelligent Agent Technology. IEEE Press, Sydney, Australia. pp. 96–99.
Lacroix, B., Mathieu, P., Kemeny, A., 2008b. The use of norms violations to model agents behavioral variety. In: Hubner, J., Matson, E., Boissier, O., Dignum, V. (Eds.), Coordination, Organizations, Institutions and Norms in Agent Systems IV. Lecture Notes in Computer Science, vol. 5428. Springer, pp. 220–234.
Lacroix, B., Mathieu, P., Kemeny, A., 2009. Generating various and consistent behaviors in simulations. In: Demazeau, Y., Pavon, J., Corchado, J., Bajo, J. (Eds.), International Conference on Practical Applications of Agents and Multi-Agent Systems. Advances in Intelligent and Soft Computing, vol. 55. Springer. pp. 110–119.
Maim, J., Yersin, B., Thalmann, D., 2009. Unique character instances for crowds. IEEE Comput. Graph. Appl. 29 (6), 82–90.
Oktal, January 2012. SCANeR Home Page. URL ⟨http://www.scanersimulation.com/⟩.
Olstam, J., 2005. Simulation of rural road traffic for driving simulators. In: Transportation Research Board. Washington, DC, USA.
Olstam, J., Espié, S., 2007. Combination of autonomous and controlled vehicles in driving simulator scenarios. In: International Conference on Road Safety and Simulation. Rome, Italy. pp. 23–32.
Pellens, B., Kleinermann, F., Troyer, O.D., 2009. A development environment using behavior patterns to facilitate building 3D/VR applications. In: Australasian Conference on Interactive Entertainment. Sydney, Australia.
Punzo, V., Ciuffo, B., 2011. Integration of driving and traffic simulation: issues and first solutions. IEEE Trans. Intell. Transp. Syst. 12 (2), 354–363.
Reymond, G., Kemeny, A., 2000. Motion cueing in the Renault driving simulator. Vehicle Syst. Dyn. 34, 249–259.
Reymond, G., Kemeny, A., Droulez, J., Berthoz, A., 2001. Role of lateral acceleration in driving: experiments on a real vehicle and a driving simulator. Hum. Factors 43 (3), 483–495.
Reynolds, C.W., 1999. Steering behaviors for autonomous characters. In: Game Developers Conference. San Francisco, USA. pp. 763–782.
Salvucci, D.D., Boer, E.R., Liu, A., 2001. Toward an integrated model of driver behavior in a cognitive architecture. Transp. Res. Rec. 1779, 9–16.
Spronk, P., Ponsen, M., Sprinkhuisen-Kuyper, I., Postma, E., 2006. Adaptive game AI with dynamic scripting. Mach. Learn. 63 (3), 217–248.
Summala, H., 2005. Traffic psychology theories: towards understanding driving behaviour and safety efforts. In: Underwood, G. (Ed.), Traffic and Transportation Psychology. Elsevier, pp. 383–394.
Toffin, D., Reymond, G., Kemeny, A., Droulez, J., 2007. Role of steering wheel feedback on driver performance: driving simulator and modeling analysis. Vehicle Syst. Dyn. 45 (4), 375–388.
Tozour, P., 2002. The evolution of game AI. In: Rabin, S. (Ed.), AI Game Programming Wisdom. Charles River Media, pp. 3–15.
Ulicny, B., de Heras Ciechomski, P., Thalmann, D., 2004. Crowdbrush: interactive authoring of real-time crowd scenes. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation. Grenoble, France. pp. 243–252.
Wright, S., Ward, N.J., Cohn, A.G., 2002. Enhanced presence in driving simulators using autonomous traffic with virtual personalities. Presence 11 (6), 578–590.