IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 7 April 2023, accepted 24 May 2023, date of publication 2 June 2023, date of current version 8 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3282176

SURVEY

Survey of Model-Based Security Testing Approaches in the Automotive Domain

FLORIAN SOMMER[1], REINER KRIESTEN[1], AND FRANK KARGL[2], (Member, IEEE)
[1]Institute of Energy Efficient Mobility, Karlsruhe University of Applied Sciences, 76133 Karlsruhe, Germany
[2]Institute of Distributed Systems, Ulm University, 89081 Ulm, Germany

Corresponding author: Florian Sommer (florian.sommer@h-ka.de)

This work was supported by the German Ministry of Education and Research (BMBF) through the Security For Connected, Autonomous caRs (SecForCARs)-Securing Automated VEhicles - Japan-Germany (SAVE) under Grant 16KIS0796.

ABSTRACT Modern connected or autonomous vehicles (AVs) are highly complex cyber-physical systems. As a result of the high number of different technologies and connectivity features involved, testing these systems to identify security vulnerabilities is a big challenge. Security testing techniques, such as penetration testing, are often manual methods that are applied comparatively late in the vehicle development process. Thus, vulnerabilities are only detected late or after development, leading to higher costs and more patching effort. To reduce the amount of testing resources in general and enable early and automated testing, model-based testing methods have been established in several domains, such as information technology and the automotive domain. The transfer of model-based testing approaches to automotive security testing could help to detect vulnerabilities earlier than other, manual methods by automatically generating, executing, or simulating security tests. In this study, we review the literature on model-based test approaches in the automotive domain. First, we consider security-independent approaches to obtain an overview of applied models, formalisms, test selection criteria, and test generation techniques. In addition, we investigate, whether and how model-based approaches are applied for automotive security testing. Overall, we identified 63 publications related to model-based testing and 29 publications with regard to model-based security testing. The aim of this study is to provide an overview and direct comparison between these approaches. In this manner, the state of model-based security testing in the automotive domain, current challenges, and potential research areas are determined.

INDEX TERMS Automotive security, model-based testing, model-based security testing.

I. INTRODUCTION

A large number of electronic and information technology components have been integrated into modern vehicles. The number of Electronic Control Units (ECUs) per vehicle has increased to 150 in recent years [1]. ECUs are connected through various communication technologies. Communication systems, such as Controller Area Network (CAN) [2], FlexRay [3], and Automotive Ethernet [4], are used. In recent years, there has also been an increase in communication between vehicles and their environments. Vehicles have evolved from closed to open systems by introducing technologies, such as Wireless Local Area Network (WLAN) [5], Bluetooth [6], and mobile communications. Thus, a modern vehicle represents a complex system of communicating entities. This complexity further increases with the trend towards autonomous driving [7]. Autonomous vehicles partially or completely drive on the road without human intervention. To realize self-driving cars, multiple sensors are required to recognize the environment, for example, cameras, Radio Detecting and Ranging (RADAR), and Light Detection and Ranging (LiDAR) systems. Image processing methods and artificial intelligence algorithms are used to process the large amount of data generated by these sensors. As a result, autonomous vehicles have a high degree of communication and data exchange, which makes testing more important and

The associate editor coordinating the review of this manuscript and approving it for publication was Agostino Forestiero.

challenging for vehicle developers and suppliers to ensure reliable operation. Software testing plays an important role in the development process. It is estimated that approximately 50 % of the total development costs is invested in testing [8]. In the automotive sector, testing plays a key role in ensuring the reliability, safety, and security of vehicles. While security is still comparatively new to the automotive domain, a great effort is currently being made to develop concepts and measures to guarantee security. Several attacks on vehicles have been recorded over the last few years [9], and some even lead to safety-critical situations, which could be life threatening for vehicle occupants and traffic participants in the worst case. Thus, security testing activities for all phases of the development cycle are necessary to protect vehicles by identifying threats and vulnerabilities.

Automotive security-related standards, such as SAE J3061 [10] and ISO/SAE 21434 [11], address cybersecurity aspects by defining a comprehensive development and test process. Therefore, activities, such as Threat Analysis and Risk Assessment (TARA), deriving security concepts, and implementing security measures, are required to protect vehicles from cyber attacks. However, to ensure that the number of potential remaining vulnerabilities in the vehicle is at an acceptable minimum, further security tests are required. ISO/SAE 21434 [11] suggests penetration testing, vulnerability scanning, and fuzzing for this purpose. However, since these test methods are typically carried out late in development after a vehicle and its components are implemented, potential vulnerabilities are found comparatively late. The elimination of these vulnerabilities can lead to increased costs and patching effort [12]. At that point, vulnerabilities that result from software or network architecture designs may not be eliminated or only at a high expense. Especially penetration testing is a manual and explorative test method that largely relies on the expertise of the tester. Thus, a significant amount of time and resources is necessary to detect vulnerabilities in a vehicle.

Marksteiner et al. [13] claim that manual test methods reach their limits in modern vehicles and propose an automation of the security test process. Jakobs et al. [14] further argue that formal methods should support security testing. Formal methods have been thoroughly employed in Model-Based Testing (MBT) to automate test activities. Several surveys (e.g., [15], [16], [17]) show that model-based testing is broadly used in the automotive domain and Khan et al. [16] highlight its ability to handle the complexity of testing modern vehicles. This raises the question whether transferring model-based methods to security testing can help to address current challenges, such as late testing and late vulnerability identification.

In this survey, we review literature to identify publications on (security-related) model-based testing methods in the automotive domain. There are several reasons for focusing on automobiles. First, there is a large number of model-based test methods in most technical domains (Section II), which must be filtered in a meaningfully for this survey. Furthermore, modern vehicles represent safety-critical and highly complex systems consisting of a multitude of different technologies that need to be tested. In addition, since the publication of UN R155 [18] and ISO/SAE 21434 [11] at the beginning of 2021, manufacturers are required to address security throughout the vehicle life cycle. Thus, there is a need for new and up-to-date security solutions for vehicles. Therefore, we exclusively address the automotive domain in this study by examining model-based test methods. The overall goal of this survey is to compare conventional and security-related model-based test approaches. We aim to evaluate whether advantages of model-based approaches can be transferred to security testing. In this way, the current state of Model-Based Security Testing (MBST) in the automotive domain, challenges, and potential research areas are determined. For this purpose, we first investigate 63 identified MBT publications and their application in the automotive sector with a focus on the vehicle development process, but without considering security. The goal of this step is to provide the state-of-the-art in terms of applied models, formalisms, test selection criteria, and test generation techniques. We further determine respective test or development stages and vehicle technologies (e.g., CAN communication or ECU applications), which can be tested by these methods. In the second step, the same investigation is performed on 29 MBST approaches. In particular, the current state of model-based security testing is highlighted. Modern vehicles combine several aspects, such as functional safety, a large number of system components and communication systems, autonomous driving, and external communication. Our goal is to evaluate whether current MBST methods cover these aspects. Based on the results, we determine current challenges in MBST and point out where further research is necessary.

This paper is structured as follows: In Section II, an introduction of fundamentals for MBT and MBST is presented. In particular, formalisms, test selection criteria, and techniques for test case generation and execution are addressed. In Section III, related publications and surveys are covered and a distinction of our survey is provided. Furthermore, we describe the procedure and realization of this literature review. The investigated databases and libraries as well as the selection process according to defined filter criteria are illustrated. In Section IV, 63 identified approaches of model-based testing in the automotive sector are presented, whereas Section V shows 29 security-relevant approaches. In Section VI, we analyze the existing approaches to compare MBT and MBST. In this way, current testing challenges and further research areas are detected. Finally, a conclusion and an outlook on future work follows in Section VII.

II. BACKGROUND

In this section, an introduction to model-based testing and model-based security testing is presented.

A. MODEL-BASED TESTING (MBT)

MBT aims to automate the test process and enable early test generation or execution in software development [19].

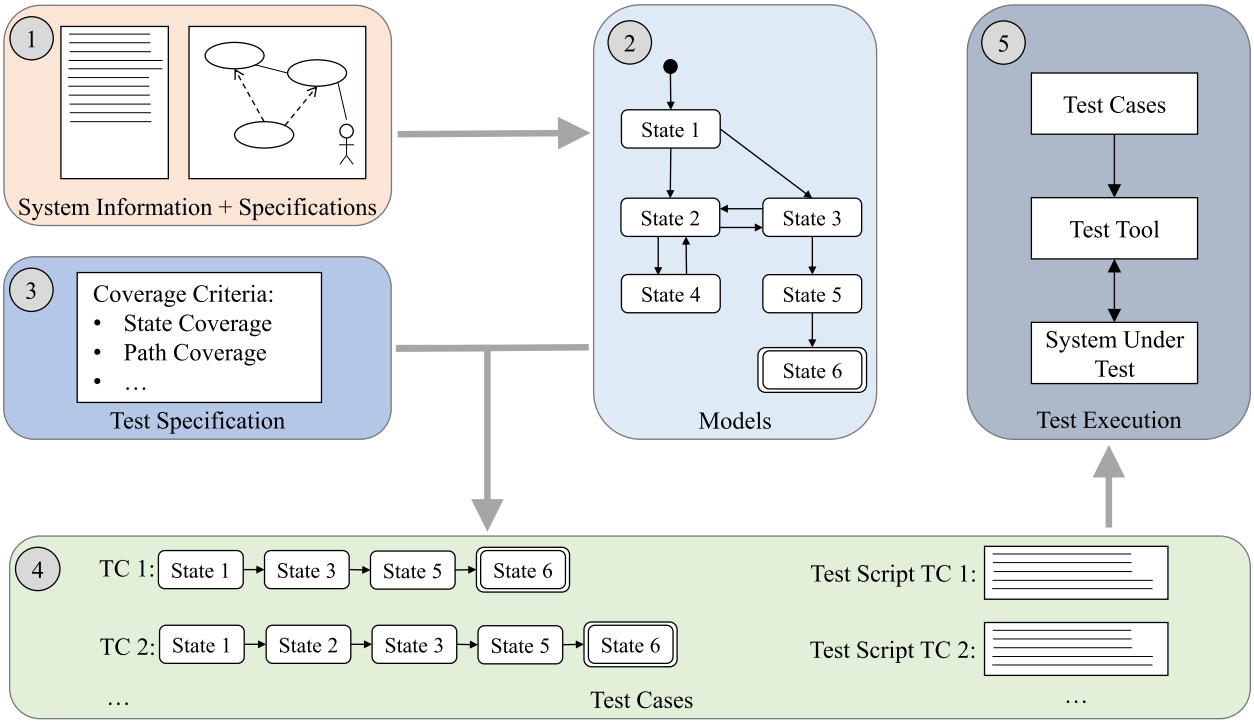

FIGURE 1. Overview of a model-based testing process. System information and specifications (1) are used to create one or multiple system models (2). A test specification (3) is applied to the model to generate test cases (4). Depending on the employed MBT approach and the availability of suitable tools, executable test scripts can be generated from the derived test cases to test the target system (5).

In literature [19], [20], [21], [22], [23], the term MBT is thoroughly described by multiple definitions. For example, Utting et al. [22] define model-based testing as a process in which a system or its environment are modeled to generate test cases for system testing. While subtle differences exist between the literature definitions, the process of system modeling and the automated derivation and execution of test cases are generally recognized as central aspects of MBT. Formal models are created based on the system requirements or specifications. Thus, the MBT process can begin early in development [19] and tests can be automatically created. If the model is executable (e.g., Simulink models, see Section II-A1), tests can be executed at model level before software code has been produced. Thus, system concepts, designs, or architectures can be examined for faults through analyzes or simulations. As a result, these faults are eliminated cost-effectively because the system design or development is not yet complete. Fig. 1 presents an exemplary MBT process.

First, system specifications, such as requirements or use cases, are specified during development. Based on these artifacts, a formal model (or several models) of this system is created. In Fig. 1, the system model is represented by an exemplary state machine. To derive test cases from the state machine, a test specification is applied to the system model. For this purpose, coverage criteria, such as state coverage, can be used. The resulting test cases are typically represented in an abstract manner (e.g., paths through the state machine). Subsequently, when the system is specified in more detail, these abstract test cases can be further refined. For example, test cases can consist of test scripts in C code with concrete test data. These test cases are then applied to the System Under Test (SUT) during test execution. This last step can be automated using specific test tools or frameworks. To discuss different aspects of model-based testing, we refer to the work of Utting et al. [22], in which a distinction is made between the categories: model specification, test specification, and test execution.

1) MODEL SPECIFICATION

Models are used to describe systems, tests, and specifications in an abstract and formal manner [19]. Typically, a distinction is made between system and test models. System models describe structural or behavioral aspects of a system, whereas test models consider test-specific aspects, such as test selection criteria, and test cases. Some approaches further apply environment models that describe the environment in which a system is located or operating. A variety of formalisms and modeling languages exist for specifying individual model types. In this context, a common and standardized modeling framework is the Unified Modeling Language (UML) [24]. It provides 14 graphical diagrams, which are classified into structural and behavioral diagrams. Structural diagrams describe static structures or connections of a system's components, such as classes of a software application. Among others, the class, component, and deployment diagrams belong to this group. Behavioral diagrams describe the dynamic behavior of a system, such as the sequential procedure of a software

application. These include use case, state, activity, sequence, and communication diagrams. As discussed in later chapters, many model-based test approaches use UML. An overview of test case generation approaches based on UML behavior diagrams is given in [25]. A common UML variant is the Systems Modeling Language (SysML) [26], which is applied in systems engineering (e.g., in the automotive sector [17]). SysML adopts a subset of UML diagram types directly or in an adapted form and introduces new diagram types, such as requirement diagrams. Another common MBT model type is represented by automata, especially Finite State Machines (FSMs) [27]. Several types of state machines exist, such as Extended Finite State Machines (EFSMs) [28], Communicating Extended Finite State Machines (CEFSMs) [29], and Statecharts [30]. Testing based on finite state machines has been extensively investigated. A comprehensive overview of the historical development of state-machine-based test approaches is presented in [27]. Simulink models are often employed in the automotive sector to specify and simulate vehicle models. Simulink [31] is an add-on environment to MATLAB [32] and models continuous or discrete systems as graphical block diagrams. It provides a library of blocks that are used to model signal flows. There is an extensive toolset for automatic C code generation from Simulink models, or interfaces to other applications, such as Vector CANoe [33], which allows simulations of bus systems and other communication technologies in vehicles. To model state machines in Simulink, the extension Stateflow [34] is used.

In general, a large number of other specification and modeling frameworks is available for model-based testing. Thus, testers must be familiar with these formalisms and related tools. The presented modeling formalisms are only a subset. A more comprehensive and detailed overview of formalisms can be found in [35].

2) TEST SPECIFICATION

A test specification contains information about test designs, procedures, and test cases. In this work, we focus on the generation of test cases, which is typically done by applying test selection criteria [22]. The criteria to consider depend on the individual test goals or modeling frameworks. As shown in Fig. 1, coverage-criteria can be used for test selection. In models, such as state machines or graphs, coverage can be based on the model structure (e.g., nodes or edges in a graph). In data-related models, coverage can be based on datasets. From a development perspective, it is also reasonable to select tests based on the requirements of a system. For systems, in which specific types of faults and failures are important (e.g., safety-critical systems), faults can be used as a test selection criterion. Test selection criteria can also depend on specific characteristics or domains of a system. For example, in cyber security development and testing, the risk or likelihood of an attack plays a major role. Therefore, risk-related test case generation [36] methods are applied for this purpose.

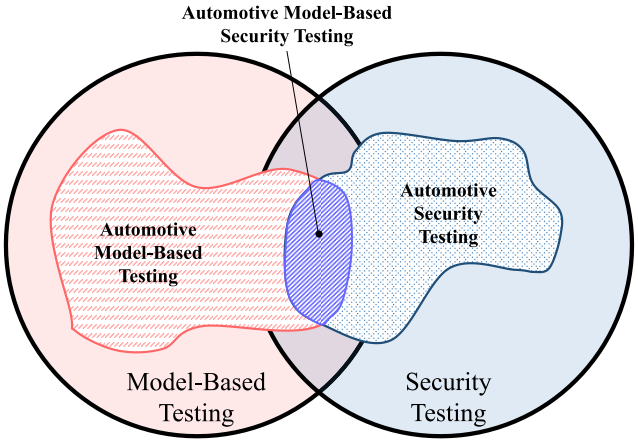

FIGURE 2. Relationship between model-based testing, security testing, and model-based security testing.

Overall, several test generation techniques exist for generating test cases based on selection criteria, whereas Utting et al. [22] differentiate model checking, random generation, search-based techniques, symbolic execution, theorem proving, and constraint solving. In Section IV, we elaborate on further details and present the approaches in which these techniques are applied in practice.

3) TEST EXECUTION

In terms of test execution, Utting et al. [22] distinguish between online and offline tests. Online tests describe a process in which test generation and execution are performed simultaneously. In offline tests, test case generation and execution are performed separately. Based on this, Zander et al. [37] introduce a taxonomy for model-based testing of embedded systems, which further divides test execution into more detailed elements. These include Model-in-the-Loop (MiL), Software-in-the-Loop (SiL), and Hardware-in-the-Loop (HiL) testing, which are common methods in the automotive sector [38]. Executing test cases on a target system depends on the SUT, system models, test generation techniques, and especially on the available testing tools. A comprehensive overview of existing testing tools and their deployment is provided in [39], which addresses different approaches to model functional black-box tests. Shafique et al. [40] introduce and investigate further model-based testing tools.

B. MODEL-BASED SECURITY TESTING (MBST)

In MBST, security testing is complemented by the advantages of MBT. This relationship is illustrated in Fig. 2.

Security testing aims to verify and validate whether a system is sufficiently protected against cyber attacks. For this purpose, two types of tests can be differentiated [41]. On the one hand, security mechanisms (e.g., encryption) are tested for correct implementation and conformance to their requirements. This process is called positive or functional testing [42] and can be conducted within traditional software

testing or MBT. However, a system can also be tested for security vulnerabilities. This process is referred to as negative, non-functional, or security vulnerability testing [42]. Penetration testing is often performed for this purpose. Traditional testing aims to test a system for faults, whereas penetration testing [43] is used to identify vulnerabilities an attacker can exploit to compromise the system. Testers act as attackers and attempt to attack the system by exploiting its vulnerabilities. Thus, cyber attacks are used as test cases. As a result, security test cases are successful when vulnerabilities are discovered or exploited.

Regarding its fundamental procedure, MBST strongly resembles MBT [44]. Models are created from existing development artifacts, such as system requirements. Based on these models, test cases are derived, which include attacks that are used for security testing. In conclusion, by applying model-based approaches to security testing, we expect to transfer the advantages of MBT, such as early testing or test automation, to the security testing domain. Through an early identification of vulnerabilities, mitigation in the concept or design phase is possible.

III. RELATED WORK AND SURVEY METHODOLOGY

In this section, an overview of related publications and surveys for model-based (security) testing in the automotive and other domains is provided. We further underline how our study differs from existing surveys. In addition, the systematic search and selection process for this literature survey is described and a brief overview of the results is provided.

A. RELATED WORK

Model-based testing has been applied in many domains. Particularly in Information Technology (IT), numerous approaches have been covered by various surveys. Dias Neto et al. [20] provide an overview of 78 model-based testing publications targeting different test stages and behavior modeling methods. Su et al. [45] focus on data-flow test techniques, for example, symbolic execution or model checking. A total of 97 approaches from 1976 to 2015 are covered. Fraser et al. [46] focus exclusively on model checking techniques for test generation. In particular, coverage-based and mutation-based approaches are addressed. Hartman et al. [47] present a survey on test modeling languages. The survey of Jia and Harman [48] covers mutation testing methods published between 1977 and 2009. Application methods, cost reduction techniques, tools, and mutant detection methods are described. Another survey was conducted by Anand et al. [49], who present methods to automatically generate test cases. In addition to classical IT, MBT is employed in other technical domains. Abbors et al. [50] present an application of MBT in the telecommunication domain. Other approaches, such as the work of Andrews et al. [51], are centered on robotics. Furthermore, safety-critical domains, such as health care [52] and aerospace [53], also use MBT. An exemplary survey of model-based testing in the automotive domain was conducted by Altinger et al. [15]. In this survey, 68 people were interviewed who work at Original Equipment Manufacturers (OEMs), suppliers, software vendors, or universities. Each participant was asked a total of 24 questions about employed tools, test environments (e.g., HiL), and automation methods. Based on the interviews, the authors conclude that formal methods are mainly used for system specification, whereas they are rarely applied in testing. Furthermore, while the generation of test cases is typically performed manually, test case execution on the SUT is largely performed in an automated manner.

Regarding security testing, Felderer et al. [54] present a large-scale survey of security testing techniques. A distinction is made between the categories of model-based security testing (applied in the analysis and design phase of the system life cycle), code-based testing and static analysis (applied in the development phase), penetration testing and dynamic analysis (applied in the deployment and maintenance phase), and security regression testing (applied in the maintenance phase). In IT, numerous MBST publications exist that focus on testing security policies of access control systems or firewalls [55]. Regarding functional security testing, Martin [56] provides an automated test generation method to test the correctness of access control policies. Non-functional security testing is addressed by Salas et al. [57]. Since many IT systems communicate via the Internet, there are numerous MBST approaches for web applications [58], [59]. Furthermore, MBST is covered in health care [60] and aerospace [61], [62].

Mahmood et al. [63] present a survey of seven test beds for security testing in the automotive field. The respective publications are examined and compared with regard to covered network technologies (e.g., CAN), attack surfaces, and attack techniques. In addition, four methods that can be used for security testing are described: penetration testing, fuzz testing, vulnerability scanning, and model-based security testing. Luo et al. [64] examine security testing methods that are used in the automotive sector. The authors distinguish between knowledge-based, automation-based, threat-based, risk-based, requirement-based, and model-based security testing. Threat-based testing is subdivided into penetration testing, fuzzing, and vulnerability testing. Furthermore, related test beds are investigated. In total, 73 publications are presented, of which 10 are related to MBST. In the automotive domain, MBST is covered in surveys regarding general security testing approaches. For example, Pekaric et al. [65] list 39 security testing approaches used in the automotive development process. The authors examine how testing activities are used at different points in the development (e.g., design phase) and life cycle of a vehicle. These methods are classified as model-based, code-based, risk-based, regression, penetration testing, and dynamic analysis. The identified methods are mapped to the different layers of the AUTomotive Open System ARchitecture (AUTOSAR) [66]. Thus, 39 publications are classified. In total, 12 approaches

are assigned as MBST. However, most of the approaches (26) are considered penetration testing methods.

Existing surveys [15], [17] suggest that MBT is commonly applied in the automotive domain. However, the available surveys treat MBT and security testing separately. To date, there has been no survey targeting both areas in combination. In contrast, our study examines MBT approaches in the automotive field and security-related (MBST) approaches. We aim to directly compare MBT and MBST. A partial goal of our survey is to investigate which MBT approaches are used and which development stages are addressed, which formalisms and tools are used. We further investigate the extent to which automotive technologies (e.g., ECU applications or communication protocols) can be tested using these approaches. Since cyber security is still relatively new to the automotive domain, the body of research on automotive security testing is comparatively young. As shown by Pekaric et al. [65], published work often refers to penetration testing and dynamic analysis techniques (e.g., fuzzing and vulnerability scanning). These test techniques are also proposed by ISO/SAE 21434 [11]. However, as outlined in Section I, these methods do not support early vulnerability testing and can only be applied late in the development cycle. Thus, the question arises how MBST approaches benefit the automotive security test process. For this purpose, we further investigate which MBST methods are currently applied in the automotive domain. The main difference from the related work is a combined examination of MBT and MBST. To the best of our knowledge, this has not been addressed in other studies. The overall goal of this survey is to investigate how widespread both approaches are, which methods and tools are used, which system components are tested, and which modeling and test generation methods are used. Based on the results, current challenges and open research areas are identified, particularly in the area of MBST.

B. SURVEY METHODOLOGY

Our review methodology is based on the guideline of Kitchenham and Charters [67] for systematic literature reviews in software engineering. This procedure has been used by other automotive security-related research, such as in [68], and also for model-based security testing research, such as in [43]. We follow two steps: First, security-independent automotive model-based testing approaches are identified. In the next step, security-related approaches are identified. We describe the identified publications and analyze both areas in comparison. The goal of this survey is to determine the current state of the art in MBT and MBST in the automotive domain and a comparison of both areas to determine if the security testing process can benefit from model-based testing. This further leads to the derivation of current challenges and research areas in MBST. To conduct our research, we chose the online libraries ACM Digital Library, IEEE Xplore, Wiley Online Library, SpringerLink, ScienceDirect, SAE Mobilus, and Google Scholar. Felderer et al. [43] and Brereton et al. [69] also consider the first five libraries as important sources for software and security engineering. We chose to add SAE Mobilus, because it specifically targets the automotive domain. Google Scholar was chosen to find publications that are not covered by the other libraries. Specific search strings, targeting MBT and MBST are used. For this purpose, we use the following terms (and related synonyms) in combination to cover a wide range of publications:

- *model-based* (*synonyms: formal, automated*)
- *testing* (*synonyms: test, verification, validation*)
- *automotive* (*synonyms: vehicle, vehicular, car, transportation*)
- *security* (*synonyms: vulnerability, attack, attacker*)

We use these search strings to gather publications from the online libraries. In order to filter the search results, four steps are conducted:

- **Step 1:** Searching for publications in online databases and libraries with the search strings.
- **Step 2:** Filtering the search results by publication title.
- **Step 3:** Filtering resulting papers by reading their abstract.
- **Step 4:** Filtering resulting papers by reading them in full text.

First, we searched for publications by applying our search strings to the online libraries and databases mentioned above. Overall, 739 publications were identified. We filtered these papers by their titles to eliminate papers that did not fit into our survey scope. After filtering, 130 publications remained. In the third step, we read the abstracts of these publications. At this stage, papers below four pages were neglected because of their limited detail of information. Only papers written in English were selected. As a result, 104 publications remained. In the last step, we read these 104 papers in full text. At this stage, we focused on papers that target the vehicle development process. This left 92 publications in total, of which 63 publications are related to MBT and 29 publications are related to MBST. The distribution of these papers according to the publication year is illustrated in Fig. 3.

Publications targeting MBT were published continuously between 2004 and 2020 (with the exception of 2007). Additionally, a total of 9 MBT papers were published before 2004. While there was another peak of publications in 2017 (eight papers), the number of publications decreased from 2018 to 2021 (five papers). In contrast to MBT, MBST has only recently become relevant. Sporadic approaches (eight papers) were published between 2009 and 2017. Since 2018, the number of MBST publications has increased (21 papers between 2018 and 2021). This can be attributed to the fact that the field of cybersecurity came into the focus of researchers and the automotive industry after various research papers (e.g., Koscher et al. [70], Checkoway et al. [71], and Miller and Valasek [72]) uncovered several security vulnerabilities in vehicles.

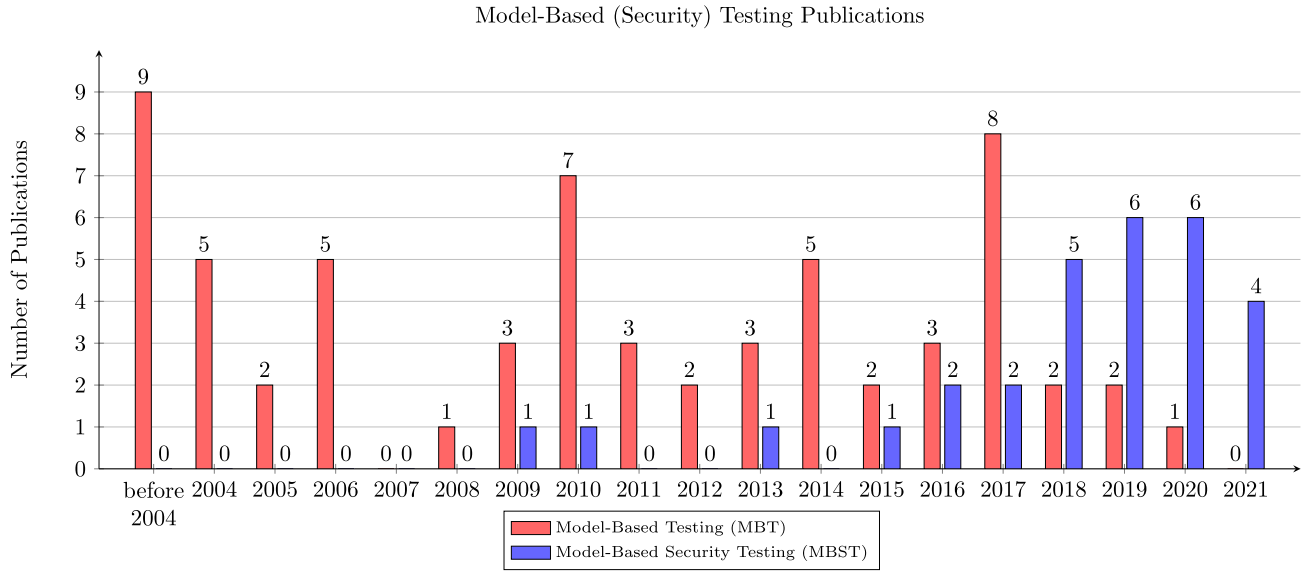

FIGURE 3. Distribution of selected approaches over the years they were published. The diagram distinguishes MBT and MBST. In total, 92 papers were selected of which 63 publications target MBT and 29 target MBST.

In the following two sections, we present the publications identified in detail. Section IV covers MBT publications, whereas Section V addresses MBST. In Section VI, an analysis and comparison of the MBT and MBST aspects is presented.

IV. MODEL-BASED TESTING (MBT) IN THE AUTOMOTIVE DOMAIN

In this section, we present MBT publications in the automotive domain. In total, 63 MBT-related publications were identified in this survey. To provide a better overview, we group the approaches according to the categories shown in Fig. 4. It should be noted that the classification in Fig. 4 is not unique as categories may overlap. For example, it is possible that XiL environments (Section V-B) are also addressed and described in state-based (Section IV-F) or fault-based (Section IV-B) approaches.

Each identified publication is assigned to one of these categories. If an approach addresses more than one category, the classification is assigned based on the primary focus of the approach. To enable a comparison between the approaches, the publications are uniformly presented in tables (e.g., Table 1). For this purpose, we distinguish the following table categories:

- **Approach:** describes the method used, for example, the Classification-Tree Method (CTM) or search-based algorithms.
- **Goal:** describes the aim or purpose of this approach. We distinguish between the generation of Test Cases or Test Data, Test Execution, Test Planning, and verification/validation of the system or its security (System/Security V/V).
- **Model:** specifies the model types or formalisms used (e.g., classification trees, UML)

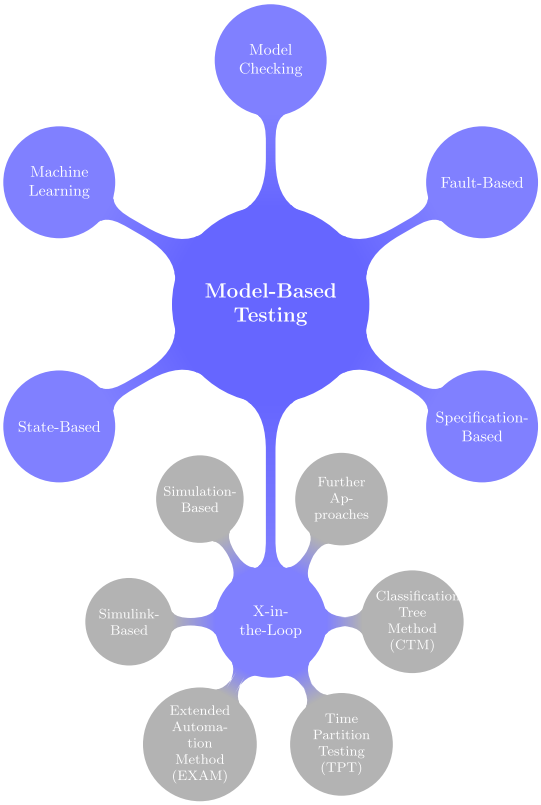

FIGURE 4. Classification of identified MBT publications.

- **Stage:** describes the development stage in which an approach is applied. We differentiate Component Test (CT), Integration Test (IT), System Test (ST), Acceptance Test (AT), Regression Test (RT), Design Level (DL), Deployment (DP), and X-in-the-Loop in terms of development stages.

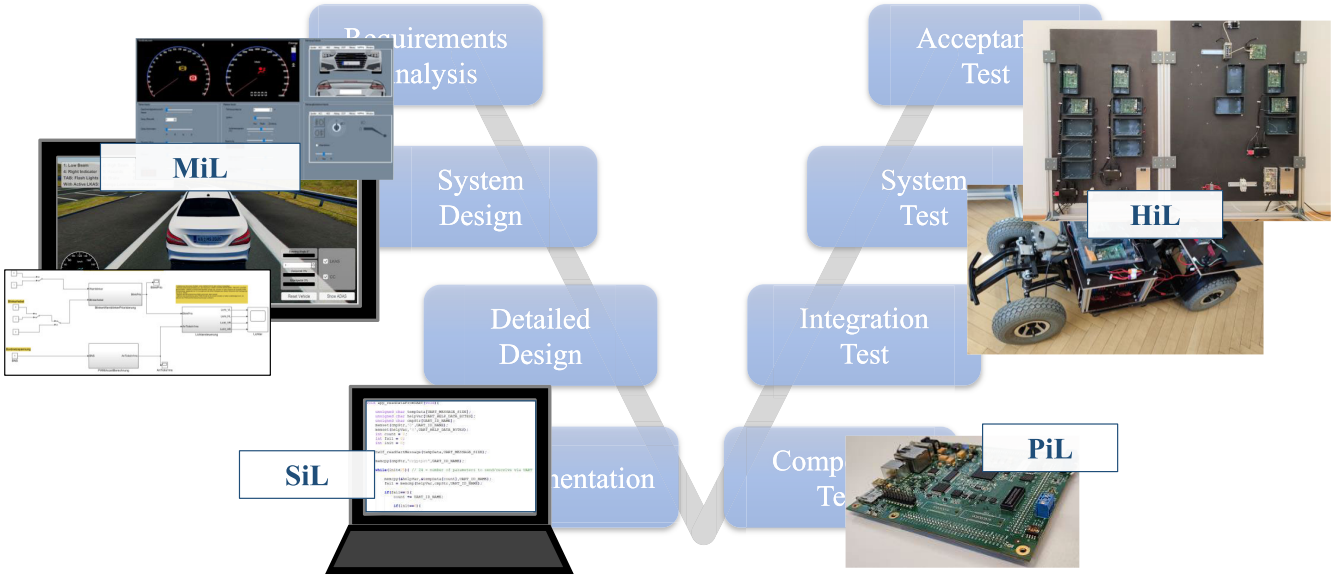

FIGURE 5. X-in-the-Loop test environments (MiL, SiL, PiL, and HiL) and related development and test stages in the V-Model (in accordance with [73]).

- **Area:** specifies the technological area addressed by the method. We distinguish ECUs, Communication Systems (COM), E/E Architectures (EEA), Sensor, Actuator, Vehicle, and Vehicle-to-X communication (V2X). If a paper mentions specific technologies (e.g., CAN communication), these will be also mentioned.
- **Application:** describes the specific use case or example (e.g., Adaptive Cruise Control (ACC)) an approach was shown or tested with.
- **Tool Support:** gives an overview of used tools or toolchains.
- **Automation (Aut.):** indicates, whether an approach has full (assigned with ✓), semi (assigned with (✓)), or no automation (assigned with ✗).

These categories result from aspects that we want to compare in this survey. The following sections describe the approaches selected for each category.

A. XiL-BASED APPROACHES

In the automotive domain, X-in-the-loop test environments are common [74]. Here, systems are developed in a model-based manner and verified at different stages of the development process. The International Software Testing Qualifications Board (ISTQB) [38] differentiates between MiL, SiL, Processor-in-the-Loop (PiL), HiL, and Vehicle-in-the-Loop (ViL), with MiL, SiL, and HiL being the most important test environments. Fig. 5 illustrates an overview of the XiL methods and their related development and test stages.

MiL is applied in the analysis and design phase of development [75]. The SUT and its environment are available as models configured and executed in a simulation. As an example, simulating and testing Simulink models can be mentioned. This often involves the use of environment models that form a closed system with a system model. The Test cases are executed in simulations. In SiL, the system model is replaced by compiled software code that can be created manually or generated automatically from the system model. Typically, the SUT (e.g., C code) is connected to the environment model, which is still available as a model and executed within a simulation environment. Interfaces are used to connect the environment model with the SUT software to execute tests. This type of testing can be applied as soon as implemented code is available. Thus, it is typically applied in the implementation or unit test phase. In HiL testing, the SUT runs on the target platform, since most parts of the SUT and environment are available as physical hardware systems. Typically, HiL frameworks or test benches represent the behavior of the environment under real conditions in real time. For example, if the ECU of a vehicle is tested, a HiL framework can consist of an ECU network connected to sensors and actuators by bus systems. This type of testing can be applied in the later stages of development, such as system and acceptance testing. Because X-in-the-loop test environments are established in automotive, MBT-related publications address these environments to a great extent.

1) CLASSIFICATION-TREE METHOD (CTM)

A long research history in MBT has been supported by Daimler AG with the Classification-Tree Method (CTM) [76]. In this approach, the test input data are separated into categories using category-partitioning. A subsequent classification of these categories results in a tree structure (classification tree). An example of a classification-tree for a turn indicator function is shown in Fig. 6.

The *Turn Indicator* function consists of the input signals of *Hazard Lights* and a *Turn Indicator Control* as well as two output signals: *Turn Indicator Left* and *Turn Indicator Right*.

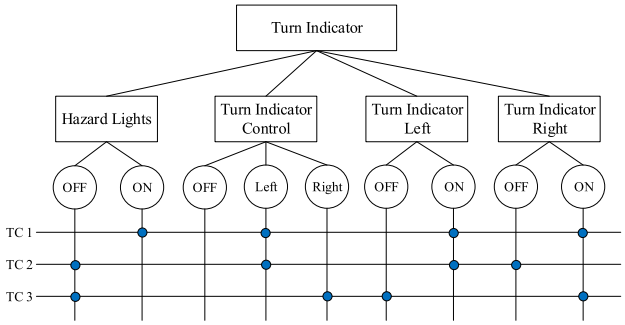

FIGURE 6. Classification-tree of a simplified turn indicator function. By combining input data, test cases (TC) are modeled.

The leaf nodes of the tree represent concrete test values for each input category (e.g., *OFF*). A combination of the individual test values results in black-box test cases, for example, to test the *Turn Indicator* function. In Table 1, publications that use classification-trees and the CTM for model-based testing are summarized.

In [77], Grochtmann et al. introduce the Classification-Tree Editor (CTE), which supports testers conducting CTM. The CTE tool enables testers to model classification-trees graphically and derive test cases by combining test values as shown in Fig. 6. Furthermore, the tool can determine test coverage. A drawback of CTM is the need to manually model classification trees and test cases. Singh et al. [78] extend this process to introduce automation. For this purpose, the input specification is described in the formal specification language Z [84]. Z allows the specification of vehicle components, for example, ECUs and their behavior, in a formal manner (set theory, first-order predicate logic) and treat them as test objects. The authors illustrate this using an ACC. The formal character of Z allows to semi-automatically generate the classification-tree. Based on the test values of the tree-classes, a disjunctive normal form of the inputs is created, which allows an automatic test case generation from the tree.

In the automotive domain, the MATLAB/Simulink framework is widely used to design vehicle systems in a virtual simulation environment. To use Simulink models in CTM, Lamberg et al. [79] present the Mtest method. The approach is illustrated by an example of a vehicle dynamics model, in which the vehicle performs a lane change maneuver. Model input signals are used as test inputs for CTM. In the same way as the turn indicator function in Fig. 6, these signals can be partitioned into test value categories for a classification-tree. The tree is automatically created using the dSPACE TargetLink and AutomationDesk tools. Furthermore, the resulting test cases can be executed in the Simulink simulation environment in a MiL scenario. The authors also highlight the ability to address SiL and PiL testing depending on the development state of the SUT.

Many vehicle functionalities and physics are time-dependent. For example, the vehicle speed increases continuously over time during acceleration. To test time-dependent functions, Conrad [80] introduce the Classification-Tree Method for Embedded Systems (CTM/ES), which extends CTM to test time-specific aspects of embedded control applications. This enables the creation of test scenarios in which test data can change between the time steps. This approach is illustrated by a pedal position interpretation. In this example, the pedal position is increased over a certain time range, which results in a change of the vehicle speed. CTM/ES is applied in a MiL test to simulate functional and structural tests.

In a subsequent work, Conrad et al. [81] demonstrate how this process can be applied to further test stages in SiL and HiL testing. For this purpose, the model-based CTM/ES approach is combined with requirement-based testing. The CTM/ES is executed on an Antilock Braking System (ABS) example to create time-dependent test cases. For requirement-based testing, tests are created directly from the system specification using CTM. In this manner, tests are traceable to their requirements and can be created at the beginning of development. To provide consistency between the requirement-based and model-based test cases, checking rules are applied that compare the results of both approaches. This enables a traceability of requirements to test cases at different stages of development.

The ability of CTM and CTM/ES to combine input and output data to derive test cases is further utilized in combination with the Automotive Validation Functions (AVF) approach from Zander-Nowicka et al. [82]. AVF compares actual and intended signal values of Simulink models. For this purpose, system requirements are used to determine intended signal values. These are implemented into validation functions, for example, to assertions or conditional statements, which compare them to actual values. The validation functions can be integrated into Simulink models to run simulations. Mjeda et al. [83] apply AVF in addition to CTM/ES. Both processes are executed in parallel to create test cases, which are combined to evaluate each other. Thus, errors in the test design can be reduced.

2) TIME PARTITION TESTING (TPT)

TPT was introduced by Lehmann [89]. This approach targets the time dependent and continuous behavior of embedded systems using an executable test language. TPT uses a graphical automaton notation to model test cases. An exemplary test case automaton is illustrated in Fig. 7.

The automaton states represent the test behavior or input stimulation (e.g., *Turn on ignition*). Transitions contain a temporal behavior (e.g., *Wait for 2s*). This representation is further refined to *Time Partition diagrams*, which characterize an executable directed graph. The TPT-related approaches identified in this survey are listed in Table 2.

Bringmann and Krämer [85] apply the TPT approach for model-based testing of automotive software. The SUT and its behavior are modeled using MATLAB/Simulink. Test cases are created manually as state machines using the TPT process. However, the authors highlight the possibility of creating test cases using CTM. The test case states specify the input signals

TABLE 1. Survey results for model-based test approaches regarding the Classification-Tree Method (CTM).

Paper	Approach	Goal	Model	Stage	Area	Application	Tool Support	Aut.
[76]	CTM	Test Cases	Class.-Tree	CT	ECU	-	-	✗
[77]	CTM	Test Cases	Class.-Tree	CT	ECU	-	CTE	✗
[78]	CTM	Test Cases	Class.-Tree, Z	CT	ECU	ACC	CTE	(✓)
[79]	CTM, MTest	Test Cases	Class.-Tree, Simulink	MiL, SiL, PiL	Vehicle	Veh. Dynamics, lane change	dSPACE (TargetLink, AutomationDesk)	✓
[80]	CTM/ES	Test Cases	Class.-Tree, Simulink	MiL, HiL	ECU	ACC	CTE/XL, CTE/ES, MATLAB	✓
[81]	CTM, CTM/ES	Test Cases	Class.-Tree, Simulink	MiL, SiL, HiL	ECU	ABS	MATLAB	✓
[82]	AVF	Test Data	Simulink	MiL	ECU	ACC	MATLAB	✗
[83]	CTM/ES, AVF	Test Data	Class.-Tree, Simulink	MiL	ECU	ACC	CTE-XL, Mtest, MATLAB	(✓)

TABLE 2. Survey results for model-based test approaches regarding Time Partition Testing (TPT).

Paper	Approach	Goal	Model	Stage	Area	Application	Tool Support	Aut.
[85]	TPT, CTM	Test Cases	Simulink, FSM	MiL, SiL, PiL, HiL	ECU	Exterior headlight	MATLAB, TPT-VM	✓
[86]	EvoTPT	Test Data	Simulink, FSM, Graph	MiL	ECU	ACC	MATLAB, ETF	✓
[87]	TPT	Test Execution	FSM (AUTOSAR-based)	HiL	ECU, COM (CAN, LIN)	Interior light	AUTOSAR tool	(✓)
[88]	TPT	Test Execution	Automaton, TTCN-3	MiL, SiL, HiL	ECU	Automated light control	-	✗

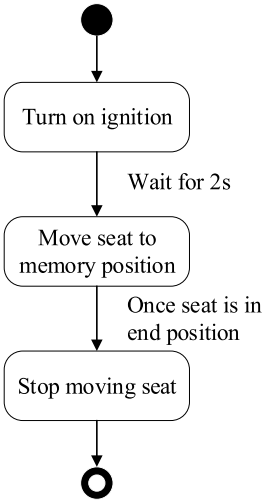

FIGURE 7. Exemplary test case in Time Partition Testing (TPT). The states represent test actions or behavior while the transitions specify time-dependent (temporal) aspects.

for the Simulink model, whereas the transitions specify the timing behavior (according to Fig. 7). The test cases are compiled into byte code to make them executable on a virtual machine that is connected to the SUT by interfaces. Thus, the continuous behavior of the system can be tested in MiL, SiL, PiL, and HiL platforms using the MATLAB/Simulink framework. The authors illustrate that approach on the example of an exterior headlight ECU, for which 72 test cases are defined in a case study.

The TPT process contains steps that have to be performed manually, such as defining the test data or parameters for test cases or evaluating test results. To automate this process, Lindlar et al. [86] extend the TPT test methodology by combining TPT with evolutionary algorithms. In this Evolutionary Time Partition Testing (EvoTPT) approach, test data generation is viewed as an optimization problem. TPT-based test cases are altered by searching for optimal test parameters to reduce the search space. After executing the test cases, the test results are provided to a fitness function that calculates a fitness value that is fed back to further refine test parameter optimization. The authors illustrate this approach using a custom Java-based tool (Evolutionary Testing Framework (ETF)) in which an ACC system is represented as a MATLAB/Simulink model and tests are executed in a MiL environment.

Modern vehicles can include 150 ECUs [1] developed by several suppliers. AUTOSAR [66] was established to provide a common interface and architecture structure. AUTOSAR consists of several layers that separate ECU specific aspects, such as communication (COM) or Software Components (SWCs). To test these individual layers, Michailidis et al. [87] apply TPT to execute tests at the system level, early in development. The authors use TPT state machines to model test cases for different AUTOSAR layers. A main state machine covers AUTOSAR SWCs, an ECU state machine covers the ECU state manager, and a COM state machine covers AUTOSAR COM services. The approach is validated using a Mercedes interior lights system. Tests are executed in a HiL environment.

The automotive domain combines several technologies. Thus, a large number of tools exist to support the test process (see Section VI-G). To introduce a common test language, which can be used across several tools, Schieferdecker et al. [88] apply the Testing and Test Control Notation Version 3 (TTCN-3) [90] to the TPT process. TTCN-3 is a standardized test language for black-box testing and is primarily used to test communication systems (e.g., telecommunication). To use TTCN-3 for TPT, streams,

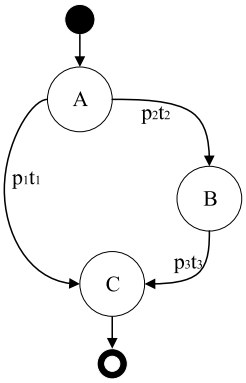

FIGURE 8. Simplified graphical representation of a Timed Usage Model in which the transition probabilities p_i of a Markov Chain are complemented by a timing parameter t_i.

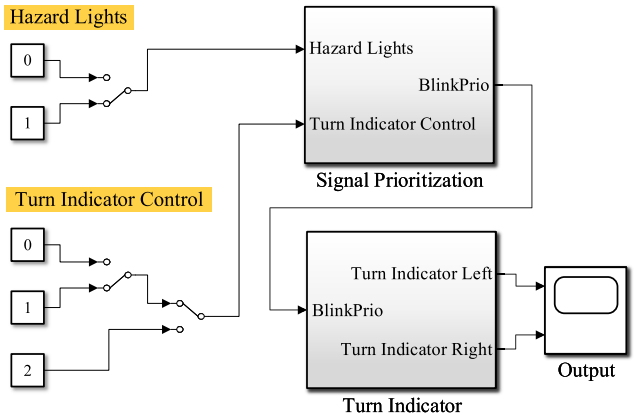

FIGURE 9. Exemplary Simulink model of the turn indicator function from Fig. 6.

channels, and continuous timing aspects are introduced to TTCN-3. The authors illustrate this extension using an exemplary automated light control system.

3) EXTENDED AUTOMATION METHOD (EXAM)

The EXAM is an automated functional ECU testing approach applied at Volkswagen AG. Thiel and Zitterell [93] and Zitterell and Thiel [94] introduce the basic abstract process of EXAM. Based on an informal test specification, test designers create a formal specification using a description language similar to the UML meta model. The resulting formal test specification includes parameters that are used to derive functional test cases. The main advantage of EXAM is that resulting test cases are independent of any test platform. This enables the reuse of tests on different test platforms, particularly in HiL environments. In Table 3 two approaches are presented that apply EXAM in a model-based test context.

In the basic EXAM process, test case generation is performed manually. Siegl et al. [91] provide an approach that automates this process. Essentially, the requirements of an engine start-stop function are used to create a Markov Chain Usage Model (MCUM) [95]. Markov Chains are similar to state machines, but include probabilities on the transitions (as shown in Fig. 8). Thus, MCUMs represent possible usage scenarios of the system under certain probabilities. The authors apply statistical random walks and coverage-based graph algorithms to automatically generate test cases from the MCUM. Resulting test cases are transferred to UML sequence diagrams. These platform independent diagrams can be further refined for specific test platforms, for example, by an automatic generation of executable test scripts for HiL environments.

In [92], the authors further enhance this approach to cover timing aspects. For this purpose, MCUMs are extended to Timed Usage Models (TUM) (Fig. 8) by complementing transition probabilities with timing parameters. Thus, time-dependent test cases can be created. This process is evaluated using an engine start-stop function and an energy management system for an electric/hybrid vehicle. For this purpose, test cases are generated to verify conformance of safety-critical system requirements demanded by ISO 26262 [96].

4) SIMULINK-BASED APPROACHES

The MATLAB/Simulink framework and its extension Stateflow are widely used in the automotive sector to model vehicle systems as block diagrams. Fig. 9 illustrates an exemplary Simulink model.

The model is based on the turn indicator functionality as shown in Fig. 6. The input signals of the hazard lights and turn indicator control are processed in a block that prioritizes the signals. The output signal triggers the left and/or right turn indicators. A large number of model-based test methods are based on Simulink models. Test cases are derived using suitable tools and executed in different test environments, such as MiL. An overview of these approaches is listed in Table 4.

Matinnejad et al. [102] present an approach for MiL testing of continuous control applications. For this purpose, a supercharger bypass flap position control unit is modeled in MATLAB/Simulink. Based on the input space of that model and functions specified in the system requirements, a random search algorithm is applied. As a result, a so-called *HeatMap* is obtained in which values of the system functions are mapped to certain regions. The map is evaluated by an expert to identify critical regions, where faults are more likely to occur. In the next step, a single-state search algorithm is applied to automatically generate test cases based on the most critical regions. The authors demonstrate that their approach provides better performance and more useful test cases than a pure random search.

Search-based algorithms are further used in Simulink and Stateflow models. For example, Hahn et al. [97] model an ACC system in two different Simulink/Stateflow variants. The first model is hybrid and combines continuous and discrete system aspects, whereas the second model contains only discrete aspects. In addition, the vehicle environment (e.g., the vehicle ahead) is considered. The discrete model, in particular the state machine of Stateflow, is used to generate test

TABLE 3. Survey results for model-based test approaches regarding the EXtended Automation Method (EXAM).

Paper	Approach	Goal	Model	Stage	Area	Application	Tool Support	Aut.
[91]	EXAM	Test Cases	UML Sequence, Markov Chain, XML	HiL	ECU, COM	Start-stop	MaTeLo graph designer	✓
[92]	EXAM	Test Cases	Timed Usage Models	HiL	ECU	Start-stop, Energy management	-	✓

TABLE 4. Survey results for model-based test approaches regarding Simulink.

Paper	Approach	Goal	Model	Stage	Area	Application	Tool Support	Aut.
[97]	Search (coverage-based)	Test Cases	Simulink Stateflow	MiL	ECU	ACC	MATLAB	✓
[98]	Failure insertion	Test Execution	Simulink, Stateflow	HiL, CT, IT, ST	ECU	Powertrain, engine	dSPACE ControlDesk	✓
[99]	Assertions	System V/V	Simulink, Stateflow	MiL	ECU	Cruise control	MATLAB, Reactis Validator	✓
[100]	Model Checking, Random Test, Constraint Solving	Test Cases	Simulink, Stateflow	MiL	ECU	Act. safety, powertr., HVAC, ESC	MATLAB, SmartTestGen	✓
[101]	Simulation	Test Cases	Simulink	MiL, SiL, HiL	ECU	Dual-Clutch Transmission	MATLAB, TestWeaver	✓
[102]	Search	Test Cases	Simulink	MiL	ECU	Supercharger bypass flap position	MATLAB	(✓)
[103]	Simulation	Test Execution	Simulink	HiL	ECU, COM (CAN, LIN)	Body controller	MATLAB, dSPACE AutomationDesk	(✓)
[104]	Simulation (signal-based)	Test Execution	Simulink, DSL	MiL	ECU	Start-Stop control	MATLAB, Arttest	✓
[105]	Simulation (signal-based)	Test Execution	Simulink, DSL	MiL, SiL	ECU	Window control	MATLAB, Arttest	✓

cases. For this purpose, a search-based algorithm is applied to the state machine, which considers coverage metrics, such as all-state-pairs and all-transition-pairs coverage. The hybrid model is used as a reference to compare the test results from the discrete model.

In a similar approach, Peranandam et al. [100] present the SmartTestGen (STGen) tool, which creates test cases based on Simulink/Stateflow models. For this purpose, a test specification is used, which includes coverage criteria, such as decision, condition, and Modified Condition/Decision Coverage (MC/DC). Based on a selected coverage metric, the tester can choose between test engines for model checking, random testing, constraint solving, and heuristics-based test case generation. To evaluate the performance of STGen, the authors perform a case study with 20 Simulink/Stateflow models of different automotive applications (for example, powertrain). In comparison to other tools, such as Reactis and Embedded Tester, STGen had a lower performance, but is able to achieve higher coverage on the majority of the evaluated models.

The approach of Cleaveland et al. [99] also applies coverage criteria (e.g., MC/DC) to Simulink/Stateflow models to test system requirements early in development. For this purpose, requirements are transferred to assertion blocks, which evaluate Simulink signals using Boolean expressions. The assertions are integrated into the Simulink/Stateflow models using the Reactis Validator tool. When the model is simulated, the assertions are checked for violations. The concept of integrating specific testing blocks into the Simulink model is further utilized in other tools, such as Arttest [104]. Arttest uses a signal-based method in which input and internal signals of Simulink models are manipulated by specific control blocks that have to be integrated into Simulink. These control blocks are provided with test signals, which override the original Simulink signals during test execution to manipulate the system behavior. Wiechowski et al. [104] demonstrate this process in a case study on the functionality of a start-stop engine. Hansen et al. [105] further demonstrate the application of Arttest in MiL and SiL testing on a window control system.

Skruch and Buchala [103] also use test blocks that are integrated into Simulink models and include test signals. The system is viewed as a set of inputs, outputs, and state variables depending on time. Tests are described as a set that includes certain time ranges, input, output, and state functions. In Simulink, this can be realized using the *Signal Builder* blocks. The resulting model can be used to derive test cases or as an oracle to compare test results with the intended system behavior. For this purpose, the Simulink model can be compiled and flashed onto a target ECU. Furthermore, the Real-Time Interface (RTI) library of dSPACE AutomationDesk is applied to connect the hardware to the Simulink model. The availability of such interfaces to connect hardware to the Simulink simulation environment enables testers to test software applications early in development in realistic HiL environments. This concept is also used by Belmon and Xu [101], which specifically targets testing of automatic transmissions in a HiL setup. For this purpose, the TestWeaver tool is used to connect a Simulink transmission control application with an environment model consisting of

a vehicle and a gearbox. Thus, specific blocks are added to the Simulink models, which are used to control the model inputs and gather the outputs. The tool is able to automatically generate test scenarios and execute them in MiL, SiL, and HiL simulations.

Nabi et al. [98] further present a HiL test bench for powertrain and engine simulation. The HiL hardware setup consists of a custom dSPACE engine simulation environment, which is controlled by a PC. The test bench is connected to a powertrain control unit that serves as a test object. Additional hardware, such as diagnostic testers or actuators, can also be integrated. Simulation models are created using MATLAB/Simulink and Stateflow. The dSPACE ControlDesk employs python scripts to apply, control, and monitor tests automatically. The authors further illustrate how failure insertion and I/O tests can be executed in the unit, integration, and system test stages.

5) SIMULATION-BASED APPROACHES

In addition to Simulink, further simulation frameworks and methodologies are used in the automotive domain. Table 5 summarizes relevant publications in this area.

Bucher et al. [106], [107] evaluate E/E architectures in vehicles. In their approach, architectures are modeled using the Electronics Architecture and Software Technology - Architecture Description Language (EAST-ADL) or Electric/Electronic Architecture (EEA)-ADL. The authors use the Vector PREEvision tool to model an exemplary ACC system in EEA-ADL. EAST-ADL and EEA-ADL are focused on structural properties, for example, to model the structural topology of an architecture. To address the system behavior, the authors developed an additional behavioral architecture layer. This layer describes the logical functions of the network, for example, message exchange between the ECUs and sensors, which realize the ACC functionality. The logical elements of this layer are mapped to the Ptolemy II simulation framework. Furthermore, a mapping to hardware-related layers is integrated, for example, for bus communication. In this way, links are created between the layers of the logical, behavioral, and hardware-related architecture to conduct simulations across multiple architecture layers. In [108], this approach is employed in a MiL environment using an E/E architecture example, which consists of several ECUs, such as RADAR, Camera, and Powertrain, as well as CAN bus communication.

Neubauer et al. [109] further extend this approach to test and verify Advanced Driver-Assistance Systems (ADASs) in a virtual simulation environment to comply with the regulations of the United Nations Economic Commission for Europe (UNECE). Test scenarios, specified and required by UNECE (regulation UN R131 [112]), are applied to an exemplary Advanced Emergency Braking System (AEBS). Each test scenario requires a driving behavior model. For this purpose, the E/E architecture simulation environment [106] is extended by the 3D simulator OpenDS, which applies the OpenDRIVE standard and OpenSCENARIO data format. Thus, the required UNECE test scenarios can be created and automatically verified through simulations using OpenDS.

Simulations can be further introduced into the completed vehicle to test sensor and actuator systems, as shown by Berger [110]. This approach focuses on validating customer requirements by performing acceptance tests. For this purpose, an autonomous vehicle is equipped with a camera and a laser scanner. To test these sensors, a Driver-in-the-Loop scenario is defined. Therefore, the sensor data are simulated in a virtual 3D environment (e.g., to provide camera pictures). For this purpose, the vehicle systems and their surroundings are described in a Domain Specific Language (DSL) to create use cases or driving scenarios using the tool MontiCore. The author further applies the Hesperia framework [113] to virtually create sensor data (e.g., for cameras). In this manner, the physical sensor systems are provided with realistic data.

Finally, Plummer [111] introduces a MiL framework in which a simulated numerical model is interacting with the physical SUT. The approach is applied to an vehicle aerodynamics and a tire model. The focus of that paper is on the specification of the numerical models and the setup of the MiL framework. However, the author further highlights how the framework can be extended by numerical or physical sensors and actuators.

6) FURTHER XiL-BASED APPROACHES

In this section, we present remaining XiL-based publications that did not match aforementioned categories from the previous sections. In Table 6, these publications are summarized.

Schoitsch et al. [114] present the Dependable Embedded Components and Systems (DECOS) test bench, which targets architectures of embedded, safety-critical systems. This test framework is used to validate and certify DECOS-related systems. For this purpose, the system must comply with safety cases, which proves that safety requirements are fulfilled. The framework can process UML and MATLAB/Simulink models and provides tools to perform safety analysis techniques, such as Fault-Tree Analyses. Thus, safety requirements demanded from standards, such as IEC 61508 or ISO 26262, can be fulfilled. Several test techniques, such as conventional functional testing, fault injection, but also theorem proving and model checking, are available to validate and verify the SUT. The authors highlight, how the test framework is applied to different domains, such as automotive (door-control and crash warning systems), aerospace, and industrial control systems.

Shin and Lim [115] present an approach to generate test cases for hardware and software testing. A power window switch module is used to create a system model as a UML state diagram. The authors implemented a custom parser, which analyzes the model and extracts an abstract syntax tree. To automatically generate test cases from this tree, a Breadth First Search (BFS) algorithm is used in combination with transition coverage. The resulting test cases are applied

TABLE 5. Survey results for model-based test approaches regarding simulations.

Paper	Approach	Goal	Model	Stage	Area	Application	Tool Support	Auto.
[106]	Simulation	System V/V	EEA/EAST-ADL, FSM	MiL, DL	EEA, COM (CAN), ECU, Sensor (RADAR)	ACC	Vector PREEvision, Ptolemy II	(✓)
[107]	Simulation	System V/V	EEA/EAST-ADL, UML State	MiL, DL	EEA, ECU, V2X, Sensor (RADAR, Camera), COM (CAN, GPS, 4G)	ACC, Lane Departure Warn., Pred. Powertrain Control Environment Perception	Vector PREEvision, Ptolemy II	(✓)
[108]	Simulation	System V/V	EEA/EAST-ADL, UML State	MiL, DL	EEA, COM (CAN), ECU, Sensor (RADAR)	ACC	Vector PREEvision, Ptolemy II	✓
[109]	Simulation	System V/V	EEA-ADL, Hierarchical FSM	MiL, DL	EEA, ECU (ADAS)	Adv. Emergency Braking System	Vector PREEvision, Ptolemy II, OpenDS	✓
[110]	Simulation	System V/V	DSL	AT	Actuator, Sensor (Camera, LiDAR)	-	MontiCore, Hesperia	(✓)
[111]	Simulation	System V/V	Numerical, physical models	MiL	ECU, Sensor, Actuator	Vehicle aerodynamics, tyres	-	(✓)

TABLE 6. Survey results for model-based test approaches regarding further XiL approaches.

Paper	Approach	Goal	Model	Stage	Area	Application	Tool Support	Aut.
[114]	Model Checking, Theorem Proving, Fault Injection	System V/V	Simulink, UML	HiL	ECU, COM (FlexRay)	Door control, crash warning system	MATLAB, DOORS, VIATRA2, EMI	(✓)
[115]	Search (coverage-based), Simulated Annealing	Test Cases	XML, UML State, Abstract Syntax Tree	HiL	ECU	Power window switch	UML tool, VectorCAST	✓
[116]	Constraint Solving	Test Cases	UML, hierarchical FSM	HiL	ECU, COM (CAN, LIN)	Turn indicator lights	RT-Tester	✓

during the unit test stage. Moreover, the authors demonstrate the reusability of the unit test cases in a path combination approach with a simulated annealing algorithm to address the integration test stage. The software test cases can be reused for hardware testing. For this purpose, the application's software variables are mapped to hardware signals. Thus, tests can also be applied in a HiL environment.

In [116], Peleska et al. present a benchmark model of turn indicator lights of a Mercedes Benz. This model is intended to benchmark testing methods and tools. The turn indicator lights are controlled by an ECU communicating via a CAN and Local Interconnect Network (LIN). To realize communication, interfaces between the system and test environment are presented in UML. The behavior of the system is described using hierarchical state machines, which are similar to Harel Statecharts [30]. In a first benchmark, the authors apply the RT-Tester tool that uses a constraint solving approach to derive symbolic test cases.

B. FAULT-BASED APPROACHES

Fault-based test approaches focus on test cases that result from faults, failures, or defects that can occur in the SUT. Different methods exist in this area, such as fault injection into a SUT or mutation testing. Table 7 presents an overview of the methods used in the automotive domain.

A variant of fault-based testing is fault injection. This test technique is comparable to security testing, in which a system is tested by executing cyber attacks. Instead of attacks, fault injection introduces potential errors into the SUT to test its behavior. Rana et al. [117] apply fault injection to Simulink models. In their work, an exemplary ABS system is modeled in Simulink. Specific blocks are integrated into the model to inject faults. This approach is similar to the test signal blocks of Arttest (Section IV-A4). However, instead of test signals, faults are introduced when the model is executed in simulations. Thus, the system behavior can be observed under different fault conditions in a MiL environment.

Faults can be injected at different technological levels, such as hardware, software, and system level. Svenninsson et al. [118] present the MODel-Implemented Fault Injection (MODIFI) tool, which covers fault injection techniques for these levels. In their paper, the authors address the software and system level based on MATLAB/Simulink models for a pedal voter system example consisting of three pedal sensors. MODIFI covers 30 fault models including bit-flip faults. Additionally, safety requirements are specified in MODIFI that must be met after test execution. Finally, timing aspects are considered to determine the point in time at which faults are injected and to enable system monitoring.

In [119], the authors address hardware-based fault injection. For this purpose, the authors present a toolchain, which includes MODIFI for software/system fault injection. Furthermore, the WAREFOLF tool is introduced that can inject faults on hardware. The test results of the software/system fault-injection with MODIFI are forwarded to WAREFOLF, which injects the faults physically on an associated hardware platform. The impacts of these tests can be investigated using a debugger. The authors apply this toolchain to an automotive application consisting of 300 Simulink blocks. In total, 1925 fault-injection experiments were conducted. For the majority of the experiments, WAREFOLF and MODIFI provided similar test results on software and hardware.

In [128], Ungermann et al. present an approach for an automated generation of fault signals to identify faulty components during vehicle diagnosis. Thus, fault-based test

TABLE 7. Survey results for model-based test approaches regarding fault-based methods.

Paper	Approach	Goal	Model	Stage	Area	Application	Tool Support	Aut.
[117]	Fault Injection, Fault Bypass Modeling	Test Execution	Simulink	MiL	ECU	ABS	MATLAB	(✓)
[118]	Fault Injection (Model)	Test Cases	Simulink, XML	DL	ECU, Sensor	Pedal sensor voter	MODIFI, MATLAB	✓
[119]	Fault Injection	Test Execution	Simulink	DL	ECU	ECU Application	MODIFI, WAREWOLF	✓
[120]	Mutation Testing	Test Cases	UML State, object-oriented action system, LTS	CT	ECU	Car alarm system	-	✓
[121]	Mutation Testing, Conformance Checking	Test Cases	UML State, object-oriented action system, LTS	CT	ECU	Car alarm system	-	✓
[122]	Mutation Testing, Constraint Solving	Test Cases	UML State, object-oriented action system, LTS, Prolog	CT	Sensor	Exhaust measurement (particle counter)	MoMuT::UML, Papyrus MDT, Z3	✓
[123]	Mutation Testing	Test Cases	Graph, FSM, Statecharts	CT	ECU	ACC	Mutant and Test Set Generator (MTSG)	(✓)
[124]	Symbolic Execution (defect-based)	Test Cases	Simulink	MiL, CT	ECU, Sensor	Temperature sensor unit	MATLAB, 8Cage, KLEE	✓
[125]	Symbolic Execution (defect-based)	Test Cases	Simulink	MiL, IT	ECU	Electric engine control	MATLAB, 8Cage, KLEE, OUTFIT	(✓)
[126]	Symbolic Execution (defect-based)	Test Cases	Simulink, Stateflow	MiL, CT, IT, ST	ECU	Electric engine control	MATLAB, 8Cage, KLEE, OUTFIT, Controller Tester	(✓)
[127]	Fault activation analysis, error propagation analysis	Test Planning	Simulink, Markov Chains	RT	ECU	Transmission gearbox	MATLAB, Reactis Tester	✓
[128]	Simulation	Test Cases	Graph, Differential equations	DP	ECU, Sensor	Spark ignition engine	-	✓

selection algorithms are applied to a structural graph of a spark ignition engine. As a result, operating situations are found that lead to faults. A custom algorithm is applied to select input signals based on these operating situations, which are used for testing the engine.

In [127], a fault-based approach for regression test suite prioritization is presented. This work aims to reduce the test effort in regression testing of MATLAB/Simulink systems. Thus, only a limited test set must be executed on the system after changes. This is done using a priority table, which assigns priority values to the test cases. Two activities are applied for this purpose. First, a fault activation analysis is performed to estimate probabilities of stimulating faults in the system. In the next step, an error propagation analysis is applied in which a dual-graph is automatically generated from the Simulink model. The resulting model is used in a Markov-based analysis to estimate the number of errors in the system and derive the priority table. The authors apply the Reactis Tester to generate a test suite for a gearbox system used for transmission in vehicles. Another variant of fault-based testing is mutation testing as illustrated in Fig. 10.

The original model or code of a system is slightly changed (mutated) to create a mutant model. Specific mutation operators are applied for this purpose.

The original model is tested using a set of test cases to evaluate its correctness. When the original model passes the tests, the mutated model is also tested with the same test set to analyze whether there is a different outcome. If the mutated model also passes a test, the test case cannot detect the introduced fault (mutation). If a test fails, the mutation is detected (the mutant is killed). Mutation testing is primarily used to evaluate and improve the quality of tests.

Mutation approaches in an automotive MBT context were investigated in the Model-based Generation of Tests for Dependable Embedded Systems (MOGENTES) project [129]. This project aims at model-based testing of

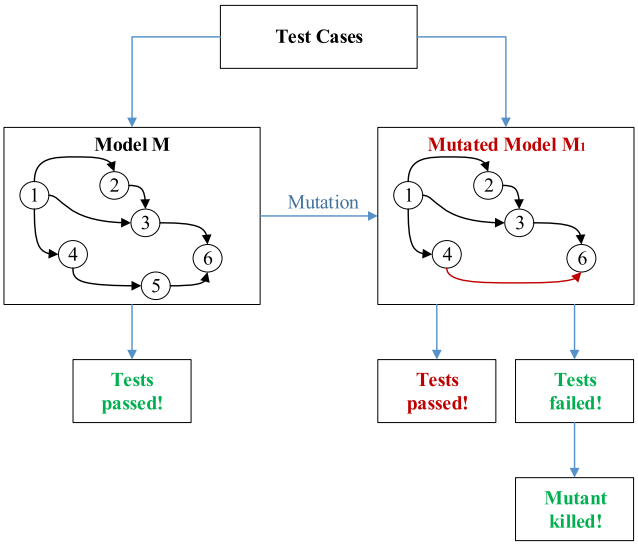

FIGURE 10. Overview of the basic mutation testing process in which a model is mutated and both models are tested by the same test case set. In this way, the quality of the test cases can be determined.

embedded systems. Herzner et al. [120] explain how this project addresses three variants using either UML, Simulink, or fault injection. The authors show an example of the UML approach in which an exemplary car alarm system is modeled as a UML state diagram and transformed to object-oriented action systems. Mutants are created by introducing faults into the original system model. Related fault models are introduced by Schlick et al. [121]. Abstract test cases are generated using the original and mutated object-oriented action systems by checking conformance between both models. The resulting test cases are transformed to concrete test cases to execute them on a target system.

An application of this approach to an automotive exhaust measurement system (particle counter) is presented by Aichernig et al. [122]. This case study suggests that a

combination of random and mutation-based test case generation provides a low number of test cases and the best fault detection rate. Belli et al. [123] show a mutation-based approach that applies a graph-based method in which SUTs are presented as graphs (e.g., directed graphs, FSMs, and Statecharts). Two mutation operators for insertion and omission are presented. These operators are integrated into the system model to create mutants. Test cases are generated based on the original system model and the mutated model. The approach is evaluated in a case study that includes an ACC.

Further fault-based test methods rely on fault or defect-models, which characterize specific failures in systems. Holling [126] aim to generate tests based on defect-models, which can be applied for unit, integration, and system testing. For this purpose, the authors introduce the tool 8Cage [124], which is able to generate test cases for MATLAB/Simulink models at the unit test stage.

This approach focuses on faults based on computational properties, such as run time failures. These types of faults are modeled as Simulink blocks with 8Cage. The application of the tool is demonstrated based on an example of two temperature sensors that provide the measured temperature to a control unit. First, a static check of block properties is carried out to detect potential fault locations in the model, for example, a block in which a division through zero could occur. Actual test case generation is performed by dynamic property checks of a block's I/O values. For this purpose, the KLEE tool is applied to symbolically execute the derived Simulink code. In this step, test input data are generated to execute tests on the target system in a MiL test environment.

Holling et al. [125] address the integration test stage by introducing the OUTFIT tool. The aim of this approach is to achieve higher coverage in integration testing and early detection of errors. OUTFIT creates integration test cases by reusing unit test cases from former unit tests, for example, from the 8Cage tool. Alternatively, KLEE can be applied to automatically generate tests using symbolic execution. The unit tests are combined with two Simulink failure models for superfluous or missing functionality and untested exception or fault handling. The SUTs of an electric engine's control software is used by KLEE to automatically generate integration test cases, which are executed in a MiL environment.

In [126], Holling extends these approaches to perform defect-based testing on the system test stage. The approach again applies MATLAB/Simulink and Stateflow system models of an electric engine control software. As previously described, the 8Cage and OUTFIT tools are used for unit and integration testing. To cover the system test stage, the Controller Tester tool is applied using several failure models from related work. The overall approach is embedded into a comprehensive lifecycle framework that supports the test process by applying defect models.

C. SPECIFICATION-BASED APPROACHES

In specification-based test approaches, formal or informal specifications of systems or tests play a primary role in the process of generating test cases. For example, a test specification typically includes criteria used to derive or select tests as shown in Fig. 1. Table 8 provides an overview on specification-based MBT in the automotive domain.

Offutt et al. [130] present a state-based specification that includes test selection criteria. The authors investigate transition coverage, full predicate coverage, transition-pair coverage, and complete transition sequences. These four criteria are applied to a cruise control system in a case study to derive test inputs for the system testing stage. In [131], Offutt et al. build up on that work and present a formal framework for their specification-based test approach. In a first step, the functional specification of a cruise control system is created as a UML state diagram or Software Cost Reduction (SCR), which is transferred into a specification graph. The SPECTEST tool is used to automatically generate test requirements from the specification graph based on transition-pair and full predicate coverage. For each requirement, test values are generated resulting in the test specifications. Finally, test scripts are generated from the test specifications, which can be executed on the cruise control system. The test results show that full predicate coverage provides the highest number of test cases, identified faults, and the highest system coverage.

Baldini et al. [133] use UML specifications to derive tests for the system test stage. The authors aim to close the gap between the design- and system-test-level stage. For this purpose, design-level test cases consist of a set of messages including timing constraints and are modeled as UML use case, interaction, and state diagrams. Design-level messages are translated into test-level messages based on test selection criteria, such as coverage. The resulting test-level messages represent test commands based on the test environment in which system tests are executed. The approach is evaluated on a real-world automotive application, including a CD reader, Global System for Mobile Communications (GSM) communication, Global Positioning System (GPS) navigation, sound amplifier, and voice control.

Zhang et al. [134] use a formal specification to derive test cases. An exemplary speed limit control system is described using differential dynamic logic, which combines continuous and discrete aspects (e.g., differential equations) of a system. Test cases are automatically derived by applying theorem proving techniques. The resulting test cases are transferred into the Modelica language, which enables an execution of tests in the Modelica simulation environment.

In [21], Pretschner et al. use a test case specification for test generation to evaluate model-based testing in comparison with manual tests. Furthermore, the quality of coverage criteria for test selection and their suitability for error detection are investigated. The AutoFocus tool is employed for this experimental evaluation. AutoFocus was introduced by Broy et al. [138] and is used to develop and verify distributed and embedded systems, such as automotive systems. In a first step, the structural setup of a Media Oriented System Transport (MOST) primary is modeled in a system structure

TABLE 8. Survey results for model-based test approaches regarding specification-based methods.

Paper	Approach	Goal	Model	Stage	Area	Application	Tool Support	Aut.
[130]	Search (coverage-based)	Test Data	UML State, SCR	ST	ECU	Cruise control	Rational Rose, UMLTest, Atac	(✓)
[131]	Search (coverage-based)	Test Data	UML State, SCR	ST	ECU	Cruise control	SPECTEST	✓
[21]	Symbolic Execution, Constraint Solving	Test Cases	EFSM	DL, CT	ECU, COM (MOST)	MOST ECU	AutoFocus	(✓)
[132]	SMArDT	Test Cases	SysML/UML/P Activity Diagrams, DSL, XML	ST	ECU	Headlight control	MontiCore	✓
[17]	SMArDT	Test Cases	SysML/UML/P Activity Diagrams, DSL, XML	ST	ECU	Headlight control	MontiCore	✓
[133]	Search (coverage-based)	Test Cases	UML Use Case, Interaction, State	DL, ST	ECU, COM (GPS, GSM)	Infotainment, CD Reader, Navigation, Voice control, Sound amplifier	Simulator	(✓)
[134]	Theorem proving, Simulation	Test Cases	Differential dynamic logic, Modelica	CT	ECU	Speed limit control	Modelica	✓

TABLE 9. Survey results for model-based test approaches regarding machine learning.

Paper	Approach	Goal	Model	Stage	Area	Application	Tool Support	Aut.
[135]	Lexical/Syntactic analysis	Test Execution	Timed automata	HiL	ECU, COM (RFID)	Keyless access system	UPPAAL Taster, NI VeriStand	✓
[136]	Failure prediction, experience-based	Test Cases	Timed automata, Support Vector Machine, Perceptron	ST	ECU	Trunk door	UPPAAL Taster	(✓)
[137]	Model Checking, Machine learning	Test Cases	PLTL	HiL	ECU, COM (CAN)	Remote engine start, Dual Circuit Steering	NuSMV, LBTest, piTest	✓

diagram. This model contains all system components and their interconnections. In addition, the processed data and related data types are specified. The behavior of the ECU is modeled in a state transition diagram, which corresponds to an EFSM. This model is converted into a Constraint Logic Programming (CLP) language. To generate tests, a test case specification is applied that consists of constraints. Test cases are generated by symbolic execution of the CLP code.

In [132] and [17], BMW's Specification Method for Requirements, Design and Test (SMArDT) and its integration in the development and test process is introduced. This approach relies on UML and SysML specifications to describe models of vehicles and their components. The system requirements are specified using SysML activity diagrams and converted to UML/P activity diagrams, which is a specific language profile of UML. Test cases are automatically derived from these models using path coverage criteria for test selection. Test cases are given in Extensible Markup Language (XML), so they can be used across several testing tools. The entire process is based on a survey conducted to identify benefits, the capability for test improvement, and potential environments for model-based testing.

D. MACHINE LEARNING APPROACHES

In more recent approaches, artificial intelligence methods (in particular machine learning) are applied to model-based testing. Table 9 presents an overview of three automotive-related publications.

In [135], a HiL testing approach is shown that uses timed automata in combination with the UPPAAL tool Taster. In this work, a system specification of an automotive keyless access system is used to model timed automata. Taster performs lexical and syntactical checks on the model. The tool is further extended in [136] with an assignment of test priorities. The timed automata models are parsed by Taster. To assign test priorities, the model is traversed and labeled with priorities. Priorities can be determined by experience aspects (safety impairment and user comfort) as well as failure prediction aspects (e.g., the number of ECUs or modifications). Failure prediction aspects are determined using Support Vector Machine (SVM) classifiers, which are trained with corresponding data sets. Experience aspects are determined using perceptron neurons. The combination of both aspects is used to assign test priorities to the model. In this way, prioritized test sequences are created by a test generator. The authors evaluate the resulting test cases on a real-world trunk door ECU.

To investigate the usefulness of learning-based approaches for testing safety-critical systems in vehicles, Khosrowjerdi et al. [137] present two case studies of safety-critical real-world automotive systems (remote engine start and dual circuit steering). This learning-based approach does not require a system model. Instead, previously specified test cases for the real SUT are used to automatically reverse engineer the system to create a reference model. This model learned its behavior from the real system. Test case generation based on the reference model is executed using the NuSMV model checker. For this purpose, the requirements of the system, which should be verified, are specified in Propositional Linear Temporal Logic (PLTL). NuSMV uses the reference model and PLTL specification to create counterexamples, which are used as test cases. The overall process is implemented in the LBTest tool. LBTest was compared to a mutation testing approach of the piTest tool. Regarding error

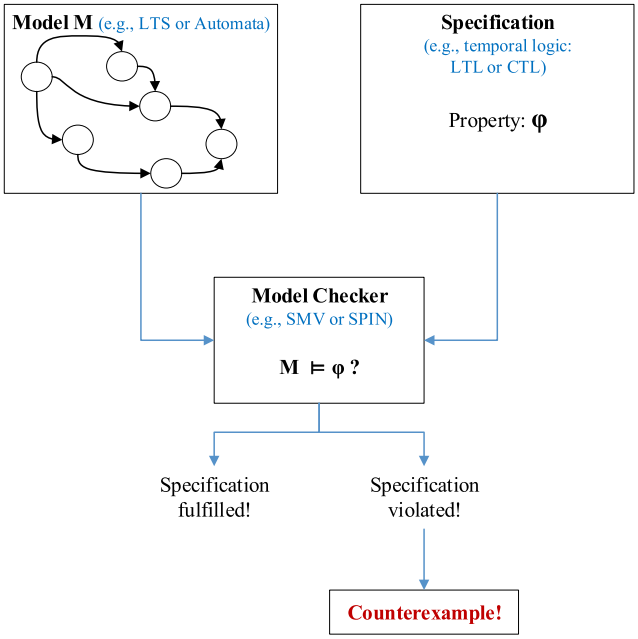

FIGURE 11. Basic model checking process in which a model is evaluated against a certain specification. In case of a violation, a counterexample is produced.

detection, LBTest detected eight out of ten errors, whereas piTest found two errors. However, the authors also highlight disadvantages of LBTest, for example, not all requirements can be formalized into PLTL and thus not be tested.

E. MODEL CHECKING APPROACHES

Model checking is a technique to verify a formal model for conformance to a given specification. The formal model is typically defined as an automaton or Labeled Transition System (LTS) [142]. As illustrated in Fig. 11, the specification is typically represented in temporal logic, for example, Linear Temporal Logic (LTL) or Computation Tree Logic (CTL) [143]).

Model checking tools, such as Symbolic Model Verifier (SMV), verify whether the model conforms to the specification. For this purpose, the model is analyzed for reachable states. If the specification is violated (i.e., a violating state is reached), a counterexample will be produced. In MBT, these counterexamples are often used as test cases. In the automotive domain, this technique is used in MBT approaches as listed in Table 10.

Gargantini and Heitmeyer [141] follow the process illustrated in Fig. 11. A case study of four different systems (e.g., an automotive cruise control system) is performed. The system requirements are described as a SCR specification. The specification is further translated into the input languages used by the SPIN and SMV model checkers. The temporal specifications are described in LTL and CTL. The model checkers derive the expected test outputs and subsequently test cases based on counterexamples.

A similar approach is presented by Ammann et al. [140]. In their work, a cruise control system is specified as a finite state machine to apply the SMV model checker. The authors use CTL to specify test coverage criteria. The focus lies on the verification safety-property violations that provoke dangerous system behavior in certain states. To generate tests, a mutation model based on a mutation analysis is created from the original system model. Test cases result from the counterexamples, which are produced during model checking.

A more comprehensive model-based test methodology is presented by Marinescu et al. [139]. In this work, several stages of the development process are addressed in an extensive toolchain. In a first step, a system designer has to specify system models for a brake-by-wire system in EAST-ADL using the ViTAL tool. For each EAST-ADL model a corresponding timed automata model is created. The timed automata and EAST-ADL models are integrated into a single formal model using the UPPAAL PORT tool. From this point, development and test activities are conducted in parallel. A developer utilizes the UPPAAL model to implement C code, which represents the SUT. At the same time, a tester generates test cases by performing model checking with UPPAAL PORT. For this purpose, a coverage criterion is applied based on the system requirements in the Timed Computation Tree Logic (TCTL). The resulting TCTL requirements are used by UPPAAL PORT as temporal specification. The produced counterexamples are used as abstract test cases. Tests are further refined into Python scripts to make them executable on the implemented C code using the Farkle tool. The results of executed tests can be fed back to the developer to correct implementation errors and test the SUT again.

F. STATE-BASED APPROACHES

State-based approaches are widely used in conventional testing. The survey of Lee and Yannakakis [27] consists of publications starting in the 1950's. Thus, there is a large body of research on MBT regarding state-based models. Table 11 lists automotive-related approaches.

Publications also use state-based models. One example is the TPT methodology (Section IV-A2) that uses state machines as test cases (Fig. 7). Additionally, the Stateflow framework of Simulink can be mentioned (Section IV-A4). In these publications, the state models are integrated into specific approaches or tools as a part of a broader process. In contrast, the state-based models in the following publications form the central aspect of the respective methodology.

A common approach to generate test cases from state-based models is the use of heuristics, or coverage criteria in combination with derivation algorithms, such as search-algorithms. A similar approach is proposed by Hierons et al. [144]. In this work, the μSZ specification language is used to specify the behavior of an ACC system. μSZ represents a Statechart in which corresponding elements, such as guard conditions, are specified in the Z language. The Statechart is a representation of an EFSM that is used as a formal model. The authors discuss several test generation techniques, such as transition tours, transition trees, or

TABLE 10. Survey results for model-based test approaches regarding Model Checking.

Paper	Approach	Goal	Model	Stage	Area	Application	Tool Support	Aut.
[139]	Model Checking	Test Cases	EAST-ADL, Timed Automata, TCTL	DL, CT	ECU, Sensor, COM, Actuator	ABS, Brake-By-Wire	ViTAL, Farkle, UPPAAL PORT	(✓)
[140]	Model Checking, Mutation analysis	Test Cases	FSM, CTL	CT	ECU	Cruise control	SMV	(✓)
[141]	Model Checking	Test Cases	SCR, LTL, CTL	CT	ECU	Cruise control	SMV, SPIN	(✓)

TABLE 11. Survey results for model-based test approaches regarding state-based models.

Paper	Approach	Goal	Model	Stage	Area	Application	Tool Support	Aut.
[144]	Search (Heuristic-/ coverage-based)	Test Cases	EFSM, Statechart, μSZ	CT	ECU	ACC	-	(✓)
[145]	Search (Breadth First)	Test Cases	UML State, LOTOS, IOLTS	ST, HiL	ECU	Vehicle diagnostics	TGV, CADP	(✓)
[146]	Simulation	Test Planning	Statecharts	CT	ECU	Cruise control	Custom	✓
[147]	Random Generation (coverage-based)	Test Cases	Perfect formal specification language	CT	ECU	Cruise control	Perfect Developer toolset	(✓)
[148]	Delta-oriented architectural testing	Test Planning	Message Sequence Chart, FSM	IT, RT	ECU, COM	Body comfort system	eDeltaMBT, Xtext, IBM Rational Rhapsody	(✓)
[149]	Search, SMT solving	Test Cases	Simulink, Stateflow, Extended Mealy FSM	DL	ECU	HVAC	MATLAB	(✓)

heuristics, and how they can be applied to the EFSM model to derive test cases.

Chimisliu et al. [145] use coverage criteria to derive test cases. In this work, the SUT is modeled as a UML state diagram. As an exemplary application, a vehicle diagnostics system is used to store diagnostic faults. The UML state diagram is transformed into a Language Of Temporal Order Specification (LOTOS) [150]. LOTOS represents a process algebra that uses mathematical expressions and structures (based on specific axioms) to describe systems and processes. This enables an application of the TGV tool from the CADP toolbox. To derive test cases from the LOTOS specification, the coverage of LOTOS processes and actions are used. TGV performs a breadth first search on the model based on specified coverage criteria.

Saifan et al. [147] combine coverage with random generation algorithms to investigate the effectiveness of formal specifications in model-based testing. For this purpose, a case study of a cruise control system is conducted. A system specification is formalized using the Perfect formal specification language, which serves as an input for the Perfect Developer tool. A static analysis and theorem proving is performed to verify the correctness of the specification syntax and semantics. Since this specification represents a finite state machine, an adjacency matrix of the connected states is extracted and used to derive paths through the state machine. For this purpose, coverage criteria, such as state coverage, all pair-transitions, or path coverage, are used. A custom C# tool automatically creates test cases from the derived paths using random test generation algorithms in which all states and an average of 77.78 % of the paths are covered.

Briand et al. [146] investigate the effectiveness of model-based test approaches that rely on Statecharts regarding their cost and ability to detect faults. For this purpose, four coverage criteria are investigated: all transitions, all transition pairs, all paths, and full predicate coverage. The investigation is applied to three case studies, including an automotive cruise control system. The system is modeled as a UML state diagram. Test cases are derived based on the coverage criteria. In parallel, mutants of the UML state diagram are created to identify faults. The test cases are executed on the mutants. As a result, a matrix is created that includes the test results. The matrix is used in further test simulations to compare the coverage criteria. The results suggest that the all transition pairs criterion provides the best fault detection capabilities. However, one drawback are the high costs (3-7 times larger than the other criteria). Furthermore, the all paths criterion is highlighted when it is used in state transition trees. Its cost-effectiveness depends on the quality of the transition tree and the test sequences. The authors point out that further research is necessary in this area.

Petrenko et al. [149] present a test approach located at model level. In this method, a system model is provided in Simulink/Stateflow to derive a test model. The test model is represented as a hierarchical extended Mealy state machine with timers. This approach applies a tester-in-the-loop method, a knowledge- and experience-based process, where the tester is responsible for guidance and decisions on the test steps. Test generation is based on a reachability problem in which an input sequence leads to a specific configuration of the state machine. To generate tests, the shortest executable path is searched in the state machine. For this purpose, a flat state machine and a hierarchical state machine are investigated by applying solving tools (for example, a SMT solver). The approach is illustrated in a case study of a HVAC ECU modeled as Simulink/Stateflow. In this case

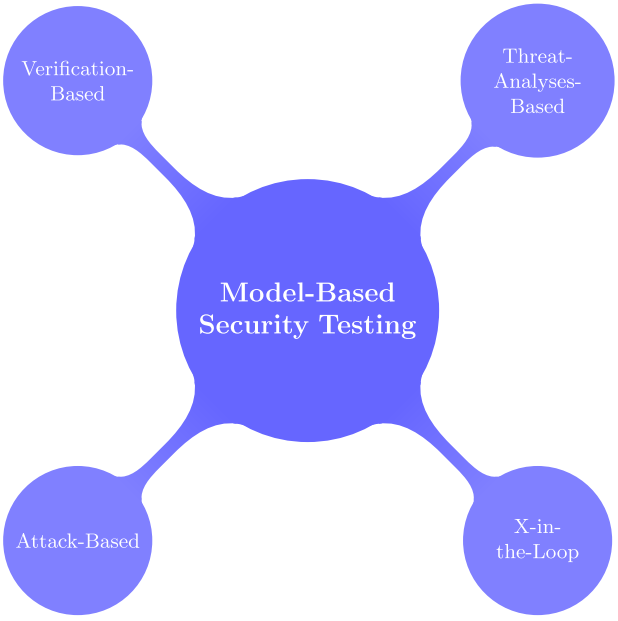

FIGURE 12. Classification of identified MBST publications.

study, an optimized test suite was identified from 40 suites, including 800 tests executed in 90 minutes.

Lochau et al. [148] present an approach that targets integration testing. For this purpose, architecture- and regression-based testing are combined in a delta modeling approach to cover different system types (e.g., vehicle variants). The vehicle architecture of a body comfort system is modeled as a set of connected components. Interactions between these components are represented as message sequence charts, whereas the behavior of an individual component is modeled as a state machine. Based on these elements, an architecture test model is created. This model can be used to define different architecture variants depending on the system configuration. Within this approach, the basic test model and subsequently all system variant models are tested. Furthermore, this process is enhanced by an incremental regression approach in which test cases are reused to test all architecture components after system changes.

V. MODEL-BASED SECURITY TESTING IN THE AUTOMOTIVE DOMAIN

During our literature review, 29 model-based test approaches were identified that specifically target security testing. In this section, these approaches are introduced and analyzed to draw conclusions about the current state of model-based security testing in the automotive domain. The publications are assigned according to the categories shown in Fig. 12.

A. APPROACHES BASED ON THREAT ANALYSIS AND RISK ASSESSMENT (TARA)

In automotive security development, the identification of potential threats to vehicles as well as their risks for exploitation plays a major role. This process is performed within a TARA, which is required by ISO/SAE 21434 [11]. The resulting threats and their risks are typically expressed as attack trees [151]. In Fig. 13, an exemplary and simplified attack tree is illustrated.

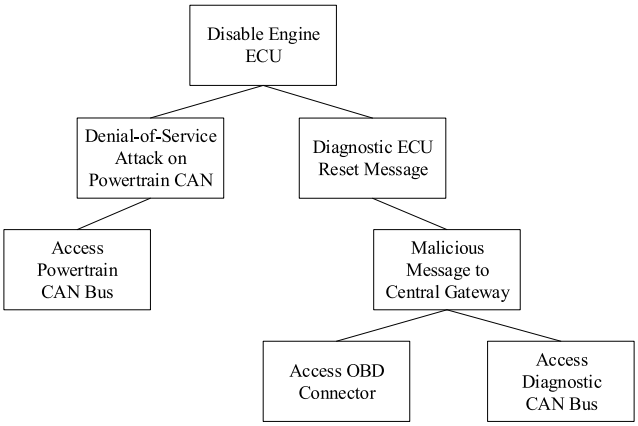

FIGURE 13. Exemplary attack tree in which an *Engine ECU* is disabled by conducting different attack steps.

The root node typically contains an attack target (e.g., *Disable Engine ECU*). The attack steps to compromise this target are listed in the edges or sub nodes of a tree (e.g., *Denial-of-Service Attack on Powertrain CAN*). In leaf nodes, which are located at the end of a branch, the initial action of an attack is described (e.g., *Access OBD Connector*). Additionally, attack trees can be labeled with risk values that describe the probability or feasibility of an attack in relation to its impact on a target system. Since attack trees describe how a system can be attacked and attacks are used to perform security tests, it makes sense to reuse attack trees for testing. An overview of TARA-related MBST publications is given in Table 12.

In [152], Cheah et al. introduce a security evaluation process for third party components, such as ECUs. Penetration tests are performed based on attack trees. Successful attacks are used to specify security requirements to mitigate the identified threats. The authors suggest to reuse these requirement specifications in the supply chain for model-based development and testing. This process is demonstrated in a case study of an automotive Bluetooth interface. In a first step, an attack tree is manually created based on Bluetooth vulnerabilities from literature and vulnerability databases (National Vulnerability Database (NVD)). In subsequent security testing, the authors, for example, mount a file system on a vehicle infotainment ECU via Bluetooth to test this system for vulnerabilities. Based on the identified vulnerabilities, requirements are formulated to mitigate threats. The resulting requirements are combined into a specification and formalized in the process algebra Communicating Sequential Processes (CSP). Based on the CSP model, formal verification and reachability analysis are performed. Thus, third party systems can be evaluated by vehicle manufacturers. This approach is the foundation of further publications from Cheah et al., which extend this concept but focus on different development and test aspects.

TABLE 12. Survey results for model-based security test approaches regarding TARA.

Paper	Approach	Goal	Model	Stage	Area	Application	Tool Support	Aut.
[152]	Formal Verification, Search (Reachability Analysis)	Security V/V	Attack Trees, CSP	PT, DL	ECU, COM (Bluetooth)	Bluetooth interface, Infotainment	-	(✓)
[153]	DREAD, PTES	Test Cases	Attack Trees, STRIDE	PT	ECU, COM (Bluetooth)	Bluetooth stack	Custom Bluetooth	(✓)
[154]	Refinement Checking	Test Cases	Attack Trees, CSP	PT	COM (Bluetooth, CAN)	OBD Device	FDR	✓
[155]	Refinement Checking	Test Cases	Attack Trees, CSP	PT	ECU, COM (Bluetooth, CAN)	OBD Device, Infotainment	Custom Bluetooth	(✓)
[156]	PTES	Test Cases	Attack Trees	PT	ECU, COM (WLAN)	OTA Update	FDR, ADTool	✓
[157]	Threat-Model-Based	Test Cases	CSP, XML, CAPL	DL	EEA, ECU, COM (CAN, FlexRay)	Gateway	FDR	✓
[158]	Simulation	Attack simulation	Graph. data-flow model, Risk matrix	DL	ECU, COM (CAN), EEA	Radio system, Firewall	securiCAD	(✓)
[159]	Simulation	Attack simulation	Attack Graph, vehicleLang vehicleLang	DL	ECU, Actuator, COM (CAN, OBD, Bluetooth, Ethernet)	Damper system, Diagnostics (UDS, KWP) (UDS, KWP)	securiCAD	(✓)
[160]	Simulation	Attack simulation	Attack Graph, vehicleLang	DL	EEA, ECU, COM (Bluetooth, Cellular)	Infotainment, Telematics, OBD, USB, CD	securiCAD	(✓)
[161]	Threat-Model-Based	Test Cases	Attack Trees, STRIDE, SysML	DL	ECU, V2X, Sensor (Camera, LiDAR, RADAR)	Autom. deriving system (lane keep/switch, emergency braking, acceleration, deceleration, steering)	MATLAB, Driving Scenario Designer	(✓)
[162]	Threat-Model-Based, OnSecta	Security V/V	STRIDE, SPARQL, SQWRL	DL	Sensor, Actuator, ECU, V2X, COM (CAN, Ethernet)	ADAS, Telematics, Gateway	ThreatGet, MORETO	(✓)

In [153], Cheah et al. suggest to use the Microsoft Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Priviledge (STRIDE) and Damage, Reproducibility, Exploitability, Affected users, Discoverability (DREAD) approach to identify vehicle threats and build a threat model. This model is represented by a graphical or textual attack tree. Based on this attack tree, penetration tests are performed following the Penetration Testing Execution Standard (PTES) [163]. The authors apply their approach to automotive Bluetooth systems of five vehicles. For this purpose, a proof-of concept tool was created based on Python and employs the *Bluez* Bluetooth stack. The tool implements an algorithm that iterates through the attack tree and executes the attack steps on the system. Thus, several weaknesses were found in the five tested vehicles, for example, a weakness allowing malicious applications to be mounted on ECUs.

In [154], Cheah et al. build up on this approach and present a formalization of their TARA-based attack tree method. For this purpose, a case study of an On-Board Diagnostics (OBD) device with Bluetooth capabilities is conducted. The device works as a gateway, which receives incoming Bluetooth messages and sends CAN bus messages to the internal vehicle network to send or read diagnostic messages. Furthermore, an overview on the attack tree specification for that SUT is provided. A tool was implemented to translate the attack tree to CSP processes, which are used as an environment model of the SUT. The SUT is also formalized in CSP. Both, the system and environment model are given to the Failures Divergences Refinement (FDR) tool. This refinement checker is used to formally compare the attack tree and system specification for automated test case generation. A test case represents a path in the tree. Fifteen test cases were automatically generated from which three tests were successfully executed on a vehicle. Thus, the authors were able, for example, to flood the CAN channel, which resulted in a denial of service in the vehicle electronics and engine function.

In a further approach [155], Cheah et al. further extend the previous methodology and semi-automatically assign severity rating values, which are based on the E-safety Vehicle Intrusion Protected Applications (EVITA) severity rating scheme. Depending on the specific values, a security assurance case is created as specified in SAE J3061 [10]. The overall process is illustrated by an example of an automotive Bluetooth interface used in infotainment systems and OBD aftermarket devices. The investigations consider denial of service attacks as well as data extraction attacks.

Mahmood et al. [156] present an approach to model-based security testing of Over-The-Air (OTA) updates. This methodology builds up on the work of Cheah et al. [154] and considers PTES [163]. The first step involves intelligence gathering to identify known vulnerabilities and entry points into the system. For this purpose, the authors use a SUT consisting of two ECUs, a switch, and a PC server. In the subsequent threat modeling phase, vulnerability information is used to generate attack trees by applying the ADTool. The attack trees are transformed into CSP processes as in [153]. This enables the FDR refinement checker to generate test cases. Subsequently, tests are automatically executed on the target system. This is demonstrated by an example of modifying or adding an ECU update image.

A similar approach based on CSP is presented by Santos et al. [157]. This method specifies vehicle bus systems (e.g., CAN or FlexRay) within a network architecture as a CSP model. A threat model is used to describe attacks on this system. For this purpose, four attacker types (e.g., thief) are characterized, which assume that an attacker has full access to the network. The attacker is modeled as a CSP process, in which attack techniques (e.g., spoofing or eavesdropping) are integrated. The system model is combined with the threat model to generate executable test cases. The FDR

refinement checker is used to generate test cases, which are available as XML or CAN Access Programming Language (CAPL) code.

In addition to conventional TARA approaches, potential threats and their risk can be assessed by simulating attacks on a system. For this purpose, tool support is necessary. In the automotive domain, the securiCAD [164] tool is used to support TARA. Overall, the tool rather focuses on risk analysis during the development process. However, simulating attacks and their ability to generate attack paths create a use case for security testing.

Xiong et al. [158] present a threat modeling approach for performing attack simulations using the securiCAD tool that is used to create attack graphs based on a vehicle system specification. The authors demonstrate this using the E/E architecture of a Jeep Cherokee and Cadillac Escalade, which were attacked by Miller and Valasek [165]. First, threat models are created for both vehicles. The network topology is modeled graphically based on the E/E architecture. In addition, a graphical model of the data flow between the network components is created. Assets and security properties are assigned to each component. Assets are also assigned to risk values based on the attack probability and impact. This enables simulations of different attacks on the network models. The authors demonstrate this by deactivating and activating the firewall of a radio system. The attack simulation results in a risk matrix in which risk changes depending on the firewall state and executed attack. In addition, exploited attack paths are illustrated graphically. A further factor is the estimated time required by an attacker to perform an attack. It is assumed that an attacker always chooses a path that takes the least time. This allows the security design of a vehicle network to be evaluated and compared. For both vehicle models and associated security settings, the authors provide examples of attacks that are considered (e.g., replay attack on the CAN bus).

Another work that uses securiCAD is presented by Ekelund [159]. In this work, a damper system is evaluated for security problems. The system architecture includes a CAN bus connected to three ECUs (one controls the damper actuators), a Bluetooth-enabled gateway ECU, and an OBD interface. In a first step, a threat model is created using vehicleLang, a specification language introduced in [166]. This language is based on the Meta Attack Language (MAL) [167]. By specifying a vehicle architecture in vehicleLang, the network can be analyzed for potential vulnerabilities. Therefore, the language provides elements, such as ECU, software, network, bus systems, and dataflow. Based on vehicleLang, a threat model is created for the damper system. Assets are created for individual network elements (for example, an asset for the Bluetooth network) in vehicleLang. In addition, the attack steps (for example, man-in-the-middle), their impact on the respective components, and suitable security mechanisms are specified. In a further step, a model of the system (E/E architecture) is created using securiCAD that imports the vehicleLang assets. This allows attack simulations to be performed based on the asset configuration. The previously specified attack steps are executed, with security mechanisms switched on or off. The attack simulation results in attack paths, for example, to manipulate the damper system via the OBD or Bluetooth interface. As in [158], this is more likely to be used in TARA than in security testing. An additional limitation is the manual modeling of attack steps that must be specified for each asset. This results in a higher threat modeling effort because further attack techniques have to be added manually. Instead, simulations can be used to evaluate and compare the use of different security mechanisms and their effectiveness in protecting against specific threats.

In the work of van der Schoot [160], vehicleLang is examined to determine its suitability for threat modeling in the automotive field in conjunction with the securiCAD tool. For this purpose, four vehicles are examined that had been successfully attacked in the past. For each vehicle, an E/E architecture model including data flows is created in securiCAD and vehicleLang. Both internal vehicle connections and wireless communication systems, such as Bluetooth or cellular communication, are considered. ECUs, such as infotainment and telematic components, as well as physical interfaces (e.g., OBD), are investigated. For the vehicleLang and securiCAD models, attack graphs are created based on previous attacks on the four vehicles. The aim is to compare the models created in threat modeling with those that are based on executed real-world attacks. It was found that some attacks could not be modeled in vehicleLang or only with high difficulty (e.g., obtaining the firmware of a component or software-specific attacks on applications). The reason for this is a lack of detail regarding assets, attacks, and security mechanisms in vehicleLang and securiCAD. In addition, some hardware details cannot be modeled (e.g., debug ports on ECUs, USB and CD ports). As a result, several real-world attacks cannot be represented. The author proposes a number of improvements to the specification of vehicle systems and attacks. These include a combination of penetration testing and modeling with vehicleLang and securiCAD to identify areas where improvement is necessary. In addition, an integration of vulnerability databases is proposed to map attacks on the models.

Suo et al. [161] present an approach to apply security testing to automated vehicles at an early stage during development before security requirements and mitigations are determined. In this work, a test-driven method is introduced, considering security and safety in a combined manner. Thus, safety engineers identify potential safety hazards for the automated vehicle. Security engineers start with threat modeling according to Microsoft STRIDE. Possible threats are identified and modeled as attack trees. These attack trees and their attack risks are mapped to the safety hazards to identify safety-critical security threats. Based on these threats, test scenarios are created to test the system. For this purpose, simulation platforms, such as MATLAB and its integrated *Driving Scenario Designer*, are used. Based on the test results mitigation strategies are derived. The authors demonstrate

their approach in a case study of a connected and automated vehicle. Especially an automated driving system is investigated, which depends on several sensors and external inputs. In particular, the threat of manipulating a sensor's perception capabilities is evaluated in MATLAB. Based on the simulation of related test scenarios, mitigation strategies, such as sensor plausibility checks, are derived. In this way, early model-based security testing supports the threat modeling process and helps to derive mitigations and security requirements.

Shaaban et al. [162] present an approach to assess vehicle threats and validate/verify related security requirements in vehicle networks. Security requirements are represented as protection profiles following Common Criteria (CC) [168] for a Target of Evaluation (ToE) in a vehicle. An ontology-based model of vehicle components is created that is used to identify threats in relation to security requirements. The authors conduct a case study in which a telematics ECU is connected to a V2X communication gateway, a sensor ECU, and ADAS via communication systems, such as CAN and Ethernet. These systems are assigned to four security protection levels and seven security requirements defined in the IEC 62443 standard. By applying the ThreatGet tool, 56 threats classified by STRIDE are identified for vehicle components. Each threat is assigned to a specific security requirement. In addition, a numerical security target value is specified for each threat. To determine if a security requirement mitigates a related threat, the authors introduce the Ontology Security Testing Algorithm (OnSecta). OnSecta checks the model for compliance with security requirements and performs a gap analysis. For each threat, the current security status is calculated as a numerical value and compared to the security target value. OnSecta includes functions that make it possible to specify additional requirements that are applied until the current security value corresponds to the target value. These additional security requirements can then be considered for the real vehicle. According to the automotive security development process specified in ISO/SAE 21434, requirements are derived based on a previously performed TARA. This process is not described in this publication. However, threats are identified automatically using the ThreatGet tool and related security requirements are automatically verified and validated.

B. APPROACHES BASED ON XiL TEST ENVIRONMENTS AND TEST BENCHES

XiL-based approaches have already been presented for MBT (see Section IV-A and Fig. 5). MBST publications also apply XiL test environments and test benches. Table 13 provides an overview of automotive-related publications.

Heneghan et al. [169] present an approach to automatically verify the security of ECU applications. This method applies several tools to an exemplary OTA ECU software update process. The ECU and OTA application are specified within the Vector CANoe tool. Each relevant ECU application is specified in a simulated vehicle network model. This includes an update server, a vehicle gateway to communicate with the server, and a target ECU to receive updates. The OTA application is implemented in CANoe using the CAPL programming language. The simulated CANoe models are translated into CSP using ANother Tool for Language Recognition (ANTLR). This enables an application of the FDR refinement checker. For this purpose, an attack or threat model as well as a model of required security properties is necessary. Regarding security properties, no exact specifications are provided by the authors. Instead, assurance of integrity of the update transmission to the target ECU is discussed as use case. The attack models are based on the Dolev-Yao model, which describes interactive cryptographic protocols. FDR uses model checking techniques to verify whether security properties of the system model are violated by the attack model. This approach is located at design stage of development because the SUT is executed in the CANoe simulation environment. However, further publications can target HiL environments, and thus physical systems can be tested.

A corresponding approach is presented by Wittenberg et al. [170], in which an automatic identification of vulnerabilities in vehicular HiL environments is targeted. This work is based on the vulnerability detection system proposed by Smith et al. [175]. An attacker's point of view is taken and three steps are differentiated: First, information on attacks, vulnerabilities, and threats is obtained from publicly available databases, such as NVD [176]. Acquired data are converted into a uniform format using the Mediation, Alignment and Information Systems for Semantic Interoperability (MAISSI) tool. In the next step, an attack plan generator applies attack-specific data to the SUT. The authors demonstrate this process in a HiL platform. The vehicle is modeled in MATLAB/Simulink and operates in a virtual environment. Individual vehicle systems, such as sensors, can also be replaced by physical hardware. System descriptions and associated attack targets are passed to a network planning tool, which generates attack scenarios for the system components. These scenarios are utilized by the SAT Planner Alloy, which generates an attack graph and applies heuristic search algorithms to identify critical attacks in the system. This involves searching for attack step sequences that lead to previously specified attack targets. Successfully identified attacks are prioritized according to their probability. The presented framework includes a variety of tools that enable the automated testing of vehicle systems based on known attacks and vulnerabilities. However, the authors highlight that high memory capacities and extensive parallel computations and simulations are necessary to perform HiL simulations in particular.

Kurachi et al. [171] present a security testing framework based on HiL simulation. This framework especially targets vulnerabilities regarding transmission timing of CAN messages. A system model is created as a Labeled Transition System (LTS) based on a system behavior specification. Additionally, a threat model is created, which includes attack

TABLE 13. Survey results for model-based test approaches regarding XiL methods.

Paper	Approach	Goal	Model	Stage	Area	Application	Tool Support	Aut.
[169]	Model Checking	Security V/V	CAPL, CSP, ANTLR	CT, DL, MiL	ECU, V2X (Backend), COM (Cellular, WLAN)	OTA update	FDR, Vector CANoe	✓
[170]	Search (Heuristic), Constraint Solving, Simulation	Test Cases	Attack Graph	HiL	Sensor (GPS, Camera, RADAR), ECU, EEA	Vehicle simulation	MATLAB, MAISSI, Alloy, Custom	✓
[171]	Model Checking	Security V/V	LTS	HiL	ECU (AUTOSAR), COM (CAN, OBD)	Direction indicator system	NuSMV, dSPACE (SystemDesk, ControlDesk)	✓
[172]	Simulation	Attack simulation	Simulink	HiL	Sensor (RADAR), ECU, COM (CAN, GPS)	Driver assistance system	MATLAB, Vector CANoe, SUMO	✓
[173]	Simulation	Security V/V	Simulink, ROS	MiL, SiL, HiL	Sensor (LiDAR, Camera), COM (CAN), ECU	Vehicle dynamics	MATLAB, CARLA, ROS	✓
[174]	Attack-based	Test Data	Attack models, XML	HiL	ECU, COM (OBD, CAN)	CAN communication	ATG, USB2CAN	(✓)

patterns, such as spoofing attacks. It is assumed that an attacker has access to the vehicle's internal CAN (e.g., via the OBD interface). In this approach, a direction indicator system that communicates via CAN bus, is used as a SUT. The SUT is simulated using the SystemDesk tool that can model AUTOSAR platforms in a virtual environment. LTS paths of the system model represent a correct system behavior. Using a test driver, system behavior is tested by applying different inputs to the model. To violate this behavior, an attack driver is used to create individual sequences of attack scenarios. The feasibility of an attack is investigated using the NuSMV model checker, which performs a timing analysis of attack scenarios. Feasible attacks are used in security testing to overwrite CAN messages on the bus. The authors employ the ControlDesk tool to execute Denial of Service (DoS) and spoofing attacks on CAN. These attacks were successful in the virtual environment. However, the authors highlight that most attacks do not work in a real-world CAN system. Furthermore, the approach is presented as a HiL framework, but so far no hardware connections have been introduced. Since the approach focuses exclusively on timing attacks on CAN, other attack scenarios are not considered. The authors outline that this limitation can be circumvented by simulating CAN hardware to enable bus-off and spoofing attacks.

Oruganti et al. [172] introduce a HiL test framework for security tests in a virtual environment. This framework focuses on vehicles and the environment in which they operate. The surrounding vehicle traffic is realized using the SUMO traffic simulation tool. Connectivity simulations (e.g., V2X) and network simulations have not yet been integrated. However, the authors plan to add these features using simulators, such as NS-3 and OMNET++. In addition, a telematics functionality will be considered for future work. The in-vehicle network is simulated using Vector CANoe [33]. Controllers, such as ECUs, and their functionality are modeled with MATLAB/Simulink. These models can directly be integrated in CANoe or on real hardware by automatic code generation. Because CANoe provides interfaces for hardware devices, physical components (ECUs, sensors, etc.) can be connected to virtual simulation tools. In this manner, a HiL framework for security testing is realized. To demonstrate the capability of this HiL framework, the authors perform simulations in which a vehicle is navigated virtually on a specific route. An attacker accesses the vehicle network and attacks the virtual RADAR sensor to manipulate sensor data that influences a simulated driver assistance system.

This approach is further extended to enable MiL and SiL testing for safety and security in [173]. In this work, the framework uses CARLA [177] to simulate the environment of an autonomous vehicle. The HiL part of that framework is used to represent an internal vehicle network based on the Robot Operating System (ROS) [178]. Currently, this network contains only a CAN bus. The vehicle behavior is represented by Simulink models, which are connected to the CARLA simulator by ROS. Since this framework is still in a development state, a focus of this work is on the framework setup rather than on procedures for security and safety assurance. It is demonstrated how the framework can be used for testing. However, no further details regarding security test creation or how attacks occur in the virtual environment are provided.

Huang et al. [174] present the Attack Traffic Generation (ATG) tool, which generates datasets for security testing of physical and simulated CAN bus systems. The datasets are based on four types of attack models: Denial-of-Service, fuzzing, target spoofing, and error handling attacks. The latter exploits CAN error handling mechanisms to influence bus communication. The tool is based on a graphical Python application and offers dataset generation, reading and replay of CAN messages, and execution of attacks. For this purpose, hardware devices, such as the USB2CAN device (including the SocketCAN driver library), are used to communicate with the real system via an interface. The authors compare their ATG tool with other well-known CAN tools, such as CANoe or CarShark. In addition, the functionality of ATG is tested on a real vehicle. For this purpose, CAN messages are captured via the OBD interface. By analyzing messages, it was possible to identify and manipulate data fields to display incorrect engine speed values in the instrument cluster.

C. APPROACHES BASED ON ATTACKS AND ATTACKER MODELS

Another possibility for deriving security test cases is the use of attack or attacker models. Attack models typically

describe attacks and their capability to exploit vulnerabilities. Attacker models represent the skills, motivation, equipment, or knowledge of an attacker. This concept is similar to threat modeling as in a TARA (Section V-A). However, in contrast to typical TARA-based or attack-tree-based approaches, the publications in this section focus on attacks and their execution. Thus, individual attack techniques, attack paths, and vulnerabilities are particularly relevant. In the automotive domain, several publications use such models for MBST, as listed in Table 14.

Volkersdorfer et al. [179] present a conceptual process to automate security tests and reviews. This method is based on a previous thesis [184] and consists of an adversary, attack, and target model. The three models are combined to specify an adversary that affects the target system for each attack. Since this approach is still in a conceptual phase, details of the automation have not yet been described. However, in [180], the authors extend this method with the Adversary-Driven Attack Modeling (ADAM) framework in which models are refined as UML class diagrams. The adversary model consists of a *toolSet*, *expertiseSet*, *motiveSet*, and *riskAversionSet*. Each of these sets contains Boolean parameters by which different adversary types can be configured. The target model contains the SUT and its environment. In addition, entry points are defined to represent access to the SUT. The attack model is concretized by an extension of the attack base. This includes known and successfully executed security attacks, for example, from [9]. The ADAM framework utilizes an attack, adversary, and target model to describe attacks on the target system. The adversary model reflects an attacker's perspective, whereas the attack model reflects a technical perspective. This allows different attacks with different attacker types to be combined to test the target system in a MiL environment. The authors demonstrate the application of their method in two attack scenarios. First, a web application is attacked considering a user input field as an entry point. In the second attack scenario, a vehicle ECU is attacked using an OBD interface as an entry point to extract data.

Sommer et al. [181] present an approach for model-based security testing of vehicle networks early in development. In particular, the creation of a security model that is used to automatically derive attack paths is addressed. The security model is based on three elements: a vehicle's E/E architecture, external entities that communicate with a vehicle, implemented security mechanisms, and attack characteristics. Based on the E/E architecture and security mechanisms, a system model is created that describes the in-vehicle components (e.g., ECUs, sensors, and actuators) and their interactions. This can be complemented by an environment model that describes external entities (e.g., smartphones and backend servers). The attack characteristics result from the Automotive Attack Database (AAD) [185] and a corresponding attacker privilege model [186]. This model describes five abstract privilege states (e.g., reading or writing data on a communication channel) an attacker can reach in case of a successful attack. The system model is extended by these privileges resulting in a security model consisting of states an attacker can acquire within a vehicular network by successfully executing attacks. Corresponding cyber attacks are represented by state transitions. In the presented approach, only the process of creating the security model is shown. The authors claim to work on a formalization of this model as an EFSM. However, as a proof of concept, the security model of a vehicle that has been successfully attacked in the past is provided as an example. The authors were able to show that the model contains attack paths that were successfully exploited in reality. This allows the model to be used to derive attack paths for security testing. However, the path derivation process is not described in the publication.

In [182], Mundhenk et al. introduce an approach for analyzing the security of a vehicle's internal network architecture at system level. An E/E architecture is used as SUT including ECUs, communication systems, messages, and interfaces. For this purpose, three exemplary architectures are investigated in which a gateway is connected to three ECUs by CAN or FlexRay. Additionally, security mechanisms, such as the Advanced Encryption Standard (AES) (128 bit) and Cipher-based Message Authentication Code (CMAC) (128 bit), are considered for communication systems. A Markov model is created for each architecture element. For ECUs and interfaces, this process is based on existing exploits. Communication systems and network aspects are seen as passive parts of connected ECUs. The Markov models for communication messages are based on Confidentiality, Integrity, and Availability (CIA). The individual Markov models for all architecture elements are combined into one single system model. Since transitions between states in Markov models are assigned with probabilities, a security and safety assessment of the architecture components is executed. For this purpose, the authors use the exploitability sub score of Common Vulnerability Scoring System (CVSS) [187] and normalize that value to a time frame to consider time aspects in their model. Furthermore, the Automotive Safety Integrity Level (ASIL) [96] metric is applied as a safety value in which patching times are also considered. The CVSS exploitability and ASIL metric are used in combination for the state transition probability in the Markov model. Overall, this model represents the exploitability of architecture components through messages at certain points in time. In this way, the security of communication between components of an E/E architecture can be analyzed. Therefore, the probabilistic model checker PRISM is used, which performs a steady state analysis to investigate, if a certain state can be reached (i.e., ECU can be exploited). The authors are able to show how different combinations of security mechanisms and other architecture designs influence the exploitability and patching rates of architecture components. This enables a more specific assessment of security designs in architectures. However, this work is limited to exploits through communication messages. For example, application vulnerabilities of the components are

TABLE 14. Survey results for model-based test approaches regarding attack and attacker models.

Paper	Approach	Goal	Model	Stage	Area	Application	Tool Support	Aut.
[179]	Simulation	Attack Simulation	UML Activity, Attack Trees	PT	ECU, COM (OBD, CAN)	ECU data extraction	-	✗
[180]	ADAM	Attack Simulation	UML Class, Attack Trees	MiL	ECU, COM (OBD, CAN)	ECU data extraction	-	✓
[181]	Search (coverage-based)	Test Cases	EFSM (graphical), STRIDE	DL	ECU, COM (CAN, GSM, USB)	EEA Attack (Telematics, Infotainment)	-	✓
[182]	Model Checking	Security V/V	Markov Chains, CIA, CVSS, ASIL	DL	ECU, COM (CAN, FlexRay, 3G)	Gateway Architecture (Telematics, Power Steering, Park Assistance), AES (128 bit), CMAC (128 bit)	PRISM	(✓)
[183]	Theorem proving, Simulation	Security V/V	Graph (Graph-based formal framework)	DL	Sensor (Camera, LiDAR, RADAR), V2X (Traffic groups)	Coord. Veh. traffic management (intersections, roundabouts, platooning)	Z3	✓

currently not considered. Furthermore, exact applied exploits are not mentioned. Still, the approach is able to find security property violations considering timing aspects.

Asplund [183] investigates the security of connected and autonomous vehicles while driving in group formations. Therefore, three different autonomous driving scenarios (managed intersections, managed roundabouts, and platooning) are presented. For each scenario, faults/attacks of a hidden vehicle, selfish behavior of a vehicle, and a platooning attack are considered. Additionally, four faults and attacks leading to such scenarios are included: communication failure, detection omission, location falsification, and sybil attack. The goal of this approach is to verify the correctness and secure operation of vehicles in group formations. Thus, vehicles and their positions inside of the group formation as well as sensors are modeled in a graph-based formal framework. The vehicles and sensors follow a group membership protocol that describes how the vehicle operates in a group. All graphs are checked for inconsistencies using a Satisfiable Modulo Theories (SMT) solver. The approach is evaluated using synthetic and realistic data of vehicles in operation. The results suggest, that even though verification works for some scenarios, this is not possible for every scenario. In addition, performance is a limiting factor when the systems and scenarios increase.

D. APPROACHES BASED ON TESTING/VERIFYING SECURITY MECHANISMS, PROPERTIES, AND REQUIREMENTS

In this section, approaches are presented that aim to verify security properties and requirements and the correct function of security mechanisms. Thus, these publications are similar to the MBT approaches described in Section II-A, because the conformance of the SUT to its functional requirements is of particular interest. The identified approaches are presented in Table 15.

Moutappa et al. [188] follow a passive test approach. The message communication of an automotive Bluetooth system is monitored to obtain execution traces. Both control- and data-specific messages are considered. In parallel, the Bluetooth system is modeled as an Input-Output Symbolic Transition System (IOSTS) based on its protocol and security requirements. An IOSTS is based on LTS and integrates additional input and output aspects. In addition, security attacks can be modeled (e.g., the Bluetooth attack *bluestabbing*). Through symbolic execution of the IOSTS, traces can be generated that correspond to protocol and security properties of the requirements. These traces are compared to monitored traces of the real system. To test whether the real system conforms to its requirements, a parametric trace slicing algorithm is applied. The traces are split and checked to determine whether formally specified protocol and security properties are satisfied in the real traces. The authors present their evaluation algorithms and their prototype tool TestSym-P. In a case study, they detected both incorrect execution traces and security attack traces using their tool.

Huang et al. [189] present a method to verify timing constraints regarding safety and security in Vehicular Ad Hoc Networks (VANET). For this purpose, the authors use an example of a cooperative automotive system in which three vehicles are driving behind each other at a safe distance. The vehicles communicate with each other and with a roadside unit. Thus, the vehicle speeds and positions are broadcasted. For communication, the roadside unit aided (RAISE) communication protocol is used, which authenticates messages using a Message Authentication Code (MAC). The authors specify 11 requirements that have to be fulfilled. These involve security-related requirements, such as integrity and freshness of the messages, but also functional requirements, such as the maximum safety distance between the vehicles, and timing requirements. The behavior model of the system is expressed as Stochastic Timed Automata (STA) in UPPAAL-SMC. This system model is extended by the RAISE authentication protocol [194] and a model for security attacks that are modeled as STA. Security attacks involve message falsification, spoofing, and replaying. To verify the system, timing constraints for safety and security are specified in the Probabilistic extension of the Clock Constraint Specification Language (PrCCSL) [195]. These constraints are transformed into a timed automaton and probabilistic queries in UPPAAL-SMC. This tool performs stochastic model checking for each attack. The results show that only falsification and spoofing attacks violate requirements, whereas replay attacks are prevented by security mechanisms. While the approach provides a strong formal background for the individual models, the modeling process can involve a large amount of time and

TABLE 15. Survey results for model-based test approaches regarding verification.

Paper	Approach	Goal	Model	Stage	Area	Application	Tool Support	Aut.
[188]	Symbolic Execution, Param. trace slicing	Test Execution	LTS (IOSTS)	PT, DP	COM (Bluetooth)	Bluetooth COM system	TestSym-P	✗
[189]	Model Checking	Security V/V	Timed automata, PrCCSL	DL	V2X (VANET)	Cooperative automotive systems, Raise, MAC	UPPAAL-SMC, ProTL, ANTLR	✓
[190]	Model Checking	Security V/V	TURTLE, CTL, (RT-)LOTOS, SysML Req.	DL	V2X	Local Danger Warning (message broadcasst)	UPPAAL, TTool	✓
[191]	Requirement-based coverage	Security V/V	Text-based	PT, CT	ECU, COM (WiFi)	Firewalls	AFT, Nmap, cve-search	(✓)
[192]	Symbolic protocol verification	Security V/V	Horn Clauses	DL	ECU, EEA	Key Distribution System (KPS), Message encryption, ECU authentication/authoris.	ProVerif	✓
[14]	Information flow evaluation	Security V/V	Graph	DL	EEA, ECU, COM (CAN, Ethernet Router)	Command messages, HSMs	-	✗
[193]	Model Checking	Security V/V	Graph, TLA$^+$, PlusCal	DL	EEA, ECU, COM (CAN, Ethernet Router)	Command/Data messages, HSMs	MoRA tool, TLC, TLAPS	✓

effort. This makes it difficult to adapt the models when requirements change. Furthermore, the approach focuses on timing aspects. As a result, other security-related aspects are not verified by this approach.

Pedroza et al. [190] introduce a method to verify vehicle applications. A local danger warning system that includes a secure exchange of warning messages between vehicles is used as an example. The Timed UML Real-Time Language Environment (TURTLE) profile is utilized to specify a system model of the danger warning application as class and activity diagrams. In a next step, a Formal Security Model (FSecM) is created, which consists of security requirements, properties, dependencies, and attacks. Security requirements are expressed as SysML requirement diagrams and describe how the system should be protected. The security dependencies specify the context and assumptions under which the security model is realized (for example, knowledge of an attacker/verifier). Security properties that a system should fulfill, such as integrity of freshness, are expressed in the temporal logic CTL. The system attacks are based on the Dolev-Yao threat model and described using TURTLE class diagrams. The authors apply the TTool and integrate their FSecM by translating TURTLE diagrams into the formal languages LOTOS, RT-LOTOS, and UPPAAL. This enables an execution of model checking to verify compliance to the specified security properties. The authors state that they applied this process for the freshness property. However, results of the verification process are not described. Only an overview of the general process and its individual modeling steps is given.

Kastebo and Nordh [191] present an MBST approach that aims for automated verification of security requirements at an early stage in development. For this purpose, the Awesome Firewall Tool (AFT) is introduced, which tests firewalls for compliance with security requirements. Positive and negative testing (Section II-B) is considered. Security requirements are used to create a textual input model that contains general information, such as a security policy or, for example, whether stateful packet inspection should take place. In addition, the incoming and outgoing data traffic and internal interfaces are recorded. Whitelists and blacklists are also considered. The resulting input model is read into AFT. The tool itself is implemented in Python and uses requirement-based coverage as a test selection criterion. Based on the requirements, a port scan of the firewall is performed using Nmap. The cve-search tool is then used to perform an automated vulnerability search using the Common Vulnerabilities and Exposures Enumeration (CVE) database. The results of the port scan and vulnerability search are presented in a test report. The authors tested AFT using a test bench that contains a Wi-Fi ECU with a firewall. Eleven requirements were tested, of which ten were covered. The firewall itself only fulfilled six of the ten tested requirements.

In [192], Lee et al. employ the ProVerif tool to formally verify security properties of network security protocols (particularly cryptographic protocols) in vehicles. ProVerif uses the Dolev-Yao attacker model and automatically verifies protocols that are given as Horn Clauses (formulas of predicate logic). The authors demonstrate this on an attestation scheme for vehicle architectures proposed by Oguma et al. [196]. Here, one or more in-vehicle ECUs serve as a primary, which represents a verification server responsible for verifying other ECUs. A Key Distribution System (KPS) is used to ensure the authenticity and authorization of the ECUs. The message communication is encrypted. Four requirements are formally verified using this protocol: only ECU recognized as valid can participate in communication, unauthorized messages are discarded or processed separately, communication is encrypted and authenticated, and an attack should not affect the entire vehicle.

Verification of security properties is also demonstrated by Jakobs et al. [14], in which an approach for integrity verification of vehicle architectures is presented. The authors view integrity as an information flow problem and discuss existing security measures, such as secure and authenticated boot, Hardware Security Modules (HSMs), or message authentication, to ensure this property. To verify integrity, a model of the architecture is created. In a first step, the logical

communication of system functions is represented as a graph, including integrity levels for each function. Furthermore, a technical vehicle architecture is modeled as a connected graph including ECUs and communication technologies, such as CAN. The technical architecture graph also includes a distribution of logical vehicle functions to individual ECUs. Thus, a relationship exists with the logical communication graph. Considering the logical and technical graph as well as security measures, a graph transformation takes place based on a custom algorithm, which verifies integrity properties for all communication paths. As a result, an evaluation of information flow between paths of the architecture components is performed to provide a statement about information flow integrity. This approach is in a conceptual state, so only the modeling and general verification process is shown for command messages in an exemplary architecture.

Jakobs et al. present an extension in [193]. The authors use the same architecture as in [14] to illustrate their approach. The architecture includes two ECUs and a gateway connected by Ethernet via a router. The gateway is also attached to a CAN bus, which includes two additional ECUs. A first extension is an integration of a risk analysis. For this purpose, the Modular Risk Assessment process (MoRA) [197] is used, which complies with the TARA process required in ISO/SAE 21434. This approach considers assessed components as well as their functions and assigns security goals according to the CIA Triad and potential damage scenarios. Furthermore, threats to these items are identified, which could violate security goals. For each threat, possible attack paths are derived and assigned to risk values. The authors use attack paths and their risk and take the most critical risk value as the integrity level that must be secured. Security controls (e.g., encryption) are considered to reduce the risk and fulfill the required integrity level. The E/E architecture is represented by a logical communication and a technical architecture graph as described above. To automatically analyze the graphs, the Temporal Logic of Actions (TLA$^+$) and PlusCal notation are used to formalize the model. This also enables an application of the TLC model checker to automatically derive counterexamples that violate the architecture's integrity. To evaluate their approach, the authors apply that process to different architectures settings of required integrity levels and security control consideration. The results suggest that realistic architectures with 100 ECUs can be analyzed in less than 10 seconds. A drawback of this approach is the focus on integrity, because other security properties, such as authentication or confidentiality, are not considered.

VI. ANALYSIS

This section presents an analysis of the approaches identified in this survey. For this purpose, the tabular overview of publications in Section IV and V is used to compare individual categories for MBT and MBST (a definition of the categories is given in Section IV). The percentages in following sections describe the relative share of the total number of publications (either 63 MBT or 29 MBST publications). The aim of this comparison is to identify common features of both domains to determine the state-of-the-art and transferability of model-based approaches to security testing. In addition, current challenges of model-based security testing are determined. In this way, further research topics on MBST are identified.

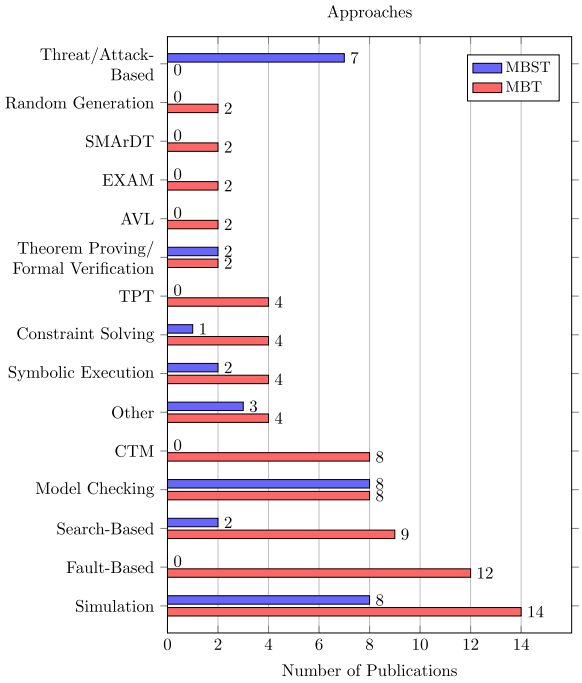

FIGURE 14. Distribution of the test approaches in relation to the frequency they are used in identified publications.

A. ANALYSIS AND COMPARISON OF TEST APPROACHES

Fig. 14 compares the number of MBT and MBST publications in terms of the applied test approaches.

For the MBT domain, simulations are most common with 14 of 63 publications (22.2 %). This suggests early testing in development, which is one of the key advantages of model-based testing. Fault-based approaches are used exclusively for MBT in 12 (19 %) publications. Search-based methods are often used in approaches based on trees or state-based models [115]. This typically involves search algorithms guided by heuristics or coverage criteria [131], [133]. In the field of MBT, 9 publications (14.3 %) use search-based methods. Specific methods, such as CTM, SMArDT, EXAM, TPT, and AVL are exclusively applied in MBT. With a total of 18 approaches (28,6 %), they account for almost one third of all MBT publications. Symbolic execution, random generation, theorem proving, and constraint solving are comparatively less represented with 12 approaches and do not differ significantly in their frequency from MBST (5 approaches). Interestingly, model checking is equally divided between MBT and MBST with 8 publications each. Simulation-based approaches are also used in 8 publications (27.6 % of all MBST publications). Threat- or attack-based methods represent the security-specific counterpart to fault-based

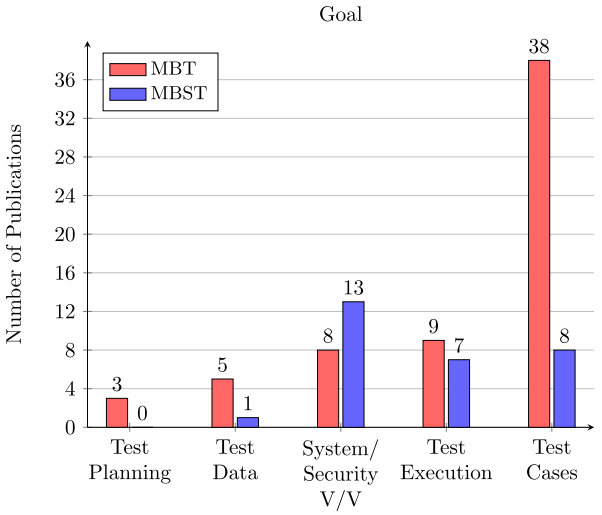

FIGURE 15. Distribution of the test goals in relation to the frequency they are addressed in identified publications. The *Test Execution* also includes *Attack simulation*.

approaches in MBT and are addressed in 7 publications (24.1 %).

Simulation-, threat/attack-based, and model checking approaches are the three most commonly employed MBST methods. In total, 23 publications are included. In 14 of these 23 papers, the design level stage (DL) or MiL is addressed, whereas 9 publications cover penetration testing (PT) or HiL. Thus, future research could investigate how these approaches can be applied in further test stages, such as component, or integration tests. Apart from these three test approaches, hardly any other methods are used in MBST. Interestingly, only two security-related publications use search-based methods, although models are often represented by attack trees or attack graphs (Fig. 16). Thus, future work could aim to investigate search-based methods for MBST more frequently. For example, in attack trees, critical attack paths can be searched for. For this purpose, model checking approaches are also conceivable. This is has already been applied in the area of TARA [198]. In addition, simulation approaches for MBST can be further investigated, because early testing is rarely used in automotive security. However, an early detection of vulnerabilities serves to increase protection against cyber attacks on vehicles. Here, a coupling or transfer of MBT simulation approaches to MBST is also conceivable. With regard to functional security testing, it would be interesting to investigate whether approaches, such as CTM, EXAM, and SMArDT, can be applied to MBST, for example, to test the function of security mechanisms based on security requirements. This could complement other methods, such as those of Lee et al. [192] and Kastebo et al. [191].

B. ANALYSIS AND COMPARISON OF TEST GOALS

This section discusses the goals pursued by the individual test approaches. Fig. 15 provides an overview of goals addressed.

With a total of 38 approaches, the generation of test cases for MBT is represented the most, in sum even more than all the other categories (25 approaches). Test case generation thus accounts for 60.3 %, whereas MBST only consists of 8 approaches (27.6 %). With regard to the execution or simulation of tests and attacks on a SUT, MBT is represented with 9 approaches (14.3 %) and MBST in 7 (24.1 %). Methods that are specifically used to verify or validate a system (e.g., ECUs) are described for MBT in 8 publications (12.7 %). One example is the validation of whether an E/E architecture or its ECUs satisfy safety-critical requirements [114]. Security-related approaches for verification and validation (V/V) are represented in 13 publications. For MBST, this is the primary test goal and accounts for the majority of publications (44.8 %). Eleven of these 13 approaches address early test stages (DL or MiL). Thus, verification and validation of security is primarily performed through early testing. Test planning is addressed in only three MBT publications. This includes the prioritization of tests [127] or determination and reduction of test effort [148]. Theoretically, this category can also include the generation of test cases and test data. Due to the large number of approaches (particularly for test case generation), these categories were considered separately.

The generation of test data is addressed in five publications for MBT and in only one approach for MBST. One reason is the difficulty of generating security test data in advance that directly relates to a successful attack that exploits a vulnerability. This is not the case with MBT (particularly in functional testing), since the behavior of SUTs is functionally specified. Thus, input and output data of a system is known. Here, for example, test input data can be generated based on system requirements, which result in correct functionality or an error in the system. This could also explain why the number of MBST approaches generating test cases is significantly smaller than the number of MBT publications. Thus, test case generation for security testing in the field of MBST has potential for future work. The challenge here, similar to generation of test data, is that it is difficult to determine whether a test case (cyber attack) can successfully exploit or expose a vulnerability. For this reason, it would be valuable to analyze which cyber attacks target which types of vulnerabilities. This can be particularly relevant for test case generation and execution in penetration tests or HiL because the result or impact of an attack is not always observable at the SUT. Thus, sufficient test cases that include knowledge of the impact an attack has can be generated using MBST approaches to support penetration tests.

Finally, another aspect can be discussed in this context. One advantage of model-based testing is the uniform description by using formal models to specify the systems under test, which potentially leads to fewer human errors. In addition, the processes for generating and executing tests can be automated. In general, this is also true for MBST, although some limitations have to be considered in this respect. In conventional testing of a software-based system, the system behavior, input values, and output values are usually known

in advance. This allows tests to be described precisely and expected test results to be specified in advance. Thus, methods of model-based testing can be examined for their efficiency (for example, by measuring the coverage of a test method on a specific SUT). For a (model-based) security test, this is only possible to a limited extent, since a successfully executed test (executed attack) depends on the existence of a vulnerability.

In addition, it may not be possible to determine which influence the exploitation of a vulnerability will have on the system in advance. Thus, the system behavior cannot be predicted or observed. This aspect complicates the evaluation of a model-based security testing method. A detailed investigation of relationships between exploits and vulnerabilities exploited as well as the resulting impact could help in this regard.

C. ANALYSIS AND COMPARISON OF MODELS

Fig. 16 compares the formalisms or types of models used in the identified approaches. A variety of model types is used in both MBT and MBST. In MBT, state-based models with 31 of 63 publications (49,2 %) and Simulink models with 24 publications (38.1 %) account for largest share of the models. For state-based models, this is a result of the large amount of research that exists for testing based on state machines and automata, dating back to the 1950s [27]. In addition, many automotive applications are based on state machines (e.g., Unified Diagnostic Services (UDS) diagnostic sessions [199]). State models are particularly used for test case generation (17 approaches). Furthermore, they are used in several test stages (Fig. 17). In total, 14 state-based approaches are applied in DL or MiL stage, 11 publications address CT, and 13 state-based approaches cover ST or HiL. The large number of Simulink-based models results from the widespread use of MATLAB/Simulink in automotive development. From 24 Simulink approaches, 20 publications address the DL or MiL stage. Aside from these two model types, the MBT publications are distributed across remaining model types, without significant differences being observed.

Since STRIDE is a security-specific classification scheme, only MBST approaches are addressed. Interestingly, CSP models occur exclusively in the MBST domain. Although CSP-based approaches are used in MBT in other domains [200], it does not occur in identified automotive MBT approaches. However, with five publications (17.2 %) it is the most common model type in the MBST approaches after tree models and graphs. Cyber threats are often represented as attack trees in the context of TARA. Thus, this model can also be reused in MBST. In total, 8 of 11 TARA-based approaches apply attack trees. The tree models are particularly used to derive test cases (5 of 8 approaches) for the penetration test stage. They further address testing of ECUs (7 publications) and communication systems (8 publications). This suggests that TARA-based MBST methods are suitable for combining threat analysis and security testing. This is confirmed by the number of TARA-based publications (11 of 29). Instead of tree models, graphs are alternatively used to represent attacks (e.g., attack graphs in [158]). With 6 publications (20.7 %), this model type is the second most frequent. The remaining MBST approaches are distributed in a similar manner to MBT.

Because of the many Simulink and state-based models in the MBT domain, it would be interesting to investigate whether these two model types are suitable for MBST as well. Only four approaches use one of these models. Since both models are used in development and in testing, there is high potential to reuse the models for MBST. The state-based models can be applied to several test stages, whereas Simulink could cover DL and MiL stages as shown in the MBT approaches.

D. ANALYSIS AND COMPARISON OF TEST STAGES

The *Stage* category describes the level at which an approach can be applied to the vehicle life cycle. Additionally, X-in-The-Loop test environments were included, as they also indicate the time of application. Fig. 17 provides an overview of the distribution of publications.

In MBT, MiL testing is most frequently represented with 24 approaches (38.1 %). Fourteen of these 24 approaches also address further test stages, such as SiL or HiL. This suggests that models on MiL are reused in further test activities and that later test stages can build up on MiL. The high number of MiL publications can be attributed to the widespread use of MATLAB/Simulink and comparable tools in development that enable early simulations. Simulink is used in 17 of 24 MiL publications. In this context, the early use of models in development further enables testing at design level (DL), which is addressed in 10 MBT approaches (15.9 %). Fig. 18 and 21 show that MBT primarily focuses on testing ECUs and their software applications. As a result, the Component Test (CT) level is the second most common test stage (18 approaches, 28.6 %). In 16 of 18 approaches, the goal is to generate test cases, and in 17 of these 18 approaches ECUs and their software applications are tested. This also includes SiL approaches, which are represented in 6 publications (9.5 %). With 16 approaches (25.4 %), HiL is addressed by MBT. 10 these 16 approaches are particularly used for test case generation. The System Test (ST) level is addressed in 9 publications (14.3 %). Other stages, such as Integration (IT), Acceptance (AT), and Regression Test (RT) as well as Deployment (DP) and PiL (10 approaches in total), play a subordinate role in MBT.

In MBST, DL is addressed most frequently with 16 publications (55.2 %). Ten of 16 approaches aim to verify or validate security (Section VI-B). In this context, it is worth noting that the number of MiL approaches is comparatively low (3 publications, 10.3 %). One possible reason is that although attacks on systems or the security-specific systems themselves can be simulated in principle (e.g., as in [158], [159], and [160]), the attack impact cannot always be estimated. However, in approaches, such as [173], vehicle systems and

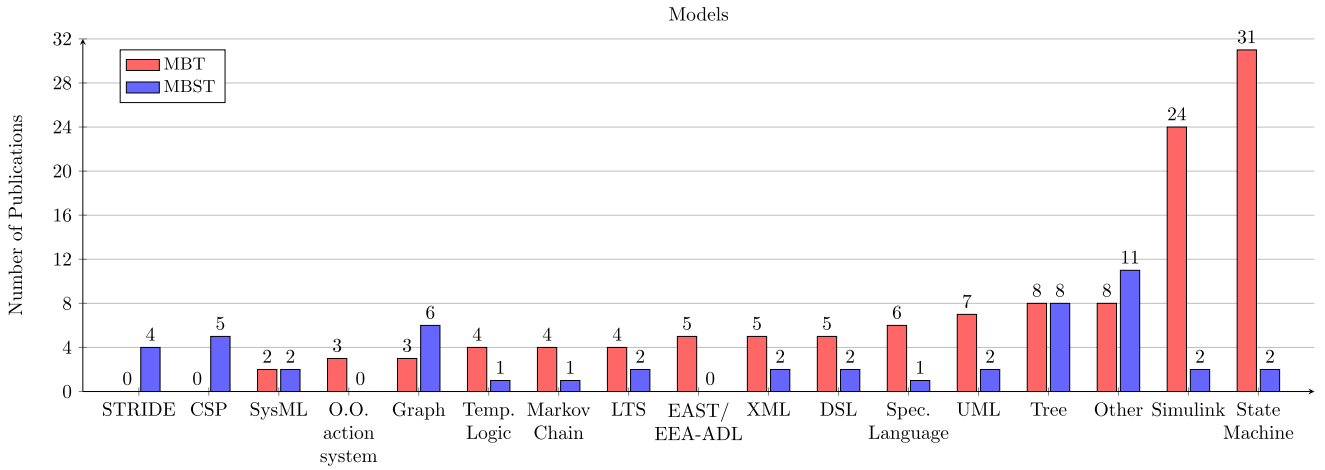

FIGURE 16. Distribution of the model types in relation to the frequency they are used in identified publications.

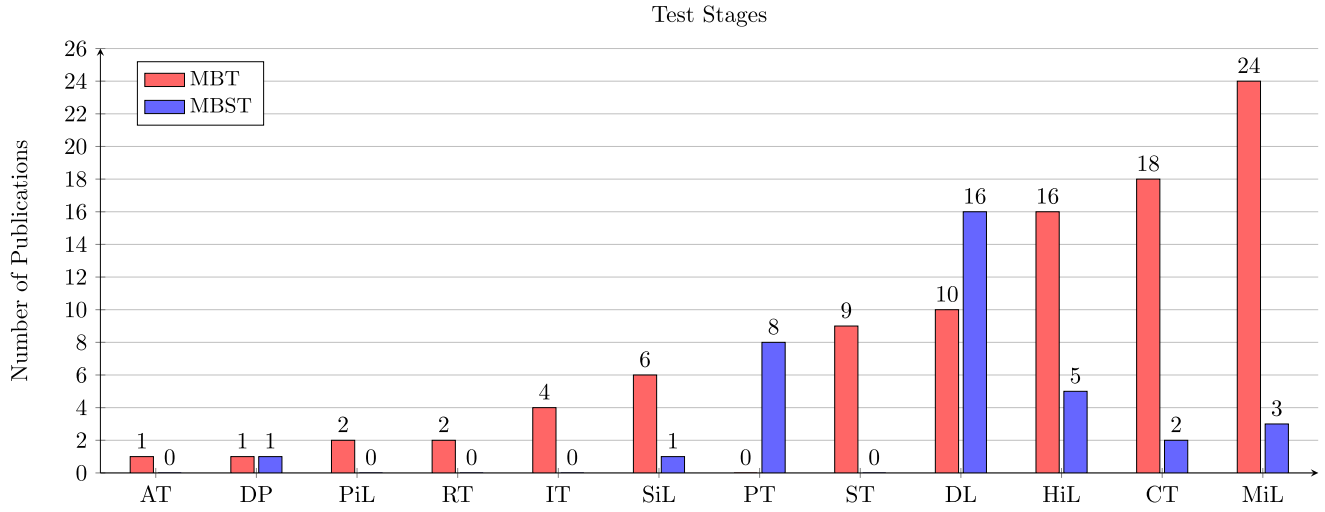

FIGURE 17. Distribution of the test stages in relation to the frequency they are target in identified publications.

surrounding traffic are simulated in Simulink and CARLA, and hardware connections are possible through ROS. This enables observing attacks on hardware and software. For this purpose, HiL frameworks are interesting for executing model-based cyber attacks on real systems to observe their reactions. HiL is addressed in five approaches, accounting for 17.2 % of MBST publications. Future work could research in the direction of MiL and HiL. Eight approaches (27.6 %) address penetration testing (PT) or use it as a supporting tool in their MBST process. Since penetration testing is typically exploratory and experience-based in black or grey box tests, this process can be supported by model-based approaches. In particular, because of the high complexity of modern vehicles, which are based on a large number of components (150 ECUs per vehicle [1]), communication systems (Fig. 20), and concepts related to autonomous driving, there is potential to support the tester with suitable models and methods.

E. ANALYSIS AND COMPARISON OF ADDRESSED TECHNOLOGICAL AREAS

The *Area* category describes the respective technical system areas of a vehicle identified within the publications. An overview of the technological areas addressed by the MBT and MBST approaches is illustrated in Fig. 18.

MBT approaches focus on testing ECUs and their software applications or functions. A total of 60 out of 63 MBT approaches (95.2 %) use ECUs applications as SUTs or as part of them. Communication systems are addressed in 14 approaches (22.2 %), whereas sensors (e.g., RADAR) are included in 10 approaches (15.9 %). Testing actuators, the vehicle as a whole, E/E architectures, and V2X communication plays only a minor role in MBT with a total of 9 approaches (14.3 %). With regard to MBST, the distribution is more balanced. With 24 approaches (82.8 %), ECUs and their software applications are almost equally addressed as communication systems (23 approaches,

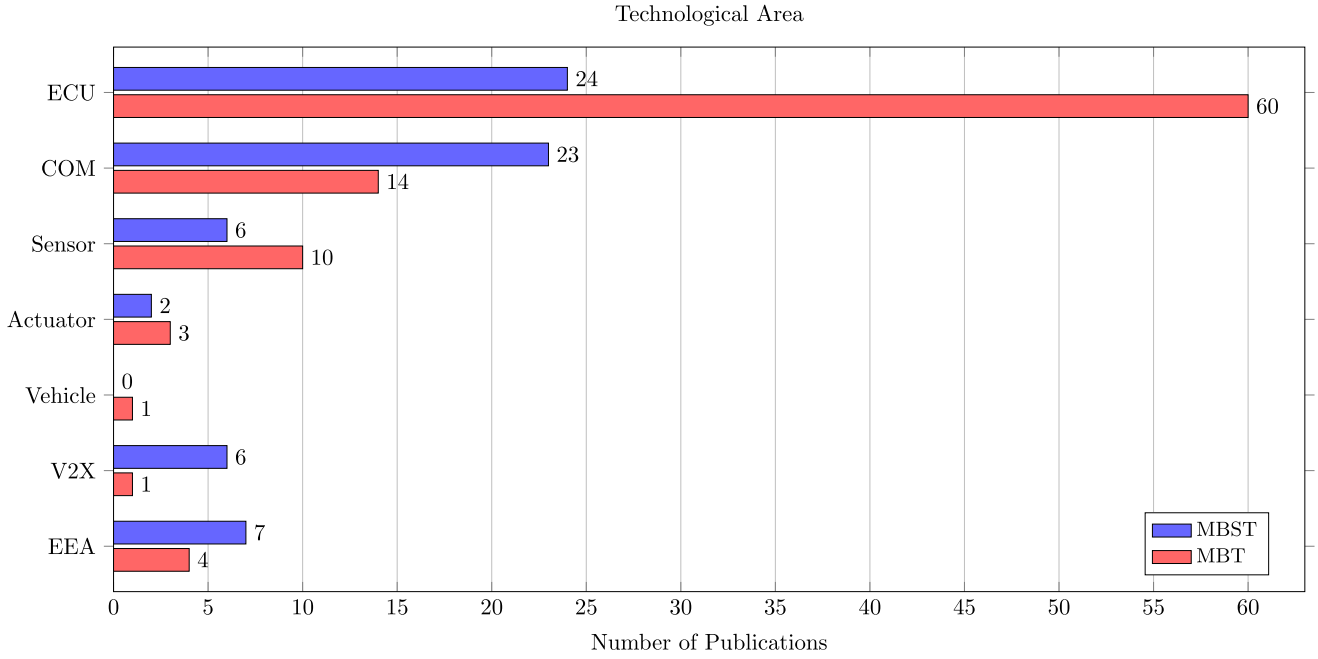

FIGURE 18. Distribution of the technological vehicle areas in relation to the frequency they are covered by identified publications.

79.3 %). Interestingly, communication systems and ECUs are almost exclusively addressed in either the early or late test stages. In 13 of 23 COM-related publications, DL or MiL is covered, 12 publications address HiL or PT. For ECUs the ratio is similar, as 15 out of 24 approaches cover DL or MiL and 11 approaches address HiL or PT. E/E architectures are addressed in 7 approaches (24.1 %), and sensors in 6 (20.7 %). In MBST, the testing of actuators and the entire vehicle also plays a subordinate role with only two approaches. However, V2X communication is addressed in 6 approaches (20.7 %). Overall, communication between vehicle systems is far more relevant in MBST than in MBT. In particular, the interaction between ECUs in a vehicle is relevant, as a manipulation of the communication systems can influence the behavior of an ECU. This is consistent with attacks on vehicles, where attack paths propagate through the vehicle's E/E architecture. Therefore, generating attack paths for testing should be investigated in future work. A more detailed look at individual technologies is provided in Fig. 19 and 20 and in the category *Application* in Fig. 21.

The distribution of individual approaches among the Camera, LiDAR, and RADAR sensors is uniform for MBT and MBST. In two MBT approaches, cameras are part of the SUT. One approach contains LiDAR sensors and three approaches contain RADAR sensors.

In MBST, four approaches include cameras and RADAR, whereas LiDAR sensors are part of the SUT in three approaches. With 11 approaches, these three types of sensors are represented almost twice as often in MBST as in MBT (relative to the number of respective approaches almost four times higher). A reason for this is the focus of vehicle manufacturers and suppliers on protecting sensors of (partially)

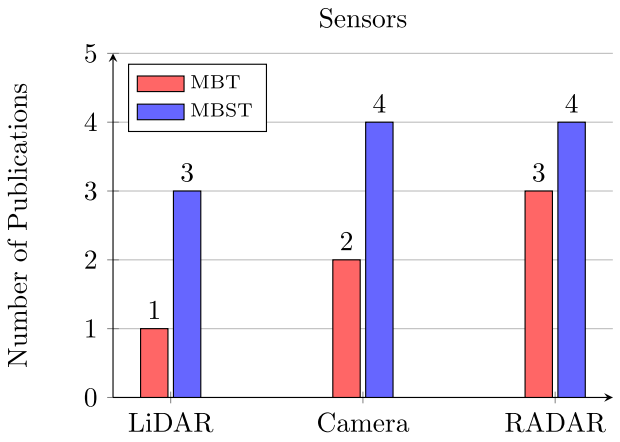

FIGURE 19. Distribution of the sensor systems in relation to the frequency they are addressed in identified publications.

autonomous vehicles against cyber attacks. Since camera, LiDAR, and RADAR sensors are used in such vehicles for environment recognition, protecting them is a high priority. This is particularly important because many sensor attacks do not require access to the vehicle. For example, for attacks on an ECU, an attacker must first gain access to the vehicle (e.g., via a wireless interface). When attacking sensors, it is sufficient to manipulate the environment sensed by the sensor. For example, Petit et al. [201] were able to blind camera and LiDAR sensors using lasers. Thus, extensive security testing and MBST is of sensors is highly relevant.

Communication systems (COM) are part of the SUT in 14 MBT publications. In contrast, in MBST they are represented 23 times. The difference between MBT and MBST can be

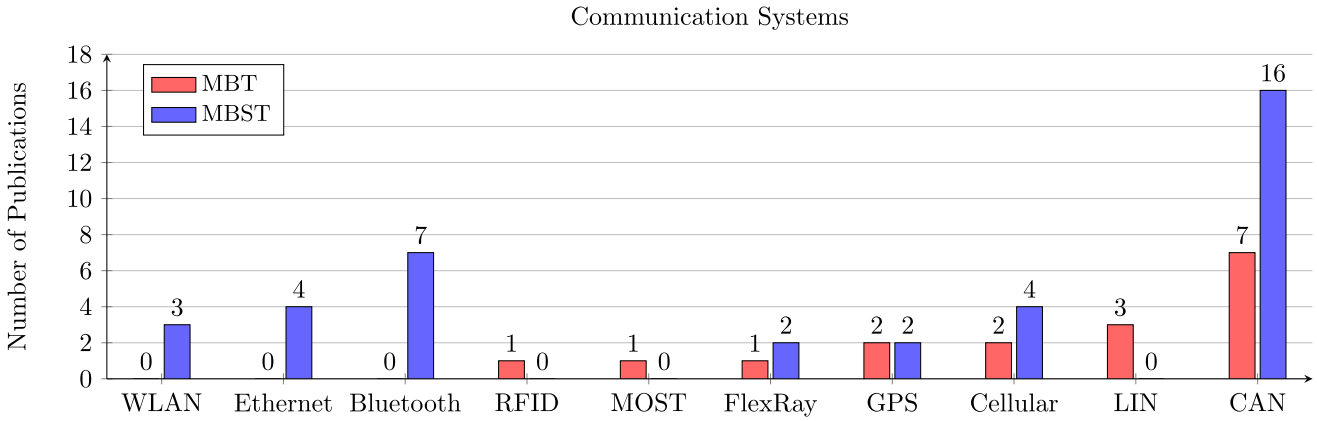

FIGURE 20. Distribution of the communication systems in relation to the frequency they are addressed in identified publications.

explained in two ways. As shown in Fig. 18, 60 out of 63 MBT approaches consider ECUs as SUTs. In these cases, the communication systems are mostly used to provide input data to an ECU (e.g., sending CAN messages to an ECU). Thus, the focus of these test approaches is primarily on testing the software application and not on communication system. Only Pretschner et al. [21] focus on testing a communication system (MOST). The second aspect concerns the cyber security of communication systems. A large number of cyber attacks on vehicles [9] have aimed at manipulating communication systems. CAN was particularly affected (e.g., in [202]) because it is most widely used in vehicles and lacks cybersecurity mechanisms (e.g., authentication). In 10 of 16 approaches in which CAN was a part of the SUT, MiL or the DL test stage is addressed. HiL or PT is covered in 7 publications. In addition, wireless communication interfaces, such as WLAN, Bluetooth, and cellular, are used to gain access to the vehicle in published attacks [9]. This is consistent with the observations in Fig. 20, as these communication systems are predominantly addressed in the published MBST approaches. Cheah et al. [152], [153], [154], [155], for example, test Bluetooth systems. CAN bus systems are addressed in 16 of 29 MBST approaches (55.2 %). Communication or signal transmission via Radio Frequency Identification (RFID) is not addressed in MBST publications. This is surprising, because RFID is commonly used in keyless access systems. In the attack database of Sommer et al. [185], attacks on keyless access systems are the most frequent target of attackers with a total of 117 attacks. In 65 of these attacks, a relay attack is conducted to open and steal vehicles. Therefore, future work could include testing keyless access systems.

F. ANALYSIS AND COMPARISON OF TEST SYSTEMS AND APPLICATIONS

This section presents the *Application* of test systems used within the identified publications as SUTs, case studies, or examples to evaluate the respective test methods. Fig. 21 lists the applications and their occurrences in the MBT and MBST publications. A variety of different vehicle functions are addressed. The vehicle as a whole is considered by MBT and MBST in two approaches each. The powertrain area is used by MBT in three approaches. In particular, the engine applications (10 approaches) show a high degree of diversity. For example, start-stop functions [91], [92], an exhaust measurement system (particle counter) [122], or a spark ignition engine [128] are used as SUT. Vehicle dynamics systems, such as ABS or brakes, are addressed in 3 MBT publications. Steering systems are used once each in MBT and MBST. A total of 18 of 63 approaches (28.6 %), cruise control and ACC systems are most frequently used as SUT in MBT. In particular, cruise control is a classic example in MBT [130], [141]. Other applications addressed by MBT are occasionally distributed over several system areas, such as ADAS, window control systems, lighting systems, and vehicle diagnostics.

MBST approaches focus on individual areas. Most frequently (7 approaches, 24.1 %), applications that implement security mechanisms (controls) are addressed. Six approaches (20.7 %) refer to infotainment or telematic applications. As these systems are typically responsible for communication with systems outside the vehicle (e.g., navigation, smartphone connection, or internet applications), they have been frequently chosen as entry points to the vehicle and attacked in the past [9]. Therefore, it makes sense to use these systems as SUT in MBST. The same applies to approaches that address communication systems or communicating units (gateways, OTA updates, and OBD devices). While these information and communication technology systems are applied in the MBST literature, driving physical systems, such as steering, driving, braking systems, and vehicle assistance systems, are comparatively little considered with a total of 7 approaches (24.1 %). Cyber attacks on these systems are particularly safety-critical, as the manipulation of vehicle physics can endanger vehicle occupants and road users. Such attacks have been demonstrated in the past by Miller and Valasek [72]. Although many MBT approaches

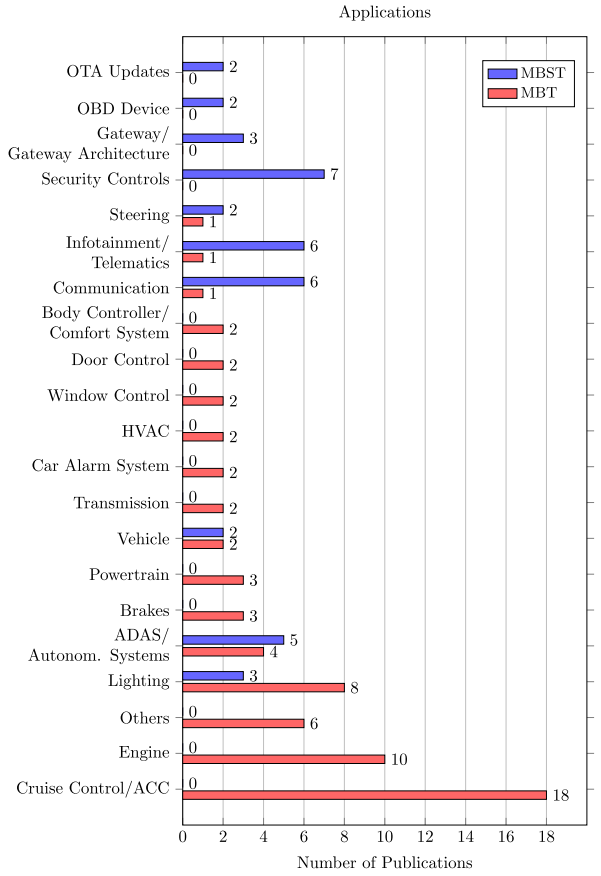

FIGURE 21. Distribution of the automotive applications in relation to the frequency they are addressed in identified publications.

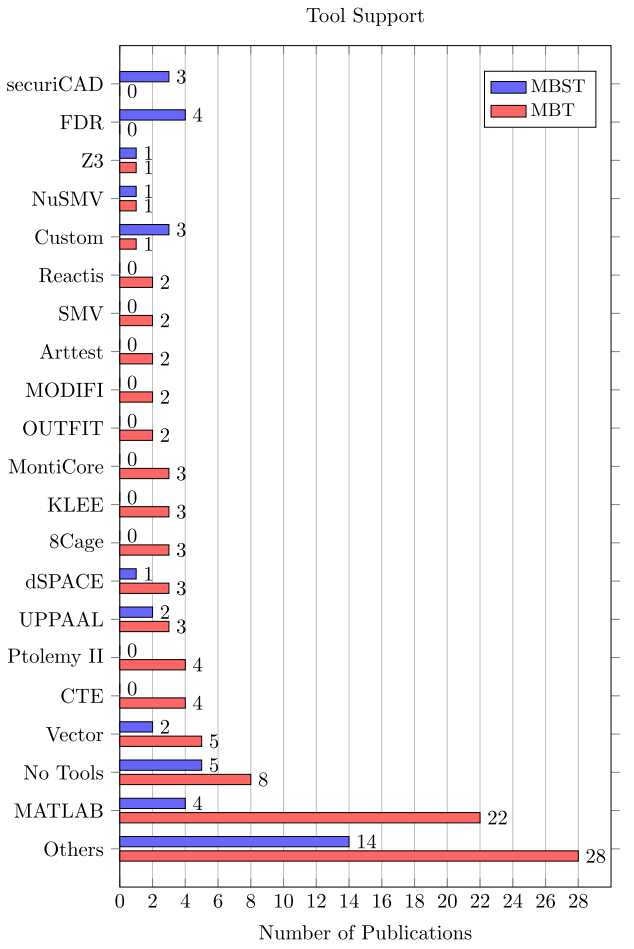

FIGURE 22. Distribution of the tools in relation to the frequency they are used in identified publications.

concerning ACC systems suggest that complex driving assistance systems can be covered by model-based testing, we suggest further research on the applicability for security-based methods.

G. ANALYSIS AND COMPARISON OF TOOL SUPPORT

In Fig. 22, an overview of the *Tool Support* in MBT and MBST is shown.

With respect to tool support for model-based test methods, high diversity exists. The only exception is MATLAB for the MBT area, which is used in 22 approaches (34.9 %), since MATLAB provides the Simulink framework, which is one of the most frequently used models (Fig. 16). The remaining MBT and MBST approaches are distributed among 89 tools. Fig. 22 explicitly shows tools that were used more than once (e.g., FDR). The category *Other* includes tools used only once in a publication. Eight MBT approaches (12.7 %) and 5 MBST approaches (17.2 %) did not receive any tool support. Apart from MATLAB/Simulink, no conclusions can be drawn regarding preferred tools.

H. ANALYSIS AND COMPARISON OF TEST AUTOMATION

The category *Automation* indicates whether an approach can be automated, semi-automated, or has no automation. Fig. 23 illustrates the distribution of the publications.

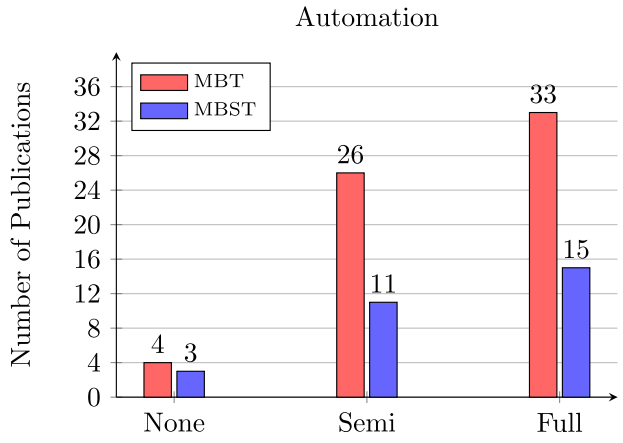

FIGURE 23. Distribution of the amount of approaches that include (partial) automation, or no automation at all.

For MBT, 33 of 63 approaches (52.4 %) are fully automated. In addition, 26 approaches (41.3 %) are semi-automated. Only 4 approaches do not use automation. Thus, 93.7 % of MBT approaches are at least partially automated. The distribution for MBST is almost the same. Of the 29 approaches, 15 (51.7 %) are fully automated,

and 11 (37.9 %) are semi-automated. Three approaches do not include any automation. Thus, 89.7 % of the MBST approaches are at least partially automated. Given that automation is a fundamental aspect of model-based testing, this high proportion is not surprising.

I. DISCUSSION

In the previous sections, 63 MBT and 29 MBST identified publications were analyzed and compared. In this section, we conclude with a discussion of the current state of MBT and MBST in the automotive sector. In addition, the challenges of current security testing activities in vehicle development outlined in Section I are addressed and whether model-based methods can provide a remedy is discussed.

The publication years in Fig. 3 (Section III-B) suggest that MBT approaches declined in the years after 2017. Recently published approaches, such as those of Bucher et al. [107], [108] or Neubauer et al. [109], primarily deal with topics, such as the simulative verification/validation of driving assistance systems. This is typically done using suitable simulation tools to simulate vehicles, their functionalities, and the surrounding traffic. This is already used in the development of vehicles, autonomous driving functions, and driving assistance systems, such as in [73], [203], and [204]. However, since such approaches have only been published in recent years, there are still open problems, such as the validity of the simulation results and their transferability to real driving situations. Thus, it can be expected that future model-based testing approaches will increasingly address these aspects.

In contrast to MBT, the number of published MBST approaches has increased since 2018. 21 of 29 MBST approaches have been published between 2018 and 2021. This accounts for 72.4 % of the total publications. Thus, it can be concluded that MBST is employed to address the current challenges of the automotive security testing process. This confirms the publications mentioned in Section I, such as Marksteiner et al. [13] or Khan et al. [16]. Based on these developments, it can be assumed that further model-based security test methods will be published in coming years.

In Sections VI-A - VI-H, data were compared between MBT and MBST for the individual categories (e.g., *Test Stage*). For MBST in particular, concrete areas were derived that can be addressed in future work. An example is the investigation of state-based models (Section VI-C) for the derivation of security test cases and the use of search-based approaches (Section VI-A), for example, for attack trees of a TARA.

In particular in MBT (Section VI-D), it can be observed that MiL testing is often addressed with further test stages (SiL, HiL, PiL, CT, ST, IT). This suggests that the models on which MiL testing is executed are reused for further test activities. Thus, MiL serves as a basis for further testing stages. For MBST, MiL has the potential to address security testing stages throughout the entire development cycle. However, only three MBST approaches address MiL. Thus, a primary field of research for future work could be to combine MiL testing with security testing, as this is already the case for the DL test stage (Section VI-D).

Additional observations can be made across individual categories. The testing of safety and security in combination has hardly been addressed. Only 4 of 29 MBST approaches [161], [173], [182], [189] cover this topic. This is surprising as these two aspects are often considered in parallel or in combination in TARA (e.g., [205], [206]). Therefore, future work could investigate a combination of model-based safety and security testing methods.

Furthermore, a comparison can be made between which vehicle systems are addressed by MBST and which systems are attacked and compromised in real-world attacks. As described in Section VI-F and VI-E, the MBST publications focus on testing ECUs and communication systems. In comparison, further areas, such as sensors, E/E architectures, and V2X, play a subordinate role. The current report of Upstream Security Ltd. [207] provides an overview of automotive-related attacks in the year 2022. According to this report, 35 % of all attacks were conducted on telematic units and application servers. In addition, 8 % of the attacks affected infotainment systems. In comparison, infotainment and telematic systems were addressed in only 6 of 29 (20.7 %) MBST publications. While MBST addresses ECUs in 24 of 29 (82.8 %) approaches, attacks on these components accounted for only 14 % in real-world attacks. Additionally, 18 % of all attacks concerned keyless entry systems, 12 % smart mobility, 4 % electric vehicle charging infrastructure, and 6 % mobile phone applications. These four areas are hardly addressed by the identified MBST publications. Only smart mobility approaches have been mentioned that deal with use cases, such as traffic management [183], OTA updates [169], or ADAS [162]. In particular, charging infrastructures and keyless access systems have not been addressed by any of the MBST approaches. In addition, according to Upstream, 97 % of all attacks in 2022 were carried out via remote access to the vehicle. This suggests that there is currently a discrepancy between the systems tested by MBST and those attacked in reality. Therefore, it would make sense to develop methods for early model-based security testing of attacked systems in future work. In conclusion, the identified topics for future research are as follows:

1) Simulative verification/validation of driving assistance systems (MBT).
2) Simulation-, threat/attack-based, and model checking approaches for component and integration tests (MBST).
3) Application of simulation- and search-based test methods (MBST).
4) Application of MBT approaches, such as CTM, EXAM, or SMArDT, for functional security testing (MBST).

5) Application of Simulink and state-based models (MBST).
6) Test case and attack path generation for security testing (MBST).
7) MiL and HiL testing (MBST).
8) Combination of model-based safety and security test methods (MBST).
9) Coherence between cyber attacks and the vulnerabilities they exploit (MBST).
10) Test approaches targeting automotive systems that are mainly attacked in reality, for example, keyless access systems (MBST).

As outlined in Section I, a main drawbacks of the current security testing process is the complexity of modern vehicles, late and manual testing techniques, such as penetration testing, and the challenge of identifying vulnerabilities as early as possible through testing. Existing surveys on (model-based) security testing, such as the work of Mahmood et al. [63], Luo et al. [64], and Pekaric et al. [65] often refer to penetration testing and dynamic analysis techniques (e.g., fuzzing and vulnerability scanning) that do not support early vulnerability testing and can only be applied late in the development cycle. Surveys, such as the work of Altinger et al. [15] and Kriebel et al. [17], suggest that model-based testing addresses these challenges. However, the presented approaches are independent from security testing. Our survey conducted in this study closes the gap between MBT and (model-based) security testing, since MBT approaches were analyzed and compared to MBST approaches. The results of this study suggest that security testing challenges can be addressed using model-based testing techniques. The early use of models in development allows a reuse for testing. The high number of MiL testing and DL publications (Fig. 17) demonstrates this. Since automation is one of the fundamental aspects of model-based testing, manual testing methods, such as penetration testing, can be supported by MBST. This is confirmed by the large number of different test tools (Fig. 22) and partially or fully automated approaches (Fig. 23). In addition, simulation-related MBT publications in particular have shown that complex vehicle systems, such as ADAS or autonomous driving functions, are covered by model-based testing and can be transferred to security testing. However, we were also able to identify topics that should be investigated in future work (mentioned above) to extend the current body of knowledge of automotive model-based security testing.

Finally, it should be noted that the validity of this survey is restricted by several limitations. Kitchenham et al. [67] cite several threats to validity of a systematic review process. One example is selection bias, which in our case describes the fact that the selection of publications may have been biased by various factors. For example, this includes selected literature libraries used for this survey and the filtering process according to which publications were selected (Section III-B). When conducting literature reviews, the risk occurs that specific types of papers or research results are published rather than others. For example, Kitchenham et al. [67] state that there is a tendency to publish positive research results rather than negative results. In order to cover this problem, we tried to consider as many publications as possible. Since we searched in the seven online libraries and databases mentioned in Section III-B, related publications of other sources could not be considered. In comparison to other surveys or reviews, such as [20], [43], and [68], we used additional sources to cover more papers.

Another limitation considers the formulation of the search strings and how this affects search results. In order to find as many relevant publications as possible, we used the search strings described in Section III-B. For each search term (e.g., *automotive*) we additionally used synonyms (e.g., *vehicle* or *vehicular*) to cover a wide range of publications. A limitation regarding the filtering process is the threat that relevant publications were sorted out, for example, while filtering publications according to their title.

Further limitations relate to the analysis and evaluation of the publications. The process of data extraction could be problematic regarding to unclear or misinterpreted data. We tried to avoid these problems by defining criteria which do not leave much room for interpretation (table categories in Sections IV and V). For example, the publications year as well as used modeling formalisms, evaluation and test methods, tool support, and application level are criteria which can be determined with a low possibility of misinterpretation, since they are rather distinct. However, some approaches addressed several categories. This led to overlaps between categories, since these papers made no clear distinction. However, we had to rely on the information given by the specific publication, e.g., the technique for test case or attack generation.

Overall, we covered a broad spectrum of approaches in this study with 92 publications (63 MBT and 29 MBST). Since we analyzed the approaches in detail, we consider risks of the described threats and limitations as moderate to low.

VII. CONCLUSION

In this work, a literature review on model-based testing and model-based security testing was conducted. We focused on the automotive domain, but examples from other domains, such as aerospace, medicine, and IT, were also mentioned. Model-based testing is common in most domains, particularly in the automotive industry, where we identified 63 relevant papers. Model-based security testing is used in many domains, such as IT. However, in comparison to MBT there are fewer approaches, particularly in the automotive sector. After the search and filtering process, 29 publications on model-based security testing in the automotive domain were identified and presented. Most of the publications were published between 2018 and 2021. Thus, MBST is comparatively new for automotive security. However, the results of this literature review suggest that MBST approaches are able to address challenges in current security testing practices.

Thus, MBST enables early testing in development and an automation of the security test process. We further identified potential research areas for future work, such as combined safety and security testing or testing according to current attack targets. Based on the results of this literature survey, we expect more model-based security testing approaches in the near future.

REFERENCES

[1] A. Winning. *Number of Automotive ECUS Continues to Rise*. Accessed: Jul. 7, 2020. [Online]. Available: https://www.eenewsautomotive.com/news/number-automotive-ecus-continues-rise
[2] *Road Vehicles—Controller Area Network (CAN)—Part 1: Data Link Layer and Physical Signalling*, Standard ISO 11898-1:2015, 1993.
[3] *Road Vehicles—Flexray Communications System—Part 1: General Information and Use Case Definition*, Standard ISO 17458-1: 2013, 2013.
[4] *Road Vehicles—In-Vehicle Ethernet—Part 1: General Information and Definitions*, Standard ISO 21111-1, 2020.
[5] *IEEE Approved Draft Standard for Information Technology—Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks—Specific Requirements: Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, Standard IEEE 802.11-2020, 2020.
[6] Bluetooth Special Interest Group. (2016). *Bluetooth Core Specification V5.0*. Accessed: Jul. 12, 2022. [Online]. Available: https://www.bluetooth.com/specifications/bluetooth-core-specification
[7] M. Maurer, J. Christian Gerdes, B. Lenz, and H. Winner, *Autonomous Driving: Technical, Legal and Social Aspects*. Cham, Switzerland: Springer, 2015.
[8] J. G. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, 2nd ed. Hoboken, NJ, USA: Wiley, 2004.
[9] F. Sommer, J. Dürrwang, and R. Kriesten, "Survey and classification of automotive security attacks," *Information*, vol. 10, no. 4, p. 148, Apr. 2019.
[10] *Cybersecurity Guidebook for Cyber-Physical Automotive Systems*, Standard SAE j3061, SAE Vehicle Electrical System Security Committee, 2016.
[11] *Road Vehicles—Cybersecurity Engineering*, ISO/SAE 21434:2021, 2021.
[12] H. Assal and S. Chiasson, "Security in the software development lifecycle," in *Proc. 14th Symp. Usable Privacy Secur.*, 2018, pp. 281–296.
[13] S. Marksteiner and Z. Ma, "Approaching the automation of cyber security testing of connected vehicles," in *Proc. 3rd Central Eur. Cybersecur. Conf.*, Nov. 2019, pp. 1–3.
[14] C. Jakobs, B. Naumann, M. Werner, and K. Schmidt, "Verification of integrity in vehicle architectures," in *Proc. 3rd Int. Conf. Netw., Inf. Syst. Secur.*, Mar. 2020, pp. 1–7.
[15] H. Altinger, F. Wotawa, and M. Schurius, "Testing methods used in the automotive industry: Results from a survey," in *Proc. Workshop Joining Academia Ind. Contrib. Test Autom. Model-Based Test.*, Jul. 2014, pp. 1–6.
[16] M. A. Khan, A. Jadoon, K. M. S. Haq, S. Mumtaz, and J. Rodrigues, "An overview of resilient and automatic model-based testing approaches for automotive industry," in *Proc. IEEE Int. Conf. Commun. Workshops*, May 2019, pp. 1–6.
[17] M. Markthaler, S. Kriebel, K. S. Salman, T. Greifenberg, S. Hillemacher, B. Rumpe, C. Schulze, A. Wortmann, P. Orth, and J. Richenhagen, "Improving model-based testing in automotive software engineering," in *Proc. IEEE/ACM 40th Int. Conf. Softw. Eng., Softw. Eng. Pract. Track*, May 2018, pp. 172–180.
[18] UNECE. (2021). *Un Regulation no. 155—Uniform Provisions Concerning the Approval of Vehicles With Regards to Cyber Security and Cyber Security Management System*. Accessed: Jul. 12, 2022. [Online]. Available: https://unece.org/sites/default/files/2021-03/R155e.pdf
[19] C. B. M. Tester, "ISTQB foundation level certified model-based tester: Syllabus," *Int. Softw. Test. Qualifications Board*, vol. 2015, pp. 1–41, Jan. 2015.
[20] C. A. D. Neto, R. Subramanyan, M. Vieira, and H. G. Travassos, "A survey on model-based testing approaches," in *Proc. 22nd IEEE ACM Int. Conf. Automated Softw. Eng.*, New York, NY, USA, Oct. 2007, pp. 31–36.
[21] A. Pretschner, W. Prenninger, S. Wagner, C. Kuhnel, B. Sostawa, R. ZoIch, and T. Stauner, "One evaluation of model-based testing and its automation," in *Proc. 27th Int. Conf. Softw. Eng.*, 2005, pp. 392–401.
[22] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Softw. Test., Verification Rel.*, vol. 22, no. 5, pp. 297–312, Aug. 2012.
[23] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz, "Model-based testing in practice," in *Proc. Int. Conf. Softw. Eng.*, May 1999, pp. 285–294.
[24] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual*. London, U.K.: Pearson, 2004.
[25] M. Shirole and R. Kumar, "UML behavioral model based test case generation: A survey," *ACM SIGSOFT Softw. Eng. Notes*, vol. 38, no. 4, pp. 1–13, Jul. 2013.
[26] M. Hause, "The SysML modelling language," in *Proc. 15th Eur. Syst. Eng. Conf.*, vol. 9, 2006, pp. 1–12.
[27] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines—A survey," *Proc. IEEE*, vol. 84, no. 8, pp. 1090–1123, Aug. 1996.
[28] A. S. Kalaji, R. M. Hierons, and S. Swift, "Generating feasible transition paths for testing from an extended finite state machine (EFSM)," in *Proc. Int. Conf. Softw. Test. Verification Validation*, Apr. 2009, pp. 230–239.
[29] J. J. Li and W. E. Wong, "Automatic test generation from communicating extended finite state machine (CEFSM)-based models," in *Proc. 5th IEEE Int. Symp. Object-Oriented Real-Time Distrib. Comput.*, 2002, pp. 181–185.
[30] D. Harel, "StateCharts: A visual formalism for complex systems," *Sci. Comput. Program.*, vol. 8, no. 3, pp. 231–274, Jun. 1987.
[31] The MathWorks. (2020). *SimuLink: Simulation und Model-Based Design*. Accessed: Jul. 22, 2020. [Online]. Available: https://de.mathworks.com/products/simulink.html
[32] The MathWorks. (2020). *StateFlow: Model and Simulate Decision Logic Using State Machines and Flow Charts*. Accessed: Jul. 22, 2020. [Online]. Available: https://nl.mathworks.com/products/stateflow.html
[33] Vector Informatik GmbH. *Testing ECUS and Networks With Canoe*. Accessed: Sep. 28, 2022. [Online]. Available: https://www.vector.com/de/de/produkte/produkte-a-z/software/canoe/
[34] The MathWorks. (2020). *MATLAB*. Accessed: Jul. 9, 2020. [Online]. Available: https://de.mathworks.com/products/MATLAB.html
[35] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lüttgen, A. Simons, S. Vilkomir, M. R. Woodward, and H. Zedan, "Using formal specifications to support testing," *ACM Comput. Surv.*, vol. 41, no. 2, pp. 1–76, 2009.
[36] M. Felderer and I. Schieferdecker, *A Taxonomy of Risk-Based Testing, Volume International Journal on Software Tools for Technology Transfer*. Berlin, Germany: Springer, 2014.
[37] J. Zander, I. Schieferdecker, and J. P. Mosterman, "A taxonomy of model-based testing for embedded systems from multiple industry domains," in *Model-Based Testing for Embedded Systems*. Boca Raton, FL, USA: CRC Press, 2011, pp. 3–22.
[38] ISTQB, "Certified automotive software tester: Certified tester foundation level specialist," *German Test. Board*, vol. 2, pp. 1–55, Mar. 2017.
[39] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*. Burlington, MA, USA: Morgan Kaufmann, 2014.
[40] M. Shafique and Y. Labiche. (2010). *A Systematic Review of Model Based Testing Tool Support*. Accessed: Jul. 13, 2020. [Online]. Available: https://ir.library.carleton.ca/pub/10271
[41] I. Schieferdecker, "Model-based testing," *IEEE Softw.*, vol. 29, no. 1, pp. 14–18, Jan. 2012.
[42] L. Wang, E. Wong, and D. Xu, "A threat model driven approach for security testing," in *Proc. 3rd Int. Workshop Softw. Eng. Secure Syst.*, May 2007, p. 10.
[43] M. Felderer, P. Zech, R. Breu, M. Büchler, and A. Pretschner, "Model-based security testing: A taxonomy and systematic classification," *Softw. Test., Verification Rel.*, vol. 26, no. 2, pp. 119–148, Mar. 2016.
[44] M. Felderer, B. Agreiter, P. Zech, and R. Breu, "A classification for model-based security testing," in *Proc. Adv. Syst. Testing Validation Lifecycle*, 2011, pp. 109–114.
[45] T. Su, K. Wu, W. Miao, G. Pu, J. He, Y. Chen, and Z. Su, "A survey on data-flow testing," *ACM Comput. Surv.*, vol. 50, no. 1, pp. 1–35, Jan. 2018.

[46] G. Fraser, F. Wotawa, and P. E. Ammann, "Testing with model checkers: A survey," *Softw. Test., Verification Rel.*, vol. 19, no. 3, pp. 215–261, Sep. 2009.

[47] A. Hartman, M. Katara, and S. Olvovsky, "Choosing a test modeling language: A survey," in *Proc. Haifa Verification Conf.*, 2006, pp. 204–218.

[48] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 649–678, Oct. 2011.

[49] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. Mcminn, A. Bertolino, J. J. Li, and H. Zhu, "An orchestrated survey of methodologies for automated software test case generation," *J. Syst. Softw.*, vol. 86, no. 8, pp. 1978–2001, Aug. 2013.

[50] F. Abbors, V.-M. Aho, J. Koivulainen, R. Teittinen, and D. Truscan, *Applying Model-Based Testing in the Telecommunication Domain*. Boca Raton, FL, USA: CRC Press, 2012.

[51] A. Andrews, M. Abdelgawad, and A. Gario, "World model for testing autonomous systems using Petri nets," in *Proc. IEEE 17th Int. Symp. High Assurance Syst. Eng. (HASE)*, Jan. 2016, pp. 65–69.

[52] B. Hasling, H. Goetz, and K. Beetz, "Model based testing of system requirements using UML use case models," in *Proc. Int. Conf. Softw. Test., Verification, Validation*, Apr. 2008, pp. 367–376.

[53] A. Gario, A. Andrews, and S. Hagerman, "Testing of safety-critical systems: An aerospace launch application," in *Proc. IEEE Aerosp. Conf.*, Mar. 2014, pp. 1–17.

[54] M. Felderer, "Security testing: A survey," in *Advances in Computers*. Amsterdam, The Netherlands: Elsevier, 2016, pp. 1–51.

[55] T. Mouelhi, F. Fleurey, B. Baudry, and Y. Traon, "A model-based framework for security policy specification, deployment and testing," in *Proc. Int. Conf. Model Driven Eng. Lang. Syst.*, 2008, pp. 537–552.

[56] E. Martin and T. Xie, "Automated test generation for access control policies via change-impact analysis," in *Proc. 3rd Int. Workshop Softw. Eng. Secure Syst.*, May 2007, p. 5.

[57] A. P. P. Salas, P. Krishnan, and J. K. Ross, "Model-based security vulnerability testing," in *Proc. Austral. Softw. Eng. Conf.*, 2007, pp. 284–296.

[58] A. Armando, R. Carbone, L. Compagna, K. Li, and G. Pellegrino, "Model-checking driven security testing of web-based applications," in *Proc. 3rd Int. Conf. Softw. Test., Verification, Validation Workshops*, Apr. 2010, pp. 361–370.

[59] A. Marback, H. Do, K. He, S. Kondamarri, and D. Xu, "A threat model-based approach to security testing," *Softw., Pract. Exper.*, vol. 43, no. 2, pp. 241–258, Feb. 2013.

[60] A. D. Brucker, L. Brügger, P. Kearney, and B. Wolff, "An approach to modular and testable security models of real-world health-care applications," in *Proc. 16th ACM Symp. Access Control Models Technol.*, Jun. 2011, pp. 133–142.

[61] S. Hagerman, A. Andrews, S. Elakeili, and A. Gario, "Security testing of an aerospace launch system," in *Proc. IEEE Aerosp. Conf.*, Mar. 2015, pp. 1–11.

[62] S. Hagerman, A. Andrews, and S. Oakes, "Security testing of an unmanned aerial vehicle (UAV)," in *Proc. Cybersecur. Symp.*, Apr. 2016, pp. 26–31.

[63] S. Mahmood, H. N. Nguyen, and A. S. Shaikh, "Automotive cybersecurity testing: Survey of testbeds and methods," in *Digital Transformation, Cyber Security and Resilience of Modern Societies*. Cham, Switzerland: Springer, 2021, pp. 219–243.

[64] F. Luo, X. Zhang, Z. Yang, Y. Jiang, J. Wang, M. Wu, and W. Feng, "Cybersecurity testing for automotive domain: A survey," *Sensors*, vol. 22, no. 23, p. 9211, Nov. 2022.

[65] I. Pekaric, C. Sauerwein, and M. Felderer, "Applying security testing techniques to automotive engineering," in *Proc. 14th Int. Conf. Availability, Rel. Secur.*, Aug. 2019, p. 61.

[66] AUTOSAR. *AutoSAR: Automotive Open System Architecture*. Accessed: Dec. 16, 2020. [Online]. Available: https://www.autosar.org/

[67] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele Univ., Keele, U.K., Tech. Rep. EBSE-2007-01, 2007.

[68] E. Lisova, I. Šljivo, and A. Causevic, "Safety and security co-analyses: A systematic literature review," *IEEE Syst. J.*, vol. 13, no. 3, pp. 2189–2200, Sep. 2019.

[69] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *J. Syst. Softw.*, vol. 80, no. 4, pp. 571–583, Apr. 2007.

[70] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *Proc. IEEE Symp. Secur. Privacy*, 2010, pp. 447–462.

[71] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Proc. USENIX Secur. Symp.*, 2011, pp. 1–13.

[72] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," Tech. Rep., 2015.

[73] F. Sommer, M. Gierl, and P. Rebling. (2023). *Vehicle Network Platforms for Automotive Security Testing*. Accessed: Apr. 28, 2023. [Online]. Available: https://zenodo.org/record/7573669

[74] H. Shokry and M. Hinchey, "Model-based verification of embedded software," *Computer*, vol. 42, no. 4, pp. 53–59, 2009.

[75] C. Gühmann, "Model-based testing of automotive electronic control units," in *Proc. 3rd Int. Conf. Mater. Test., Test*, 2005, pp. 1–6.

[76] M. Grochtmann and K. Grimm, "Classification trees for partition testing," *Softw. Test., Verification Rel.*, vol. 3, no. 2, pp. 63–82, Jun. 1993.

[77] M. Grochtmann, K. Grimm, and J. Wegener, "Tool-supported test case design for black-box testing by means of the classification-tree editor," in *Proc. EuroSTAR*, vol. 93, 1993, pp. 169–176.

[78] H. Singh, M. Conrad, and S. Sadeghipour, "Test case design based on Z and the classification-tree method," in *Proc. 1st IEEE Int. Conf. Formal Eng. Methods*, Apr. 1997, pp. 81–90.

[79] K. Lamberg, M. Beine, M. Eschmann, R. Otterbach, M. Conrad, and A. I. Fey, "Model-based testing of embedded automotive software using MTest," *SAE Trans.*, vol. 10, pp. 132–140, Jan. 2004.

[80] M. Conrad. (2004). *A Systematic Approach to Testing Automotive Control Software*. Accessed: Jul. 13, 2020. [Online]. Available: https://www.win.tue.nl/~mvdbrand/courses/sse/0809/papers/con04_a_systematic_approach.pdf

[81] M. Conrad, I. Fey, and S. Sadeghipour, "Systematic model-based testing of embedded automotive software," *Electron. Notes Theor. Comput. Sci.*, vol. 111, pp. 13–26, Jan. 2005.

[82] J. Zander-Nowicka, I. Schieferdecker, and A. Marrero Perez, "Automotive validation functions for on-line test evaluation of hybrid real-time systems," in *Proc. IEEE Autotestcon*, Sep. 2006, pp. 799–805.

[83] A. Mjeda, P. McElligott, K. Ryan, and S. Thiel, "Model-based testing design for embedded automotive software," in *Proc. SAE World Congr.*, 2009, pp. 1–12.

[84] *Information Technology—Z Formal Specification Notation—Syntax, Type System and Semantics*, Standard ISO/IEC 13568:2002, 2002.

[85] E. Bringmann and A. Kr, "Model-based testing of automotive systems," in *Proc. Int. Conf. Softw. Test., Verification, Validation*, Apr. 2008, pp. 485–493.

[86] F. Lindlar, A. Windisch, and J. Wegener, "Integrating model-based testing with evolutionary functional testing," in *Proc. 3rd Int. Conf. Softw. Test., Verification, Validation Workshops*, Apr. 2010, pp. 163–172.

[87] A. Michailidis, U. Spieth, T. Ringler, B. Hedenetz, and S. Kowalewski, "Test front loading in early stages of automotive software development based on AUTOSAR," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2010, pp. 435–440.

[88] I. Schieferdecker, E. Bringmann, and J. Großmann, "Continuous TTCN-3: Testing of embedded control systems," in *Proc. Int. workshop Softw. Eng. Automot. Syst.*, May 2006, pp. 29–36.

[89] E. Lehmann, "Time partition testing: Systematischer test des kontinuierlichen Verhaltens von eingebetteten systemen," Ph.D. thesis, Fac. IV Elect. Eng. Comput. Sci., Berlin Techn. Univ., Berlin, Germany, 2004.

[90] *Methods for Testing and Specification (MTS); The Testing and Test Control Notation Version 3; Part 1: TTCN-3 Core Language*, Standard ETSI ES 201 873-1, 2017.

[91] S. Siegl, W. Dulz, R. German, and G. Kiffe, "Model-driven testing based on Markov chain usage models in the automotive domain," in *Proc. 12th Eur. Workshop Dependable Comput.*, 2009, pp. 1–6.

[92] S. Siegl, K.-S. Hielscher, and R. German, "Model based requirements analysis and testing of automotive systems with timed usage models," in *Proc. 18th IEEE Int. Requirements Eng. Conf.*, Sep. 2010, pp. 345–350.

[93] S. Thiel and D. Zitterell, "Extended automation method (EXAM) zur automatisierten funktionserprobung von Steuergeräten in der automobilindustrie," in *INFORMATIK 2008. Beherrschbare Systeme—Dank Informatik. Band 2*, H.-G. Hegering, A. Lehmann, H. J. Ohlbach, and C. Scheideler, Eds. Bonn, Germany: Gesellschaft für Informatik e.V., 2008, pp. 625–630.

[94] D. Zitterell and S. Thiel, "Automatisierter funktionaler steuergerätetest mit der extended automation method (EXAM)," in *INFORMATIK 2010. Service Science—Neue Perspektiven für die Informatik. Band 2*, K.-P. Fähnrich and B. Franczyk, Eds. Bonn, Germany: Gesellschaft für Informatik e.V., 2010, pp. 351–356.
[95] P. S. Meyn and L. R. Tweedie, *Markov Chains and Stochastic Stability*. Cham, Switzerland: Springer, 2012.
[96] *Road Vehicles—Functional Safety: Part 1: Vocabulary*, ISO 26262-1:2018, Dec. 2018.
[97] G. Hahn, J. Philipps, A. Pretschner, and T. Stauner, "Prototype-based tests for hybrid reactive systems," in *Proc. 14th IEEE Int. Workshop Rapid Syst. Prototyping*, 2003, pp. 78–84.
[98] S. Nabi, M. Balike, J. Allen, and K. Rzemien, "An overview of hardware-in-the-loop testing systems at Visteon: SAE technical paper series," in *Proc. SAE World Congr.*, Mar. 2004, pp. 1–6.
[99] R. Cleaveland, A. S. Smolka, and T. S. Sims, "An instrumentation-based approach to controller model validation," in *Proc. Automotive Softw. Workshop*, 2008, pp. 84–97.
[100] P. Peranandam, S. Raviram, M. Satpathy, A. Yeolekar, A. Gadkari, and S. Ramesh, "An integrated test generation tool for enhanced coverage of simulink/stateflow models," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2012, pp. 308–311.
[101] L. Belmon and Y. Xu, "Intelligent test-case generation for automated validation of TCUS," in *Proc. Int. CTI Symp. Innov. Automotive Transmiss.*, 2012, pp. 1–12.
[102] R. Matinnejad, S. Nejati, L. Briand, T. Bruckmann, and C. Poull, "Automated model-in-the-loop testing of continuous controllers using search," in *Proc. Int. Symp. Search Based Softw. Eng.*, 2013, pp. 141–157.
[103] P. Skruch and G. Buchala, "Model-based real-time testing of embedded automotive systems," *SAE Int. J. Passenger Cars-Electron. Electr. Syst.*, vol. 7, no. 2, pp. 337–344, Apr. 2014.
[104] N. Wiechowski, T. Rambow, R. Busch, A. Kugler, N. Hansen, and S. Kowalewski, "Arttest—A new test environment for model-based software development," SAE Tech. Paper 2017-01-0004, 2017.
[105] N. Hansen, N. Wiechowski, A. Kugler, S. Kowalewski, T. Rambow, and R. Busch, "Model-in-the-loop and software-in-the-loop testing of closed-loop automotive software with arttest," in *Informatik*, M. Eibl and M. Gaedke, Eds. Bonn, Germany: Gesellschaft für Informatik, 2017, pp. 1537–1549, doi: 10.18420/in2017_154.
[106] H. Bucher, C. Reichmann, and J. Becker, "An integrated approach enabling cross-domain simulation of model-based E/E-architectures," SAE Tech. Paper 2017-01, Jun. 2017.
[107] H. Bucher, S. Kamm, and J. Becker, "Cross-layer behavioral modeling and simulation of E/E-architectures using preevision and Ptolemy II," *SNE Simul. Notes Eur.*, vol. 29, no. 2, pp. 73–78, Jun. 2019.
[108] H. Bucher, K. Neubauer, and J. Becker, "Automated assessment of E/E-architecture variants using an integrated model-and simulation-based approach," SAE Tech. Paper 2019-01-0111, 2019.
[109] K. Neubauer, H. Bucher, B. Haas, and J. Becker, "Model-based development and simulative verification of logical vehicle functions using executable un/ece regulations," in *Proc. Summer Simulation Conf.*, 2020, pp. 1–12.
[110] C. Berger, *Automating Acceptance Tests for Sensor- and Actuator based Systems on the Example of Autonomous Vehicles*. Maastricht, The Netherlands: Shaker, 2010.
[111] R. A. Plummer, "Model-in-the-loop testing," *Proc. Inst. Mech. Eng., I, J. Syst. Control Eng.*, vol. 220, no. 3, pp. 183–199, 2006.
[112] UNECE. (2013). *Un Regulation no. 131—Uniform Provisions Concerning the Approval of Motor Vehicles With Regard to the Advanced Emergency Braking Systems (AEBS)*. Accessed: Apr. 7, 2023. [Online]. Available: https://unece.org/fileadmin/DAM/trans/main/wp29/wp29regs/2013/R131e.pdf
[113] C. Berger and B. Rumpe, "Hesperia: Framework zur szenario-gestützten modellierung und entwicklung sensor-basierter systeme," in *Informatik 2009—Im Focus das Leben*, S. Fischer, E. Maehle, and R. Reischuk, Eds. Bonn, Germany: Gesellschaft für Informatik e.V., 2009, pp. 328–328.
[114] E. Schoitsch, E. Althammer, H. Eriksson, J. Vinter, L. Gönczy, A. Pataricza, and G. Csertan, "Validation and certification of safety-critical embedded systems—The decos test bench," in *Proc. Int. Conf. Comput. Saf., Rel., Secur.*, 2006, pp. 372–385.
[115] K.-W. Shin and D.-J. Lim, "Model-based automatic test case generation for automotive embedded software testing," *Int. J. Automot. Technol.*, vol. 19, no. 1, pp. 107–119, Feb. 2018.
[116] J. Peleska, A. Honisch, F. Lapschies, H. Löding, H. Schmid, P. Smuda, E. Vorobev, and C. Zahlten, "A real-world benchmark model for testing concurrent real-time systems in the automotive domain," in *Proc. IFIP Int. Conf. Test. Softw. Syst.*, 2011, pp. 146–161.
[117] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, "Improving fault injection in automotive model based development using fault bypass modeling," in *INFORMATIK 2013—Informatik Angepasst an Mensch, Organisation und Umwelt*, M. Horbach, Ed. Bonn, Germany: Gesellschaft für Informatik e.V., 2013, pp. 2577–2591.
[118] R. Svenningsson, J. Vinter, H. Eriksson, and M. Törngren, "MODIFI: A model-implemented fault injection tool," in *Proc. Int. Conf. Comput. Saf., Rel., Secur.*, 2010, pp. 210–222.
[119] R. Svenningsson, H. Eriksson, J. Vinter, and M. Törngren, "Model-implemented fault injection for hardware fault simulation," in *Proc. Workshop Model-Driven Eng., Verification, Validation*, Oct. 2010, pp. 31–36.
[120] W. Herzner, R. Schlick, H. Brandl, and J. Wiessalla, "Towards fault-based generation of test cases for dependable embedded software," in *Softwaretechnik-Trends Band 31, Heft 3*. Bonn, Germany: Geselllschaft für Informatik e.V., 2011.
[121] R. Schlick, W. Herzner, and E. Jöbstl, "Fault-based generation of test cases from UML-models—Approach and some experiences," in *Proc. Int. Conf. Comput. Saf., Rel., Secur.*, 2011, pp. 270–283.
[122] K. B. Aichernig, J. Auer, E. Jöbstl, R. Korošec, W. Krenn, R. Schlick, and B. V. Schmidt, "Model-based mutation testing of an industrial measurement device," in *Proc. Int. Conf. Tests Proofs*, 2014 pp. 1–19.
[123] F. Belli, C. J. Budnik, A. Hollmann, T. Tuglular, and W. E. Wong, "Model-based mutation testing—Approach and case studies," *Sci. Comput. Program.*, vol. 120, pp. 25–48, May 2016.
[124] D. Holling, A. Pretschner, and M. Gemmar, "8Cage: Lightweight fault-based test generation for Simulink," in *Proc. 29th ACM/IEEE Int. Conf. Automated Softw. Eng.*, Sep. 2014, pp. 859–862.
[125] D. Holling, A. Hofbauer, A. Pretschner, and M. Gemmar, "Profiting from unit tests for integration testing," in *Proc. IEEE Int. Conf. Softw. Test., Verification Validation (ICST)*, Apr. 2016, pp. 353–363.
[126] D. Holling, "Defect-based quality assurance with defect models," Ph.D. thesis, School Comput., Inf. Technol., Technische Universität München, Munich, Germany, 2016.
[127] A. Morozov, K. Ding, T. Chen, and K. Janschek, "Test suite prioritization for efficient regression testing of model-based automotive software," in *Proc. Int. Conf. Softw. Anal., Test. Evol. (SATE)*, Nov. 2017, pp. 20–29.
[128] M. Ungermann, J. Lunze, and D. Scharzmann, "Model-based test signal generation for service diagnosis of automotive systems," *IFAC Proc. Volumes*, vol. 43, no. 7, pp. 117–122, Jul. 2010.
[129] E. Zurich and C. P. D. Kroening. (2010). *Mogentes—Model-Based Generation of Test-Cases for Embedded Systems: Modelling Languages: Final Version*. Accessed: May 21, 2019 [Online]. Available: http://www.mogentes.eu/public/deliverables/MOGENTES_3-13_1.0r_D3.2b_ModellingLanguages.pdf
[130] A. J. Offutt, Y. Xiong, and S. Liu, "Criteria for generating specification-based tests," in *Proc. 5th IEEE Int. Conf. Eng. Complex Comput. Syst.*, Oct. 1999, pp. 119–129.
[131] J. Offutt, S. Liu, A. Abdurazik, and P. Ammann, "Generating test data from state-based specifications," *Softw. Test., Verification Rel.*, vol. 13, no. 1, pp. 25–53, 2003.
[132] I. Drave, S. Hillemacher, T. Greifenberg, S. Kriebel, E. Kusmenko, M. Markthaler, P. Orth, K. S. Salman, J. Richenhagen, B. Rumpe, C. Schulze, M. von Wenckstern, and A. Wortmann, "SMArDT modeling for automotive software testing," *Softw., Pract. Exper.*, vol. 49, no. 2, pp. 301–328, Feb. 2019.
[133] A. Baldini, A. Benso, and P. Prinetto, "System-level functional testing from UML specifications in end-of-production industrial environments," *Int. J. Softw. Tools Technol. Transf.*, vol. 7, no. 4, pp. 326–340, Aug. 2005.
[134] L. Zhang, J. He, and W. Yu, "Test case generation from formal models of cyber physical system," *Int. J. Hybrid Inf. Technol.*, vol. 6, no. 3, pp. 15–24, 2013.
[135] J. Sobotka and J. Novák, "Testing automotive reactive systems using timed automata," in *Proc. 9th IEEE Int. Conf. Intell. Data Acquisition Adv. Comput. Syst., Technol. Appl. (IDAACS)*, vol. 1, Sep. 2017, pp. 510–513.
[136] L. Krejcí and J. Novák, "Model-based testing of automotive distributed systems with automated prioritization," in *Proc. 9th IEEE Int. Conf. Intell. Data Acquisition Adv. Comput. Syst., Technol. Appl. (IDAACS)*, vol. 2, Sep. 2017, pp. 668–673.

[137] H. Khosrowjerdi, K. Meinke, and A. Rasmusson, "Learning-based testing for safety critical automotive applications," in *Proc. Int. Symp. Model-Based Saf. Assessment*, 2017, pp. 197–211.
[138] M. Broy, F. Huber, and B. Schätz, "AutoF ocus–Ein werkzeugprototyp zur entwicklung eingebetteter systeme," *Informatik-Forschung und Entwicklung*, vol. 14, no. 3, pp. 121–134, Sep. 1999.
[139] R. Marinescu, M. Saadatmand, A. Bucaioni, C. Seceleanu, and P. Pettersson, "A model-based testing framework for automotive embedded systems," in *Proc. 40th EUROMICRO Conf. Softw. Eng. Adv. Appl.*, Aug. 2014, pp. 38–47.
[140] P. Ammann, W. Ding, and D. Xu, "Using a model checker to test safety properties," in *Proc. 7th IEEE Int. Conf. Eng. Complex Comput. Syst.*, May 2001, pp. 212–221.
[141] A. Gargantini and C. Heitmeyer, "Using model checking to generate tests from requirements specifications," *ACM SIGSOFT Softw. Eng. Notes*, vol. 24, no. 6, pp. 146–162, 1999.
[142] S. V. Alagar and K. Periyasamy, *Specification of Software Systems*. Cham, Switzerland: Springer, 2011.
[143] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press, 2008.
[144] R. M. Hierons, S. Sadeghipour, and H. Singh, "Testing a system specified using statecharts and Z," *Inf. Softw. Technol.*, vol. 43, no. 2, pp. 137–149, Feb. 2001.
[145] V. Chimisliu, C. Schwarzl, and B. Peischl, "Test case generation for embedded automotive systems: A semantics preserving model transformation," *Model-Based Test. Pract.*, vol. 10, p. 43, Jan. 2009.
[146] L. C. Briand, Y. Labiche, and Y. Wang, "Using simulation to empirically investigate test coverage criteria based on statechart," in *Proc. 26th Int. Conf. Softw. Eng.*, 2004, pp. 86–95.
[147] A. Saifan and W. Mustafa, "Using formal methods for test case generation according to transition-based coverage criteria," *Jordanian J. Comput. Inf. Technol.*, vol. 1, no. 1, p. 15, 2015.
[148] M. Lochau, S. Lity, R. Lachmann, I. Schaefer, and U. Goltz, "Delta-oriented model-based integration testing of large-scale systems," *J. Syst. Softw.*, vol. 91, pp. 63–84, May 2014.
[149] A. Petrenko, O. N. Timo, and S. Ramesh, "Model-based testing of automotive software: Some challenges and solutions," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2015, pp. 1–6.
[150] *Information Processing Systems—Open Systems Interconnection—Lotos—A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, Standard ISO 8807:1989, Dec. 1989.
[151] B. Schneier, "Attack trees," *Dr. Dobb's J.*, vol. 24, no. 12, pp. 21–29, 1999.
[152] M. Cheah, A. S. Shaikh, J. Bryans, and H. N. Nguyen, "Combining third party components securely in automotive systems," in *Proc. IFIP Int. Conf. Inf. Secur. Theory Pract.*, 2016, pp. 262–269.
[153] M. Cheah, H. N. Nguyen, J. Bryans, and A. S. Shaikh, "Formalising systematic security evaluations using attack trees for automotive applications," in *Proc. IFIP Int. Conf. Inf. Secur. Theory Pract.*, 2017, pp. 113–129.
[154] M. Cheah, S. A. Shaikh, O. Haas, and A. Ruddle, "Towards a systematic security evaluation of the automotive Bluetooth interface," *Veh. Commun.*, vol. 9, pp. 8–18, Jul. 2017.
[155] M. Cheah, S. A. Shaikh, J. Bryans, and P. Wooderson, "Building an automotive security assurance case using systematic security evaluations," *Comput. Secur.*, vol. 77, pp. 360–379, Aug. 2018.
[156] S. Mahmood, A. Fouillade, H. N. Nguyen, and A. S. Shaikh, "A model-based security testing approach for automotive over-the-air updates," in *Proc. IEEE Int. Conf. Softw. Test., Verification Validation Workshops (ICSTW)*, 2020, pp. 6–13.
[157] E. dos Santos, A. Simpson, and D. Schoop, "A formal model to facilitate security testing in modern automotive systems," 2018, *arXiv:1805.05520*.
[158] W. Xiong, F. Krantz, and R. Lagerström, "Threat modeling and attack simulations of connected vehicles: A research outlook," in *Proc. 5th Int. Conf. Inf. Syst. Secur. Privacy*, 2019, pp. 272–287.
[159] J. Ekelund, "Security evaluation of damper system's communication and update process: Threat modeling using vehicleLang and securiCAD," School of Electrical Engineering and Computer Science, Stockholm, Sweden, Tech. Rep. TRITA-EECS-EX; 2021:720, 2021.
[160] W. D. Schoot, "Validating vehicleLang, a domain-specific threat modelling language, from an attacker and industry perspective," Ph.D. thesis, KTH Roy. Inst. Technol., School Elect. Eng. Comput. Sci., Stockholm, Sweden, 2020.
[161] D. Suo and S. E. Sarma, "A test-driven approach for security designs of automated vehicles," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2019, pp. 26–32.
[162] A. M. Shaaban, C. Schmittner, T. Gruber, A. B. Mohamed, G. Quirchmayr, and E. Schikuta, "Ontology-based model for automotive security verification and validation," in *Proc. 21st Int. Conf. Inf. Integr. Web-Based Appl. Services*, Dec. 2019, pp. 73–82.
[163] PTES. (2014). *The Penetration Testing Execution Standard*. Accessed: Jan. 2, 2023. [Online]. Available: http://www.pentest-standard.org/index.php/Main_Page
[164] Foreseeti. (2021). *Securicad Enterprise*. Accessed: Mar. 17, 2023. [Online]. Available: https://foreseeti.com/securicad-enterprise/
[165] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," IOActive, Tech. Rep., 2014.
[166] S. Katsikeas, P. Johnson, S. Hacks, and R. Lagerström, "Probabilistic modeling and simulation of vehicular cyber attacks: An application of the meta attack language," in *Proc. 5th Int. Conf. Inf. Syst. Secur. Privacy*, 2019, pp. 1–13.
[167] P. Johnson, R. Lagerström, and M. Ekstedt, "A meta language for threat modeling and attack simulations," in *Proc. 13th Int. Conf. Availability, Rel. Secur.*, Aug. 2018, pp. 1–8.
[168] *Information Security, Cybersecurity and Privacy Protection—Evaluation Criteria for it Security—Part 1: Introduction and General Model*, Standard ISO/IEC 15408-1:2022, 2022. [Online]. Available: https://www.iso.org/standard/72891.html
[169] J. Heneghan, S. A. Shaikh, J. Bryans, M. Cheah, and P. Wooderson, "Enabling security checking of automotive ECUs with formal CSP models," in *Proc. 49th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2019, pp. 90–97.
[170] K. D. Wittenberg, J. Smith, R. Gray, and G. Eakman. (2016). *Automotive Vulnerability Detection System*. Accessed: Jul. 13, 2020. [Online]. Available: https://www.cs.brandeis.edu/ dkw/papers/ESCARVDS4.pdf
[171] R. Kurachi and T. Fujikura, "Proposal of HILS-based in-vehicle network security verification environment," SAE Tech. Paper 2018-01-0013, 2018, doi: 10.4271/2018-01-0013.
[172] P. S. Oruganti, M. Appel, and Q. Ahmed, "Hardware-in-loop based automotive embedded systems cybersecurity evaluation testbed," in *Proc. ACM Workshop Automot. Cybersecur.*, Mar. 2019, pp. 41–44.
[173] M. Appel, P. S. Oruganti, Q. Ahmed, J. Wilkerson, and A. Sekar, "A safety and security testbed for assured autonomy in vehicles," *SAE Int.*, vol. 10, p. 8, Apr. 2020.
[174] T. Huang, J. Zhou, and A. Bytes, "ATG: An attack traffic generation tool for security testing of in-vehicle CAN bus," in *Proc. 13th Int. Conf. Availability, Rel. Secur.*, Aug. 2018, p. 32.
[175] J. Smith and M. Figueroa, "Reduced realistic attack plan surface for identification of prioritized attack goals," in *Proc. IEEE Int. Conf. Technol. Homeland Secur. (HST)*, Nov. 2013, pp. 716–721.
[176] National Vulnerability Database. (2019). *CVE-2019-14951*. Accessed: Jul. 30, 2022. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-14951
[177] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," 2017, *arXiv:1711.03938*.
[178] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, vol. 3, 2009, p. 5.
[179] T. Volkersdorfer and H.-J. Hof, "A concept of an attack model for a model-based security testing framework: Introducing a holistic perspective of cyberattacks in software engineering," in *Proc. 14th Int. Conf. Emerg. Secur. Inf., Syst. Technol.*, 2020, pp. 96–101.
[180] T. Volkersdorfer and H.-J. Hof, "Adam: An adversary-driven attack modelling framework for model-based security testing," *Int. J. Adv. Secur.*, vol. 14, no. 2, pp. 12–25, 2021.
[181] F. Sommer, R. Kriesten, and F. Kargl, "Model-based security testing of vehicle networks," in *Proc. Int. Conf. Comput. Sci. Comput. Intell. (CSCI)*, Dec. 2021, pp. 685–691.
[182] P. Mundhenk, S. Steinhorst, M. Lukasiewycz, S. A. Fahmy, and S. Chakraborty, "Security analysis of automotive architectures using probabilistic model checking," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2015, pp. 1–6.
[183] M. Asplund, "Combining detection and verification for secure vehicular cooperation groups," *ACM Trans. Cyber-Phys. Syst.*, vol. 4, no. 1, pp. 1–31, Jan. 2020.
[184] T. Volkersdorfer, "Methodik zur angriffsmodellierung für security-tests," Ph.D. thesis, Fac. Comput. Sci., Technische Hochschule Ingolstadt, Ingolstadt, Germany, 2020.

[185] F. Sommer and J. Dürrwang. (2019). *IEEM-HSKA/AAD: Automotive Attack Database (AAD)*. Accessed: Jul. 12, 2022. [Online]. Available: https://github.com/IEEM-HsKA/AAD
[186] J. Dürrwang, F. Sommer, and R. Kriesten, "Automation in automotive security by using attacker privileges," in *Proc. 19th ESCAR*, 2021, pp. 137–152.
[187] CVSS Special Interest Group. (2019). *Common Vulnerability Scoring System V3.0: Specification Document*. Accessed: Apr. 28, 2023. [Online]. Available: https://www.first.org/cvss/specification-document
[188] P. Mouttappa, S. Maag, and A. Cavalli, "Monitoring based on IOSTS for testing functional and security properties: Application to an automotive case study," in *Proc. IEEE 37th Annu. Comput. Softw. Appl. Conf.*, Jul. 2013, pp. 1–10.
[189] L. Huang and E.-Y. Kang, "Formal verification of safety & security related timing constraints for a cooperative automotive system," in *Proc. Int. Conf. Fundam. Approaches Softw. Eng.*, 2019, pp. 210–227.
[190] G. Pedroza, L. Apvrille, and R. Pacalet, "A formal security model for verification of automotive embedded applications," in *Proc. SAFA Annu. Workshop Formal Techn. (SAFA)*, Sophia-Antipolis, France, Oct. 2010.
[191] M. Kastebo and V. Nordh, "Model-based security testing in automotive industry," M.S. thesis, CHALMERS Univ. Technol., Univ. Gothenburg, Gothenburg, Sweden, 2017.
[192] G. Lee, H. Oguma, A. Yoshioka, R. Shigetomi, A. Otsuka, and H. Imai, "Formally verifiable features in embedded vehicular security systems," in *Proc. IEEE Veh. Netw. Conf. (VNC)*, Oct. 2009, pp. 1–7.
[193] C. Jakobs, M. Werner, K. Schmidt, and G. Hansch, "Following the white rabbit: Integrity verification based on risk analysis results," in *Proc. Comput. Sci. Cars Symp.*, Nov. 2021, pp. 1–9.
[194] Y. Zhang, Y. Liu, L. Zhang, Z. Ma, and H. Mei, "Modeling and checking for non-functional attributes in extended UML class diagram," in *Proc. 32nd Annu. IEEE Int. Comput. Softw. Appl. Conf.*, Oct. 2008, pp. 100–107.
[195] E.-Y. Kang, D. Mu, and L. Huang, "Probabilistic verification of timing constraints in automotive systems using UPPAAL-SMC," in *Proc. Int. Conf. Integr. Formal Methods*, 2018, pp. 236–254.
[196] H. Oguma, A. Yoshioka, M. Nishikawa, R. Shigetomi, A. Otsuka, and H. Imai, "New attestation based security architecture for in-vehicle communication," in *Proc. IEEE Global Telecommun. Conf.*, Apr. 2008, pp. 1–6.
[197] J. Eichler and D. Angermeier, "Modular risk assessment for the development of secure automotive systems," in *Proc. 31st VDI/VW joint Conf. Automot. Secur.*, Wolfsburg, Germany, 2015, pp. 21–22.
[198] J. Dürrwang, "Steigerung der betriebssicherheit von personenkraftwagen durch bedrohungsanalysen für die informationssicherheit," Ph.D. dissertation, School Comput., Inf. Technol., Technische Universität München, München, Germany, 2022.
[199] *Road Vehicles—Unified Diagnostic Services (UDS)—Specification and Requirements*, Standard ISO 14229:2006, 2006.
[200] J. Hartmann, C. Imoberdorf, and M. Meisinger, "UML-based integration testing," in *Proc. ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Aug. 2000, pp. 60–70.
[201] J. Petit, B. Stottelaar, M. Feiri, and F. Kargl, "Remote attacks on automated vehicles sensors: Experiments on camera and LiDAR," in *Proc. Black Hat Eur.*, vol. 11, 2015, p. 2015.
[202] C. Miller and C. Valasek, "Adventures in automotive networks and control units," *Def Con*, vol. 21, pp. 260–264, Aug. 2013.
[203] D. R. Niranjan and B. C. VinayKarthik, "Deep learning based object detection model for autonomous driving research using CARLA simulator," in *Proc. 2nd Int. Conf. Smart Electron. Commun. (ICOSEC)*, Oct. 2021, pp. 1251–1258.
[204] C. Gómez-Huélamo, J. D. Egido, L. M. Bergasa, R. Barea, F. Arango, J. Araluce, and J. López, "Train here, drive there: Simulating real-world use cases with fully-autonomous driving architecture in Carla simulator," in *Proc. 21st Int. Workshop Phys. Agents*, 2021, pp. 44–59.
[205] J. Dürrwang, K. Beckers, and R. Kriesten, "A lightweight threat analysis approach intertwining safety and security for the automotive domain," in *Proc. 36th Int. Conf. Comput. Saf., Rel., Secur.*, Trento, Italy, Sep. 2017, pp. 305–319.
[206] G. Macher, H. Sporer, R. Berlach, E. Armengaud, and C. Kreiner, "SAHARA: A security-aware hazard and risk analysis method," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2015, pp. 621–624.
[207] Upstream Security Ltd. (2023). *Global Automotive Cybersecurity Report*. Accessed: Apr. 28, 2023. [Online]. Available: https://upstream.auto/reports/global-automotive-cybersecurity-report/

FLORIAN SOMMER received the B.Eng. and M.Sc. degrees in automotive systems engineering and vehicular technology. He is currently pursuing the Ph.D. degree with the Institute of Distributed Systems, Ulm University. He has been an Academic Researcher with the Institute of Energy Efficient Mobility, Karlsruhe University of Applied Sciences, since 2018.

REINER KRIESTEN received the Dr.-Ing. degree. Since 2003, he has been with Robert Bosch GmbH, where his applied research is based on a strong connection to the automotive industry, such as due to SW/system engineering and project management activities for automotive gateways and body computers. He is currently the Head and a Speaker with the Institute of Energy Efficient Mobility, Karlsruhe University of Applied Sciences. His research interests include software (SW) and systems engineering of cyber-physical and embedded systems and research in automotive security.

FRANK KARGL (Member, IEEE) is currently a Professor of distributed systems with Ulm University, Germany. Earlier, he held a tenured position with the University of Twente, The Netherlands. He was a PI in several research projects on European and national levels. His main research interest includes the security and privacy of automotive systems. Recently, his focus extended toward the question of how a combination of communication and automated driving in vehicles creates new challenges for their security.