Model-Driven Testing based on Markov Chain Usage Models in the Automotive Domain

Sebastian Siegl, Winfried Dulz, Reinhard German, Gerhard Kiffe

Model-Driven Testing based on Markov Chain Usage Models in the Automotive Domain

Sebastian Siegl, Winfried Dulz, Reinhard German
Department of Computer Science 7
University of Erlangen-Nuremberg
Martensstrasse 3, 91058 Erlangen, Germany
{sebastian.siegl|dulz|german}@informatik.uni-erlangen.de

Gerhard Kiffe
AUDI AG
85045 Ingolstadt
gerhard.kiffe@audi.de

***Abstract*—The Extended Automation Method (EXAM) is employed by AUDI AG - one of the leading car manufacturers in Europe - to design and execute test cases for the purpose of component and system-testing. Test cases are specified using platform independent UML sequence diagrams. In order to provide an executable test suite platform dependent code is automatically generated from a formal description. The test case execution itself is mainly performed on Hardware-in-the-loop (HIL) simulators. The main drawback of the existing procedure in EXAM is correlated to the fact that each test case must be created individually, which is apparently awkward and error-prone. Furthermore the development of increasingly complex and interconnected features poses new challenges to the prevalent validation and test routines in the industry. We therefore have evaluated whether the appliance of a Markov Chain Usage Model (MCUM) would help to overcome this drawback. The MCUM was used to describe at best all possible usage scenarios of the System Under Test (SUT) and provided a basis to derive systematically test cases without human interaction. These test cases can be further processed inside EXAM. This approach makes it also possible to get both indicators for the test-planning in advance and to obtain dependability measures after the test cases have been processed. The usage of MCUMs in a fully automated process has been demonstrated within the QoS Enhanced test Development (Q.E.D.) test framework. In addition, the automotive domain requires a variety of problems to be resolved.**

Keywords: **Software Testing, Automated Test Generation, Model-based Test Generation, Test coverage of specifications, Test execution, Testing strategies, Automotive Systems, Electronic Control Units**

I. Introduction

Driven by customer and mandatory requirements more and more complex and interconnected functionality in terms of software is installed in modern cars. This development emphasizes the need to establish test methods that can cope with this development.

A. The EXAM Testing Method

The test method used by the AUDI AG and within the Volkswagen AG to perform tests at component and system levels is called Extended Automation Method (EXAM) [1]. EXAM defines a process, the roles, and the tools used to

1) model test cases graphically and platform independently in UML. Sequence diagrams are used for this task and build the formal basis for test case specifications.
2) generate platform dependent test scripts automatically from the formal description in UML. In this way a separation between the test case description and its concrete implementation is achieved.
3) to use sharable test automation functionalities from a structured database. Thus, test cases can be developed in independent test teams and test know-how is accumulated enterprise-wide.

So far, test automation in the scope of EXAM means the automated generation of platform dependent code and the execution of the derived test suite without human interactions. However each single test case must be invented, be developed, and be designed by the person which is instructed to test the SUT. This procedure has many drawbacks, e.g. the estimation of the test coverage implies a very hard task. Moreover it can not be prevented that one or more important test scenarios remain undiscovered.

B. MCUMs and the Q.E.D. process

MBT (Model-based Testing) techniques make use of formal descriptions (models) of either the SUT or the expected usage of the users of the SUT. In the former case a behavioral specification is derived from the requirement definitions and serves as a starting basis to automatically generate test cases in order to test the SUT [2]. In the latter case usage models are deduced from the requirements and may be considered as independent of the specification. Because exhaustive testing of real systems is infeasible in practice an appropriate set of test cases is derived to accomplish a given test goal. At this point Markovian statistics come into play.

In a MCUM the possible usage at a specific abstraction layer is represented by a Markov chain that consists of states and transitions between states [3]. This graph structure describes all possible usage scenarios for a given SUT. Transitions are augmented by probabilities from a usage profile that reflects usage choices users will have when they interact with the SUT. It is possible to provide more than one usage profile for a given Markov chain structure [4]. Therefore, the statistical selection of test cases may also consider distinct properties of different

roles or user classes, e.g. car occupants, in interaction with the SUT. Furthermore the results of statistically derived test cases can be used for the calculation of dependability measures of the SUT [5]. The *Miller Reliability Model* offers a possibility to assess the dependability of a system even in the case of the non-observance of failures, making use of pretest reliability information[6].

MCUMs have been successfully employed in the Q.E.D. test development process. Q.E.D. defines a complete process for statistical testing that starts from formal UML use case descriptions [7], [8]. Q.E.D. enables the automatic generation of test cases for both testing functional and non-functional requirements. The central structure of the framework is represented by a MCUM, which is generated automatically from use case scenarios and can be employed to derive test cases on-the-fly inside an executable TTCN-3 test suite as shown in Fig. 1.

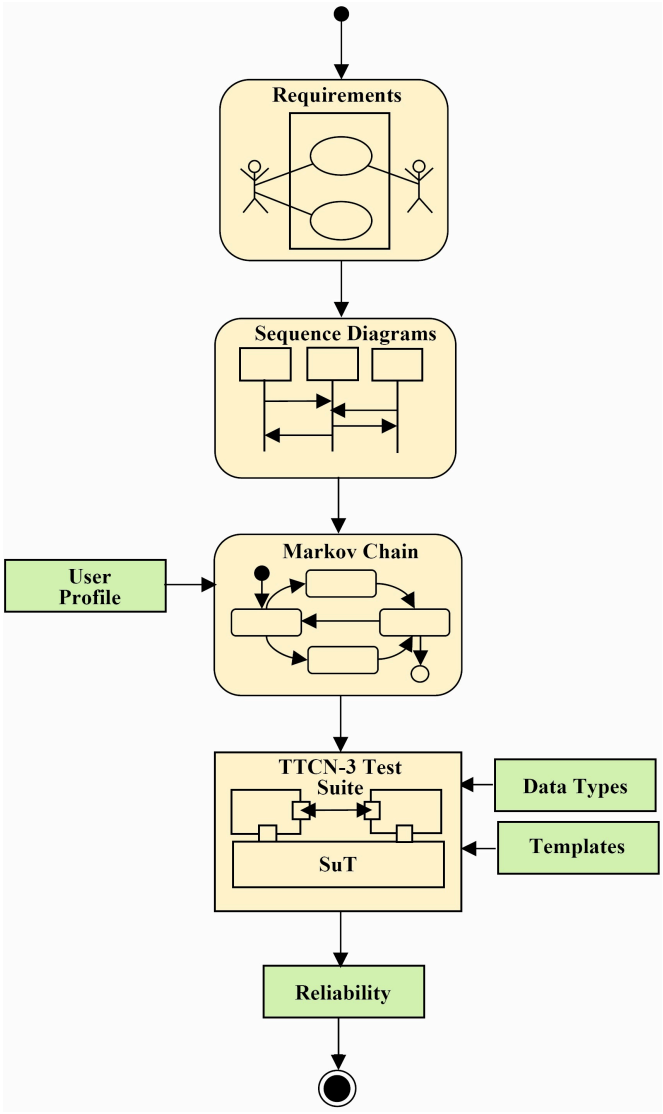

Fig. 1. Q.E.D. process: on-the-fly generation of test cases

The results of statistically generated test cases can be used to obtain specific metrics of the SUT or e.g. the Kullback-Leibler divergence with respect to the given profile, the mean number of test cases to reach a certain confidence level or even the expected reliability of the future system [9]. In this paper we focus on the automatic generation of test cases from a MCUM usage model that can be further processed inside the EXAM tool chain.

One has to keep in mind that in the automotive domain the input domain of a SUT consists of operations by the car driver and its occupants, of environmental influences and of stimuli that have their origin in other components of the car. Furthermore, one has to keep in mind that the HIL test benches should be used as efficiently as possible because testing time is an expensive resource. The main goal is therefore to validate the specified functionality of the SUT by using a minimal sample of test cases that also comprises extreme and unlikely conditions.

For that reason we checked whether the existing test method EXAM could be enhanced by a test case generation out of a MCUM. Test cases should be processable by EXAM tools and fit the requirements from the automotive domain at best. All in all, the whole test process should benefit from the availability of a MCUM in order to pre-assess the coverage and test-depth achieved by the selected test cases.

II. General Approach

Our main goal was to enhance the EXAM test process by a method to automatically generate test cases for system testing from given requirement documents. In this extended approach, the invention and manual description of each single test case is substituted by the modeling of a single MCUM. The primary function of the usage model is to create specific test suites that contain test cases according to preset test purposes and conditions. For example, characteristic test aims can be time constraints for the maximum test duration or exceeding the bound of a dependability indicator. Moreover, benefits from existing methods should be combined and preserved in an optimal manner.

The method to construct the MCUM followed the principles of sequence-based software specification [10], that has already been applied in many application domains, e.g. for testing a feedback control system in Simulink [11]. It consists basically of the following steps:

1) Identify the system boundary
2) Enumerate all sequences of stimuli and their responses across the system boundary

This process is explained in more detail. First the system boundary of the SUT is defined. Based on the requirements all stimuli that go across this boundary and involve the functionality to be tested are extracted. This comprises stimuli S that represent operations on the SUT and responses R from the SUT. Next all theoretically possible sequences $S*$ of stimuli to the system are enumerated and mapped to a response, stated as $u \rightarrow r$. The enumerations are ordered ascending by their length. This process follows specific rules.

Starting with the empty sequence λ new sequences v are derived from the existing u by increasing their length through concatenation with any legal $x \in S$. Illegal sequences are such that are either impossible to occur, never invoked or not observable by definition of the functionality to be tested.
Two sequences u and v are called equivalent if extended by any sequence w the future response is the same to uw and vw. In this case u is said to be reduced to v and not extended further. Summarized, only legal and unreducible sequences are extended.
By this procedure, a complete and consistent usage model is created. An additional feature is the easy traceability of requirements. Beside the annotation of the response to each stimuli sequence u the corresponding requirement is assigned. This ensures the correctness of the enumeration and in doing so incomplete and inconsistent requirements are discovered. If no requirement or desired system's response is found for a sequence this must be documented and a requirement must be derived. So this procedure describes also a technique to analyze the requirements.

The procedure to draft the MCUM followed this approach. In a first step the system boundary of the SUT, i.e. in this case of the Start-Stop functionality, is to be identified. Next all stimuli that have influence on the functionality were extracted from the specification. A stimulus in this context is always represented by a discrete event that has an effect on the SUT's behavior. These stimuli can have their origin in different "users" of the functionality, where a user may be the car driver, other components of the car, and environmental influences such as temperature. In the latter case, continuous changes are mapped to events that provide the new values.
After this, an initial usage state is defined. Starting with this state one identifies all possible stimulus sequences that can occur. A system response is assigned to every possible stimulus sequence. Stimulus sequences are considered as equivalent, as mentioned before, if they provoke the same system's behavior when extended by the same non empty stimulus sequence. In this case they lead to the same usage state. An example for such a usage state in our case study is given by the usage state after which the car has come to a standstill and the ignition has been turned off and on. The extension of this usage state by many different stimuli sequences led to the same response, independent of the way this usage state has been reached. In this case these sequences are considered as equivalent with respect to their future behavior, so they all lead to the same usage state. An appropriate counterpart of the EXAM repository is assigned to each transition, i.e. a platform independent interface for the stimulus or a sequence of stimuli. This way the usage description is kept generic and the specific implementation is resolved when the target system for each test case is chosen. In the case of using equivalence classes for stimuli values one representative can be selected later on. The application of equivalence classes makes it easy to combine test vectors together with the MCUM in a manner that one can define actual parameter values in a further step. The procedure is continued by adding at each usage state every possible and legal operation that could be performed at this usage state. After that, it has to be considered what the expected reaction of the SUT would be if this sequence of stimuli were executed. If it is observable, a description of it or a method to check the expected behavior is added to the transition. As a next step it has to be considered which usage states are reachable next. This procedure is continued until all reasonable sequences of stimuli can be found in the MCUM.

Measurements and checks on the HILs, e.g. evaluations of signal curves are performed in various ways. Unfortunately, a fully automated approach starting solely from the MCUM to establish all necessary actions seems to be hardly feasible. Platform independent information to customize the monitoring process can be entered into the MCUM in a semi formal way. Remaining steps in order to prepare the execution are performed on the EXAM layer. These aspects are not considered during usage modeling, because they form a part of the evaluation and not of the usage. The test person that runs the test case can then decide which check-methods to use from the EXAM repository in a step after the generation of the test cases and before the automated execution on the test bench.

The MaTeLo toolkit (www.all4tec.net), which is a commercial offspring of the European IST project MaTeLo [12], has been chosen as the main facility to design the MCUM. By using the MaTeLo graph editor it is relatively easy to create a structured usage model that consists of state groups and macro-states that further lead to referenced submodels. The structuring concept is not only important to keep the model easily understandable and manageable but also helps to specify information for the composition of the test cases in the EXAM process. According to the EXAM modeling rules test cases are structured and consist of three parts. First, the SUT is initialized with a starting state that builds the basis for the following test actions. Next, the actual test sequence is described and in the third part it is checked whether the SUT behavior confirms to the expected results. Macro-states are applied to assign adherent nodes and transitions to one of these describing parts. In this way, all derived test cases automatically obey the EXAM test case structuring rules.

Test case generation can be performed automatically by random walk based on the stochastic profile information or by a graph-based minimal arc coverage algorithm. In the latter case a test suite is created where each transition is traversed at least once. All derived test cases are described in an adapted XML dialect in order to pass the necessary information to the EXAM initialization procedure and to process it automatically. Test sequence steps and stimuli calls of test cases that are described in XML are imported automatically into EXAM in this way. The overall procedure is depicted in Fig. 2. Compared to the Q.E.D. process where an executable test suite can directly be generated from the MCUM, the test cases derived from the MCUM are sequence diagrams that have to be completed in a pre-execution step with administrative information, measurements, and monitoring handlers. Equipped with these information the test cases are ready for execution within EXAM.

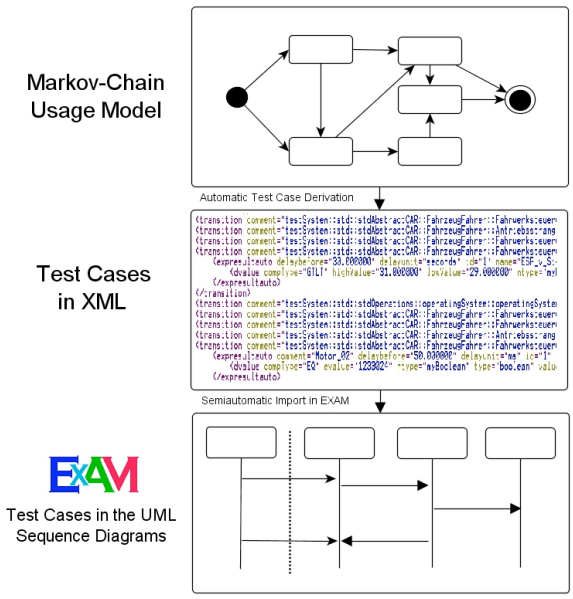

Fig. 2. EXAM enhanced by a MCUM

III. Case Study Start-Stop Functionality

In order to evaluate the MCUM-driven approach we have chosen the Start-Stop functionality because it represents a fairly complex application scenario and requires many ECUs (Electronic Control Unit) to interact via car communication networks. Start-Stop performs an automatic shutdown of the engine during longer standing periods and an automatic restart [13], which reduces the fuel consumption and noxious emissions of the car. Start-Stop is highly interconnected within the ECUs of the car, and the implementation is leading to a high system complexity. This is mainly caused by the requirement that the motor shutdown and restart should happen as imperceptible as possible and the familiar functionality and comfort of the car should be sustained.

Thus, for testing the Start-Stop functionality we had to consider many entities performing the Start-Stop function within the car as well as other ECUs of the car and environmental influences. Furthermore, we assumed high level evaluation functions in EXAM in order to keep the abstraction level in the MCUM consistent.

The MCUM for the Start-Stop functionality was designed by using the MaTeLo graph editor. It consists of about 300 states and covers the functional specification in a car with manual gearshift control. The specification consisted mainly of textual descriptions in a requirement management system plus graphs and tables that bring out certain aspects of the textual specification. Traceability of requirements was achieved by linking requirements with transitions and making use of state groups to structure them. In Fig. 3 the central part of the MCUM is depicted. It describes usage that affects the start-stop functionality, including inputs from other ECUs in the car and environmental influences.

Already during the modeling of the MCUM deficiencies in the specifications could be identified. The detection is straightforward, because following each possible usage sequence the expected system's reactions are described based on the requirements specification. During this procedure could be identified

- one clearly misplaced erroneous specification,
- three contradictions within the requirements specification and
- ambiguities in the specification, that have origin in the omission of possible usage scenarios.

Naturally the identification of all these deficiencies is important because they can lead to unexpected or even undesired behavior of the final system. This demonstrates that as early as possible potential implementation errors due to an erroneous behavior model can be identified and avoided even before test cases are executed in order to test the SUT.

Test cases have been derived via random walk based on the profile information. Since no statistics for the future usage behavior have been available we have assumed a uniform probability distribution, which guarantees maximum entropy. Applying 100 test cases an arc coverage of already 78% was achieved and resulted in a mean test case length of 49 transitions. The shortest possible path traversing the MCUM required 31 transitions.

In addition, an algorithm was employed to create a test suite for minimal arc coverage. The generated test suite consisted of 37 test cases with a mean test case length of 42 transitions. This is quite a good result because each transitions is traversed at least once in a comparably small number of test cases. Therefore, this test suite can be used as the minimal basis for testing the Start-Stop functionality.

Considering the test requirements in the automotive domain new coverage algorithms for distinct testing criteria have been invented. The generation of a test cases for arc coverage is often performed by algorithms that address the Chinese Postman Problem [14]. These work fine, however tend to create a few but lengthy test cases. Thus the known established algorithms for the derivation of test cases for coverage omit the fact that with the length of an offline-executed test case the risk increases that requirements are left untested during a test run. This is due to the fact that a test case is aborted when an error occurs. If this happens at an early test step and the whole test case is aborted many steps remain untested and a new test run has to be set up to test them. A family of algorithms has been developed to overcome this drawback. Mainly they aim for creating test suites for coverage consisting of shortest individual test cases with regard to a cost function. This cost function can be number of steps required or the estimated execution time of a test case. Also the coverage not of the whole model but of distinct states and transitions with minimal length test cases - w.r.t. a cost function - is made possible [15]. These are based on A^* and Dijkstra's algorithms. Their enhancement and integration into EXAM makes part of our future work.

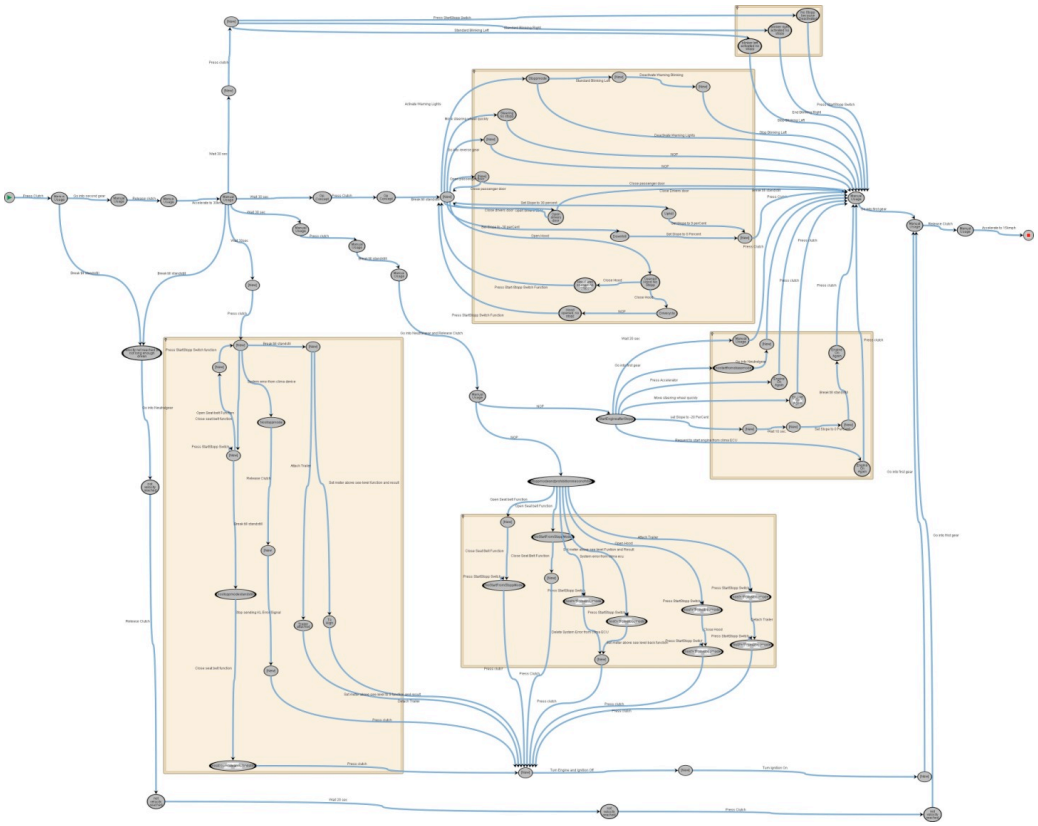

Fig. 3. Central part of the MCUM for the Start-Stop scenario

For providing an objective assessment of the extended EXAM approach test cases for the Start-Stop functionality were created manually by an experienced test designer. The resulting test cases were compared to those that have been automatically derived from the MCUM. The analysis revealed that test cases derived from the MCUM by random walk generation covered a broader spectrum of operation sequences that may occur in real life scenarios than those, which have been manually created. Among the random test cases we also found situations that otherwise would not have been tested. These experiments have shown that by means of a MCUM real life scenarios were revealed that can occur in practice and which are difficult to anticipate in advance by pure consideration.

Furthermore, a comparison with test cases resulting from the algorithm to achieve minimal arc coverage identified scenarios that were not contained in the crafted test cases. Due to this fact scenarios exist that would have probably been neglected using the established EXAM method and therefore could not be tested.

It is hardly possible to compare the time needed for creating the MCUM and sketching the test cases manually in an exact manner. This is correlated to the fact that creating a MCUM imposes another quality in modeling, since it implicates requirements analyses as stated in section II. Moreover, it is difficult to compare the efforts of the manual way directly with the efforts expended by our approach. This is due to the fact, that not only the efforts but also the output of the methods would have to be made comparable in a gaugeable manner, which is obviously a hard task. Though we can claim from our experience that the effort for creating the MCUM and deriving the test cases has been less or equal than creating the test cases manually and we achieved the stated improvements.

All in all many benefits have been determined by applying a MCUM, i.e deficiencies in the specification could be identified even before test case execution, new test scenarios could be obtained, and gaps in the test case coverage could be closed. However, we did not establish a fully automated way yet to be able to derive test cases from a MCUM at the considered abstraction layer. Questions for reaching this goal are not resolved yet, e.g. how to establish full traceability from the EXAM repository back to the MCUM and how to enable regression testing based on the MCUM. Furthermore, it is difficult to sustain the established modularity, reusability and thus exchangability of test sequences if one would pursue the fully automated approach.

Hence, our future goal is to create an architecture, which keeps and reflects the benefits of the existing EXAM process.

IV. Future Work

In our future work we will examine how to make MCUMs an integrated part of EXAM. Two promising approaches into

this direction arised from our work. The first one pursues the extended approach presented in this paper but does not provide a complete automated process starting from the MCUM up to the test case generation (cp. Fig. 4). Instead, the usage of a MCUM as a tool for improving the test management and detecting test cases should be further optimized. Just as specifying single test cases in the UML our future research is focusing on describing a MCUM for EXAM completely in the UML, including functional and non-functional requirements.

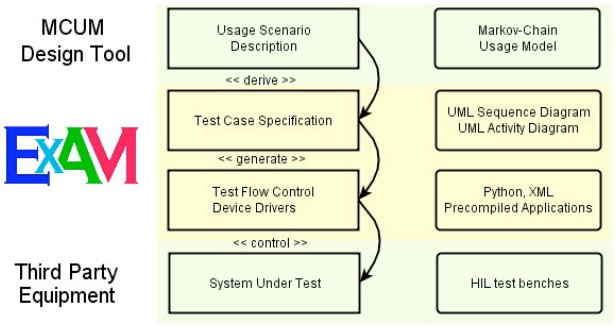

Fig. 4. EXAM enhanced by MCUM for test case generation

In the second approach MCUMs in EXAM are employed at the same layer as sequence diagrams (cp. Fig. 5). In this way, the automatic derivation of dependability measures [6] will be possible and testing time could be dynamically adjusted to the realization of desired test targets. In addition, recorded usage data of components can be used to calculate specific usage profiles, which control the actual test case generation and thus, the executed stimuli and evaluation sequences. In this way real usage data could be directly used to test a new system or a future system release.

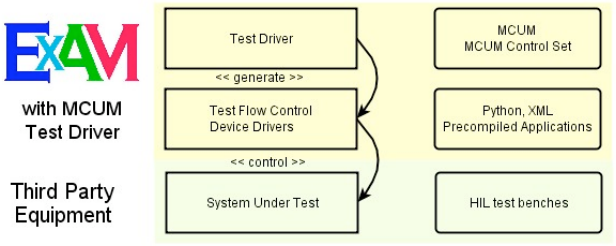

Fig. 5. EXAM and a MCUM as test driver

Optimizing the test case derivation is also planned in our future work. We intend to develop algorithms that allow the systematic discovery of test cases according to specific test benches and test requirements. Hereby, the main benefits of using a MCUM, e.g. the calculation of dependability indicators will be sustained.

V. Conclusions

In this paper we have demonstrated how to apply a MCUM for the automatic deduction of test cases in the automotive domain. In our Start-Stop case study substantial advantages are discovered that result from creating a MCUM usage scenario. Main findings are that even before executing the test cases deficiencies and ambiguities in the system specification can be identified. Potential failures can be detected and clarified in an early validation step even before a single test case will be executed. Test cases derived by random sampling from a MCUM describe usage scenarios that would otherwise not have been tested. The test suite for minimal arc coverage can be used to close coverage gaps in a manually created test suite. A fully automated test case generation process starting from the MCUM up to the execution on the test bench has not been established yet. Our goal was the optimal enhancement of the EXAM process by an automatic test case generation from MCUMs. We also have detected perspectives for algorithms and future testing architectures that are more promising though they do not provide a fully automated test case generation environment.

References

[1] G. Kiffe, *EXAM Konzeptpapier*, Audi AG, Ingolstadt, Dezember 2007.

[2] S. Rosaria and H. Robinson, "Applying models in your testing process," *Information and Software Technology*, vol. 42, pp. 815–824, 2000.

[3] J. A. Whittaker and M. G. Thomason, "A Markov chain model for statistical software testing," *IEEE Transactions on Software Engineering*, vol. 20, no. 10, 1994.

[4] S. Prowell and J. Poore, "Computing system reliability using Markov chain usage models," *J. Syst. Softw.*, vol. 73(2), pp. 219–225, 2004.

[5] W. J. Gutjahr, "Software Dependability Evaluation Based on Markov Usage Models," *Performance Evaluation*, vol. 40, no. 4, pp. 199–222, 2000. [Online]. Available: citeseer.ist.psu.edu/gutjahr00software.html

[6] K. Sayre and J. Poore, "A Reliability Estimator for Model Based Software Testing," in *ISSRE '02: Proceedings of the 13th International Symposium on Software Reliability Engineering (ISSRE'02)*. Washington, DC, USA: IEEE Computer Society, 2002, p. 53.

[7] M. Beyer, "Q.E.D. - Ein Entwurfsprozess fuer statistische Tests mit Betrachtung von Zeit- und Leistungsanforderungen," Ph.D. dissertation, Department of Computer Science 7, University Erlangen-Nuremberg, 2008.

[8] M. Beyer, W. Dulz, and K.-S. J. Hielscher, "Performance Issues in Statistical Testing," in *Proceedings of 13th GI/ITG Conference on Measurement, Modeling, and Evaluation of Computer and Communication Systems (MMB 2006)*. VDE Verlag GmbH, 2006, pp. 191–207.

[9] S. J. Prowell, "Computations for Markov Chain Usage Models," Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, USA, Tech. Rep., 2000, uT-CS-03-505.

[10] S. J. Prowell and J. H. Poore, "Foundations of sequence-based software specification," *IEEE Trans. Softw. Eng.*, vol. 29, no. 5, pp. 417–429, 2003.

[11] J. M. Carter and J. H. Poore, "Sequence-based specification of feedback control systems in simulink®," in *CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*. New York, NY, USA: ACM, 2007, pp. 332–345.

[12] W. Dulz and F. Zhen, "MaTeLo - Statistical Usage Testing by Annotated Sequence Diagrams, Markov Chains and TTCN-3," in *IEEE Proc. of Third International Conference on Quality Software (QSIC 2003)*, 2003, pp. 336–342.

[13] *Systemlastenheft Start-Stopp-Funktion*, Volkswagen AG, 38436 Wolfsburg, March 2008.

[14] H. Thimbleby, "The directed chinese postman problem," in *In journal of Software Practice and Experience*, 2003, p. 2003.

[15] S. Siegl, "Evaluation of Model-Driven Test Case Generation based on Markov-Chain Usage Models in the Automotive Domain," Master's thesis, Department of Computer Science, University of Erlangen-Nuremberg, Germany, June 2008.