Reliability Engineering and System Safety 174 (2018) 29–40

Contents lists available at ScienceDirect

Reliability Engineering and System Safety

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

A formal framework for the safe design of the Autonomous Driving supervision

Romain Cuer[a,b,*], Laurent Piétrac[a], Eric Niel[a], Saidou Diallo[b], Nicoleta Minoiu-Enache[b], Christophe Dang-Van-Nhan[b]

[a] Université de Lyon, CNRS, INSA-Lyon, AMPERE, F-69621 Villeurbanne, France
[b] Renault S.A.S., 1 avenue du Golf, 78280 Guyancourt, France

ARTICLE INFO

Keywords:
Autonomous vehicle
Systems engineering
Safety analysis
Requirements analysis
Design systems
Discrete-event dynamic systems
Redundancy control

ABSTRACT

The autonomous vehicle is meant to drive by itself, without any driver intervention (for the levels 4 and 5 of automated driving, according to the National Highway Traffic Safety Administration(NHTSA)). This car includes a new function, called Autonomous Driving (AD) function, in charge of driving the vehicle when it is authorized. This function may be in different states (basically active or inactive), that shall be managed by a sub-function, named supervision. The main focus of this work is to ensure that the supervision of a function, performed by a safety critical embedded automotive control system (controlled systems are not considered), respects functional and safety requirements. Usually two processes are involved in the system design: the systems engineering process and the safety one. The first process defines the functional requirements on the function while the safety one specifies redundant sub-functions (realizing together the function) allowing to ensure a continuous service under failure. Since two different aspects of the system are specified, it is a major challenge to make all requirements consistent, from the outset of the design process. In this paper, a method is precisely proposed to address this issue. A progressive reinforcement of the treated requirements is achieved by means of formal state models. In fact, the proposed approach permits to build state models from requirements initially expressed in natural language. Potential ambiguities, incompletenesses or undertones in requirements are in this way gradually deleted. The enrichment of conventional formal verification of control properties with safety requirements constitutes the main originality of the deployed method and contributes to solve inconsistencies between functional and safety verification processes. In addition, the application of the method to the design of AD function supervision highlights its efficiency in an industrial context.

1. Introduction

The autonomous vehicle causes a break in the automotive embedded systems design mainly because it is no more possible to count on the driver reaction in order to keep always the vehicle safe. One of the main arguments in favor of autonomous driving is actually the potential huge reduction of crashes, precisely by eliminating common drivers mistakes [1,2]. This paradigm change will deeply impact the design process. But the autonomous vehicle design must also take into account the constraints of the existing and be built on the know-how, considering the high time-to-market pressure [3,4]. The autonomous vehicle design can be carried out following the usual automotive engineering process. Indeed, the AD system, in charge of the new function Autonomous Driving, is integrated in a specific type of vehicle, already equipped with several ADAS (Advanced Driver Assistance Systems such as Adaptive Cruise Controller or Automatic Parking). This approach is then consistent with the introduction of other ADAS functions. However, if these systems were already safety critical [5,6], the challenge is higher for the AD system because the driver is no more the ultimate safety barrier [7]. In addition, prove the AD system safety only by validation tests appears almost impossible [8]. It is consequently crucial to ensure safe design of the AD system. Moreover, parallel processes of systems engineering and safety are difficult to integrate, given that the differences in terms of planning, constraints, objectives and work teams, as recently highlighted in [9]. Taofifenua [10] also emphases this issue and illustrates it in Fig. 1.

On related fields, like aerospace and railways, this topic is also central. Specific methods (such as Safety driven design methodology [11]) and software environment, like SCADE [12], are implemented in

* Corresponding author.
E-mail address: romain.cuer@insa-lyon.fr (R. Cuer).

https://doi.org/10.1016/j.ress.2018.01.014
Received 19 May 2017; Received in revised form 14 November 2017; Accepted 20 January 2018
Available online 2 February 2018
0951-8320/

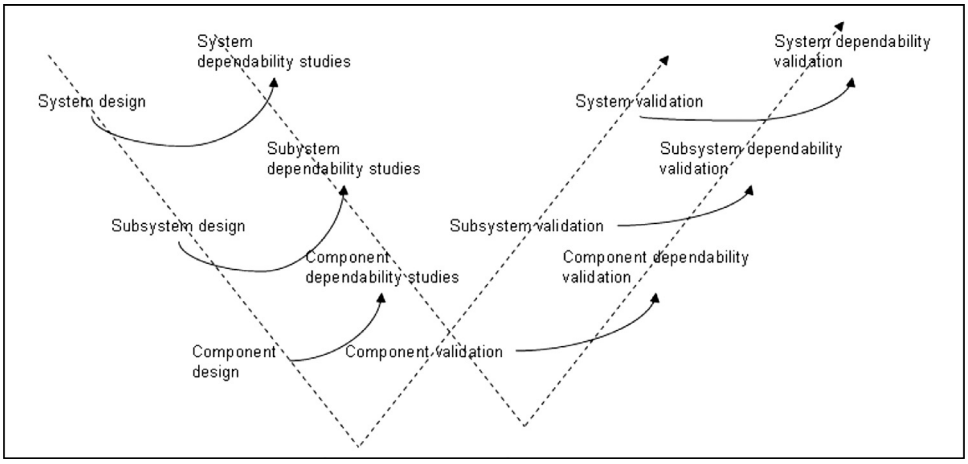

Fig. 1. Integration of safety approach in systems engineering process [10].

aerospace sector. Regarding railways area, the B-method [13,14] has already proven its efficiency. These methods are clearly effective in their application domains, as illustrated by the proven safety of trains and aircrafts. Nevertheless, it should be noted that the particular constraints of automotive field in terms of time-to-market pressure, extremely variable conditions (countries, regulations, driver abilities, climatic conditions…) strength constraints of the existing, costs reduction, limited available space and volume, and organization chart (formal methods are much more common for aeronautics and railways engineers than for automotive ones) [3,4,15] make the adaptation of these methods difficult. Consequently, accurate and adapted means, methods and tools, inspired by this experience, have to be proposed in the context of automotive area. The proposed approach contributes to address this issue. More particularly, it deals with the control system (realizing the designed function): the controlled systems are out of the scope of this study. Special emphasis is focused on verification of deterministic requirements specifying expected behavior in normal conditions on one hand, and safety requirements addressing redundancy and reconfiguration management in case of failures on the other hand.

The paper is structured as follows. In the Section 2, related works are presented and our position is specified. The Section 3 is dedicated to the framework proposed in this paper, that allows safely designing a function performed by an automotive embedded system. Specifically, the method deployed is centered on the behavior of the function: the main aim is to ensure that the intended function remains always in a safe state, whatever happens. This section contains the main contributions: the approach itself, that specifies a formal behavior model, correct by construction, from requirements written in natural language; and the reinforcement of the requirements, notably by highlighting ambiguities, incompleteness (in the sense of incomplete formulation of a requirement itself, not completeness of all requirements), inconsistencies or implicit early in the design process. The contributions of this study are focused on methodological aspects, by improving the current engineering processes, explicitly the design and safety ones. The Section 4 illustrates the interest of the approach by applying it to the AD system design. Lastly, the Section 5 remains the principal conclusions and contributions and outlines future works.

2. Related works

The consideration of the risks analysis at the first step of the system design process is an acknowledged problem for safety critical systems [16] and in particular for the automotive embedded systems [5,6,17,18]. More precisely, the main problem addressed concerns the impacts of safety requirements (requirements provided from safety analyses) on system design and the verification of design compliance with such requirements. It shall also be guaranteed that the system respects an expected behavior in normal conditions, determined by functional requirements. Many works already address this issue.

The most shared way to deal with this topic consists in modeling safety-critical embedded systems in a unique formal, or semi-formal, model [17–23]. It actually eases the merging between system design activities and safety ones, in terms of modeling activities. However, in the context of this work, we more specifically focus on methods aiming at verifying requirements compliance. The three main verification methods implemented [22], both in industrial domains and in the research community, are:

- *The simulation*: this widespread technique [19,24] is based on the symbolic execution of models and the realization of compliance tests corresponding to the users' needs. The symbolic execution requires an operational semantic defining in a deterministic manner the model behavior in reaction to input stimuli. The **main limit** is the completeness of the tested scenarios. The simulation gives only a presumption of correct behavior but is strongly correlated to the expertise and experience of the practitioners;
- *The theorem proving*: the verification is viewed as a theorem to prove from a set of axioms. The program and the properties shall be transformed in mathematical objects. Conclusions are inferred from the description of events or operations allowing to animate the system [25]. The **principal limits** are that the complete automation is rarely possible and the proof preparation requires the determination of elements exceeding the framework of the specifications;
- *The model checking*: automatized technique that, considering a finite state model of a system and a formal property, systematically verifies if this property is valid for this model [26]. This method is divided into three phases: the *modeling* of the intended system, the *execution* of the *model checking* algorithm, and the *analysis* of the results (property satisfied, non satisfied or saturate memory). Different tools implement this method: UPPAAL[1] [22,23,27], UPPAAL-Port[2] [18] or NuSMV[3] [28]. Among the **main limitations** of model checking, one can cite [26] that its applicability is subject to decidability issues. Thus, for infinite-state systems, model checking is, in general, not computable. Besides, it suffers from the combinatorial explosion problem. Finally, it requires expertise in finding appropriate abstractions to obtain smaller system models and to state properties in the logical formalism used.

[1] http://www.uppaal.org/.
[2] http://www.it.uu.se/research/group/darts/uppaal/port/.
[3] http://nusmv.fbk.eu/.

The technique of *model checking* is retained for the deployed methodology. Indeed, the simulation is certainly a widespread and proven practice in the automotive field, but it remains conditional on a trial and error approach. Moreover, the safety of the autonomous vehicle cannot be reasonably proven only by validation tests and simulations [8]. Furthermore, the limits of the *theorem proving* prevent, up to now, its application in an operational automotive context. The proposed approach contributes to reduce the last limitation of the *model checking* mentioned above. Hence, this article aims more specifically at analyzing the problem of the expressiveness of properties to check. Some works tackle the requirements analysis, their decomposition and formalization [29,30]. Nevertheless, the required assumptions to proceed to formalization are not highlighted and the impacts of such assumptions not analyzed. Ghazel et al. [31] proposes a process for progressive formalization of requirements initially expressed in natural language. However, the emphasis is on the refinement of the requirements (translation of gross requirement into a requirement called "formalizable") but the formalization itself is less detailed. The method of *algebraic synthesis* [32] permits to formalize (in Boolean expressions) requirements expressed in natural language, while pruning their potential inconsistencies. Nonetheless, the Boolean expressions employed hardly account for temporal notions. Lastly, the Supervisory Control Theory (SCT) [33] allows building formal models of supervisors, correct by construction, from a description of the uncontrolled system behavior (without constraints) and the specifications that it shall respect. But, its application as part of this study is made difficult by the absence of appropriated description of the uncontrolled system behavior. In addition, we do not attempt to generate supervisor. Besides, the formalization of requirements, generally initially informal, remains a challenging issue [34].

This state of the art shows that there is a lack of approaches progressively improving both control system modeling and requirements that the system has to meet. The method deployed in this article precisely contributes to fill this gap, inspiring from main concepts previously presented. Such as B-method and algebraic synthesis, the requirements are gradually consolidated to eliminate errors, ambiguities, inconsistencies... However, contrarily to those methods, the goal of the proposed process is to provide clear and unambiguous specifications to subcontractors (who realize the designed system) instead of generating executable code. The approach, using formal methods (model checking and principle of automata composition applied in the modeling steps of SCT) and supported by expert advice, allows highlighting errors in requirements formulation at the beginning of the design process. Moreover, the proposed method enriches conventional formal verification of control properties with some properties as given by safety issues. In this way, it originally contributes to solve the problematic of combining functional and safety verification processes. New requirements are formulated by analyzing the formal verification results with different experts. Moreover, the next step of the proposed framework (not included in this paper) will permit to consider implementation constraints.

3. Proposed methodology

3.1. Automotive embedded systems design

The automotive embedded systems design follows the usual (in industry) V-model. It is roughly composed of a design phase, during which the models of the system are built, then the realization is done, after which the integration of the system in the vehicle is undertaken. As part of this study, we focus on the safety requirements, resulting from risks analysis (part of the *subsystem dependability studies* in Fig. 1) and on functional requirements arising from functional analysis (part of the *subsystem design* in Fig. 1). These two elements constitute the input data of this work. Risks analysis is made in the context of the safety process, which is different from the systems engineering one (Fig. 1). In fact, the main objective of the systems engineering process is to design as soon as possible and as cost-effective as possible the system intended.

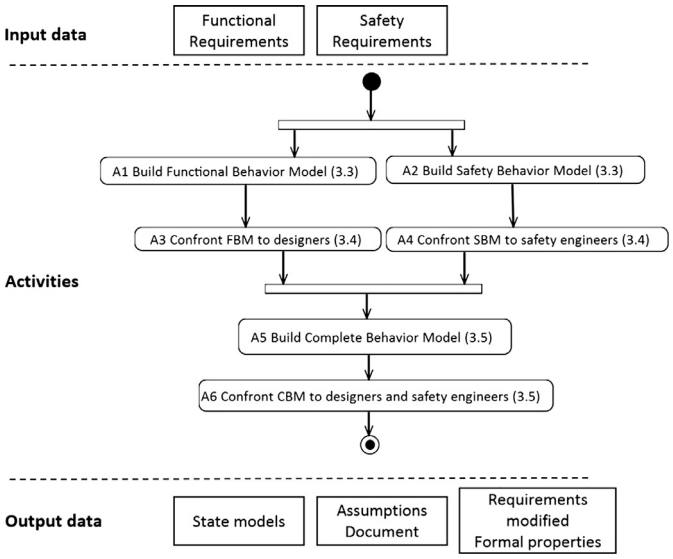

Fig. 2. Approach deployed.

Regarding the safety process, it aims at guaranteeing the system safety under any circumstances. As outlined in the literature [5,10,16,17], the planning of the two processes is difficult to synchronize. The following paragraphs precisely present a methodology facilitating these processes integration. This method is applicable to the supervision of a generic function performed by a safety-critical automotive embedded system. The role of the supervision is to manage the states of the designed function, for safety reasons. One verifies, in this study, if the behavior model of the supervision respects its specifications. So, modeling activities, and no supervisor synthesis, are undertaken.

3.2. Approach overview

Fig. 2 gives an overview of the method adopted. This figure shows that two input data launch the process: the functional requirements, and the safety requirements, both related to the function safely intended. Then, two activities are led in parallel (A1 and A2): building of the Functional Behavior Model (FBM) and building of the Safety Behavior Model (SBM). These models are conditioned from the associated assumptions, needed to construct the state models from requirements, and listed in the *Assumptions Document* (see Section 3.3). To achieve these first activities, the tool UPPAAL is used. UPPAAL is an integrated tool environment for modeling, validation and verification of real-time systems, modeled as networks of timed automata. As explained in [35], UPPAAL consists of three main parts: a description language, a simulator and a model checker. In accordance with this work, only the description language and the model checker will be used. Three reasons drive this choice of tool. The first one is that the behavior of systems communicating with each other (and particularly their synchronization) can be easily modeled, that precisely fit to the studied system. Moreover, the graphical view of state models are common for engineers. Secondly, it is possible in UPPAAL to express formal properties in a textual form (in the tab named *Verifier*). Those properties are directly inferred from requirements analyzed, that considerably eases the traceability between the design (state models on which properties are checked) and the requirements. In addition, UPPAAL can provide a formal proof of properties compliance. The last reason concerns the temporal possibilities offered by UPPAAL that could be utilized when further implementation details will be known. Besides its functionalities and an easy handling, UPPAAL is also a largely well-known tool, both in academic field [36,37] and in a more industrial context [38–40].

The two following activities (A3 and A4) consist in confronting the models obtained to the experts. Since two aspects of the function were

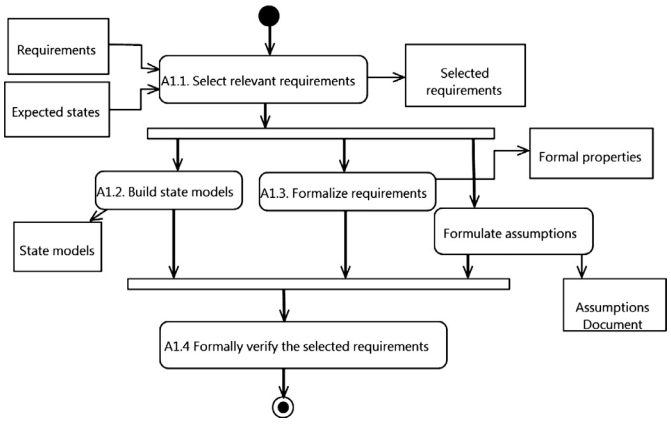

Fig. 3. Details on models construction.

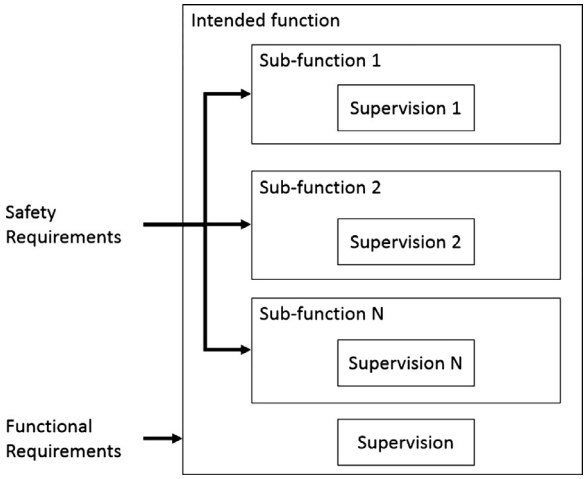

Fig. 4. Requirements allocation.

studied, two activities are actually carried out. On the one hand, the FBM is confronted to the system designers (A3). On the other hand, the SBM is analyzed with safety engineers (A4). It can be then stated that resulting discrete state models (named FBM_1 and SBM_1) actually correspond to the expected behavior of the function supervision (see Section 3.4). This stage is crucial because it shows a main interest of the approach. Indeed, two assumptions types are made in the precedent activities: modeling assumptions, that remain hypotheses, and interpretation assumptions leading to requirements modification and even new requirements formulation (details in Section 3.3). All these assumptions are submitted to the experts knowledge. A new set of requirements is then issued and stored in the document *Requirements modified* (see Fig. 2). These requirements are complete, unambiguous, consistent, clear and verifiable (usually required qualities for requirement [41–43]). They are accompanied by associated modeling assumptions as well as formal properties inferred.

Nevertheless, one cannot verify the safety requirements on the FBM_1 and the functional requirements on SBM_1. Furthermore, the safety requirements address redundant sub-functions while the functional requirements are allocated to the whole function intended. Since the abstraction levels are different, it is required to build a Complete Behavior Model (CBM) on which all requirements are verifiable (see Section 3.5). This is precisely the aim of the last activities (A5 and A6 in Fig. 2) of the approach. Given that the activity A5 relies on the operation of parallel composition of automata (see Section 3.5), the tool Supremica is used. This tool is an integrated environment for verification, synthesis and simulation of discrete event systems [44]. As UPPAAL, Supremica is also an acknowledge tool [45–47]. Lastly, as previously, the resulting state model is conditioned from the assumptions and leads to modify requirements or determine new requirements (activity A6).

3.3. State models construction

Fig. 3 shows the details of the activities A1 and A2 (these two activities are done similarly, sub-activities of A2 are named A2.1, A2.2, A2.3 and A2.4) of the whole approach depicted in the Fig. 2. These activities structure this paragraph. The bold arrows represent control flow whereas the other arrows correspond to data flow.

Fig. 4 illustrates how the intended function is structured: it is globally composed of N redundant sub-functions (for safety and availability reasons) that together perform the function. Each sub-function includes, among other functions, a functional block in charge of managing the states of the sub-function, called *Supervision i* (for the i^{th} sub-function). The designed function also contains a sub-function *Supervision* that handles the states of the whole intended function. The architecture depicted in Fig. 4 is a statical view of the designed function structure. It constitutes an input data and shows how considered requirements are allocated to the statical functional architecture. It does not show however the precise redundancy policy and reconfiguration management, that are specified by certain safety requirements (see Section 4.5). The *Requirements* are composed of the functional requirements and the safety ones, both expressed in natural language. The functional requirements define the behavior of the global function studied, from a user's perspective. They determine actions required in the different states of the function as well as the conditions to commute from one state to another. As shown in the Fig. 4, all the functional requirements are allocated to the developed function. The safety requirements result from the risks analysis. The fundamental difference face to the functional requirements is that the safety requirements address the N redundant sub-functions that together perform the whole function (whose nominal behavior is determined by functional requirements).

The second available input data consist in two lists of states (*Expected states* in Fig. 3): one from the functional perspective and another one regarding the safety viewpoint.

To proceed to the selection of the relevant requirements (A1.1 in Fig. 3), one **criterion**, directly inferred from the main objective of this study (ensure that the intended function remains always in a safe state, see Section 1), is defined. This criterion stipulates:

The **requirement content** shall be relative to a **state change**.

It can be determined from the list of *Expected states*. The criterion application actually corresponds to the selection of the requirements allocated to the *Supervision*, at sub-function level and at global function level (see Fig. 4). For instance, some functional requirements might specify particular actions required in a given state. Thus this type of requirements is rejected. In fact, it is assumed that the actuators control functions are inherently consistent. So, when one (and only one) sub-function is in a state in which it shall command actuators, no contradictory commands can be computed. On the contrary, some risks appear when two sub-functions simultaneously command actuators, because the commands may be contradictory and lead to global instability. One aim of this study is to avoid this type of situations, by ensuring that only one sub-function really controls actuators. Otherwise, since risks analysis starts from the undesirable events, safety requirements can address other aspects (such as acquisition of data, data processing or diagnosis) than the supervision of the sub-function. These requirements are not retained too.

From the selected requirements and by formulating additional assumptions, the state models can be built, by means of UPPAAL (A1.2 in the Fig. 3). Two types of assumptions are made:

1. *Interpretation assumption*: it is an assumption taken to complete a selected requirement. A *complete requirement* is a requirement defining one or several initial state(s), one or several condition(s) and one final state. The automata built in the context of this study are *de-*

terministic because a unique decision shall be taken. It is a question of command models (aiming to serve as specifications) and not of analysis models, where all situations are studied. To transform the incomplete requirements, a rule is in addition stated: the final state shall be different from the initial one. Then an incomplete requirement may lead to several (complete) requirements (precise number depending on the information contained in the requirement), conditioned from their interpretation assumptions. For each interpretation, a state model might be built. One design (possibly with some variants) will finally be chosen (during activities A3 and A4) to carry out the design process.

2. *Modeling assumption*: it is an assumption on the overall framework of the study. It is also related to the way to practically build the state models. In any case, a modeling assumption remains an hypothesis, even after design choices.

UPPAAL is the tool chosen to build state models. It shall be precised how this tool is used as part of this study. Formally, the tool UPPAAL manipulates *Temporized automata*, that is to say tuples (L, l_0, C, A, E, I) where L is the set of locations, $l_0 \in L$ the initial location, C the set of clocks, A the set of actions, co-actions and internal actions, E the set of edges between the locations with an action, a guard and a set of clocks to be reset, and I assigns invariants to the locations [48]. As part of the proposed approach, the temporized aspect and the invariants are not considered. Thus, the built models are *Finite state automata* in the form (L, l_0, A, E) where edges E may just be provided from an action and a guard. Besides, we will thereafter only use the term “state”, understood as “location” if a UPPAAL automaton is considered. Along with the state models construction, it is necessary to formalize the selected requirements (A1.3 in Fig. 3). To that end, the language adopted is the *Computation Tree Logic (CTL)* because its simplified version constitutes the input language of UPPAAL and it is a commonly used language in the context of formal methods [13,29,49]. The chosen structure is the following:

AG (a imply b)

where **a, b** are propositions, **A** is the *All* operator (all future possible paths) and **G** is the operator *Globally* meaning *always in the future*. So, this statement signifies: in a given state, for all possible paths and for all future states in each possible path, if **a** becomes true, then **b** becomes true too. In the context of this study, **a** is a condition for a change of state and (possibly) a state, while **b** is a state. To express the condition **a** contained in a requirement written in natural language, some *variables* and *parameters* are extracted. The *variables* represent the evolution of measurable data, such as velocity or acceleration, while the *parameters* are special values of the variables. These data may be defined in other documents (than *Requirements Document*) or have to be determined, to be further discussed with experts. Up to now, the simplest formula has been chosen. Other operators, allowing to express more advanced temporal notions, can be used at a latter stage. For instance, this will enable to precise the maximum time between the fulfillment of the condition and the effective state change, as illustrated the following formula:

AG (a imply (AX b))

It means that the state **b** is reached exactly at the next state (or after one clock tick if the modeling is temporized [49], operator **X** standing for “next”) once the condition **a** is verified. These operators types can particularly be employed when more implementation details will be available. It may also give a criterion of comparison between different possible implementations of the designed function.

All requirements are translated in this way and formal expressions for all transitions are determined, then verified thanks to UPPAAL. This permits to proceed to the last activity of the *state models construction* (A1.4 of the Fig. 3).

Finally, two state models with their associated assumptions, listed in the *Assumptions Document* (interpretation assumptions and modeling assumptions) are obtained. This concludes the first two activities (A1 and A2) of the whole deployed method (Fig. 2).

3.4. Experts competence confrontation

The taken assumptions must be confronted to the experts. This is the purpose of the activities A3 and A4 of the approach (Fig. 2). Some inconsistencies and weaknesses in the requirements can already be handled (so at a very early stage of the design process). It is a question of a confrontation between the formal behavior models, done by modeling experts (building the behavior models), and the system experts (designers and safety engineers) knowledge. The modeling and interpretation assumptions are reviewed, during working sessions, involving both modeling experts and system experts that have provided analyzed requirements. These activities result in two models: FBM_1 and SBM_1.

Each incomplete requirement shall be treated. Indeed, these requirements may be interpreted in several ways. However, one interpretation has to be chosen for each requirement. In this way, modeling experts submit one main design and some variants to system experts. Several possible simulations deduced from the initial requirements are presented. Because these simulations are limited to only one state change, it is not relevant to use UPPAAL abilities to undertake this task (the simulations are in fact direct). After that, one interpretation (corresponding to a given simulation) of each requirement can be chosen by systems experts. The initial requirements are then reformulated, according to the selected interpretation, and might be source of new requirement(s) too. Regarding modeling assumptions, they are actually validated before performing the study.

Besides, the graphical view and the relevant aggregation of requirements, initially expressed in massive documents, dramatically ease the error detection in requirements, without loss of rigor.

The confrontation with experts allows underlining an added value of the proposed method. Additional assumptions required to build correct state models lead to modify the requirements, and to clearly formulate some taken hypotheses (often corresponding to undertones). So far, there are no automatized (but semi-automatized) links between requirements, inferred formal properties and state models. A work prospect is precisely to create these links allowing to automatically update the specifications according to both functional and safety viewpoints. This might significantly facilitate the two processes integration.

In conclusion of these activities, two correct state models are obtained but one cannot verify functional requirements on SBM_1 and safety requirements on FBM_1. In fact, the functional viewpoint does not consider any failure occurrence and addresses the whole function while the safety requirements are allocated to sub-functions and focus on failure events. Given that the two requirements types concern the same function, one must ensure that the function supervision effectively verifies all selected requirements. For this purpose, it is needed to construct a behavior model on which all selected requirements can be verified. This is the aim of the so-called Complete Behavior Model (CBM).

3.5. Complete behavior model development

The two state models FBM_1 and SBM_1 specify the supervision of the same function upon two different viewpoints. The correctness of these two models with regard to the selected requirements has been proved by formal verification on the one hand, and thanks to the experts assessment, on the other hand. To make the two points of view consistent, we propose an approach based on automata composition. That is why the tool Supremica is used to achieve the activity A5 (Fig. 2) of the proposed method. As for UPPAAL, the use of Supremica in the context of this work shall be precised. The objects manipulated by this tool are *Finite state automata* defined by the tuples (Q, q_i, Q_x, Q_m, Σ, δ) where Q is the finite set of states, q_i the initial state, Q_x the set of forbidden states, Q_m the set of marked states, Σ the alphabet, finite set of events, and δ the transition function defined such that $\delta(q_k, \sigma) = q_l$ where $\sigma \in \Sigma$ [44]. The notions of *marked states* and *forbidden states* are not used in this study. This means that the tuples (Q, q_i, Σ, δ) are manipulated. As part of this study the same types of objects, than those previously presented in

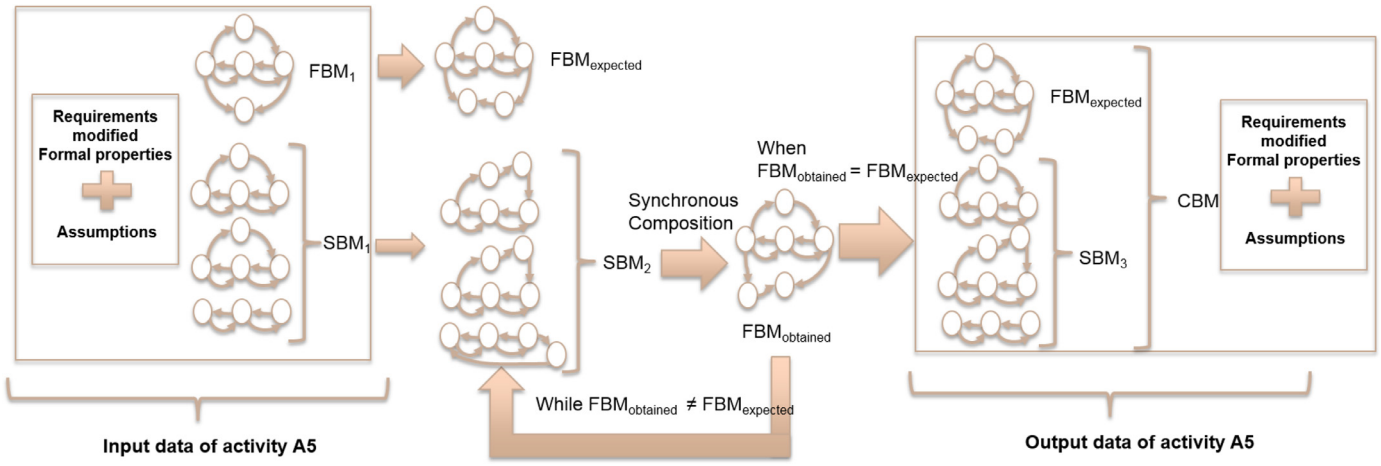

Fig. 5. Graphical overview of the activity A5.

UPPAAL context, are manipulated, with L and Q the set of locations (UPPAAL) and states (Supremica), l_0 and q_i the initial location (UPPAAL) and state (Supremica), A and Σ the set of actions (UPPAAL) and events (Supremica), E and δ the set of edges (UPPAAL) and transition functions (Supremica). The difficulties of translation of an automaton described in the tool UPPAAL to Supremica, raised in [50], are not encountered here. In fact, as underlined in [50], an event in Supremica is possible only if it is allowed in each synchronized automaton. However, in UPPAAL, two types of communication are possible. The communication via *binary channels* involving two automata and ensuring that the event is executed only if both automata allow it. *Broadcast channels* implement the other type of communication. It involves several (more than two) automata. The shared event can occur even if all automata are not in a location allowing it. Given that only communication between two automata (involving binary channels in UPPAAL environment) are considered in this study, the translation does not cause particular problems. Nevertheless, if broadcast channels had been considered, the translation would be less straightforward.

As shown in Fig. 5, the basic idea is to determine the whole function supervision in case of a sub-function failure through the state model called $FBM_{expected}$ (**first step**). In practice, starting from the global function supervision, described by FBM_1, the reaction in case of a sub-function failure is determined (so at the global function level). $FBM_{expected}$ corresponds then to FBM_1 enriched by the specification of behavior in case of any sub-function failure. Then the automata of SBM_1 are iteratively modified (automata of SBM_2 in Fig. 5) until their parallel composition (named $FBM_{obtained}$) actually corresponds to $FBM_{expected}$ (**second step**). Eventually, a Complete Behavior Model (CBM) can be determined. It is composed of $FBM_{expected}$ and SBM_3, as well as the corresponding requirements, formal properties and assumptions taken.

More precisely, the **first step** consists in determining $FBM_{expected} = (Q_{ex}, q_{iex}, \Sigma_{ex}, \delta_{ex})$ from $FBM_1 = (Q_{f1}, q_{if1}, \Sigma_{f1}, \delta_{f1})$ in such a way that $Q_{ex} = Q_{f1} \cup Q_{safe}, q_{iex} = q_{if1}, \Sigma_{ex} = \Sigma_{f1} \cup \Sigma_{fail}, \delta_{ex}(q_{f1}, \sigma_{fail}) = q_{safe}$ where Q_{safe} is the set of safe states, Σ_{fail} the set of foreseen failures, $q_{safe} \in Q_{safe}$ and $\sigma_{fail} \in \Sigma_{fail}$. This activity is done jointly with designers and safety engineers in order to define together Q_{safe}, Σ_{fail} and δ_{ex}.

The **second step** is the iterative modification of the automata of SBM_1 until the resulting automaton of their *synchronized product* is identical to $FBM_{expected}$ defined in the precedent step. It means that the concurrent behaviors of the N sub-functions (see Fig. 4) is then compliant with the global expected behavior. Specifically, we have $SBM_1 = \{A^1_{s1}, A^2_{s1}, \cdots, A^N_{s1}\}$, where $A^j_{s1} (j \in [1, N])$ is the *automaton* describing the behavior of the ith sub-function supervision, and $FBM_{expected}$; and one searches $SBM_3 = \{A^1_{s3}, A^2_{s3}, \cdots, A^N_{s3}\}$, such that $\mathbf{FBM_{expected}} = A^1_{s3} \| A^2_{s3} \| \cdots \| A^N_{s3}$ (where $\|$ is the symbol of the *synchronized product*).

The states of the automata A^j_{s3} (j $\in$ [1,N]) are defined such that $q_{ex} = (q^1_{s3}, q^2_{s3}, \cdots, q^N_{s3})$. As the determination of $FBM_{expected}$, it is achieved in collaboration with designers and safety engineers. Then the transitions of the automata of SBM_1 are modified until the obtention of the desired equality ($FBM_{expected} = A^1_{s3} \| A^2_{s3} \| \cdots \| A^N_{s3}$). This **second step** seeks make the vocabulary between functional view and safety one consistent and ensure, through the definitions of the states q_{ex}, that no potentially dangerous global states (such as mentioned in Section 3.3) may be reached. Currently, this step remains a modeling process led in collaboration with system designers and safety engineers. Its aim is to reach the adequacy between the perceptions of the engineering and the safety. In order to improve the consistency of this process, this step is integrated in a systems engineering approach.

As the precedent activities of the approach, new assumptions are issued. During the activity A6 (see Fig. 2), these assumptions are then confronted with the experts in the same way that for the activities A3 and A4. As previously, some requirements are reformulated and new requirements are defined.

To illustrate the interest of the proposed method, the next section describes its application in an industrial context.

4. Case study

4.1. Autonomous driving function

Most of current vehicles are equipped with many Electric/Electronic (E/E) components, such as: sensors, actuators, Electronic Control Units (ECUs), harnesses wires... The E/E equipment implements some functions, organized in a functional architecture. As highlighted in [51], the main advantage of the functional architecture is to be stable and independent from physical implementation (one functional architecture could be the common root for a wide variety of implementations). The functional architecture of autonomous vehicle includes the AD function in the way depicted in Fig. 6. According to the data (relative to the environmental conditions, the driver behavior and the vehicle state) sent by the blocks *Localization, Perception* and *Enabling systems*, the AD function computes continuously a trajectory command that is transmitted to the actuators (block *Acting*). The perception functionalities and the trajectory calculation constitute significant and active search fields [52–55]. Moreover, the AD function can be in different states (*active, available*... more details on these states are given in the paragraphs 4.4, 4.5). The functional block *AD management level* rightly manages the states of the AD function and, more specifically, the sub-block *Supervision* has to permanently determine in which state the AD function must be. The supervision of these states is at the core of this work. In particular, in a specific state of the AD function, called MRM, standing for Minimal Risk

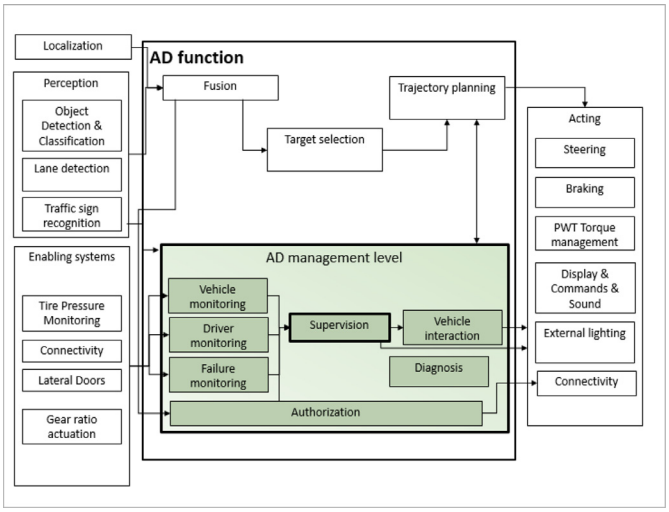

Fig. 6. Simplified functional architecture of an autonomous vehicle.

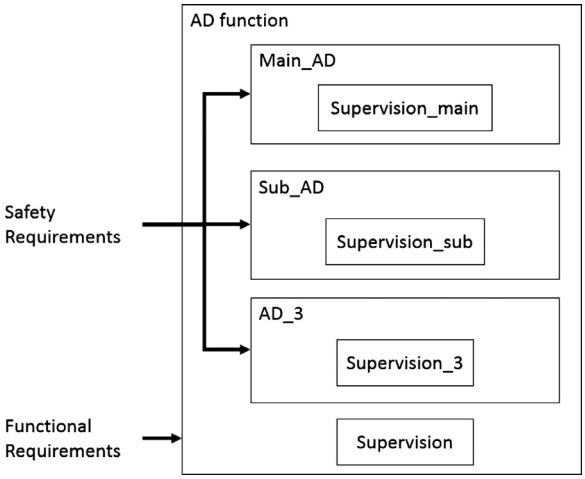

Fig. 7. Simplified architecture of AD function.

Maneuver, the vehicle carries out different predetermined maneuvers to keep the vehicle safe under any circumstances. The safety requirements precisely specify this aspect. It highlights the crucial significance of the consideration of risks analysis results at the first step of the AD system design.

4.2. Description of the input data

As recently underlined in [7], it is particularly challenging to clearly define requirements on AD function, given that the multiple areas involved. In the context of this study, it has been stated that the two requirements types studied are defined in the way described in 3.3.

The functional requirements define then conditions to commute from one state to another that mainly depend on environment evolutions, vehicle global state and driver behavior. They also determine specific actions to perform in different states. Functional requirements are about 175. Here follow two examples:

FR1: "AD function shall not be available within a construction area."
FR2: "If AD function is active, then the HMI shall always give information of the autonomous vehicle status"

For this application, the safety requirements are allocated to three redundant sub-functions (called main_AD, sub_AD and AD-3) that together perform the global AD function and are structured according to the architecture depicted in the Fig. 4, as illustrates the Fig. 7. The safety requirements specify redundancy policy and reconfiguration management. More particularly, they determine safety barriers that shall be implemented by the blocks of the functional architecture (Fig. 6). About 350 safety requirements have been treated. Below are two examples of such requirements:

SR1: "In case of an excessive deceleration due to a failure of sub_AD, AD-3 shall switch itself on"
SR2: "SPF (Single Point Fault) Metric of AD-3 shall comply with target value 99 percent"

The expected states for the two points of view, determined after discussions with AD system designers and safety engineers, are the following:

- The states of the AD function from the functional perspective: *Off, Not_available, Available, Activatable* and *Active*;
- The states of the three sub-functions regarding the safety viewpoint: main_AD: *Off, Active, MRM*, sub_AD: *Off, Standby, On* and AD-3: *Standby, On*.

4.3. Selection of relevant requirements (A1.1 and A2.1)

The **FR1** and the **SR1** are chosen while the **FR2** and the **SR2** are rejected. In fact, **FR1** gives a condition to enter in the state *Not_available* while **SR1** also precises a condition to switch to the state *ON*. So they both respect the defined criterion (in Section 3.3). Conversely, **FR2** indicates what the AD function must do when it is in the state *Active*, so no state change is mentioned and the criterion is not respected. **SR2** concerns the reliability of the component performing AD-3. More precisely, the *Single Point Fault Metric* is a notion of the ISO-26262 standard [56] reflecting the robustness of a system to single point faults (faults that directly cause the violation of a vehicle level safety requirement). This metric quantifies, by means of failure rates ratio, the coverage of the single point fault. Hence it is not relative to a state change too.

Finally, about 20 percent of the initial safety requirements (73) and 40 percent of the initial functional requirements (70) were selected.

4.4. Functional Behavior Model construction (A1.2)

After selecting relevant functional requirements, the construction of the Functional Behavior Model may be undertaken. To do this, some assumptions have to be made. For example, **FR1** specifies that the AD function shall enter in the state *Not_available* if the vehicle stands in a construction area (understood as zone of roadworks). However, one cannot determine if the initial state is *Off, Available, Activatable* or *Active*. That is why 15 possibilities, that could be gathered in four cases, have been considered:

1. the initial state is one of the four states (4 possibilities). It means that 4 state models may be built: in the first model, the initial state is *Off*, in the second one, it is *Available*, in the third one, *Activatable* and in the last one, *Active*;
2. the initial states are two of the four states (6 possibilities);
3. the initial states are three of the four states (4 possibilities);
4. the initial states are the four states (1 possibility).

This constitutes an *interpretation assumption* that is reported in the *Assumptions Document* (Fig. 3). **FR1** actually belongs to a group of requirements that define conditions to enter in the state *Not_available*. The interpretation assumption concerns in reality all the requirements of this group. In the same way, 4 other groups of requirements determining conditions to enter or to exit the states of AD function have been created. Each group conducts to interpretation assumptions, so to different possible versions of the state model. The interpretation assumptions for the 4 other groups respectively lead to 3, 4, 15 and 46 possibilities. This potentially corresponds to a huge number of possibilities (so state

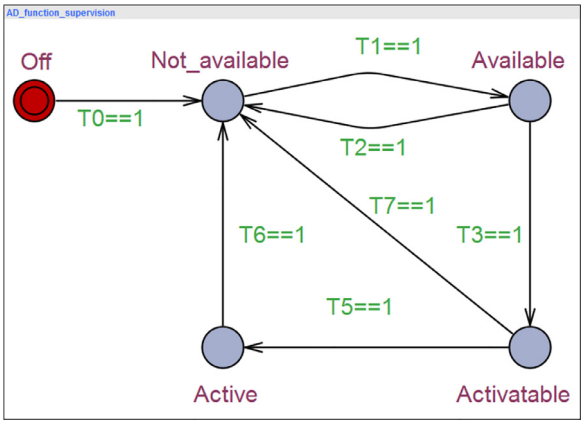

Fig. 8. Functional Behavior Model (FBM).

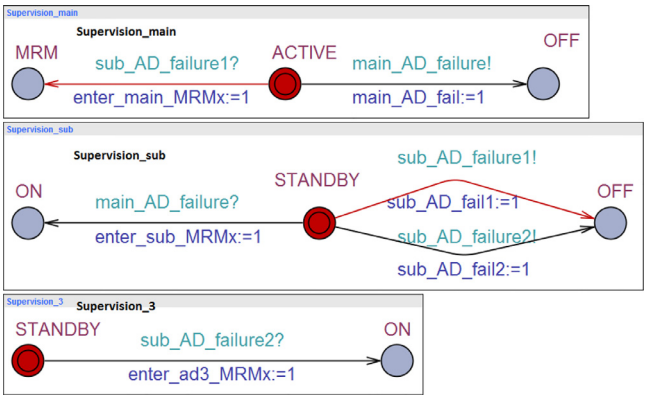

Fig. 9. Safety Behavior Model (SBM).

model versions), but, similarly to the **FR1** group, the possibilities can be gathered in cases, easing the forthcoming confrontation work.

The Fig. 8 represents a version of the FBM. The transitions *Ti* (i=0 to 7) contain the different conditions to commute from one state to another, defined in the selected functional requirements. The state *Active* is the only state perceptible by the driver. Indeed, in this state the driving is completely delegated to the AD system and no driver intervention is required, whatever the situation. The other states correspond to the initialization phase, they are intermediate states permitting to propose the AD function to the driver only when required conditions are met. It has been stated that the transitions to enter in a "more active" state (*Tj* with j={0, 1, 3, 5}) are defined in this way: *Tj* is true if all conditions of its guard are true (**AND** logical) while the transitions to switch to a "less active" state (*Tk* with k={2, 6, 7}) are determined as follows: *Tk* is true if only one condition of its guard is true (**OR** logical). This is a *modeling assumption* written in the *Assumptions Document* (Fig. 3).

Otherwise, the four cases above-mentioned (interpretation assumption of **FR1**) are taken into account in the transitions content:

1. The guard of each transition that leads to the state *Not_available* (T0, T2, T6, T7) includes the condition about construction area;
2. The guards of the transitions between two of four states (all states except *Not_available*) and the state *Not_available* contain the condition about construction area.

The same applies for the two last cases. So the 15 versions of FBM, corresponding to the 15 possible interpretations of **FR1** described earlier, have practically (*T7* is not always present) the same graphical aspect. In fact, only the contents of the guards of the transitions leading to the state *Not_available* are modified according to each interpretation.

4.5. Safety Behavior Model construction (A2.2)

The same approach is adopted to build the Safety Behavior Model. An aggregated version of the SBM obtained is represented in the Fig. 9. It is called *aggregated* because all elements are not depicted to improve readability. For instance, many consequences of a main_AD failure are actually considered but they can be all gathered in the generic event *main_AD_failure!* because they lead to the state *Off* of the *Supervision_main*. SBM is composed of three automata describing the behavior under failure of the three sub-functions performing the global AD function. The safety point of view focuses on the sub-functions reaction in case of one sub-function failure. The redundancy policy specified is a cold redundancy. In fact, the sub-function main_AD is normally always active and the sub-function sub_AD intervenes only in case of main_AD loss. If the sub_AD is lost, either main_AD takes action or AD-3 intervenes, according to the failure type (*sub_AD_failure1* or *sub_AD_failure2*) causing the sub_AD loss. With regard to the safety viewpoint, only modeling assumptions have to be made. Effectively, observing parameters have to be introduced to enable modeling. For example, the boolean parameter *enter_main_MRMx* indicates if the sub-function main_AD is entered in the *MRM* state by activating a Minimal Risk Maneuver (MRM) *x* (there are actually several MRM according to the failure types). The reason to add these observing parameters is to enable the properties verification in UPPAAL. Indeed, in the query language, it is not possible to include directly the channels in formal properties. So to verify if the event *main_AD_failure!* has occurred, one checks the value of *main_AD_fail*.

4.6. Formalization of requirements (A1.3 and A2.3)

Concerning the functional requirements, the translation is not immediate. Take the example of **FR1** ("AD function shall not be available within a construction area."). To translate this requirement, we choose to define a boolean variable *construction_area* in this way: *construction_area==0* means that the vehicle is not in a construction area while *construction_area==1* signifies otherwise. As explained above, different interpretations of **FR1** have been proposed. Consequently, **FR1** could be translated in different ways, according to the possibilities previously presented. For example, if we choose that there is one initial state and this state is *Available*, we obtain the following statement (**FP** stands for Functional Property):

FP1-1: AG((construction_area==1 && AD_function.Available) imply (AD_function.Not_Available))

The index **1-1** indicates that **FP1-1** is the interpretation **1** of the initial **FR1** (there are actually 14 others).

For the safety requirements, the translation is more straightforward. In fact, all the selected requirements are formulated in the same way of the example requirement **SR1** ("In case of an excessive deceleration due to a failure of sub AD, AD-3 shall switch itself on"). This statement is then translated like this (**SP1** standing for Safety Property 1):

SP1: AG(sub_excess_decel2==1 imply AD-3.ON)

sub_excess_decel2 is here a particular case of *sub_AD_fail2* of the SBM (see Fig. 9). This requirement can be formulated more generally in this way:

AG(sub_AD_fail2==1 imply AD-3.ON)

All the selected requirements have been translated in this manner and formally verified in UPPAAL. This ensures the correctness of the state models with regard to the selected requirements.

4.7. Confront state models to designers (A3 and A4)

With respect to the FBM, the correct interpretation of each requirement necessitating one has to be chosen. For the requirement example

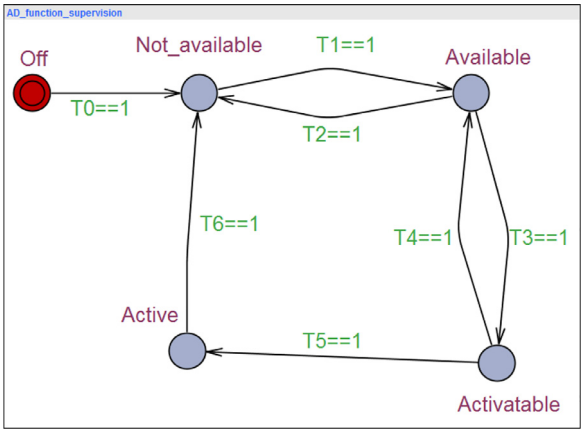

Fig. 10. Functional Behavior Model$_1$ (FBM_1).

(**FR1**), the interpretation 1, earlier described, was the good one, for the reasons given in the next paragraph. Accordingly, a new requirement formulation for initial **FR1**, called **FR1-1**, has to be written:

FR1-1: "If AD function is in the state *Available* and the autonomous vehicle stands in a construction area, then AD function shall switch to the state *Not_available*"

For each interpretation assumption, a choice has to be made. It should be noted that certain choices are very significant for the design. For the taken example (**FR1**), the interpretation adopted does not mean that AD function does nothing when it is in the state *Active* and the vehicle enters in a construction area. Indeed, when it happens, more specific requirements based on more precise environment information indicate the actions required. The information about construction area mentioned in **FR1** just allows avoiding to propose AD function to driver at a wrong time. Moreover, the analysis of **FR1** conducts to the definition of another requirement, named **FR1a**. In fact, if the vehicle enters in a construction area while the AD function is in the state *Activatable*, an action is required and has to be specified. This is precisely the role of the new requirement **FR1a**:

FR1a: "If AD function is in the state *Activatable* and the autonomous vehicle stands in a construction area, then AD function shall switch to the state *Available*"

Given their complexities and their number, all choices have still not been made. Consequently, the modified FBM, named FBM_1, represented in the Fig. 10, is the most recent and stable version of the state model. The FBM_1 contains an additional transition *T4* face to the FBM depicted in Fig. 8. Indeed, **FR1a**, for example, gives a condition to commute from the state *Activatable* to the state *Available*, so the guard of *T4* contains (at least) this condition. Furthermore, the transition *T7* has been finally deleted. This transition corresponded actually to interpretations of **FR1** finally not retained. Each transition has to be defined by choosing the correct interpretation of the initial functional requirements. Otherwise, these initial requirements have been meanwhile modified and completed too.

For the SBM, the Supervision_3 specified by the selected requirements did not actually correspond to the experts expectation. In fact, AD-3 shall intervene in case of simultaneous loss of sub-functions main_AD and sub_AD (due to common cause failure). Yet, it has been specified that AD-3 had to take action in case of particular failures affecting the sub_AD function (see Fig. 9). That is why, some initial requirements have to be deleted, and new requirements have been expressed in order to be compliant with the experts advice. For instance, since **SR1** specified reaction of AD-3 in case of sub_AD loss, a modified requirement, replaced the original **SR1**, and called **SR1a**, has been written:

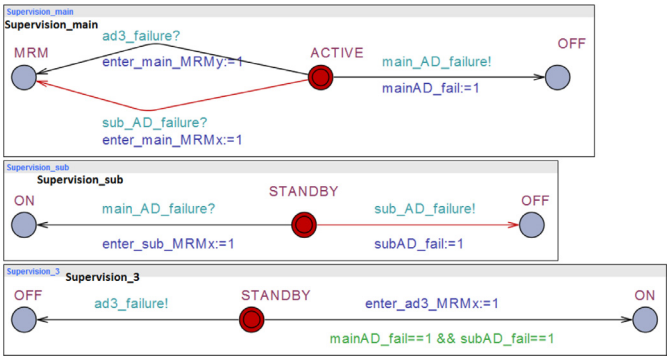

Fig. 11. Safety Behavior Model$_1$ (SBM_1).

SR1a: "In case of an excessive deceleration due to a failure of sub_AD **and a failure of main_AD**, AD-3 shall switch itself on" Fig. 11 illustrates the modified SBM resultant, called SBM_1 (like SBM, it is a question of an aggregated model). In addition, the main_AD function has to enter in the state *MRM* in case of failure of AD-3. This event were actually not considered in SBM (Fig. 9).

Finally, 29 new requirements have been written to specify the AD-3 in case of simultaneous failure affecting main_AD and sub_AD while 5 requirements have been deleted. Indeed, they did not correctly specified the AD-3.

As sated in Section 3.4, the two models FBM_1 and SBM_1 are different in terms of states, transitions, events and abstraction levels. To ensure global consistency and completeness, it is needed to build a Complete Behavior Model, compliant both with functional requirements and with safety requirements.

4.8. Complete Behavior Model construction (A5)

As explained in the description of the activity A5 (see Section 3.5), the **first step** consists in determining expected global AD function supervision (state model named $FBM_{expected}$) in case of foreseen failures occurrence. Since four groups of failures (*main_AD_failure, sub_AD_failure, ad3_failure* and *AD_failure*) have been identified and treated, four global states, called *MRMm, MRMs, MRM3, MRMms* and reached respectively after the events *main_AD_failure, sub_AD_failure, ad3_failure* and *AD_failure*, shall be added to the FBM_1. These global states are, in accordance with the local safety requirements (see Fig. 11), reached only from the state *Active*. The four groups of failures make up Σ_{fail} (defined in Section 3.5) the four global states constitute Q_{safe}, and the events leading to these states correspond to δ_{ex}. The resulting expected state model (called $FBM_{expected}$) is represented in Fig. 12.

Then, to start the **second step**, the states of the automata SBM_3 are determined such as $q_{ex} = (q^1_{s3}, q^2_{s3}, q^3_{s3})$. In fact, the parallel composition of the automata forming SBM_3 gives an automaton whose each state is a configuration of the states of the three local supervision functions. It is then necessary to determine the authorized states configurations, corresponding to the nominal global states. The main idea is to avoid that two control functions are in states in which they can command actuators (see Section 3.3). Effectively, if actuators receive two or more commands, it could lead to vehicle instability. The states concerned are, from the functional point of view: *Active* and, from the safety one: *Active* and *MRM* (broken into four states). This operation is done, also in conjunction with safety engineers and AD system designers, and the result is shown in Table 1.

Once this correspondence is achieved, it is then possible to proceed to local behavior models modifications (automata of SBM_1). This operation is done from two perspectives:

- *Functional point of view*: according to the redundancy policy, the main_AD function is ordinary active and ensures the AD function

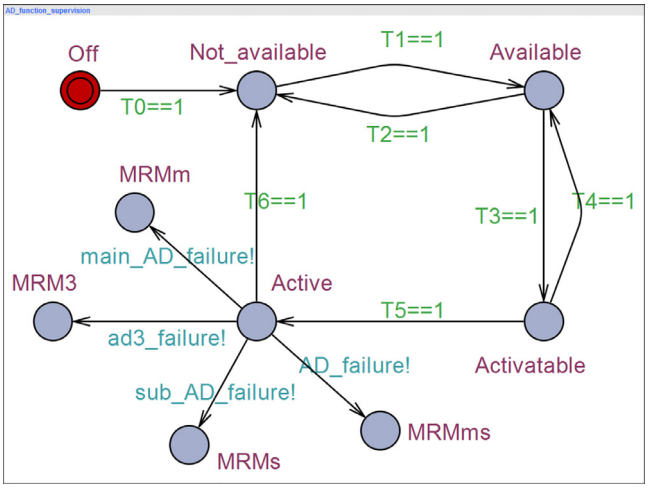

Fig. 12. FBM$_{expected}$.

Table 1
Authorized states configurations.

FBM$_{expected}$	Main_AD	Sub_AD	AD_3
Off	Off	Off	Off
Not_available	Not_available	Not_available	Available
Available	Available	Available	Available
Activatable	Activatable	Activatable	Available
Active	Active	Activatable	Available
MRMs	MRM	Silent	Available
MRM3	MRM	Activatable	Silent
MRMm	Silent	MRM	Available
MRMms	Silent	Silent	MRM

when it is authorized. That is why, the state *Active* shall only appear on the model of the *Supervision_main*. On the contrary, the sub_AD function and the AD-3 functions are not ordinary active and take control uniquely after a failure occurrence. Thus no state *Active* must appear in the *Supervision_sub* model and in the *Supervision_3* one. Besides, all functions can be in an initialization state. That is why the states *Not_available, Available* and *Activatable* replace, by detailing, the state *Standby* of *Supervision_main* and *Supervision_sub*. Since the AD-3 function takes control solely in case of common cause failure affecting both main_AD and sub_AD, it is just required that AD-3 is in the state *Available* when the other functions are active;

- *Safety viewpoint*: each AD sub-function is capable to ensure secured maneuvers in case of treated failures occurrence. Thus, the state *MRM* can actually be reached by each local supervision. For the sake of clarity, the states called *ON* in *Supervision_sub* and *Supervision_3* (see Fig. 11) are renamed *MRM*. Note that the state *MRM* is only reachable from the *Active* state. It means that the reaction of the AD function in case of sub-function failure is, for now, specified only when the whole AD function is active (in accordance with the safety requirements). In the same way, the state named *OFF* is recalled *Silent*, because it is reached after a failure of the sub-function considered. The state *Off* is kept but in the functional sense, *i.e.* it is reached when power supply is switched off.

These considerations help to complete the automata of SBM_1 by giving indications for adjusting the states of the automata. Then, one verifies if the automata modified in this way are correct by performing their parallel composition thanks to Supremica. This result is compared to the *FBM$_{expected}$* depicted in the Fig. 12. When the automaton thus obtained is identical (in the sense defined in the description of the **second step** of activity A5, in Section 3.5), the local automata forming SBM are correct

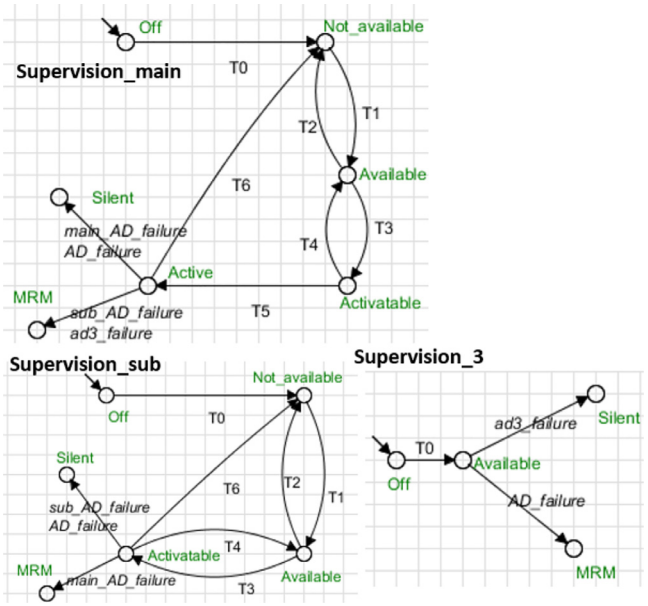

Fig. 13. Safety Behavior Model$_3$ (SBM_3).

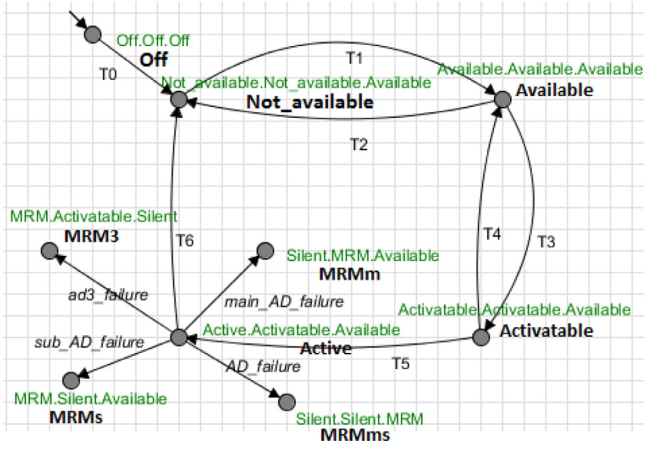

Fig. 14. FBM$_{obtained}$.

with respect to global AD function supervision. The correct automata thus obtained (forming SBM_3) are represented in the Fig. 13. The automaton resulting from their composition is represented in the Fig. 14. The states presented in bold type are the states of the *FBM$_{expected}$* such as defined in the Table 1. This proves that the three state models for the *Supervision_main*, the *Supervision_sub* and the *Supervision_3* are correct with regard to the expected global AD function supervision.

Nonetheless, some assumptions have again been formulated to build the CBM. Certain are realistic and might result in new requirements formulation, such as the states repartition reported in the Table 1. Thus, after confronting the taken assumptions with safety engineers and designers (activity A6 of the method, Fig. 2), about 70 new requirements for the main_AD and 35 for the sub_AD have been formulated. Other hypotheses are however not acceptable, like the consideration of a sub-function loss only when the global AD function is active, or the implicit assumption of perfect transitions synchronization (for example, *T1* corresponds to the same event for main_AD and sub_AD in the Fig. 13). This points out the necessity to take into account certain constraints of the implementation in the proposed modeling. This aspect constitutes a perspective of the works presented in this paper.

5. Conclusions

The introduction of the Autonomous Driving function makes the automotive embedded system in charge of its realization particularly safety critical. An appropriate approach has to be determined in order to avoid all design errors and to take into account as many requirements as possible from the design phase. This paper precisely contributes to this important work and proposes a formal-based method that both improve requirements formulation and system modeling at the beginning of the design process.

The approach proposed has allowed to build from requirements, originally expressed in natural language, a unique and complete formal behavior model, correct by construction. Besides, requirements are consolidated at the outset of the design process. Mainly, two means allow to improve requirements. First, the formalization underlines certain deficiencies, such as requirements formulation incompleteness. Secondly, the clear and readable presentation of the results (in the graphical form) largely facilitates the search for requirements errors. Indeed, it is more convenient to find mistakes in behavior specification when it is represented in the form of state models, without loss of rigor. Lastly, it should be noted that this approach, even it is partially automatized, relies on expert knowledge too.

Generally speaking, the proposed method takes place in the context of the so-called *intrinsic safety*, as defined in [57]. Thus it ensures that all already foreseen events are properly considered: for each possible event, a system reaction is specified. Unplanned events from the design beginning are then out of scope. Otherwise, the method is focused on verification process and requirements formulation, and is not enough integrated in the whole systems engineering process. Moreover, the behaviors of components supporting the function studied are, in this study, not sufficiently rigorously taken into account too.

Consequently, two aspects will then further investigate. On the one hand, the use of modeling language (SysML, EAST-ADL) and tools already available (*Rational Doors,*[4] SysML modeler) to integrate the approach in more general MBSE (Model Based Systems Engineering) context and increase its re-usability will be studied. On the other hand, the formalization of constraints implementations in order to seamlessly consider them in the framework proposed will be addressed.

References

[1] Fagnant DJ, Kockelman K. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. Transp Res Part A 2015;77:167–81.

[2] Anderson JM, Kalra N, Stanley KD, Sorensen P, Samaras C, Oluwatola OA. Autonomous vehicle technology: A Guide for policymakers. RAND Corporation; 2014. 9780833083982.

[3] Taofifenua O, Chale H, Gaudré T, Topa A, Levy N, Boulanger J. Reducing the gap between formal and informal worlds in automotive safety-critical systems. In: 21th annual INCOSE international symposium, Denver, USA; 2011. Presented also at IEEE 5th Annual International System Conference, Montreal, April 2011.

[4] Maurer M, Winner H. Automotive systems engineering. Springer Science & Business Media; 2013. 978-3-642-36455-6.

[5] Kaiser B, Klaas V, Schulz S, Herbst C, Lascych P. Integrating system modelling with safety activities. In: International conference on computer safety, reliability, and security. Springer; 2010. p. 452–65.

[6] Weissnegger R., Pistauer M., Kreiner C., Römer K., Steger C.. A novel method to speed-up the evaluation of cyber-physical systems (ISO 26262). In: 12th International workshop on intelligent solutions in embedded systems, WISES 2015, Ancona, Italy, October 29–30. p. 109–114.

[7] Koopman P, Wagner MD. Autonomous vehicle safety: an interdisciplinary challenge. IEEE Intell Transp Syst Mag 2017;9(1):90–6.

[8] Kalra N, Paddock SM. Driving to safety: how many miles of driving would it take to demonstrate autonomous vehicle reliability? Transp Res Part A 2016;94:182–93.

[9] Mauborgne P, Deniaud S, Levrat E, Bonjour E, Micalli J-P, Loise D. Operational and system hazard analysis in a safe systems requirement engineering process application to automotive industry. Saf Sci 2016;87:256–68.

[10] Taofifenua O. Ontology centric design process : sharing a conceptualization. Conservatoire national des arts et metiers - CNAM; 2012. Phd dissertation.

[11] Owens BD, Herring MS, Dulac N, Leveson NG, Ingham MD, Weiss KA. Application of a safety-driven design methodology to an outer planet exploration mission. In: 2008 IEEE aerospace conference; 2008. p. 1–24.

[12] Märtin L, Schatalov M, Hagner M, Goltz U, Maibaum O. A methodology for model-based development and automated verification of software for aerospace systems. In: 2013 IEEE aerospace conference; 2013. p. 1–19.

[13] Boulanger J-L. Formal methods applied to industrial complex systems: implementation of the b method. 1st. Wiley-IEEE Press; 2014. 1848217099, 9781848217096.

[14] Behm P, Benoit P, Faivre A, Meynadier J-M. Mtor: asuccessful application of B in a large project. Springer Berlin Heidelberg; 1999. p. 369–87.

[15] Broy M. Challenges in automotive software engineering. In: Proceedings of the 28th international conference on software engineering. New York, NY, USA: ACM; 2006.

[16] Leveson NG. An approach to designing safe embedded software. In: International workshop on embedded software. Springer; 2002. p. 15–29.

[17] Chen D, Johansson R, Lnn H, Blom H, Walker M, Papadopoulos Y, et al. Integrated safety and architecture modeling for automotive embedded systems. E & I Elektrotechnik und Informationstechnik 2011;128(6):196–202.

[18] Kang E-Y, Enoiu EP, Marinescu R, Seceleanu C, Schobbens P-Y, Pettersson P. A methodology for formal analysis and verification of EAST-ADL models. Reliab Eng Syst Saf 2013;120:pp.127–138.

[19] Cressent R, Idasiak V, Kratz F, David P. Mastering safety and reliability in a model based process. In: Proceedings - annual reliability and maintainability symposium; 2011. p. pp.1–6.

[20] David P, Idasiak V, Kratz F. Reliability study of complex physical systems using sysml. Reliab Eng Syst Saf 2010;95(4):431–50.

[21] Güdemann M, Ortmeier F. A framework for qualitative and quantitative formal model-based safety analysis. In: 12th IEEE high assurance systems engineering symposium, HASE 2010, San Jose, CA, USA, November 3–4, 2010; 2010. p. 132–41.

[22] Pétin J-F, Evrot D, Morel G, Lamy P. Combining SysML and formal methods for safety requirements verification. In: 22nd international conference on software & systems engineering and their applications, Paris, France; 2010. p. CDROM.

[23] Liu X, Zhu Z. Construct aspectual models from requirement documents for model-driven development of automotive software. Electron Notes Theor Comput Sci 2011;274:pp.33–50.

[24] Nouacer R, Djemal M, Niar S, Mouchard G, Rapin N, Gallois J, et al. EQUITAS: a tool-chain for functional safety and reliability improvement in automotive systems. Microprocess Microsyst - Embedded Hardware Des 2016;47:252–61.

[25] Roussel J-M, Denis B. Safety properties verification of ladder diagram programs. Journal Européen des Systèmes Automatisés (JESA) 2002;36(7):pp.905–917.

[26] Baier C, Katoen J-P. Principles of model checking (representation and mind series). The MIT Press; 2008. 026202649X, 9780262026499.

[27] Apvrille L, Becoulet A. Prototyping an embedded automotive system from its UML/sysml models. Proc Embedded Real Time SystSoftw 2012:pp.87–124.

[28] Sharvia S, Papadopoulos Y. Integrating model checking with HiP-HOPS in model-based safety analysis. Reliab Eng Syst Saf 2015;135:64–80.

[29] Bitsch F. Safety patterns-the key to formal specification of safety requirements. In: SAFECOMP, 2187. Springer; 2001. p. 176–89.

[30] Evrot D, Pétin J-F, Morel G, Lamy P. Using sysml for identification and refinement of machinery safety properties. IFAC Proc 2007;40(6):127–32. 1st IFAC Workshop on Dependable Control of Discrete Systems.

[31] Ghazel M, Yang J, El-Koursi E-M. A pattern-based method for refining and formalizing informal specifications in critical control systems. J Innovation Digital Ecosyst 2015;2(1):32–44.

[32] Roussel J-M, Lesage J-J. Algebraic synthesis of logical controllers despite inconsistencies in specifications. In: 11th international workshop on discrete event systems, WODES 2012, Guadalajara, Mexico; 2012. p. 307–14. 8 pages.

[33] Ramadge PJ, Wonham WM. Supervision of discrete event processes. In: 1982 21st IEEE conference on decision and control; 1982. p. 1228–9.

[34] Zaytoon J, Riera B. Synthesis and implementation of logic controllers a review. Annu Rev Control 2017;43(Supplement C):152–68.

[35] Larsen KG, Pettersson P, Yi W. Uppaal in a nutshell. Int J Softw Tools TechnolTransfer 1997;1:134–52.

[36] Larsson F, Larsen KG, Pettersson P, Yi W. Efficient verification of real-time systems: compact data structures and state-space reduction. In: Proc. of the *18th IEEE real-time systems symposium*. IEEE Computer Society Press; 1997. p. 14–24.

[37] Larsen KG, Pettersson P, Yi W. Compositional and symbolic model-checking of real-time systems. In: Proc. of the 16*th* IEEE real-time systems symposium. IEEE Computer Society Press; 1995. p. 76–87.

[38] Lindahl M, Pettersson P, Yi W. Formal design and analysis of a gear controller. In: International conference on tools and algorithms for the construction and analysis of systems. Springer; 1998. p. 281–97.

[39] David A, Yi W. Modelling and analysis of a commercial field bus protocol. In: Proceedings of the 12th Euromicro conference on real time systems. IEEE Computer Society; 2000. p. 165–72. 0-7695-0734-4.

[40] Hessel A, Pettersson P. Model-based testing of a wap gateway: an industrial case-study. Tech. Rep.; 2006. Technical Report 2006-045, ISSN 1404–3203.

[41] Pohl K, Rupp C. Requirements engineering fundamentals: a study guide for the certified professional for requirements engineering exam - foundation level - IREB compliant. Rocky Nook, Inc.; 2011. 978-1-4571-1192-1.

[42] Holt J, Perry SA, Brownsword M. Model-based requirements engineering. Institution of Engineering and Technology; 2012. 978-1-84919-487-7

[43] of Electrical I., Engineers E.. ISO/IEC/IEEE 29148:2011(E) systems and software engineering life cycle processes requirements engineering. 2011.

[44] Åkesson K, Fabian M, Flordal H, Vahidi A. Supremica – a tool for verification and synthesis of discrete event supervisors. In: Proceedings of the 11th mediterranean conference on control and automation. Rhodos, Greece; 2003.

[45] Markovski J, van de Mortel-Fronczak JM. Modeling for safety in a synthesis-centric systems engineering framework. In: International conference on computer safety, reliability, and security. Springer; 2012. p. 36–49.

[4] http://www-03.ibm.com/software/products/fr/ratidoor.

[46] Mohajerani S, Malik R, Fabian M. Compositional synthesis of supervisors in the form of state machines and state maps. Automatica 2017;76:277–81.
[47] Rohée B, Riera B, Carré-Ménétrier V, Roussel J-M. A methodology to design and check a plant model. In: 3rd IFAC workshop on discrete-event system design (DES-Des'06). Rydzyna, Poland; 2006. p. pp.246–250.
[48] Behrmann G, David A, Larsen K. A tutorial on UPPAAL. In: Lecture Notes in Computer Science. Springer; 2004. p. 200–36. VII/3185.
[49] Bitsch F. Classification of safety requirements for formal verification of software models of industrial automation systems. In: Proceedings of the 13th conference on software and systems engineering and their applications. Citeseer; 2000.
[50] Koolmees B. Reniers M. Markovski J.. Validation of modeled behavior using uppaal, Master's thesis, University of Technology Eindhoven.
[51] Behere S, Törngren M. A functional reference architecture for autonomous driving. Inf Softw Technol 2016;73:136–50.
[52] Do QH, Niknejad HT, Mita S, Egawa M, Muto K, Yoneda K. Human drivers based active-passive model for automated lane change. IEEE Intell Transp Syst Mag 2017;9(1):42–56.
[53] Falcone P, Borrelli F, Asgari J, Tseng HE, Hrovat D. Predictive active steering control for autonomous vehicle systems. IEEE Trans Control Syst Technol 2007;15(3):566–80.
[54] You F, Zhang R, Lie G, Wang H, Wen H, Xu J. Trajectory planning and tracking control for autonomous lane change maneuver based on the cooperative vehicle infrastructure system. Expert Syst Appl 2015;42(14):5932–46.
[55] Mu G, Xinyu Z, Deyi L, Tianlei Z, Lifeng A. Traffic light detection and recognition for autonomous vehicles. J China Univ Posts Telecommun 2015;22(1):50–6.
[56] ISO 26262 - Road vehicles Functional safety. Tech. Rep. Geneva, Switzerland; 2011.
[57] Boulanger J. CENELEC 50128 And IEC 62279 standards. Wiley; 2015. 9781119005056.