Requirements to Models of Automotive Software: Application to the Automatic Park Assist function

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 the Automatic Park Assist (APA) function.**

***Index Terms*—Requirements analysis, reliable systems, model-based design.**

I. Introduction

Usually, software project involves a huge set of requirements, that can be written by different stakeholders (software and requirements engineers, etc). These requirements are usually expressed in natural language which arises a problem with the expressiveness, the completeness, and the accuracy. The design problem in system engineering is to ensure that the relevant functional and safety requirements are respected at all stages during the development phase. Within that context, many efforts have been done in the processes of software development and in particular in the validation phase. However, although methodologies and tools have been proposed for that purpose, very few works have been devoted to supporting rigorous design, in particular to provide a systematic stepwise design approach for building a validated design model. Such an approach would be valuable for software developer, it should provide confidence that the requirements are consistent and realizable.

The idea behind this paper is to propose a rigorous model-based process leading from requirements to correct implementations. The advocated approach is applied for designing software for reliable autonomous vehicles, will provide assistance and guidance to the developer in the development phase to assess the correctness of his design. Indeed, our methodology allows to early specify and validate a system design from a collection of functional requirements. In order to highlight and assess our methodology, we applied it on a real industrial case study, that is the the APA (Automatic Park Assists) function. This function is implemented into the vehicle so it can park itself. Inside this function, there is many sub-functions which have different autonomous levels. From the HFP (Hand Free Parking) function, the system only controls the steering wheel to the RPK (Remote Park) function where the driver is outside the vehicle and the system controls all the elements involved in parking the car. The systems requirements are specified in an excel file called a System Technical Requirement Component (STRComp, for short). This file involves all the requirements (functional and safety) and the definitions of a given system. In this particular instance, the APA's STRComp consists of 327 requirements among them functional requirements and safety requirements.

II. Framework – From Requirements to Formal Models

Our goal is to use formal methods both to increase the quality of the systems developed at Renault through enhancing the verification activity, and to prevent unnecessary tests. We propose a general model-based approach for the software development activity, that aims to the systematic development of a design solution for a set of system requirements. The big picture of the proposed approach is shown in Figure 1, highlighting the relevant steps towards fulfilling the transformation of requirements written in natural language into exploitable design models that can be automatically verified for errors and validated against behavioural properties.

Step1. Requirements Analysis: Aiming to provide assistance and guidance to the engineers to ensure the quality of their developed software with respect to functional and safety requirements. Our methodology requires proceeding with the specification of the identified requirements. Actually, the analysis is naturally focused on the requirements with clear relevance to the system under design (software under development). A system is defined by a collection of states and the set of relevant requirements are those that can affect states of the collection. 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. Parsing requirements is not a trivial task, as those requirements, are often written by different engineers in various style, and use natural language that is inherently ambiguous, as it is not tied to a formal semantics. This activity should be conducted together with domain experts (requirements engineers). At this final parsing stage unambiguous specification of requirements are derived.

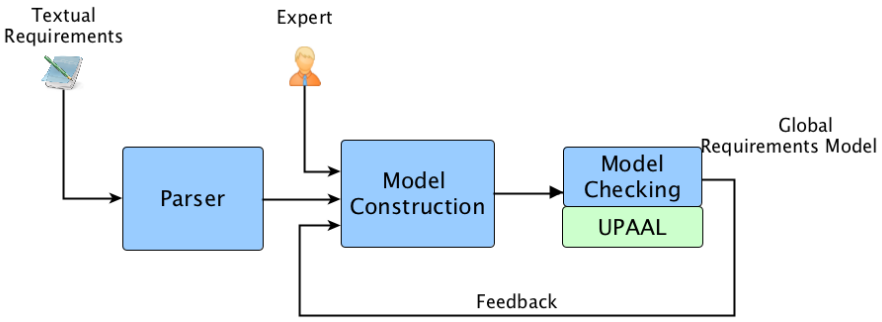

Fig. 1. Framework for the formal analysis of requirements

The specification are patterns formalized in the following form:

IF [*Initial State*] **AND** [*Condition*] **THEN** [*Final State*] where **IF**, **AND**, and **THEN** are fixed syntax terms, while *Initial State*, *Condition*, and *Final State* are attributes of the requirement. *Initial State* and *Final State* express a source state and a target state resp. of the system under design, and [*Condition*] expresses the condition that enables/disables a change of state. Thus, through the parsing analysis a set of states and transitions are generated.

Step2. Model Construction: Once all the requirements of a system (software or architecture) have been analysed and the patterns generated, an initial design model is automatically built from all the patterns. Actually, patterns bring a set of states and transitions between them. So the design model in its initial form is a state machine built from all the set of obtained states and transitions. However, as requirements may be incorrect, missing and conflicting, so the set of states may 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 step so that this state is potentially accessible from all other states of the state machine. In a second step, the plausibility of the additional information is checked by using to the priority table – this table is provided by requirement engineers that provides the forbidden transitions by specifying the final states. If the extra-transitions are not coherent with the priority 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 verification of compliance with requirements. The model checking method is applied to determine the error-free design of the generated state machine model and automatically find the logical errors. The strength of this method [4] is returning a feasible sequence through a counter-example and simultaneously verifying its correctness using properties. To facilitate the production of tool chains the popular model checker UPPAAL [6] has been used to verify the generated design models. If assumed properties are satisfied, then so is the whole set of requirements; otherwise, the design model should be refined or certain unsatisfied requirements have to be revised.

III. Case Study

The APA system is related to the ADAS (Advanced Driver Assistance System) ECU system. This system assists the driver during the parking maneuver. The automatic park assist contains various sub-functions with different level of autonomy. We have developed an early version of the parser, it has been implemented in Python language.

1) **Requirement analysis.** An initial step in the parsing is to extract information about states and conditions are extracted from the set of analysed requirements. For our case study we identified six states: *Safe State 1*, *Safe State 2*, *Safe State 3*, *Safe State 4*, *Idle State* and *Ramp State*. The last two states they are also called *out of maneuver* state and *in maneuver* state resp. elsewhere in the STRComp document.
2) **Requirements selection.** When the process of gathering knowledge has been completed, the set of collected states is submitted to an expert. He will select accordingly the set of most relevant requirements for the analysis. Actually, the set of relevant requirements are all the requirements that can affect the list of provided states. For the analysed system, only 15 functional and safety requirements are kept. Let us consider the following two example requirements:
   REQ1: **IF** (ADAS ECU is in "APA ADAS ECU Safe State 1" **AND** Standstill is Activated **AND** pgen_PLC_ParkingSequenceRequest=vgen_stop2) **THEN** (ADAS ECU shall continuously request Standstill **AND** set pgen_PLC_SequenceStatus to vgen_PLC_Move_Ready).
   REQ2: **IF** (ADAS ECU is in "APA ADAS ECU Safe state 1" **AND** pgen_PLC_ParkingSequenceRequest=vgen_PLCRamp) **THEN** ADAS ECU shall switch to Ramp state.
   The two requirements concern the same function APA ADAS ECU within the function ADAS ECU. Both REQ1 and REQ2 refer to selected states *Safe State 1* and *Ramp state*, but as one can notice REQ1 causes no change of

states whereas REQ2 do. So, only REQ2 is kept for the rest of the analysis.

3) **Completion.** The requirement REQ2 is defined in a complete manner, it provides an initial state, a final state, and a condition enabling the transition. However, some requirements are incomplete, the source state or the destination state are not specified. As for the following requirement:
   REQ3: **IF** ADAS receives: -pgen_APA_Failure= vgen_Fail **THEN** the ADAS ECU shall switch to "APA ADAS ECU Safe State 3".
   This requirement describes that the function should switch to *Safe State 3*, so specifying only the target state and making no mention of source state; So, this incomplete requirement generates a state on the initial model that should be completed in the next step. Indeed, in the model construction phase a transition will be added from each state of the predefined list.
4) **Check the plausibility.** However, the transitions added are not all semantically correct. The model construction relies on the priority table to cut out the meaningfulness and undesirable transitions. For the example we are considering, the priority table reveals that the transitions from *Safe State 2* to *Safe State 3*, and *Safe State 4* to *Safe State 3* 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 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 [5] or simply as observer automata [9]. Fig. 2 shows the resulting automaton of our use case, with 6 reachable states and guarded transitions. 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 and associated values, a new variable 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 variable exists or not. More specifically, we simplify the case study encoding in Fig. 2, as shown by denoting the state *Safe State* with *SS* and the variable *pgen_PLC_ParkingSequenceStatus* with *Pstatus*. Additionally, as the *pgen_PLC_ParkingSequenceStatus* variable can take 7 different values *vgen_PLCIdle, vgen_PLCRamp, vgen_PCLStanby, vgen_MovePause, etc*, we encode these values using numerical numbers *0, 1, 3, etc*.
   It is also noted that, still for the sake of readability we encode some conditions by these practical meaning, for instance the condition *pgen_APA_Failure = vgen_ Fail* corresponds to a failed situation, so we denote it by *Fail* predicate over the associated transition.

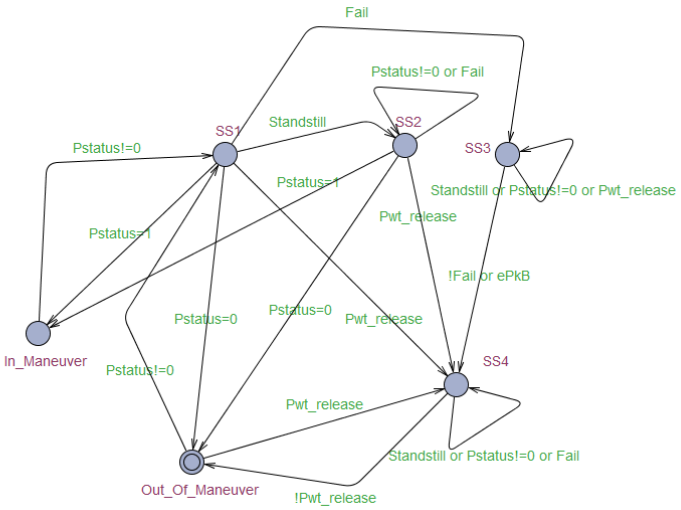

Fig. 2. The final model for the APA case study

IV. Related Works

In the last decade, companies, in particular in the area of aeronautics and aerospace have made great efforts to reduce their development times and to improve the quality of their products. Indeed, for developing complex systems, they are increasingly adopting model-based design tools in system development, e.g. Simulink [2] and SCADE suite [1]. 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., [7] and [8], 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 (i) 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 to automotive projects; (ii) and furthermore, the adoption of formal methods in the analysis process. Indeed, the model checking technique is used as a validation technique in order to verify the correction of system requirements, otherwise to guide their revision.

V. Conclusion

This paper introduces a systematic process for building design model from functional and safety requirements with the aim of reducing the validation testing during the late stages of 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 the APA 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 goals is to propose a requirements specific language for the expression of requirements that can fit the specific automotive domain. This language with constrained grammar and well-defined semantics will lead to express system requirements in a language close to the natural language, and will make possible to prove formally their correctness.

References

[1] Anonymous. SCADE Suite - Esterel Technologies Home Page. Available at https://www.ansys.com/products/embedded-software

[2] Anonymous. The MathWorks Home Page. Available at https://fr.mathworks.com/products/simulink.html

[3] Anonymous. The Argosim Home Page. Available at https://www.argosim.com/

[4] C. Baier, J. Katoen. it Principles of model checking. MIT Press, 2008

[5] Clarke, E. M.; Emerson, E. A. and Sistla, A. P. *Automatic verification of finite-state concurrent systems using temporal logic specifications*. ACM Transactions on Programming Languages and Systems, 1986

[6] K.GLarsen, P. Pettersson, and Wang Yi. *UPPAAL in a nutshell*. Journal of Software Tools for Technology Transfer, 1(1-2):134-152,1997

[7] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Panagiotis Katsaros, Konstantinos Mokos, Viet Yen Nguyen, Thomas Noll, Bart Postma, and Marco Roveri. *Spacecraft early design validation using formal methods*. Reliability Engineering & System Safety, 132:20-35, 2014.

[8] Steven P. Miller, Alan C. Tribble, Michael W. Whalen, and Mats P. E. Heimdahl. *Proving the shalls: Early validation of requirements through formal methods*. Journal of Software Tools for Technology Transfer, 8(4):303-319, August 2006.

[9] Nicolas Halbwachs, Fabienne Lagnier, and Pascal Raymond. *Synchronous observers and the verification of reactive systems*. In Proceedings 26 of the Third International Conference on Methodology and Software Technology: Algebraic Methodology and Software Technology, AMAST'93, pages 83-96, London, UK, UK, 1994. Springer-Verlag.