Towards Formal Verification of Autonomous Driving Supervisor Functions

Yasmine Assioua, Rabéa Ameur-Boulifa, Patricia Guitton-Ouhamou

Towards Formal Verification of Autonomous Driving Supervisor Functions

Assioua Yasmine
Renault Software Labs
France
yasmine.assioua@renault.com

Ameur-Boulifa Rabea
LTCI, Télécom Paris, Institut polytechnique de Paris
France
rabea.ameur-boulifa@telecom-paris.fr

Guitton-Ouhamou Patricia
Renault Software Labs
France
patricia.guitton-ouhamou@renault.com

***Abstract*—In the software development lifecycle, errors and flaws can be introduced in the different phases and lead to failures. Establishing a set of functional requirements helps producing safe software. However, ensuring that the (being) developed software is compliant with those requirements is a challenging task due to the lack of automatic and formal means to lead this verification. In this paper, we present our approach that aims at analysing a collection of automotive requirements by using formal methods. The proposed approach for formal verification is evaluated by the application to supervisor functions of the autonomous driving (AD) system, the system in charge of self-driving.**

***Index Terms*—Requirements analysis, Reliable systems, Model-based design, Systems engineering.**

I. Introduction

With the growing complexity of systems and the shortening of development times, system design requires a robust engineering process. Conventional development approaches provide an unified process for system design from requirement engineering, analysis to design and implementation. Such approaches play a major advancement role in software engineering practices, but quality of designed product in terms of correctness and robustness still remain very critical. Indeed, testing is one of the most practical methods of software quality assurance in industry. But, the process of software testing cannot show the absence of errors. It can only show the presence of errors in software systems [22]. Furthermore, design flaws detected in a late stage can lead to much rework of earlier performed activities, and increase the risk of budget and schedule overruns.

A few years ago, the reliability of a software system was largely determined by the distribution of errors. Thus, there was a great interest in error rates in software engineering, as example study published in [15] reporting that 64% of all errors are introduced during requirements specification and design, and 36% of the errors are introduced during the implementation phase. Today, especially in industrial context reliability engineering focuses on costs of failure caused by bugs and errors. According to [1] and [28] financial losses caused by failures represent more than 5% of the overall turnover of the companies.

Allowing early validation of system requirements through use of formal methods will significantly improve software quality and increase productivity of a production in industry. Indeed, formal methods are deemed as a rigorous software engineering approach for developing highly reliable systems because these methods are based on mathematics and logic.

To facilitate the design of correct and safe systems, we suggest enhancing design process in automotive software engineering, by offering a rigorous model-based approach for the formalization of system requirements, and their early validation. In the paper, we aim to assist developers in their task by automating and speeding the conformance tests that must normally be carried out after development. Such tests usually take some time before a program can be deemed correct or simply thrown back to the developer for rework. To this end, we propose a model-based approach for the software design process, which relies on formal verification technique to systematically develop a design solution for a set of system requirements and to completely verify a developed software. A design solution exists, only if the requirements are complete, consistent, correct and realizable. In fact, the ability to ensure that a system design meets system requirements is crucial for any software from both an economic and a safety point of view. The avionics industry has long recognized the need for formal verification inside the development [35] and has spearheaded the development of several methodologies for adopting rigourous system formal design, in particular, in the field of requirements engineering, e.g. Simulink [5] and SCADE suite [7]. However, the various tools have been developed with the aim of improving the specifications development are mostly focused on requirements management and trace-ability. Unfortunately, even if there is a great interest about the use of formal methods for early validation of system requirements, e.g., [11] and [32], there are very few requirements validation tools available for checking their correctness and functional consistency before any design or coding fulfills. Among them Argosim [3] that provides practical tool for debugging the requirements, and modelling and simulation capabilities for the validation of systems, from functional requirements engineering to automatic test-case generation. Similarly to the Argosim approach our work aims at a design process that allows for the early validation of system requirements. Important differences from the existing approaches are our solution is automotive-specific. The approach focuses on functional and safety requirements coming from the automotive domain. This means that the validated models can be optimally adapted

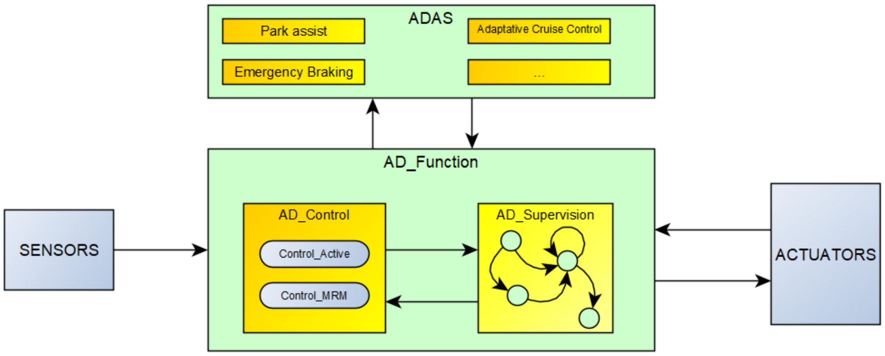

Fig. 1: Simplified view of the autonomous vehicle functional architecture

to automotive projects. Furthermore, the adoption of formal methods in the analysis process could be improved. Indeed, the model checking technique is used as an automatic verification technique in order to validate of system requirements, otherwise to guide their revision.

The advocated approach is semi-automated through a new tools and existing verification tool. We have assessed its effectiveness on a set of requirements for the software design of the autonomous vehicle. The experiment is performed on the supervisor functions of the autonomous driving (AD) system. A simplified view of the autonomous vehicle functional architecture is illustrated in Fig. 1 (as descibed in [18]). It shows the role of AD-function within the self-driving vehicle. It gathers data from cameras, radars and other sensors on different levels to build up a surrounding map, interpret the situation, determine a trajectory and take up the appropriate actions to control the actuators of the vehicle. The AD-function consists of two sub-functions: the AD-Control that is in charge through inner functions of controlling the activity of the automated driving function; The AD-supervisor is a fundamental safety function that prevents unsafe manoeuvres by supervising the operation of the automated driving function can disengage the system at any time. The behaviour of the AD-supervisor is described by a set of states (Active, Activable, Available,. . .) and transitions that are detailed in a set of requirements of the AD-function. These requirements detail a set of states (Active, Activable, Available,. . .) of AD-function and the set of transitions allowed for controlling the operation of this function in order to ensure that no unsafe mode could occur, i.e, the unsafe states are not reachable by the AD-function.

The paper flow is organized as follows: we provide in Section II an overview of the proposed approach to validate of requirements specific to automotive domain. In Section III we apply the approach to restricted set of requirements to illustrate how models are built from specifications written in natural language. In Section IV we give arguments validating our approach by giving the result of applying our methodology on a realistic case-study. In Section V, we present existing approaches that dealt with requirements. Section VI concludes the paper and discusses the limitations as well as possible directions of our work.

II. Approach

To assess the correctness of a software design and to provide assistance and guidance to the engineers to ensure that the developed software meets predefined requirements, we propose a systematic model-based approach for the software development activity. The advocated methodology automate requirements analysis, and through incremental build, whenever possible, a validated design model which provides evidence of consistency of requirements and that they are realizable.

Figure 2 outlines the architecture of our framework that illustrates the proposed approach to help automating the systematic verification of requirements. The figure highlights the relevant steps towards fulfilling the transformation of requirements from natural language to exploitable design models that can be automatically verified. Our framework is composed of mainly two big parts: the **model construction** that consists in constructing a formal model of the software to be developed, and the **automatic verification** of the behavioural properties over the model. This formal verification framework will allow automotive software to comply with ASILs C and D unit recommended guidelines in ISO 26262. Our methodology consists in several steps.

Step1. Requirements Analysis: The first step is to analyse the requirements of the system under design namely software under development. Actually, the analysis is naturally focused on the requirements with clear relevance to the system under design. A system is defined by a collection of states and the set of relevant requirements are those that can affect states of the collection. Requirements are generally expressed in natural language that is inherently ambiguous, as it is not tied to a formal semantics; and are often written by different engineers in various style, which raises a problem with the expressiveness, the completeness, and the accuracy. Starting from those requirements described in an informal manner, we proceed by analysing them for gathering knowledge and extracting the key elements/key-concepts from their textual description. During this analysis phase conducted with domain

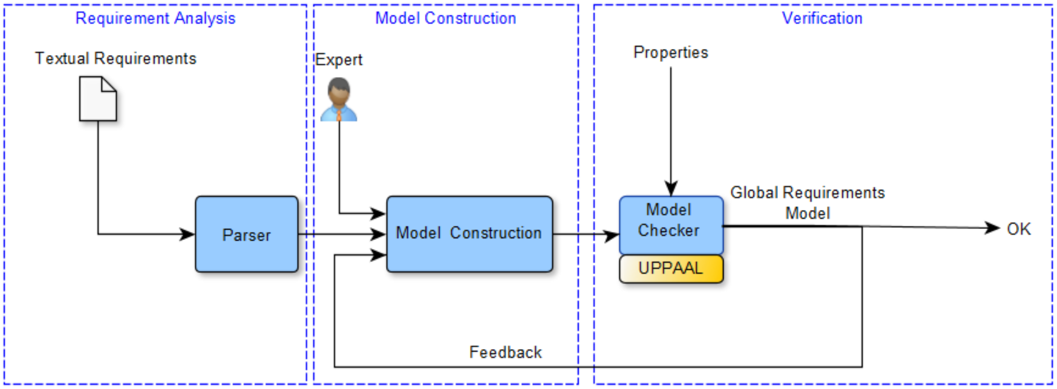

Fig. 2: Framework for the formal analysis of requirements

experts (requirements engineers), requirements are cleaned, inconsistencies discovered, and completed if necessary .

At this final parsing stage unambiguous specification of requirements are derived.

Step2. Model Construction: An initial design model is automatically built form all the derived requirement patterns. Actually, the generated requirement patterns are a set of states and transitions between them. So, a design model in its initial from is built from these states and transitions. However, as requirements might be incorrect, missing and conflicting, the set of states might be incomplete and not sound. For instance, some requirement patterns specify only the target state (source state is lacking), so the state machine should be completed; in a first attempt, we suppose that this state is potentially accessible from all other states of the state machine, thus the associated transition is added. Secondly, the plausibility of the additional information is checked by using a table called the plausibility table – this table is provided by requirement engineers that provides the forbidden states by specifying explicitly the unlawful transitions. If the extra-transitions are not coherent with the plausibility table, we correct the design model. After that, if inconsistencies remain the system software engineer role is required, he can choose among the transitions those that make sense for creating design model that satisfy a set of properties. So the design model is progressively revised and completed until a coherent and complete design model has been obtained.

Step3. Verification: This step aims at automating the systematic validation of requirements. The obtained models are analysed using standard model checking techniques [8], the analysis determines whether the generated model is error-free design or not. If the design is correct it proves it automatically; otherwise it provides an invalidating execution. To facilitate the production of tool chains the popular model checker UPPAAL [29] has been used to verify the generated design models. If assumed properties are satisfied, then the whole set of requirements is validated; otherwise the design model should be refined or certain unsatisfied requirements have to be revised (by returning to the previous stage). Nominal outputs of the model checker gives a valuable feedback reporting the requirements that are not compliant and potential failures that can occur on the generated design solution.

III. Model construction

Any system under design is intended to perform a set of functions. The functions are captured in requirements, i.e, statements that describes what system must do and could impose constraints on it. In the model construction design phase, the generated models intend to describe the functionality of a software design. This section gives the language for the requirements specification and the rules for the transformation of requirements into formal models that are precise in meaning and amenable to formal analysis, in particular accepted by model checking algorithms. In the context of our work, the models that we use in our framework are state machines (labelled transition systems) that represent the behavioural semantics of function as a set of reachable states and boolean (condition) that enables a change of state. Indeed, the set of states expresses the possible states of the function. The transitions describe at any given time, how current state can change upon a condition. State machines formalism presents number of subsequent benefits for the engineer, it is supported descriptive and graphical, especially it is formal specification languge. It is worth to note that the work presented in this paper only covers functional requirements, for non-functional requirements analysis we are aware of the need to extend the state machines, in order to allow specifying non-functional aspects e.g. timing and performance aspects like has been done in [34] and [30].

One of the main objectives of our approach is to tackle the ambiguity requirement specifications through the use of structured approach for the system design. In order to make our framework user-friendly tool and to improve the requirements analysis process, in the same way as in CESAR reference technology platform [33], we propose the use of guided natural language for a domain specific automotive. This language can be used for the requirement definition by requirement engineers. So that, the problem of requirement definition will be simplified, especially through the use of dictionary for the used words. All requirements we analysed have fixed terms and others are attributes that are expressed from one requirement to another with different syntax. So, we propose as the guided language the following natural-language-like templates from which can be derived the set of requirements:

$$\langle \textit{function} \rangle \ \mathbf{shall\ be} \ \langle \textit{state} \rangle \langle \textit{condition} \rangle$$

or

$$\mathbf{if} \ \langle \textit{function} \rangle \ \mathbf{is} \ \langle \textit{state} \rangle \ \mathbf{and} \ \langle \textit{condition} \rangle \ \mathbf{then} \ \langle \textit{function} \rangle \ \mathbf{shall\ be} \ \langle \textit{state} \rangle$$

where **if**, **then**, **and**, **is**, and **shall be** are fixed syntax terms, while elements between symbol $\langle . \rangle$ are attributes of the requirement that are filled by a specialist engineer: $\langle \textit{function} \rangle$ expresses a function of the system under design (that is considered by the requirement), $\langle \textit{state} \rangle$ expresses a given state of the function, and $\langle \textit{condition} \rangle$ a condition that enables/disables a change of state. $\langle \textit{function} \rangle$ and $\langle \textit{state} \rangle$ are specified by their names (identifiers), whereas $\langle \textit{condition} \rangle$s are boolean expressions built from variables and parameters compared to given values. Semantically, the requirements of the first form express that a given function $\langle \textit{function} \rangle$ shall eventually reach a given state $\langle \textit{state} \rangle$ when a set of conditions $\langle \textit{condition} \rangle$ are satisfied. Notice that the requirements of the first form specify only the target state, consequently this state is potentially accessible from all other states. The requirements of the second form express that if the function is at a given state and some conditions are satisfied, then another state shall be reached, resulting in a change of state.

To illustrate our approach and the different phases of the models construction, we will use throughout the paper requirements considered in the analysis of AD-function. Our methodology is general, but in this work it is restricted to functional requirements that specify the functional behaviour of the analysed system.

Example 1: Let us consider the following functional requirements:

- *"The lateral jerk requested by the AD-function shall be limited to a threshold"* (FR1)
- *"AD-function shall be available on verified road sections"* (FR2)
- *"AD-function shall be available at dawn and dusk"* (FR3)
- *"if AD-function is not available and vehicle is in Germany or in France, then AD-function shall be available"* (FR4)

Consider the requirements given in Example 1, they are all related to the same function: *AD-function*. The first three requirements are expressed in the first form, while the last one is expressed in the second form. Note that both FR2 and FR3 refer to a same state AVAILABLE, but with different conditions relating environmental parameters to associated values. Also note that FR1 (by using the keyword **request**) refers to another function, meaning that the AD-function requests another function called THE LATERAL JERK.

In order to reduce unnecessary complexity of the model building process of our methodology takes into consideration only the requirements referring to the analysed function. Thus, within the context of the analysis of AD-function, the analysis should be limited to this function and should not consider other functions, so the requirement FR1 will be taken away.

We have developed an early version of the parser, it has been implemented in Python language; it is currently 600 lines of code.

1) **Requirement analysis.** An initial step in the parsing is to extract information about states and about conditions from the set of analysed requirements. For our case study we identified several states: OFF, NOT_AVAILABLE, AVAILABLE, ACTIVABLE and ACTIVE, and several variables e.g. DAY_TIME and ROAD_SECTIONS that are compared to different values to form the conditions, as ROAD_SECTIONS=OK, DAY_TIME=DAWN and DAY_TIME=DUSK. Sometimes some conditions are not directly expressed making the parsing task more strenuous. For example the condition of requirement FR4 can be interpreted by the condition VEHICLE_LOCATION=GERMANY or VEHICLE_LOCATION=FRANCE only with the help of the experts who define the variable names (VEHICLE_LOCATION) and their possible values (GERMANY and FRANCE).
2) **Requirements selection.** When the process of gathering knowledge has been completed, the set of collected states is submitted to an expert who will select accordingly the set of most relevant requirements for the analysis. Let us consider the requirements given in Example 1. Suppose that the list of selected states are: OFF, NOT_AVAILABLE, AVAILABLE, ACTIVABLE, ACTIVE. The outcome of the selection task is as follows:
   - FR1 is rejected because it does not refer to any state in the list.
   - FR2, FR3 and FR4 are selected, they refer to the considered states AVAILABLE and NOT_AVAILABLE.

   Actually, the set of relevant requirements are all those that can affect the list of provided states.
3) **Completion.** As mentioned before some requirements are incomplete, as the source state or the targeted state are not specified, in particular for the requirements expressed in the first form, in this phase these states will then be completed. In fact, an incomplete requirement generates a state on the initial model that should be completed in the next step. For instance, for the requirement FR3 saying that the AD-function shall reach the state AVAILABLE if the condition DAY_TIME=DAWN and DAY_TIME=DUSK is satisfied. Consequently, transitions should be added. These transitions coming from all other states to this state are labelled with the condition (DAY_TIME=DAWN $\wedge$ DAY_TIME=DUSK). Thus, we apply the same process to all incomplete requirements. At the end of this phase, we generate a complete state machine. However, the additional transitions might be

not compliant with the expected functionality, they lead to forbidden behaviour.

4) **Check the plausibility.** This step aims to check the correctness of the additional transitions. The model construction relies on the priority table (which is sometimes provided in the specification documents) to cut out the meaningfulness and undesirable transitions. For the example we are considering, the priority table reveals that the transitions from OFF to ACTIVABLE, also from NOT_AVAILABLE to AVAILABLE are not plausible, so the model is modified accordingly. In this way, the plausibility rules are incrementally applied to the design model. These rules are those defined in the priority table or provided by software engineer.

5) **Model Generation.** Where there is no more rule to apply, the final design model is generated. We use the UPPAAL model checker tool to display the obtained design model, and especially to verify safety properties that can be expressed as logical formulas [16] or simply as observer automata [23]. Figures Fig. 3 and Fig. 4 show two examples of the resulting global state machines of AD-function as captured in UPPAAL's tool. The two state machines consists of 5 states, but the number of transitions is different. To improve the readability of the generated models, we code the names of states and the names of the variables used over transitions in a concise manner. Indeed in practice, during the parsing phase for each identified variable, parameter, data and associated values, a new name is created and inserted in a table together with the domain from where it gets its value. Such that for each variable encountered the table is initially consulted to check whether the name of variable or parameter exists or not. For instance, in Fig. 4 the guard denoted *guardInActivable()* represents a conjunction of 16 sub-guards, $\mathit{guardInActivable()} = \mathit{guard}_1 \wedge \ldots \wedge \mathit{guard}_{16}$, such that each encoding environmental parameters or some variables: the guard guard_2 denotes the boolean expression VEHICLE_LOCATION=GERMANY $\vee$ VEHICLE_LOCATION=FRANCE; and the guard guard_6 denotes the boolean expression ROAD_SECTIONS=OK and so on.

IV. EXPREPRIMENTS

In order to highlight and assess our methodology, we applied it on a real industrial case study, that is Autonoums Driving system (AD-system) which is a major function of the autonomous vehicles. International organisms, such as NHTSA [6] and OICA [4], have defined autonomous levels to introduce progressively this innovation. It states that an Automated Driving System on the vehicle can do all the driving in all circumstances. The human occupants are just passengers and need never be involved in driving. Failure in such system may cause multiple fatalities. So functional safety assumes critical importance. To trust them, these products should be designed to enable high-performance, and targeting highest automotive safety integrity level (ASIL D).

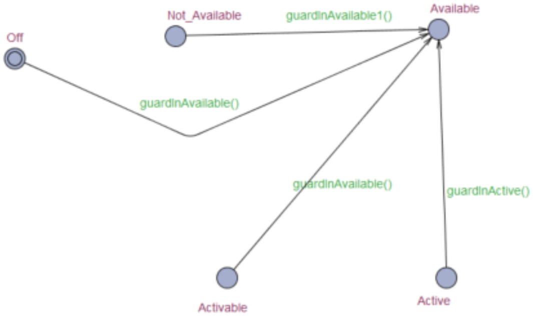

Fig. 3: A non-valid model for the AD-function – An intermediate result of the framewok

We handled several case studies, they include Automatic Park Assist (APA) function that helps make parallel parking easy. In this paper we illustrated our approach with the AD-supervisor component that controls state changes of AD-function, and to ensure that no unsafe can occur, i.e, the unsafe states are unreachable by the function. Behavioural specification of this component is described through 188 functional requirements. After the first analysis and selection phase 120 requirements are selected. Indeed, the analysis keeps only the set of relevant requirements: the requirements that can affect the states of the system. The model is then constructed through several iterations of refinement and completion. Once the model construction have been completed 162 state machines are generated, but not all are valid, i.e, some of them contain wells or non-plausible transitions. Fig 3 shows an example of of a generated model that is not valid. As one can notice, states ACTIVABLE, NOT_ACTIVABLE and ACTIVE are not accessible from the initial state OFF. After removing of non-valid models, only 17 models remain. These models correspond to possible implementations of the system under analysis.

Graphical user-interface built on top of UPPAAL allows engineers to simultaneously see the results of the verification and accordingly continue modelling. Also, in order to ease debugging the tool provides a trace showing how a given state is reachable, and it returns to the engineer a finite error trace, when it is available (when some properties are not satisfied). For instance, we can see in Fig. 5 a result of the reachability analysis. This trace is automatically constructed based on the trace issued by the tool and displayed as a sequence of states.

Discussion: The experiment and the obtained results helped in identifying some issues and open challenges. Actually, as one can notice the model is not too huge in terms of the number of states. But the conditions over transitions might be potentially wide because of the amount of data: parameters, variables, values and constants used by the system; and for which the satisfiability analysis may become very hard. A practical way to do this checking is to use of Satisfiability

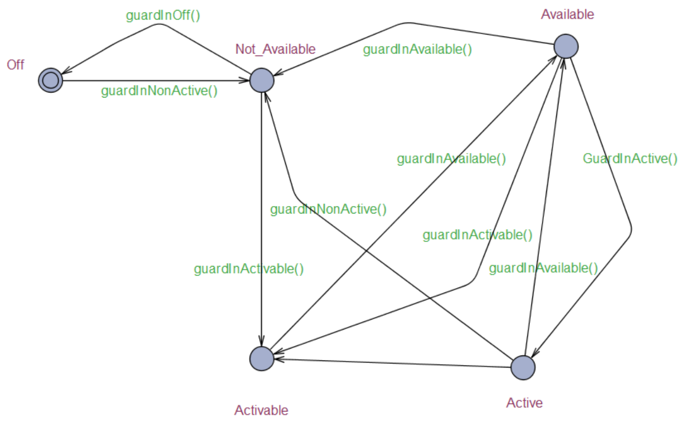

Fig. 4: A valid model for the AD-function – A result generated by the framework

Modulo Theory (SMT) technology, which has a great ability for solving satisfiable problems of first order logic [20]. Moreover, these multiple informations (data, parameters, variables, and constants) are sometimes written with different syntaxes and terminologies. Even if the parsing process is performed by the tool in cooperation with the domain experts in charge of the system (software) design, it remains hard and heavy. There is a clear need for a standardized formalism, to reduce some of the burden on the parsing process (requirements interpretation and analysis phase). We are currently working on the definition of a requirements specific language (like EARS [31] and RSL [19]) for the expression of requirements that can fit the specific automotive domain. This language designed to be user-friendly will be defined with a user-defined glossary terms and using a fixed set of syntactical rules. So, it will lead to express system requirements in a constrained natural language for capturing safety and functional concerns, will ease analysis and will make possible the formal analysis of models. A second important issue is the expression of guards, and whether for a given state there are several requirements with different conditions. So the global condition is constructed by gathering all subconditions in two ways either by using a conjunction operator (as with our example) or by using a disjunction operator. For instance from requirements FR2 and FR3 we get several transitions that are labelled either by a guard in conjunctive form as:
(VEHICLE_LOCATION=GERMANY ∨ VEHICLE_LOCATION=FRANCE) ∧ (ROAD_SECTIONS=OK)
or by a guard in disjunctive form as:
(VEHICLE_LOCATION=GERMANY ∨ VEHICLE_LOCATION=FRANCE) ∨ (ROAD_SECTIONS=OK).

This problem is similar to the one encountered in the program analysis that is invoked may-must analysis [38]. Within our framework both models are relevant, patently the conjunctive form and the disjunctive form offer complementary approaches to analyse the design model. The disjunctive model can be used to prove that all executions of the program satisfy some properties, while the conjunctive model can be used to prove the existence of some program execution that violates a given property.

V. Related work

In recent years the early validation of requirements and using formal methods has become a focus for industrial research such as [11], [32] and [37]. Formal methods have provided evidence for cost-effectiveness by their successful use of in industrial context, in different areas railway, aeronautics and aerospace [10].

More broadly, the validation and verification of system requirements has attracted the industrial research initiatives. Most researches in this area focus on improving the information provided to stakeholders for feedback, including simulation as in [39] and [27] and animations as in [24] and [25]. On the other hand, the principle of validation by using formal methods is very little explored considering their reliability. In this context, verification techniques are used to prove that the developed software meets given requirements. Such proofs often take the form of checking that a specification model satisfies some constraints. For instance, authors in [14], [36] and [21] use model checking technique and show how to check requirements through examples, such as small communication protocols, mutual exclusion algorithms, and small circuits. Additionally, in [26] the author introduces an approach to software design based on model satisfiability to find flaws. In all technical approaches it is assumed that a formal description of the requirements exists, this is not the case in industrial setting.

The use of natural language in the formalisation of requirements was surveyed, in [9] the authors target the specification and analysis of requirements by using the pattern-based Requirements Specification Language (RSL) [19] which is a formal language with a fixed semantics that is still readable like natural language.

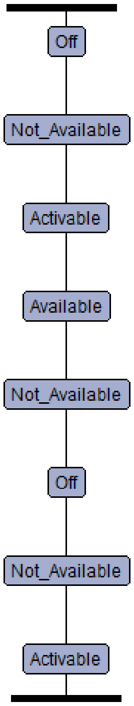

Fig. 5: Trace expressing an example of execution on the model given in Fig 4

The process of producing a global requirements models on which it is possible to detect underspecified or inconsistencies have been the focus of intensive research. Several studies, for example [2], [12] and [40] focus on formalizing the semantics of informal or semi-formal modelling notations as SysML and UML languages. However, these languages are not the traditional language for expressing requirements in all companies. Furthermore, they do not have a formal logic-based semantics, which makes them not suitable for use within approaches integrating formal methods.

In the context previously described, the objective of our work is to propose an automatic method helping system and software architect engineers to visualize, verify and check the delivered requirements. Most of such architect engineers are not aware of checking modelling but need to check coherency, deadlocks, etc …when they define or analyse a lot of requirements. Indeed, most of the time delivered requirements consist of hundreds (and sometimes more) expressions written in natural language by different stakeholders. The analysis of requirements is not straightforward for a human spirit. It is crucial to have a global requirements model to validate automatically the way the stakeholders think the software-to-be and to help to find the appropriate corrections.

VI. Conclusion

This paper introduces a systematic process for building design model from functional and safety requirements with the aim of reducing the effort of testing and defect fixes late in the software development lifecycle. The design model provides an early assurance that the requirements specification are complete, correct and realizable. This first effort led to a formalisation of a set of requirements of the AD-function, and we showed a first proof-of-concept regarding the feasibility of our approach that aims at the exhaustive verification of the generated model; the correctness of the model is proved by using the model checking technique. Naturally, our next goal is to propose a requirements specific language for the expression of requirements that can fit the specific automotive domain. As shown in Section III, we introduced the first version of a structured textual language for the requirements definition. This language with controlled vocabulary with respect to dictionary of automotive domain, constrained grammar and well-defined semantics will lead to express requirements a (formal) natural-language-like representation that will make possible to prove formally the requirements consistency and the design correctness. This language is different from the languages that already exist for the description of automotive standards, as [17] and [13]. In the sense, it allows the specification of systems at a high abstraction level, without any prior knowledge about architectural considerations and how the functions are then allocated to the components of the physical architecture. Giving the benefits of early verification of the functional architecture before its mapping onto the physical architecture. Nevertheless, a practical aim of our approach is to feed EAST-ADL approach with validated models, that can help to define an appropriate electronic system.

We are currently extending the proposed language, looking at further requirements expressing other aspects. Naturally, this growth requires to annotate models with information required to perform specific analysis. For timing aspects of systems that we are dealing with, we propose to enrich our models with timing information and replace automata with timed automata that are in fact supported by UPPAAL tool.

REFERENCES

[1] National Survey: The costs of poor quality in industry. Technical report, AFNOR, October 2017.

[2] E. Andrianarison and J.-D. Piques. SysML for embedded automotive Systems: a practical approach. In *ERTS2 2010, Embedded Real Time Software & Systems*, Toulouse, France, May 2010.

[3] Anonymous. The argosim home page. *Available at https://www.argosim.com/*.

[4] Anonymous. International organization of motor vehicle manufacturers home page. *Available at http://www.oica.net/*.

[5] Anonymous. The mathworks home page. *Available at https://fr.mathworks.com/products/simulink.html*.

[6] Anonymous. National highway traffic safety administration home page. *Available at https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety*.

[7] Anonymous. Scade suite – esterel technologies home page. *Available at https://www.ansys.com/products/embedded-software*.

[8] C. Baier and J. Katoen. *Principles of model checking*. MIT Press, 2008.

[9] M. Böschen, R. Bogusch, A. Fraga, and C. Rudat. Bridging the gap between natural language requirements and formal specifications. In *Joint Proceedings of REFSQ-2016 Workshops, Doctoral Symposium, Research Method Track, and Poster Track co-located with the 22nd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2016), Gothenburg, Sweden, March 14, 2016.*, 2016.

[10] J. Boulanger. *Industrial Use of Formal Methods: Formal Verification*. Wiley, 2013.

[11] M. Bozzano, A. Cimatti, J.-P. Katoen, P. Katsaros, K. Mokos, V. Y. Nguyen, T. Noll, B. Postma, and M. Roveri. Spacecraft early design validation using formal methods. *Reliability Engineering & System Safety*, 132:20–35, 2014.

[12] E. Brottier, B. Baudry, Y. Le Traon, D. Touzet, and B. Nicolas. Producing a global requirement model from multiple requirement specifications. In *EDOC'07 (Entreprise Distributed Object Computing Conference)*, Annapolis, MD, USA, 2007.

[13] S. Bunzel. Autosar - the standardized software architecture. *Informatik Spektrum*, 34(1):79–83, 2011.

[14] W. Chan, R. J. Anderson, P. Beame, S. Burns, F. Modugno, D. Notkin, and J. D. Reese. Model checking large software specifications. *IEEE Transactions on Software Engineering*, 24(7):498–520, July 1998.

[15] R. N. Charette. *Software engineering environments : concepts and technology*. Intertext Publications, New York, NY, 1986.

[16] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, Apr. 1986.

[17] P. Cuenot, P. Frey, R. Johansson, H. Lönn, Y. Papadopoulos, M.-O. Reiser, A. Sandberg, D. Servat, R. Tavakoli Kolagari, M. Törngren, e. H. Weber, Matthias", G. Karsai, E. Lee, B. Rumpe, and B. Schätz. *The EAST-ADL Architecture Description Language for Automotive Embedded Software*, pages 297–307. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[18] R. CUER. *Démarche de conception sûre de la Supervision de la fonction de Conduite Autonome*. PhD thesis, INSA Lyon – Institut National des Sciences Appliquées de Lyon, 2018.

[19] W. Damm, H. Hungar, B. Josko, T. Peikenkamp, and I. Stierand. Using contract-based component specifications for virtual integration testing and architecture design. In *2011 Design, Automation Test in Europe*, pages 1–6, March 2011.

[20] L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[21] S. Easterbrook and M. Chechik. A framework for multi-valued reasoning over inconsistent viewpoints. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 411–420, Washington, DC, USA, 2001. IEEE Computer Society.

[22] W. E. H. Edward Miller. *Tutorial, software testing & validation techniques*. IEEE Computer Society Press, 1981.

[23] N. Halbwachs, F. Lagnier, and P. Raymond. Synchronous observers and the verification of reactive systems. In *Proceedings of the Third International Conference on Methodology and Software Technology: Algebraic Methodology and Software Technology*, AMAST'93, pages 83–96, Berlin, Heidelberg, 1994. Springer-Verlag.

[24] C. Heitmeyer, J. Kirby, B. Labaw, and R. Bharadwaj. Scr: A toolset for specifying and analyzing software requirements. In A. J. Hu and M. Y. Vardi, editors, *Computer Aided Verification*, pages 526–531, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[25] Hung Tran Van, A. van Lamsweerde, P. Massonet, and C. Ponsard. Goal-oriented requirements animation. In *Proceedings. 12th IEEE International Requirements Engineering Conference, 2004.*, pages 218–228, Sep. 2004.

[26] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2012.

[27] B. Jeannet and F. Gaucher. Debugging Embedded Systems Requirements with STIMULUS: an Automotive Case-Study. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, TOULOUSE, France, Jan. 2016.

[28] H. Krasner. The Cost of Poor Quality Software in the US: A 2018 Report. Technical report, CISQ Consortium for IT Software Quality, September 2018.

[29] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Int. J. Softw. Tools Technol. Transf.*, 1(1-2):134–152, Dec. 1997.

[30] R. Mateescu and J. I. Requeno. On-the-Fly Model Checking for Extended Action-Based Probabilistic Operators. *International Journal on Software Tools for Technology Transfer*, 20(5):563–587, Oct. 2018.

[31] A. Mavin and P. Wilkinson. Big Ears (The Return of "Easy Approach to Requirements Engineering"). In *RE 2010, 18th IEEE International Requirements Engineering Conference, Sydney, New South Wales, Australia, September 27 - October 1, 2010*, pages 277–282, 2010.

[32] S. P. Miller, A. C. Tribble, M. W. Whalen, and M. P. E. Heimdahl. Proving the shalls: Early validation of requirements through formal methods. *Int. J. Softw. Tools Technol. Transf.*, 8(4):303–319, 2006.

[33] A. Rajan and T. Wahl. *CESAR - Cost-efficient Methods and Processes for Safety-relevant Embedded Systems*. Springer, Vienna, 2013.

[34] L. Rioux, R. Henia, and N. Sordon. Using model-checking for timing verification in industrial system design. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 377–378, March 2017.

[35] J. Rushby. Formal methods and their role in the certification of critical systems. In R. Shaw, editor, *Safety and Reliability of Software Based Systems*, pages 1–42, London, 1997. Springer London.

[36] T. Sreemani and J. M. Atlee. Feasibility of model checking software requirements: a case study. In *Proceedings of 11th Annual Conference on Computer Assurance. COMPASS'96*, pages 77–88, June 1996.

[37] E. Stachtiari, A. Mavridou, P. Katsaros, S. Bliudze, and J. Sifakis. Early validation of system requirements and design through correctness-by-construction. *Journal of Systems and Software*, 145:52–78, 2018.

[38] A. Stone, M. M. Strout, and S. Behere. May/must analysis and the dfagen data-flow analysis generator. *Information & Software Technology*, 51:1440–1453, 2009.

[39] M. Taisch and B. Stahl. Requirements analysis and definition for eco-factories: The case of emc2. In C. Emmanouilidis, M. Taisch, and D. Kiritsis, editors, *Advances in Production Management Systems. Competitive Manufacturing for Innovative Products and Services*, pages 111–118, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[40] A. Taleghani and J. M. Atlee. Semantic variations among uml statemachines. In O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, editors, *Model Driven Engineering Languages and Systems*, pages 245–259, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.