Application of Extended Timed Automata to Automotive Integration Testing

Jan Sobotka [1], Jiří Novák [1]
[1] Czech Technical University in Prague, Faculty of Electrical Engineering,
Prague, Czech Republic
email:jan.sobotka@fel.cvut.cz, www.fel.cvut.cz

Abstract— **Deployment of the Model-based Testing methods in practice has not achieved the level it deserves. To help dissemination, as well as to improve the testing process in a particular domain, this paper presents a new Test Generation tool on a case study. The domain is automotive integration testing. The new tool is named Taster and utilizes Timed Automata for the online Model-based test generation. The objective of these tests is testing of integration of automotive comfort systems. The proposed concept, the system modeling, and the new software tool is evaluated on testing of a Keyless Access System. Purpose of the paper is to present an approach for automatic test generation intended for automotive integration testing.**

Keywords- Model-Based; Integration; Testing; Timed; Automaton; Automotive; ECU

I. INTRODUCTION

This paper presents an application (new tool implementation) of Model-Based Testing (MBT) approach to the integration testing of automotive electronics systems - i.e. a cluster of Electronic Control Units (ECU). In current practice, the original test suite is developed by test engineers as a sequence diagrams. The test suite development is labor intensive, and amount of work should not grow in future. On the other hand complexity of automotive systems still grows. Test suite complement in the form of automatically generated test cases is offered, to keep the amount of test development works reasonably.

The key idea is to supplement a test suite developed traditionally by a test suite generated using the MBT principles. These additional test cases are generated by a software tool called Taster. MBT process is driven by a Timed Automata model. One of the limitations of traditional test suites is that test cases are designed using driver-oriented point of view. One of the goals of presented work is to overlook from this narrow use case and, using the MBT techniques, to produce traces, which examine system by more diverse, but still reasonable stimuli. In other words, we presume that usefully complement test suite created by engineers with machine-generated ones can significantly increase the number of revealed faults. The proposed solution is evaluated on a short case study with the objective to judge the suitability of the developed test generation tool Taster for future research. The proposed solution is depicted in Fig. 1. A short overview of related work follows.

Many model checkers or formal verification tools are based on the Theory of Timed Automata. Also, there are at least hundreds of Timed Automata variants [1]. Usage of this theory for testing is less common than for model verification. TRON [2] and CoVer [3] from the UPPAAL connected tools family are probably the most relevant research for presented approach. Both tools use the UPPAAL model checking engine for the test generation. Presented tool Taster uses a different technique. It uses own algorithms based on graph search theory. Taster tool evolves on a basis of our conceptual work [4]. This paragraph is focused on Timed Automata linked works. Besides introduced tools, RT -Tester [5] can be mentioned as it is targeted to similar SUT class. RT-Tester uses a subset of UML or SysML as modeling format.

Paper is organized in following structure. Section *II* describes the problems of the current practice of integration testing in more details. Section *III* shows overall testing concept and system modeling - Timed Automata model with an extension. In Section *IV* are presented algorithms developed for the test generation and our test generation tool Taster. Section *V* contains the Keyless Access System (KESSY) case study, which is subdivided into four subsections. Last two sections are Conclusion (*VI*) and Future work (*VII*).

II. PROBLEM STATEMENT

Consider an automotive integration testing scenario. For a System under Test (SUT), a test suite based on the test plan is developed. This test plan covers demands from compulsory road regulation, internal standards, and a system specification. The specification is usually created and maintained by requirement management software (e.g., Rational DOORS). An SUT is tested using black box approach – no internal structure or similar information is employed to test suite design. All test cases are developed from a driver point of view (a typical driver use case). The testing process itself is driven by EXAM [6], which covers tasks from individual test case implementation to management, execution, and assessment of complex test suites. Despite careful and hard testing work, it is not possible to cover all possibilities by a manually developed test suite. The reason is in the system complexity and associated well-known State space explosion problem [7] together with limited time and cost resources.

In this process, various improvement possibilities can be identified. First of all, manual development of the integration tests manually is very labor intensive. Therefore, decreasing of a size of original test suite in behalf of automatically generated one should be significantly beneficial. Also, complementing the suite by test cases developed differently

 ISBN: 978-1-61208-593-7

could help achieve better test diversity. This work addresses these challenges by the development of an MBT test tool able to operate with an SUT controlled by EXAM.

III. CONCEPT AND SYSTEM MODELING

Overview of proposed concept is depicted in Fig. 1. The textual specification is presumed as the basis for the model development. SUT is modeled as a network of Timed Automata. UPPAAL [8] is used in the role of model editor. Complete testing is driven by this model – no other models or configuration files are used.

Figure 1. Concept of implemented solution

Taster tool explores the models using graph theory algorithms. Test inputs are produced online, and the SUT outputs are checked against expected ones. The model simulation is directly used for test generation. The algorithms are described in detail in Section *VI*. Interaction with an SUT is done by NI VeriStand or EXAM test adapter.

The proposed solution employs UPPAAL tool [8] as a modeling environment. The system is modeled as a network of Timed Automata. The underlying theoretical concept is denoted Timed Safely Automata. Beyond theoretically described [1] time and transition properties, the UPPAAL implementation offers usage of variables, conditions, synchronization channels and another language construct to provide a certain level of expressivity (in theory summarized by term action). The used modeling language is a subset of UPPAAL modeling language summarized in Tab 1.

TABLE I. SUPPORTED SUBSET OF UPPAAL MODELLING LANGUAGE

	Action		
Data type	**Guard**	**Update**	**Sync**
clock	✓	reset	×
chan	×	×	✓
bool	✓	✓	×
int	✓	✓	×

The subset is chosen concerning tested system class. Supported data types are *bool*, *int*, *clock* and synchronization type *chan*. Edges can contain *guard*, *sync*, and *update* expressions.

Besides Timed Automata modeling language implemented by UPPAAL tool, Taster additionally utilizes labeling of Timed Automata states by relevance. *Relevance* is a natural number assigned to an automaton state. The parameter expresses the importance of a model state summarized by one number. The higher value implies the higher testing priority. *Relevancies* are assigned according to the SUT expert knowledge. It is determined in a manual or automatic way and originates in for example safety impact, the impact on rest of the system operability, and previously revealed issues (bugs). KESSY testing example presented later in this paper shows usage of *Relevancies* on environment models of start and door buttons.

The success of proposed solution strongly depends on the reasonability of used model. In this work, two types of models are used. The first category is environment Models, which produces inputs for an SUT during simulation. The second category is observer models which have the Oracle function – they check expected SUT outputs. Correct output is expressed as an invariant condition. SUT behavior is considered valid if invariant in an active state is satisfied. The division of the model into environment and observer parts is only imaginary, and it is possible to combine input and output actions to single Timed Automaton. Nevertheless, partitioning of entire model to these two model types is recommended to keep clarity.

IV. ALGORITHMS AND TOOL

The test generation is based on the model exploration using graph search techniques. Timed automata model is simulated in a Real-time. The SUT input and outputs are linked to the model variables. Consequently, variable assignment on model edges produces test stimuli. The SUT outputs are observed using variables utilized in location invariant conditions. The progress of the time is discrete, and it is equal to the time of simulation step. Developed algorithms are described in pseudo code. The first algorithm (Fig. 2) describes the overall testing process.

```
Algorithm 1
DoTesting (Model):
  while invSatisfied and covCritNotSatisfied
    ReadInputs
    DoStep (Rand. | Prized R. | Sys.)
    UpdateClocks
    WriteOutputs
```

Figure 2. Algorithm 1

Three different strategies are used to choose next step (i.e. edge to be taken). First step common for all strategies is the creation of a list of allowed edges in that time. Allowed edge is an edge with satisfied guard condition. If the edge triggers synchronization, corresponding edge waiting for synchronization has to be allowed. In case the list of allowed edges is empty, no edge is taken, clocks are incremented and the loop continues to next iteration. The loop is stopped if the invariant is violated, coverage criterion is satisfied, or test is stopped by the test operator. Algorithms 2 and 3 take next edge randomly from the list of allowed edges. Algorithm 2 (Fig. 3) works with discrete uniform distribution – probability of pickup is the same for all edges in the list.

```
Algorithm 2
DoStepRandom (Model):
for each ActiveNode in Mod. Templates:
  for each OutEdge in OutEdges:
    if GuardIsSatisfied(OutEdge)
      listOfAllowedEdges.Add(OutEdge)
nextEdge = Rand(listOfAllowedEdges)
```

Figure 3. Algorithm 2

Algorithm 3 (Fig. 4) modifies discrete uniform distribution using *Relevance* numbers defined in the previous section. Probability of taking for an edge *i* from the list of allowed edges is:

$$P(E_i) = \frac{rel_{E_i}}{\Sigma rel_E} \quad (1)$$

Implicit *Relevance* is equal to one. *Relevance* is assigned to an edge from its target node. It may be confusing, but reason is to preserve compatibility with UPPAAL model format. *Relevance* is stored as a node comment, which is not possible with an edge.

```
Algorithm 3
DoStepPrioritizedRandom (Model):
for each ActiveNode in Mod. Templates:
  for each OutEdge in OutEdges:
    if GuardIsSatisfied(OutEdge)
      listOfAllowedEdges.Add(OutEdge)
nextEdg = PrizedR.(listOfAllowedEdges)
```

Figure 4. Algorithm 3

```
Algorithm 4
DoStepSystematic (Model):
for each ActiveNode in Mod. Templates:
  for each OutEdge in OutEdges:
    if GuardIsSatisfied(OutEdge)
      listOfAllowedEdges.Add(OutEdge)
nextEdge =
PickLowestTakenEdg(listOfAllowedEdges)
```

Figure 5. Algorithm 4

Algorithm 4 (Fig. 5) refers edge with the lowest takes count from a list of allowed edges. Model exploration is more deterministic than in the case of the random strategy. This behavior may speed up structural model coverage if it is desired. Concerning black box testing approach only, the model coverage criteria are feasible. Condition `coverageCriterionNotSatisfied` unify coverage of all Timed Automata model nodes or edges.

The proposed concept with the MBT theory and algorithms results in the implementation of a testing tool named Taster. The first version of this tool was implemented in [9]. Taster works with models stored in UPPAAL 4 format. After a model is loaded, it is possible to performed testing as proposed. The SUT is connected by a test adapter. In following sections, the Taster is described in the order same as testing workflow. First is the model parser and last is the result viewer. The code is written in C# using .NET Framework 4.5.

The software architecture is divided into the model parser and the test execution part. First part is responsible for syntactical check and data structures preparation. Second part implements testing engine itself. The execution part also contains trace logger with replay function.

Syntactical analyzer code is generated by language recognition software ANTLR [10]. Expression evaluation is solved by Shunting-yard algorithm [11]. Syntactical analyzer code is generated by language recognition software ANTLR [10]. Expression evaluation is solved by Shunting-yard algorithm [11]. Traces are stored in an XML file for further analysis.

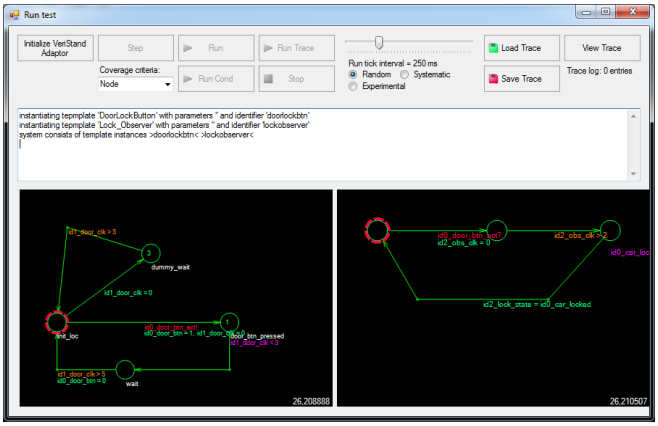

Figure 6. Taster – Test Execution and Control

The usage of the testing tool is displayed in Fig. 6. First is screen (not depicted) is the model viewer where a model is loaded. Second screen is the run screen. On the run screen the test adapter is initialized and a test run is executed according to selected algorithm. A test run is terminated by one of the following actions: invariant is violated, coverage criterion is satisfied, and the test time has passed or by termination by a user request.

V. KESSY SYSTEM TESTING

The following example shows the application of presented concept on testing of a keyless access system. As the name suggest, the purpose of keyless access system is to allow vehicle entry and engine start without using a key (only the key's RFID tag have to be detected).

A. Specification

The example system function (i.e. textual specification) is captured by this description. The door locking system is controlled by the lock button built in the driver side door handle. The Start/stop button works as the ignition switch. The short press is designated to turn the ignition on and button long press is the command for engine start operation. Button press longer than one second is considered long. Ability to lock and unlock the door, as well as start or stop the engine, is determined by detected key position. The system contains two equal keys RFID tags marked Key 1 and Key 2. The system recognizes following states of each key: detected outside the car, detected inside the car and not detected. The door unlocking is not possible if no key is detected outside the car. The door locking is not allowed if a key is detected inside the car. The engine start requires a key to be detected inside a car.

In modern Vehicles, the KESSY system is implemented as distributed system. Let's consider a system composed from three ECUs. First ECU is the KESSY system itself which cooperates with a Body Control Module (BCM) and Engine Control Unit. All ECUs are interconnected by a Controller Area Network (CAN) bus. The KESSY ECU is responsible for keys and button status monitoring. Based on its commands, the BCM controls power supply system and individual door locks. Engine start and stop procedure are driven by the ECU. The system inputs are described in the previous paragraph. The system is observable by four digital outputs. Door locking system has locked and unlocked states. Start button controls ignition system which controls three power supply branches. German language prefix Klemme (KL) for individual clamps is preserved as, it is standardized by DIN 72552 standard and ISO equivalent does not exist. The system controls KL 15, 50 and S. KL 15 is active if the ignition is on. KL 50 is active during engine startup only and in this work it is used for engine start monitoring. KL S is turned on by switching the ignition on, and it is active until the car is locked. Its common usage is for audio system powering.

B. Models

The example system is modeled by the network of nine Timed Automata. Every system input is modeled by a single automaton. The KESSY system provides two functions – ignition switch controlled by Start/stop button and door locking system control. Correct behavior of each is observed by separate automaton. The relation between keys position and corresponding system behavior is modeled by another three automata. Overall characteristic of the model is stated in Tab. 2.

TABLE II. KESSY MODEL SUMMARY

Entity	Count
Templates	8
Instances	9
Nodes	30
Edges	39
Clocks	7
Variables	15
In/Out variables	6

Due to limited paper range, only Start/stop button model and one of two observer models are described. The first model, the start/stop button, is depicted in Fig. 7. In the case of the Start/stop button, the long and short press are distinguished, in opposite to the door lock button, which has two states only (press and release). Models also contain auxiliary wait states which are used for simulation of user inactivity. Labeling the automaton states by *Relevance* parameter is used for preferring inactivity (wait state) against button pressed. Similarly, it is favored short press before the long press.

The most important parts for distinguishing between expected and incorrect SUT reactions are observer models. Our example uses two observers. Locking System Observer and Ignition System Observer, which is shown in Fig. 8. The system behavior is checked by location invariants. Variables in invariant conditions are mapped to the system outputs. Observers are synchronized with button handling models by synchronization channels.

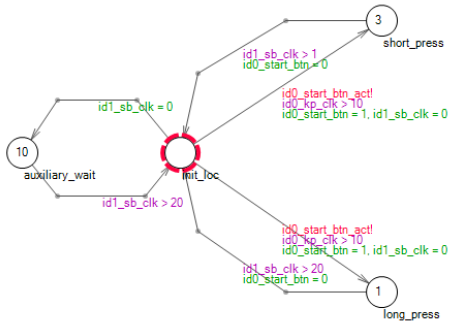

Figure 7. Start button model

A set of simple models capture relation between keys position and allowed system behavior. A key location determines whether it is possible to unlock, lock and start the car. The models produce corresponding signals to handle this situation.

C. Implementation

The experimental SUT was developed using NI VeriStand Real-Time Testing Software. This platform was chosen on the basis of previous experience during development of an HIL test place with our industrial partner. Real KESSY ECU was replaced by an own implementation, as fault injection is much less complicated in its case.

 ISBN: 978-1-61208-593-7

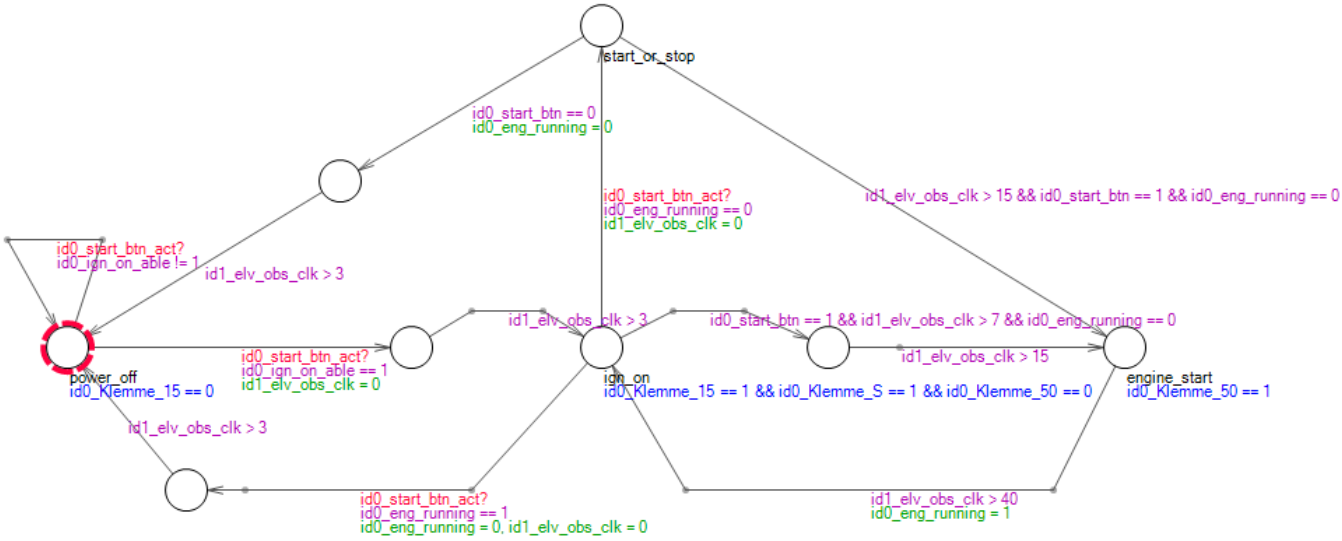

Figure 8. Ignition System Observer

KESSY system, described by specification in Section *VIII.A*, is implemented as three simulation models executed by NI VeriStand Real-Time engine. Partitioning of the SUT into multiple models better reflects distributed nature of a real automotive system. Communication between models is realized by VeriStand channels. The connection of Taster tool to the SUT is done by test adapter which uses VeriStand .NET API [12].

D. Results

Example KESSY system specification was implemented in LabView as VeriStand simulation models and modeled in UPPAAL using Taster supported language constructs. The implementation was afterwards tested by the Taster. Test step period was 100 ms and frequency of the Primary Control Loop of VeriStand was 50 Hz. The objective of the first set of tests was an error detection capability and it was evaluated by injection of three independent faults into SUT.

Fault 1: KL S goes off after 2s from turning the ignition on.
Fault 2: Short start button press is not recognized while the ignition is on – it is not possible to shut down the ignition.
Fault 3: Car is not able to be locked if both keys are detected outside the car.

Faults were injected by modification of implementation models. All inserted faults were successfully detected. Detailed results are summarized in Tab. 3.

TABLE III. KESSY - FAULT INJECTION RESULTS

Fault	Detected	Time to detect	Trace steps	State of detection
F1	✓	39s	392	ign_on
F2	✓	28s	276	power_off
F3	✓	99s	992	check_locked

Detection time is lengthened by key position templates; they wait for arbitrary time quanta between key position changes. Fault 1 and 2 needs at least one key inside the car to allow switching ignition on. Otherwise, the fault is not detectable. Fault 3 is exposed if Key 1 and 2 detected outside the car. Optimization for time or step count is possible in future work and was not objective of presented work. Algorithm 2 – Random strategy was used for fault detection experiment.

Correct implementation was evaluated by performing nine test runs summarized in Tab. 4. No fault was revealed. Node coverage of complete model was selected as stop condition. The SUT model was not optimized for the fastest possible node coverage.

TABLE IV. KESSY – OVERVIEW OF PERFORMED TEST RUNS

Trace	Strategy	Test Time [s]	Steps	Lock state change	Engine Start-Stop
TR 1	Random	45	446	2	2
TR 2	Random	46	458	10	1
TR 3	Random	195	1950	57	2
TR 4	Relevance Random	152	1525	43	1
TR 5	Relevance Random	36	363	1	1
TR 6	Relevance Random	84	841	25	2
TR 7	Systematic	300*	3005	78	9
TR 8	Systematic	300*	3002	78	9
TR 9	Systematic	300*	3004	79	9

*test run was terminated after 300s because it is not possible to reach node coverage

The testing ability for the specific SUT is expressed by a number of locking state changes and engine start and stop cycles included in a test trace. If these two features are examined, the major part of SUT functionality is tested, because they are conditioned by the correct reaction to key position and door and start buttons. Test runs that use Algorithm 4 (systematic) discovered impossibility to achieve node coverage. The reason is in demand of generation of two short start button presses in a row, which is not possible with systematic exploration. Without two consequent short presses, it is not possible to test switching ignition on and off without engine start.

 ISBN: 978-1-61208-593-7

VI. CONCLUSION

New practical approach to the Online MBT test generation based on the simulation of Timed Automata based model is proposed. The suggested method utilizes well-known Timed Automata formalism as system description language. Test traces are generated directly during the model simulation. Model is simulated by randomized exploration. Testing stimuli are produced by variables included in the model and connected to an SUT by a test adaptor. Correct tested system behavior is checked by invariant conditions. Three variants of exploration algorithm are presented. The proposed concept is implemented as a part of MBT tool Taster. The implementation utilizes UPPAAL tool as model editor and uses a subset of UPPAAL modeling language in UPPAAL 4 file format. On the other hand, *Relevance* parameter coupled with nodes was defined to allow prioritization of certain part of the state space. The target application is the testing of comfort part of vehicle electronics systems, such as door locking, exterior/interior lighting, and air condition. Test adapters for NI VeriStand and EXAM were implemented for connection to a testbed. A case study on a KESSY system was done to validate expected Taster capabilities. Results are promising and indicate the suitability of presented approach for automotive testing application. Last but not least, utilization of UPPAAL models allows performing formal verification besides the testing for the specific part of the SUT.

VII. FUTURE WORK

Presented work can be viewed as a basic step or creation of background for future research in the area of automotive integration testing. Implementation of proposed concept had to address many challenges. The range of performed works was too broad to addressed individual problems thoroughly. Further work is planned to be much specific. In the following paragraphs, two of these specific objectives are outlined.

Described solution is presented as a complement to the original test suite. Let's presume it is developed in EXAM. Both suites should not contain similar test cases. The question is how to algorithmically analyze Taster test runs and EXAM test cases in a comparable way. Comparison of the content of Timed Automata traces with sequence diagrams, or underlying Python code will be useful for assurance of required diversity between Taster and EXAM test cases. Solving of this problem is necessary for the truly synergic effect of the deployment of presented Taster tool beside human developed test cases.

The concept of labeling of Timed Automaton states by *Relevancies* numbers was proposed in System modeling section. In this work are these number assigned manually by an expert knowledge. This concept could become powerful with a machine extraction of the *Relevancies*. Source of information could be some test hypothesis or test results database.

ACKNOWLEDGMENT

We would like to thank former master degree student Tomáš Grus for the first implementation of the test tool and the help with the project in the beginning.

This work was supported by the Grant Agency of the Czech Technical University in Prague, project Model-Based Testing methods for automotive electronics systems (Grant No. SGS16/171/OHK3/2T/13) and by the support of EU Regional Development Fund in OP R&D for Innovations (OP VaVpI) and The Ministry of Education, Youth and Sports, Czech Republic, project # CZ.1.05/2.1.00/03.0125 Acquisition of Technology for Vehicle Center of Sustainable Mobility.

REFERENCES

[1] M. T. B. Waez, J. Dingel, and K. Rudie, "A survey of timed automata for the development of real-time systems," *Computer Science Review*, *9*, pp. 1–26, doi:10.1016/j.cosrev.2013.05.001

[2] K. G. Larsen, M. Mikucionis, B. Nielsen, and A. Skou, "Testing real-time embedded software using UPPAAL-TRON: an industrial case study," In Proceedings of the 5th ACM international conference on Embedded software (EMSOFT '05). ACM, New York, NY, USA, pp. 299-306. DOI=http://dx.doi.org/10.1145/1086228.1086283

[3] A. Hessel, and P. Pettersson, "CoVer-a real-time test case generation tool," In: 19th IFIP International Conference on Testing of Communicating Systems and 7th International Workshop on Formal Approaches to Testing of Software, 2007

[4] J. Sobotka, and J. Novak, "Automation of automotive integration testing process," Proceedings of the 2013 IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems, IDAACS 2013, pp. 349–352. doi:10.1109/IDAACS.2013.6662704

[5] J. Peleska, E. Vorobev, F. Lapschies, and C. Zahlten, "Automated model-based testing with RT-Tester," Technical report, University of Bremen. 2011-05-25

[6] EXAM, MicroNova AG, 2017-09-04, URL:http://www.exam-ta.de/en.html, Accessed: 9, 2017, (Archived by WebCite® at http://www.webcitation.org/6tESeHUn6)

[7] J. F. Groote, T. W. D. M. Kouters, and A. Osaiweran, "Specification guidelines to avoid the state space explosion problem," *Software Testing, Verification and Reliability*, vol. 25, no. 1, pp. 4–33, 2015. http://doi.org/10.1002/stvr.1536

[8] K. G. Larsen, P. Pettersson, and Wang Yi, "UPPAAL in a nutshell," *International Journal on Software Tools for Technology Transfer (STTT)*, 1, pp. 134-152, 1997

[9] T. Grus, "Implementation of Integration Testing Test Cases Generation Tool," Master's Thesis, CTU in Prague, 2014.

[10] T. Parr, "*The Definite ANTLR 4 Reference*," *The Pragmatic Programmers*, http://doi.org/10.1016/j.anbehav.2003.06.004

[11] N. Reed, "The Shunting-Yard Algorithm," 2017-09-04, URL:http://www.reedbeta.com/blog/2011/12/11/the-shunting-yard-algorithm/, Accessed: 9, 2017. (Archived by WebCite® at http://www.webcitation.org/6tETNRkl3)

[12] "NI VeriStand™ .NET API Help," National Instruments, URL:http://zone.ni.com/reference/en-XX/help/372846J-01/vsnetapis/bp_vsnetapis/, Accessed: 9, 2017, (Archived by WebCite® at http://www.webcitation.org/6tETbzWi6)