Using Reference Traces for Validation of Communication in Embedded Systems

Falk Langer, Erik Oswald
Fraunhofer Institute for Embedded Systems and Communication Technologies ESK
Munich, Germany
Email:{falk.langer,erik.oswald}@esk.fraunhofer.de

***Abstract*—This paper addresses the problem of evaluating the communication behavior of embedded systems. An important problem is missing, wrong or incomplete specification for the interaction in the distributed system. In this paper, a new approach for evaluating the communication behavior based on reference traces is introduced. The benefit of the approach is that it works automatically, with low additional effort and without using any specification. The introduced methodology uses algorithms from the field of machine learning to extract behavior models out of a reference trace. With the presented algorithm, the complexity of the learning problem can be reduced significantly by identifying parallel execution paths. The efficiency of the proposed algorithm is evaluated with real vehicle network data. At this data the self-learning algorithm covers up to 69% of the behavior from the presented trace.**

Keywords*—embedded system validation, testing procedures,network trace analysis, self-learning test methods

I. INTRODUCTION

This paper focuses on test and validation of the communication behavior from embedded systems. In systems with highly distributed functionality like it can be found in modern car's electronics, the communication behavior is an important aspect on system validation. At a cars development cycle, it is important to analyze the network traffic in a fully assembled car. Even if all single electronic control units are tested exhaustively, a significant portion of remaining bugs resulting in errors or malfunction is lately found at real driving tests. Because network traffic represents the internal behavior of a distributed system, its analysis can help to detect possible bugs earlier and faster. But especially on system level test it is not easy to rate about the correctness of communication at the network.

The most important problem of ensuring the correct interaction at system level is missing, wrong or incomplete specification of the interaction of functions in the distributed system (compare [1] and [2]). There are many works of research in progress that tries to improve the process of creating system specification, with the goal of building better test cases for validating the communication on system level. Nevertheless it is still an extensive process to get sufficient test models.

In this paper, a new approach for evaluating the communication behavior automatically, with low additional effort and without using any specification will be presented. The goal is to detect problems early, best before detectable errors or malfunctioning occurs. The proposed approach shall help to detect these remaining bugs faster without a significant increasing of testing effort.

This paper is structured as follows. Section II gives a short overview of the state of the art and the gaps that the proposed approach addresses. Section III describes the expected usage, benefit and outcome of the approach. Section IV provides the technical background of the learning problem. In Section V, the methodology for identifying parallel execution paths in traces is discussed an evaluated and Section VI offers an optimization. The paper closes with section VII that presents the conclusion and future work.

II. OVERVIEW AND RELATED WORK

The car's network can be seen as a closed but distributed system. The network behavior mostly depends on sensors and actors and its input or output, which results from different environment or user interaction. Therefore, in the most cases it is only possible to observe the communication behavior. Because of the nature of a closed system, it is not possible to stimulate a behavior on network level and evaluate the response. To rate about the correctness of network communication it is necessary to build more or less passive observer models. To build such models it is important to have a detailed description or specification of the communication protocols between the applications. In difference to well-known protocols like TCP/IP, this is a kind of *meta* communication protocols because they are mostly not noticed as a protocol. In [3], *meta* states that a communicating systems can internally take place, are described and it is pointed out that this meta states are often the cause of malfunction because they are mostly not known.

Basically, it did not surprise that one of the main causes of malfunctions detected at system level test is wrong, incomplete or missing specification (compare [1] and [2]). Therefore, the focus of most research projects working on testing network behavior is to enhance the specification. The key aspects in research are requirement engineering and its interaction with test methods. For this reason a popular approach is to use additional description languages to describe the systems behavior more accurate and build better test cases([4, 5]). Another approach for getting better specification is the automatic update of specification from already developed systems ([6, 7]). This shall help to get the specification up-to-date and provides the tester an overview about yet not specified behavior. Obviously, this approach stands in contradiction to top down software engineering methods like the V-Model ([8]), which is very popular in embedded systems development. Nevertheless incomplete specification is an unavoidable problem in software engineering and because of this reason nearly all methods that help to close this gap, will enhance software quality.

 ISBN: 978-1-61208-319-3

In this background, the focus of the proposed method within this paper, is the analysis of network communication without the need of specification as it is described in [9]. The communication is recorded within a trace which can be analyzed offline. Therefore, a trace represents all data observed in the network within a given time. The proposed method basically uses reference traces as replacement for missing specification. A reference trace represents the allowed behavior or the possible states a system can take place at the surrounding influences provided to the system at recording time. If the reference trace represents most of the possible behavior of the system, it could be interpreted as the normal or norm behavior. This comes close to the idea to use examples as specification like it is described in [10], but differs in the kind the specification is represented.

III. EXPECTED OUTCOME

The goal of this work is to construct a method that allows a qualitative comparison between the reference trace and newly recorded traces with respect to the represented system behavior. The essential outcome of the proposed procedure is the awareness, that the newly recorded network trace represents a new system behavior, which is not represented within the reference trace. If such a behavior is recognized, the method outputs a trigger or some equivalent information to the tester. At this point two potential expectations about the tested network behavior can be made: 1) A newly implemented or just jet not observed behavior was found, or 2) A bug in in communication behavior is detected. Just at this point a system expert has to decide if the proposed method detects case 1) or 2). Surely it is not possible to detect bugs, which are already within the reference trace included, but if no other tests detect these bugs und these bugs did not lead to malfunction, it is not sure if it is a bug or just unspecified behavior.

The described scenario has some analogy to regression tests. But at system test level, regression tests are usually not interpreted or executed on network level. On network level it is only possible to observe some kind of internal system reaction as consequence to external test stimuli. The internal behavior represented within a network trace is hard to interpret. As mentioned before, this is mostly done by using passive reference models ([11, 12]). Because these models are hard to build, in many cases only search of negative examples is done on the network trace. This is mostly a search of error codes or bad sequences, which are known from previous bugs.

With the proposed method a kind of reference model shall be extracted from the reference trace. In comparison to manually build reference models this method comes for free and can be applied without any specification. Therefore, the proposed method shall help to improve the evaluation of network communication at system tests.

IV. THE LEARNING PROBLEM

This section describes the algorithmic foundations and the basic functioning of the proposed self-learning trace analyzing approach. The goal of the approach is the qualitative evaluation of network traces, with the focus on interpreting the sequence of observed events. It was pointed out above that such sequences can be potentially described by protocol automata which are finite state machines. This leads to the basic assumption that a network trace can be described by one or more finite state machines. According to the intention of learning reference models, it is only needed to accept the trace and not to generate it. So, one can use the definition of a 5-tupel acceptor automaton for describing the network trace:

$$A := (\Sigma, S, s_0, \sigma, F) \quad (1)$$

Where: Σ is the input alphabet consisting from events $\in \Sigma$, S a finite set of non-empty-states, s_0 is the initial state with $s_0 \in S$, σ as state transition function with : $\sigma : \Sigma \times S \mapsto S$ and F the set of final states $F \subseteq S$.

It can be pointed out, that a state in A is represented by a sequence of events $\varphi \rightarrow S$ with $\varphi = e_1, e_2 \ldots e_n$ and $e \in \Sigma$. This means that the a learned reference model must predict for any given sequence $\varphi = e_1, e_2 \ldots e_n$ the next event e_{n+1}. This can be repeated in an unlimited manner that $n \rightarrow \infty$, which means that a sequence is potentially endless.

Another important expectation about the network behavior results from the paradigm of parallelism in distributed systems ([13]). This results in the expectation that there exist several independent automata A_i with disjoint input alphabets. A trace would then be observable by an automata A^*, which is a product of all automata A_i

$$A^* = A_0 \times A_1 \times A_2 \times \ldots A_i \quad (2)$$

with $\Sigma^* = \biguplus \Sigma_i$ (non overlapping alphabets)

These assumptions describe a basically system hypothesis for the network trace. With this hypothesis it should be possible to describe the learning problem, which is the first step to find applicable learning algorithms. If this hypothesis is correct the network trace would consist of several sub traces describing the execution path of a single automaton A_i.

For learning sequences even if they are infinite long, a lot of algorithms can be used. For example neural networks ([14,15]), Markov chains ([16, 17]) and Angluin Style automata learning ([18]) algorithms are usable. It was shown that the fundamental problem by applying these learning algorithms, is the parallelism resulting from (2). This leads to an exponential growing of complexity with the number of parallel executed automata. In ([18] this effect was shown by using a CAN trace from a cars powertrain.

With these results it can be pointed out that the major problem for learning behavior or reference models, is the identification of parallel execution paths within the reference trace. If it is possible to extract group off events, where each group belongs to an independent executed automata, the complexity of the learning problem can be reduced exponential.

V. IDENTIFYING PARALLISM

The grouping of events can be seen as a clustering problem. Clustering means to identifying groups of elements with most similar properties. But what are the properties of events belonging to the same execution path? It is most likely the

 ISBN: 978-1-61208-319-3

property that these events are generated by the same automaton. But this property is not directly measurable. Because of that, the properties of an event need to be checked if they give an advice to that not measurable property.

There are only two properties of an event that are somehow usable for this problem. The first one is the recording time of the event and the second one is the information theoretic probability of an event. For clustering the usage of these two properties is not straight forward. In the following, the usage of the recording time within clustering algorithms will be introduced. The proposed method basically interprets the timestamp by applying them to a behavior model and extract new features that are more suitable for clustering methods. This kind of process is called feature extraction or feature construction [19].

A. Feature construction based on timestamps

The absolute value of the timestamp of an event is not usable for clustering because a dedicated event can occur several times in a trace. But for clustering it is important to calculate a distance between two different events. Additionally the calculated distance must be related to the problem of identifying groups of events that belonging to the same execution graphs. To interpret the timestamps in that way, a hypothesis of the system is needed, that explains the values of the timestamps. For this reason the definition of acceptor automata (1) needs to be extended to deal with time. Such automata are called timed automata and are invented by [20]. Timed automata extend the classical automata with a finite set of clocks and the transition function with are enriched with guards that control the time when a transition is allowed to be executed. A timed automaton A_t can be defined as follows:

$$A_t := (\Sigma, S, s_0, X, \phi, F) \tag{3}$$

Σ, S, s_0, F are equivalent to (), X is a finite set of clocks, and ϕ replaces the transition function σ. $\phi: \Sigma \times S \times G(X) \mapsto S$ were $G(X)$ describes the timing guard.

This definition is a very general description of timed behavior. To use a model of timed automata for feature construction it is necessary to made some simplification or assumption of a typical use. For this reason it is argued that every transition will take place in in a fixed time. Even if there is more flexible timing behavior allowed or thinkable, it could be expected that in an implementation in most cases a fixed timing values will be used.

With this simplification it is possible to argue about the expected timing behavior of a dedicated event. For a continuously executed automaton where the transitions add a fixed delay between emitted events, it can be expected that the delay in a fixed loop is constant. If the automata have different ways for an execution loop, different but countable delays between the emitting of the same event can be expected. For example consider the automaton in Fig. 1, there the time period t_P between two occurrences of event **a** can be $t_P = t_b + t_e$ or $t_P = t_b + (n+1)t_e + n(t_c + t_d)$ and for event **b** it can be $t_P = t_e + t_a$ or $t_P = t_a + (n+1)t_e + n(t_c + t_d)$. For this example one can see that there would be a good chance that there are time periods of event **a** and **b** that are in the range of $t_P = (n+1)t_e + n(t_c + t_d)$. That means that the automaton takes sometimes the lower path through event $\mathbf{c} \rightarrow \mathbf{d} \rightarrow \mathbf{e}$ before it enters the upper path $\mathbf{a} \rightarrow \mathbf{b}$.

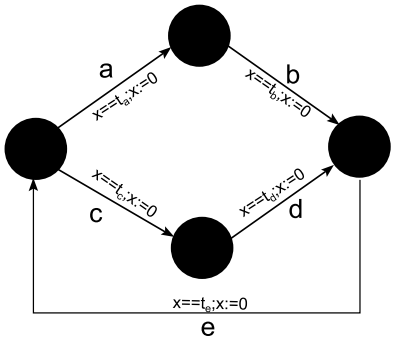

Fig. 1. An example of a looped automaton

If this assumptions are correct, it is supposable that events that belonging to the same automaton having same or similar frequency components. These frequency components should be better usable to calculate a distance between two different events. This distance is later usable for clustering reasons.

B. Calculating the frequency components

A dedicated event can occur several times within a trace. Because of this, an event is located in time. It describes more or less a signal function with different frequencies (see Fig. 2). To get the main frequency components out of the signal curve the most common algorithm to use is the Fourier transformation.

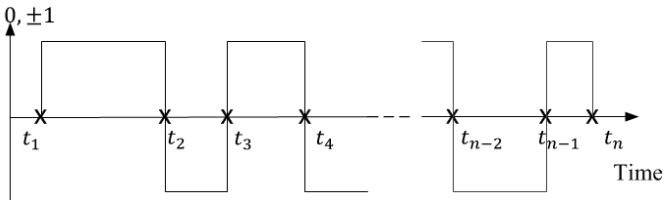

Fig. 2. Signal characteristic of a dedicated event

After applying the Fourier transformation to the signal curve, the result looks like the chart in Fig. 3. Basically after applying the Matlab function "*findpeaks*" several times, until less than 10 frequency components are left, the results looks like in Fig. 4. In the next steps these frequency components are used for calculating a distance between different events.

C. Distance metrics

For clustering a simple density based hierarchical clustering method is used. To get good clustering results the usage of the right distance metric is essential. For this reason four different metrics are tested within the evaluation.

The well-known distance metrics Euclidian, Manhattan and the Hamming distances are used for a first analysis. The definitions of this metrics are as follows:

- Euclidian distance

$$d_E(x,y) = \sqrt{\sum_{i=1}^{max_{freq}} (freq_{x,i} - freq_{y,i})^2} \tag{4}$$

- Manhattan distance

$$d_M(x,y) = \sum_{i=1}^{max_{freq}} |freq_{x,i} - freq_{y,i}| \quad (5)$$

- Hamming distance

$$d_H = \sum_{freq_{x,i} - freq_{y,i} > \varepsilon} 1 \,;\, i = 1, \dots, max_{freq} \quad (6)$$

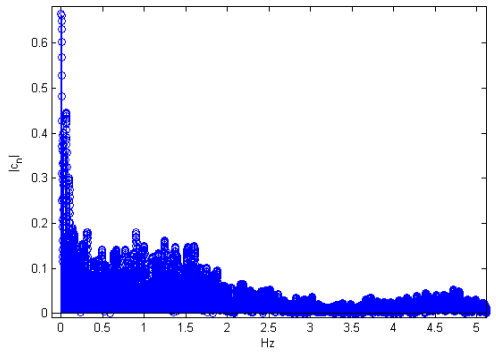

Fig. 3. Unfiltered frequency spectrum

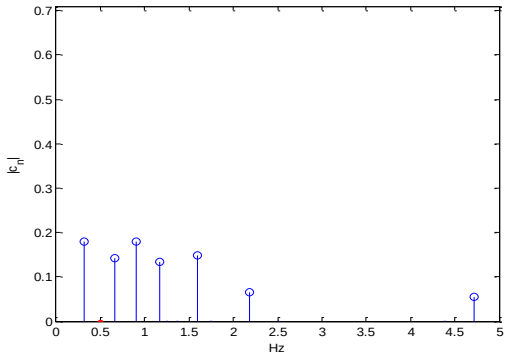

Fig. 4. Filtered frequency spectrum with only 7 main frequencies

It was mentioned before, that if two different events has one equal frequency component the probability that this two events belonging to the same automaton is very high. Because of this, a new distance metric is introduced and will be called in the next *frequency distance*.

This new metric is derived from the hamming distance. The difference to the hamming distance is that it does not care about the ordering of the points that are to be compared. Because for the proposed problem mainly the equality of frequency components is important, not it's ordering.

- Definition of frequency distance

$$d_{freq}(x,y) = max_{freq} - sum\_eqal\_freq_{x,y} \quad (7)$$

With $\text{sum\_eqal\_freq}_{x,y} = \sum_{|\text{freq}_{x,i} - \text{freq}_{y,j}| > \varepsilon} 1$ and $i, j = 1, \dots, \text{max}_{\text{freq}}$

D. Evaluation Criteria

To interpret the clustering results in the manner of identifying parallel execution graphs within a trace, it is important have an evaluation procedure. This helps to rate if the events in one group are belonging to the same execution graph.

As mentioned before the assumption of clustering is, that if events belonging to the same execution graph, they should be generated by the same protocol automaton. It was also shown, that if an automaton is constructed from events that belonging to different independent automata, the resulting automaton has a significant greater complexity. This is because of the exponential growing of possible states, if two or more independent automata are merged to one automaton. This behavior is called the product of automata (2).

If it would be possible to build an automaton, which describes the behavior of a group of events, the complexity of the build automaton should correlate to the quality of the clustering results. The fewer the complexity of the automata, the better the clustering results are.

A good starting point for such a rating mechanism would be the L* algorithm introduced by [21]. This algorithm constructs the smallest automaton describing a given set of sequences. It was shown in [18] that this algorithm can be applied to the given problem were the sequences are located within a trace. On the first glance one would calculate the complexity from the structure of the inferred automaton. But [18] has furthermore shown, that if L* can learn an automaton with a tested sequence length larger than five events within a short time, the resulting automaton will be less complex.

With this background, the evaluation of the clustered event groups is done by applying the L* algorithm. If the algorithm learns an automaton, which can be tested successfully with a test length greater than five events, within a time frame of three minutes, a good event group is argued. A good event group means its events are not generated by independent automata and therefore, the describing automaton of the event group is minimal.

E. Example Application

For proofing the described method of learning behavior models from network traces by identifying parallel execution paths an example with real network data is provided. The network data are recorded at a cars powertrain CAN network within real road tests. The example application takes a reference trace of approximately 13 min driving time.

From this reference trace all discrete events are extracted. A discrete event is represented by discrete signals. That means no measurement values like speed, temperature or similar continues information are used. Discrete events are most likely internal states like the engine status, discrete input values like the position of light or blinker levers and internal protocol values of interfaces. Because on CAN usually most information are sent periodically, duplicated send events are explicitly filtered.

After the preprocessing the CAN-trace, approximately 7500 different events are detected. Afterwards events are deleted that occur less than two times. From the resulting 7170 different events the frequency components were calculated. The overall quantity of events in the reference trace is about $2.4\ 10^6$ events.

 ISBN: 978-1-61208-319-3

The count of events containing to one cluster was limited to 127 events, because the implementation of the L* can only handle that amount of events per automaton. This is not necessary a limitation because in [18] it was shown that the performance of L* drops rapidly with more than 100 events.

F. Discussion of results from sample application

In TABLE I, the count of resulting clusters by applying the different distance metrics are shown. It can be seen that for the given example, a range of 400 to 700 clusters per metric are identified.

The results from the evaluation of the identified clusters with the L* algorithm are shown in TABLE II. As result the highest percentage of the identified clusters that can be inferred to valid automata with L*, offers the frequency distance with 73%. The most dedicated events describe by inferred automata was reached with the Manhattan distance with 46%. Even the coverage of the trace is best with Manhattan distance.

These results could be rated as success. In [18] a hand sorted list of events that describe less than 30% of the trace were used to learn an automaton. The learning process for this set of events was even not successful. With the usage of the Manhattan distance it is possible to describe about 49% of the behavior of the trace without any prior knowledge about dependencies of events.

TABLE I. CLUSTERING RESULTS BEFORE EVALUATION

Distance metric	**Results from different distance metrics**	
	Count of clusters	***Events per cluster***
Frequency	694	9,7
Euclidian	422	16,4
Manhattan	469	14,7
Hamming	570	12,6
Sum	2155	12,9

Within an additionally executed test, with randomly created clusters, the learning rate of successfully inferred automata was less than 4%. This leads to the expectation that the proposed clustering methodology gets a significant improvement for learnability of network traces with L* algorithm.

VI. IMPROVING TRACE COVERAGE

The presented clustering method extracts non overlapping events groups from a given set of events. From this event groups only these are usable, that lead to a learnable automaton by L*. For describing the system behavior represented within a trace, the description would be that better the more parts of the trace are considered to be used.

TABLE II. CLUSTERING RESULTS AFTER EVALUATION WITH L*

	Results from different distance metrics			
	Frequency	***Euclidian***	***Manhattan***	***Hamming***
Cluster successfully learned automaton (percentage of found clusters)	506 (73 %)	254 (60 %)	295 (61 %)	316 (55 %)
Ratio of successful clustered single events (abseolute count)	37 % (2,689)	40 % (2,883)	46 % (3,298)	37 % (2,676)
Ratio of event quantity (trace covery) (abseolute count of events)	27 % (678,820)	36 % (917,442)	49 % (1,259,452)	32 % (805,294)

Additionally, there is no strict requirement that the groups of events need not to overlap with each other. It would be quite the contrary if there are overlapping groups usable. Because it could be suggested, that clustering did not lead to a perfect separation in the sense of (2), the clusters will most likely describe only parts of independent automata. If there are overlapping event groups the merging of this group will lead to an automaton with describes a more complex but no parallel execution graphs.

This consideration leads to the conclusion that all identified clusters shall be used for describing the systems behavior. For that reason the results from the former clustering results are merged. The merge in that case is basically done by interpreting the learning results of L* from all clustering approaches. Like it is shown in Fig. 5, the different event groups resulting from clustering are overlapping. For the merge it should not be necessary to build new event groups and infer new automata. Instead the different sets are analyzed and the union set of all identified groups is calculated.

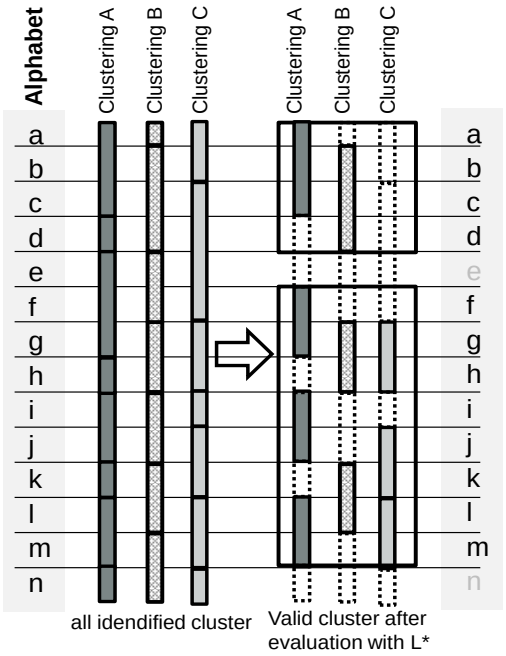

Fig. 5. Combining different clustering results

TABLE III. COVERAGE WITH MERGED CLUSTERS

	Results from combining the clustering results from Frequency, Euclidian, Manhattan and Hamming distance
Union set of dedicated events coverd (abseloute count)	69 % (4,981)
Union set of trace covered (abseloute count)	65 % (1,664,373)

A. Discussion of results

One can see in TABLE III. that the union set of all covered events in relation to the total amount of events increases from 46% at Hamming distance to 69% for the union set of all clustering results. Also the coverage of the trace increases from 49% to 65%. As a result of combination of different clustering approaches the behavior of one single event will be described with more than one automaton. The mapping of events to automata is over-determined. One dedicated event is now mapped to 2.3 automata at average.

VII. CONCLUSION AND FUTURE WORK

In this paper, a method for learning behavior models from network traces is proposed. These behavior models are the foundation for using reference traces to validate network communication in distributed embedded systems.

To enable learning algorithm the problem of high complexity within a network trace must be solved. Based on the assumption that a trace contains several parallel and independent activities, a system hypothesis was formulated. With the help of this hypothesis a technique for clustering and an evaluation method is derived and evaluated.

This technique basically uses the timestamps of the events and generates frequency components of each dedicated event. It was shown that clustering, based on these frequency components, produces sufficient results. The inferred automata from the L* algorithm helps to evaluate the clustering results and provide at the same time the behavior models to compare other network traces with the reference trace.

The next steps in feature work will be an estimation of the false positive and false negative rate, when the learned models from L* are used to compare the reference trace with other network traces. Additionally the concept of merging several clustering approaches promises good results and needs to be improved. For that reasons it would be a good idea to try some completely different clustering technics, that the union set and so and the coverage of events will be increased more efficient.

Based on a real world example it was shown that it is possible to separate a CAN trace in different sub-traces in the manner that these sub-traces contain independent execution graphs. The proposed methodology covers 69% of the events from the example trace. That is in comparison to previous work a significant improvement. With respect to the huge amount of 7,500 dedicated events and a trace length of 2.4 million events of the example trace, this is high amount of successfully interpreted data. With this results a step ahead to establish an unsupervised self-learning approach for validating network communication in scenarios were no specification is available.

REFERENCES

[1] R. R. Lutz, “Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems,” in Proceedings of the IEEE International Symposium on Requirements Engineering, 1993, pp. 126-133.

[2] R. R. Lutz ,“Requirements Discovery during the Testing of Safety-Critical Software,” in Proc. 25th Int’l Conf. Software Eng. (ICSE 03), IEEE CS: Press, 2003, pp. 578-583.

[3] P. Peti, R. Obermaisser, and H. Kopetz, “Out-of-norm assertions,” in Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. IEEE, 2005, pp. 280–291.

[4] G. Weiss, M. Zeller, D. Eilers, and R. Knorr, “Approach for iterative validation of automotive embedded systems,” in Proceedings of the 3rd International Workshop on Model Based Architecting and Construction of Embedded Systems, 2010, pp. 69–83.

[5] K. Becker, M. Zeller, and G. Weiss, “Towards Efficient On-line Schedulability Tests for Adaptive Networked Embedded Real-time Systems,” in PECCS 2012 - Proceedings of the 2nd International Conference on Pervasive Embedded Computing and Communication Systems, Rome, Italy, 24-26 February, 2012: SciTePress, 2012, pp. 440–449.

[6] O. Grinchtein, B. Jonsson, and M. Leucker, “Learning of event-recording automata,” in Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems: Springer, 2004, pp. 379–395.

[7] B. Bollig, J.-P. Katoen, C. Kern, and M. Leucker, “Replaying Play in and Play out: Synthesis of Design Models from Scenarios by Learning,” in Proceedings of the 13th International Conference on Tools and Algorithms for Construction and Analysis of Systems, Braga, Portugal: Springer, 2007. pp. 435–450.

[8] V-MODELL®XT, “V-Modell-XT Complete 1.2.1.1,” IABG, Oct. 2008. Available: http://v-modell.iabg.de/dmdocuments/V-Modell-XT-Complete-1.2.1.1-english.pdf.

[9] F. Langer and C. Prehofer, “Anomaly detection in embedded safety critical software,” in International Workshop on Principles of Diagnosis (DX) , 2011, pp. 163–166.

[10] G. Adzic, “Specification by example: How successful teams deliver the right software.” Shelter Island, N.Y: Manning, 2011.

[11] T. E. Daniels, “A functional reference model of passive systems for tracing network traffic,” Digital Investigation, vol. 1, no. 1, 2004, pp. 69–81

[12] G. Weiss, D. Eilers, "Device for creating a marked reference data stream", Germany EP20090015132, June 9, 2010

[13] A. S. Tanenbaum and M. van Steen, “Distributed Systems: Principles and Paradigms” (2nd Edition), 2nd ed.: Prentice Hall, 2006.

[14] F. A. Gers, J. Schmidhuber, and F. Cummins, “Continual prediction using LSTM with forget gates,” in Neural Nets WIRN Vietri-99: Springer, 1999, pp. 133–138.

[15] F. Langer, D. Eilers, and R. Knorr, “Fault Detection in Discrete Event Based Distributed Systems by Forecasting Message Sequences with Neural Networks,” in Lecture Notes in Computer Science, KI 2009: Advances in Artificial Intelligence, B. Mertsching, M. Hund, and Z. Aziz, Eds.: Springer Berlin / Heidelberg, 2009, pp. 411–418.

[16] S. Jha, K. Tan, and R. A. Maxion, “Markov Chains, Classifiers, and Intrusion Detection,” in Proceedings of the 14th IEEE Workshop on Computer Security Foundations, Washington, DC, USA: IEEE Computer Society, 2001, pp. 206-219.

[17] R. A. Maxion and K. M. C. Tan, “Anomaly detection in embedded systems,” Computers, IEEE Transactions on, vol. 51, no. 2, 2002, pp. 108–120

[18] F. Langer, K. Bertulies, and F. Hoffmann, “Self Learning Anomaly Detection for Embedded Safety Critical Systems,” in Schriftenreihe des Instituts für Angewandte Informatik, Automatisierungstechnik am Karlsruher Institut für Technologie: KIT Scientific Publishing, 2011, pp. 31–45.

[19] H. Liu and H. Motoda, “Feature extraction, construction and selection: A data mining perspective.” Boston: Kluwer Academic, 1998.

[20] R. Alur and D. L. Dill, “A theory of timed automata,” Theoretical Computer Science, vol. 126, no. 2, 1994, pp. 183–235.

[21] D. Angluin, “Learning regular sets from queries and counterexamples,” Information and computation, vol. 75, no. 2, 1987, pp. 87–106.

 ISBN: 978-1-61208-319-3