Using Obstacle Analysis to Support SysML-Based Model Testing for Cyber Physical Systems

Mounifah Alenazi*, Nan Niu*, Wentao Wang*, and Juha Savolainen†
* Department of Electrical Engineering and Computer Science, University of Cincinnati, USA
† Global Software and Control R&D, Danfoss Drives A/S, Denmark
Email: alenazmh@mail.uc.edu, nan.niu@uc.edu, wang2wt@mail.uc.edu, juha.savolainen@danfoss.com

***Abstract*—Cyber-physical systems play a crucial role in various applications, ranging from critical infrastructure control like power grid to the technological revolution of Industry 4.0 aimed to integrate and automate the manufacturing value chain. The Systems Modeling Language (SysML) represents a significant and increasing segment of industrial support for the development of cyber-physical systems partly due to the language's built-in mechanisms for modeling the requirements. In this paper, we leverage goal-oriented obstacle analysis to systematically identify the impediments to the fulfillment of requirements, and further examine several machine learning algorithms' capabilities of classifying these impediments into the components that constitute the cyber physical systems. We then investigate the extent to which a state-of-the-practice SysML tool simulates these obstacles, thereby assessing the risks of system failures at the requirements level. Our work offers concrete insights into model testing which incorporates the deep intertwining of software with hardware in order to improve the robustness of the cyber-physical systems.**

***Index Terms*—obstacle analysis, KAOS, cyber-physical systems, system modeling language (SysML), machine learning**

I. Introduction

A cyber-physical system (CPS) tightly integrates computing and communication technologies in order to monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa [1]. The applications of CPS have not only spanned domains like medical devices and autonomous vehicles, but also become the hallmark of Industry 4.0 which revolutionizes manufacturing via transdisciplinary approaches rooted in theories of cybernetics, mechatronics, and design science.

For CPS developers, the Systems Modeling Language (SysML) [2] gains increasing use in their industrial applications as SysML is becoming the *de facto* modeling standard for systems engineering [3]. SysML allows for precisely specifying hardware devices, software control components, and their interfaces [4]. State-of-the-practice tools like Magic Draw [5] facilitate the systems engineers to define SysML diagrams, track design progress, and communicate with clients and other CPS stakeholders about requirements satisfactions.

Currently, the fulfillment of requirements can be supported by traceability, e.g., SysML design slices could link fragments of the activity diagram to a safety concern of the requirements diagram, thereby assisting in safety inspections [6], [7] and change impact analysis [8]. While traceability exploits mainly the textual content of the model elements [9], [10], [11], the Protos framework [12] aims to assure the requirements of a sociotechnical system like CPS are formally specified through autonomous parties' conditional interactions via commitments. Such interaction-oriented requirements specifications are motivated by promoting openness and accountability [13], which helps to realize the vision of Industry 4.0 by connecting autonomous parties (e.g., machines, software controllers, and businesses) along the entire value chain.

An intrinsic challenge to specifying and tracing CPS requirements is rooted in the deep intertwining of software with hardware, making it insufficient to interrelate only the modeled elements and to make assumptions without exposing necessary details of the sociotechnical interactions (networks, devices, software, people, etc.). To overcome this challenge, Briand *et al.* [14] proposed a practical solution called *model testing*. The idea is to raise the level of abstraction of testing from a unit of operational software, where factors such as hardware constraints are commonly neglected, to models such as the ones built using SysML where systems-level behaviors and properties can be automatically checked. Recent work by Ali and his colleagues [15], [16], [17] made major strides in CPS model testing by understanding the multifaceted natures of uncertainty, performing variability-aware multiobjective test optimizations, learning new uncertainties during evolution, and generating test cases from the evolved models.

In this paper, we build on the latest developments and significantly extend the literature by exploring the role of KAOS-based [18] obstacle analysis in CPS model testing. Obstacles are a dual notion to goals: while goals capture desired conditions, obstacles capture undesirable (but nevertheless possible) ones [19]. In KAOS-based requirements elaboration, not only are goals operationalized into specifications [18], but the obstacles can also be systematically generated from goal formulations [19]. An obstacle obstructs some goal and is a precondition for non-satisfaction of stakeholders' requirements. Thus, we are interested in leveraging the obstacles to guide model testing so as to improve the CPS robustness at the requirements level.

Our work makes two main contributions: (1) investigating machine learning's capabilities of automatically classifying the obstacles generated from KAOS into cyber, physical, cyber-physical, and uncertainty, and (2) deriving test cases based on the classified obstacles to automatically execute model testing within a state-of-the-practice SysML tool, namely,

978-1-5386-8406-1/18/$31.00 ©2018 IEEE
DOI 10.1109/MoDRE.2018.00012

Magic Draw (formerly known as Cameo System Modeler). Our results on the basis of three sets of CPS requirements show that Naive Bayes is among the most stable obstacle classifiers, the classified obstacles can be model tested with Magic Draw's built-in support, and surprisingly, physical obstacles are addressed more readily than those obstructing cyber conditions. Our study offers operational insights into goal-oriented obstacle analysis as an innovative solution to CPS model testing, and meanwhile provides a mechanism for making obstacle analysis more robust.

The rest of this paper is organized as follows: Section II provides background information and reviews related work. Section III presents the subject systems, along with the sets of CPS requirements, of our study. Section IV discusses the identification and classification of the obstacles. Section V builds upon the obstacle classifications to perform model testing. Section VI and Section VII discuss threats to validity and practical applications. Finally, Section VIII draws some concluding remarks and outlines future work.

II. Background and Related Work

A. Cyber-Physical Systems and SysML

A CPS is defined by Ali and Yue [17] as: A set of heterogeneous physical units (e.g., sensors, control modules) communicating via heterogeneous networks (using networking equipment) and potentially interacting with applications deployed on cloud infrastructures and/or humans to achieve a common goal. Applications of CPS have the potential to both dwarf the 20th century IT revolution [20] and to trigger a new industrial revolution termed Industry 4.0. An example is the smart factory like a car or yogurt manufacturing system characterized by its flexibility, resource efficiency, ergonomic design, and the ability to integrate customer and business partner into the value creation process [21].

The CPS challenges can stem from both physical and cyber environments. From the mechatronics perspective, a CPS is stochastic in nature due to factors such as actuator inaccuracies, sensor readings, the rate of arrivals, and component failure rates. In a smart yogurt factory, for example, the temperature of the freezing cylinder may be sensed stochastically. From the cybernetics perspective, a CPS has to cope with the fast increase of software scale and complexity while striving for high levels of computational intelligence. With wide variations in customizing yogurt (lactic acid, sweetness, mix-ins, etc.), the product-line software could compute an optimal decision ordering to minimize the overall manual configuration steps [16]; however, the optimization objectives may change as stakeholders requirements evolve [17], [22].

The requirements of CPS must be engineered at a systems level by explicitly considering the hardware-software interdependencies. One of such modeling frameworks that have gained much industrial adoption is SysML, which represents a significant and increasing segment of the embedded software industry, particularly in safety-critical domains [8]. SysML extensively reuses UML, while also providing certain extensions to it. Compared to UML, a couple of SysML features offer advantages for systems engineers [23]: (1) the use of blocks to unify structural concepts of a CPS, thereby reducing the bias that UML classes have towards software in encapsulating data abstractions, and (2) the built-in requirements diagram allowing for natural language requirements to be modeled and traced to design elements.

The requirements tracing capabilities equipped with state of-the-practice SysML tools like Magic Draw and enhanced by state-of-the-art methods like model slicing [24] facilitate activities such as verification and validation, safety certification, and change impact analysis. For example, to verify the design meets such safety requirements as: *"The feed belt conveys a blank to table if the table is in load position"* [6], the slices (fragments) from a SysML internal block diagram can help the inspector to efficiently locate the relevant model elements without having to analyze all the elements. However, automated requirements traceability [8] relies mainly on the textual descriptions of requirements and (SysML) models. To complement traceability's static nature, model testing was proposed by Briand and his colleagues [14] as a dynamic mechanism for reasoning about requirements satisfactions.

B. Model Testing

Testing is a widely used approach toward software quality because the degrees of automatically executing test cases help tackle the scalability issues, such as size and complexity, in practice. For the software under test (SUT), a particular testing strategy essentially guides two practices: *what* and *how many* test cases to generate with the latter being bounded by resources like the amount of available or allocated time to perform testing. For example, usage-based statistical testing manages the test cases proportional to the probability distribution obtained from the operational profiles of the SUT's environment [25], whereas boundary testing focuses on developing test cases around the corner cases of the input domain (e.g., maximum, minimum, just inside/outside boundaries, typical values, and error values) when the SUT takes numeric input(s) [26].

Such traditional testing methods rely on after–the–fact testing for verification. However, due to the stochastic and intrinsic nature of a CPS and the lack of a precise understanding of the expected system behavior (i.e., uncertainty) these methods are infeasible to apply. Model-based-testing (MBT), a systematic and automated test case generation technique, has been used for many years in industry [27]. MBT has also shown promise for ensuring CPS during testing [28]. However, due to the scalability issue of CPS and the large test suites generated by MBT, MBT techniques are not practical [28].

To overcome this challenge, recent studies were proposed. Uncertainty-wise testing modeling framework called *UncerTum* [28] is a new testing paradigm that was proposed to explicitly address *known uncertainty* about the behavior of CPS. *UncerTum* relies on Test Ready Models (TRM) with uncertainty. TRM are models that represent the behavior of the CPS in such details that test cases can be generated from them [28]. In other words, TRM take models as inputs and generate test cases either automatically or semiautomatically.

The aim of *UncerTum* is to improve the quality of these TRM by using the real operational data of CPS, which can be used to generate additional test cases as compared to the initial ones. The main idea of this framework is that uncertainty must be explicitly captured in TRM. Briand *et al* [14] propose an approach called *model testing* for untestable systems. As the authors claim, since models represent system behavior, environment, properties, and structure, they can be used as a basis for test execution. The basic idea of this approach is to run as much as test scenarios while executing the models in an automated fashion, taking in consideration uncertainty and level of details for detecting faults [14].

One of the main objectives of model testing [14] is to execute a much larger number of test scenarios by means of model executions. Thus, in this paper, we investigate the role of KAOS-obstacle analysis into CPS model testing where model-driven engineering practices such as SysML models are adopted.

C. KOAS-Based Obstacle Analysis

Requirements violation caused from the interactions between cyber and physical components can wreak havoc in critical systems [29]. CPS interacts with the physical world and, given the complexity of problems being solved by CPS in critical domains, these systems must function safely even when experiencing uncertainty in their physical environment [15]. However, due to the intertwining of software with hardware, predicting the exact behavior of the physical environment of a CPS is not possible. An example of safety violations is the Alice incident [29]. Alice, an autonomous Ford Econoline van, dangerously deviated from the generated path of the computer and started stuttering in the middle of a busy intersection [29]. The reason was the unexpected interaction of the path planner and the steering systems which was involved during making a sharp left turn while merging into traffic. Alice deviated from the path and thus the reactive obstacle avoidance system slowed it down, the path planner generated a new path with a more sharper turn to be able to merge into the traffic. As a result, Alice could not follow, deviated, and caused a collision. This unexpected interaction among the path planner and the physical environment was not witnessed during more than 300 miles of autonomous test-driving and hours of extensive simulations [29]. According to Lee and Seshia [30] , as an intellectual challenge, CPS is about the intersection, not the union, of the physical and the cyber. Therefore, dealing with such obstacles is crucial for CPS.

KOAS-based obstacle analysis has been successfully used in a variety of safety-critical systems [31]. It is a goal-oriented activity, and begins with exploring the goal model and with negating each goal in turn [31]. A goal *G* in the goal refinement model is refined into *G1, ..., Gn iff G1,...., Gn imply G*. If *G* is violated, then it is because at least one of the subgoals is violated. Goals are refined using AND/OR refinements until they are assignable to agents. The type of goals assigned to agents is either a requirement or an expectation (i.e., leaf goals in refinement graph). Each negated goal is refined to possible obstacles that obstruct that goal from being fulfilled. Obstacles are situations in which a goal, a requirement or an expectation is violated [31]. To identify obstacles to goal *G*: negate *G*; find as many obstacles as possible that obstruct *G* in view of domain properties. After identifying obstacles, conditions for these obstacles are looked for. Various techniques are used to generate these obstacles ranging from formal calculus of preconditions for obstruction to the use of heuristics as an informal alternative to formal techniques [19].

In this paper, we use two types of techniques for obstacles identification: obstruction refinement patterns and informal obstacle identification. In obstruction refinement patterns, obstacles are shown in a form of a tree. It shows through AND/OR refinements how the goal may be violated. The root of the tree is the goal negation; the leaves are elementary obstruction conditions that are satisfiable by the environment [32]; an AND-refinement captures a combination of sub-obstacles entailing the parent obstacle; an OR refinement captures alternative ways of entailing the parent obstacle. Since OR-refinements are in general desirable for critical goals [19], we apply it in our study. Informal obstacle identification takes the form “if the specification has such or such characteristics then consider such or such type of obstacle to it”.

Integrating obstacle analysis in CPS model testing not only gives systems engineers the ability to more easily identify scenarios in which goals are not satisfied, but also to reason incrementally about new alternatives for handling obstacles that can occur during runtime. This is useful in that obstacles indicate which scenarios would ensure coverage of exception cases. All these obstacles must be investigated at requirements engineering time if the system is to remain robust.

III. Subject Systems

In this section we present the subject systems that we examined in this paper: AntiLock-Braking System (ABS), Transmission Control Module (TCM) [33], and the barbados Car Crash Management System (bCMS) [34].

An example of CPS is a car equipped with an ABS [35]. The ABS may interact directly or indirectly with other systems, such as adaptive-cruise control and lane keeping control [36]. ABS is designed to prevent a vehicle’s wheels from locking up and skidding during heavy braking. ABS can also reduce the distance required to come to a stop, thereby reducing the risk of skidding and allowing the driver to keep control of the vehicle and improving safety. The safety requirement we have considered for this system is: *The vehicle shall be steerable and stable during heavy braking by preventing wheel lock.* An obstacle to this requirement, for simplicity, is {*loss of control*}, i.e., the negation of the associated leaf expectation *Achieve[Full control of vehicle]* in the goal model.

TCM, another example of CPS [37], consists of not only mechanical parts but also electronics. It controls gearbox and switch between gears based on input from several sensors as well as data provided by engine control module (ECM).

TABLE I
SUBJECT SYSTEMS STATISTICS

Case Study	# Req.s	# Obstacles/tree (std)	# Leaf obstacles /req. (std)	Avg. height of obstacle tree (std)	Avg. branch factor (std)
ABS	1	24	13.00	5.00	1.00
bCMS	9	5 (3.36)	3.33 (2.40)	2.78 (0.67)	0.75 (0.17)
TCM	1	25	18.00	3.00	0.96

It then processes this input to calculate how and when to shift gears in the transmission and generates the signals that drive actuators to complete this shifting. Electronic sensors monitor the selection of gear position, vehicle speed, throttle position, and many other attributes. This information helps the control module to adjust the current supplied to solenoids in the transmission that control the position of various valves and gears. The safety requirement we have considered for this system is: *Revolution Per Minute (RPM) value shall not exceed 3900*. An obstacle to this goal is {*RPM value exceed 3900*}.

bCMS system is a distributed crash management system that is responsible for coordinating the communication between different systems to handle crisis [34]. It has several requirements and expectations including hazard requirements, threat requirements, inaccuracy requirements, and other types of requirements as identified by van Lamsweerde and Letier [19]. The high-level goal we consider for this system is that *bCMS shall handle a crisis in a timely manner*. An obstacle to this goal is *crisis is not resolved in a timely manner*. Based on goal refinement in the goal model of this system, we can refine this goal to more specific requirements.

We create our own obstacle tree for ABS (i.e., 24 obstacles) and TCM (i.e., 25 obstacles). For bCMS, we include all obstacles (i.e., 54 obstacles) generated by Cailliau *et al* [32]. The underlying rationale for the choice of these subject systems is due to the following reasons:

- Each subject system has different types of requirements and expectations. As a result, we can investigate various types of obstacles (e.g., safety requirements, security requirements).
- The complexities are different for each subject system in terms of obstacles. For example, we calculate the number of obstacles per requirement or obstacle tree. The results show that obstacle trees in ABS and TCM contain more nodes than trees in bCMS. In addition, although the height of obstacle trees is very different in the three subject systems; the highest one (i.e., 5 in ABS) is almost twice the lowest one (i.e., 2.78 in bCMS). The percentages of leaf obstacles in each obstacle tree are similar in three datasets (i.e., 54.17%, 66.60%, and 72% in ABS, bCMS, and RPM respectively). Table I shows the statistics for the subject systems used in this paper.

In each of these subject systems, the goal is too ideal; it presupposes that nothing in the environment will prevent the goal from being achieved. Obstacles for preventing this goal from being fulfilled could be related to software, hardware, or interaction of both. Examples of such obstacles are: wheel speed sensors malfunctions (physical), ABS master controller failure (cyber), or any other obstacles (system behavior under different road conditions). Identifying obstacle under which goals may be violated is essential for not missing requirements that would prescribe what the system should do to properly handle such unexpected situations [38].

As mentioned earlier, we used a combination of two techniques for obstacles generation: OR refinement pattern and informal obstacle identification. Figure 1 shows the resulting obstacle tree for the ABS case study. Objectiver[1], a tool designed to support KAOS, was used for creating the trees in our study. The root of the tree is the leaf goal *Achieve[Full control of vehicle]*. Its root obstacle is obtained by negating the goal (i.e., loss of control) using an obstruction link. The root obstacle is then refined to sub-obstacles, and the leaves are elementary obstruction conditions that are satisfiable by the environment (e.g., dump valve fails).

Obstacles at the leaves should correspond to cyber, physical, but not cyber-physical components. However, in practice, especially in CPS, it is not clear whether the obstacles are due to cyber or physical components or both. Thus, we incorporate machine learning algorithms to automatically classify the obstacles generated from KAOS into four components (i.e., cyber, physical, cyber-physical, and uncertain). Figure 2 shows an overview of our approach. Obstacles are identified and then classified using machine learning algorithms. The obstacles then are imported to Magic Draw tool in which test cases are derived from the obstacles and executed using the Cameo Simulation Toolkit. Section V demonstrates our approach. Next, we present our experiments of classifying these obstacles.

IV. CLASSIFYING OBSTACLES

In machine learning, classification is the problem of classifying a new observation into a given set of categories based on a training data set (past observations whose category memberships are known). Many problems could be formulated as the classification problem, which has wide applications, including object recognition, behavior analysis, speech recognition, face recognition and verification, etc [39]. For example, object recognition and behavior analysis with camera sensors data will help smart transportation systems have a better understanding of their surrounding environments, which is necessary for autonomous vehicles to support automated planning and control to achieve desired destinations. The uncertainty in physical environment makes future CPS

[1]http://www.objectiver.com/index.php?id=4

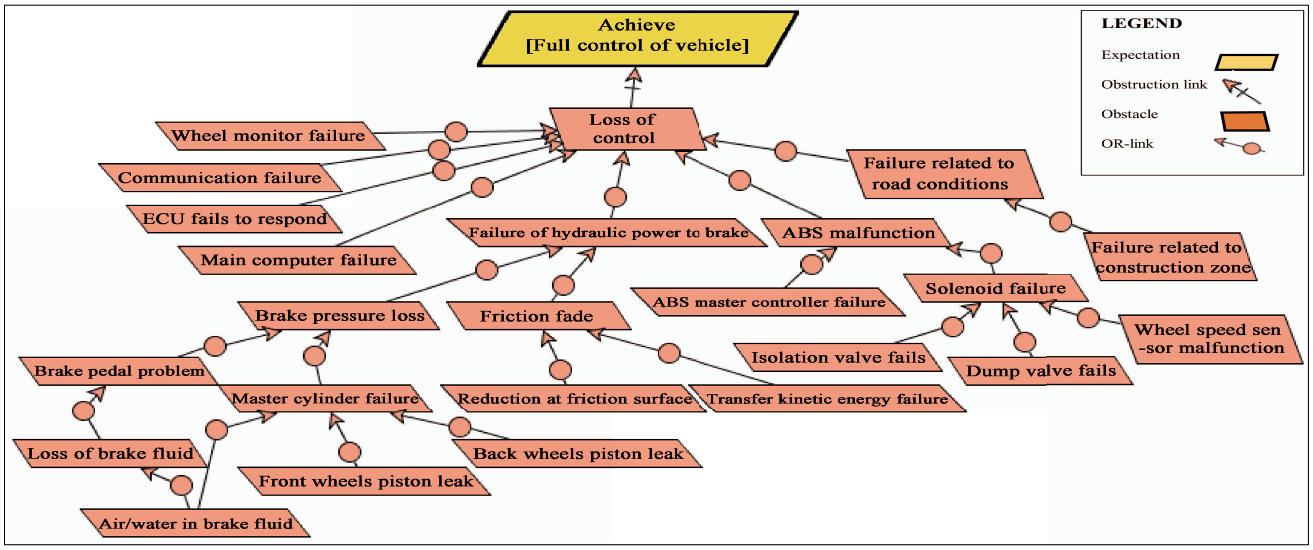
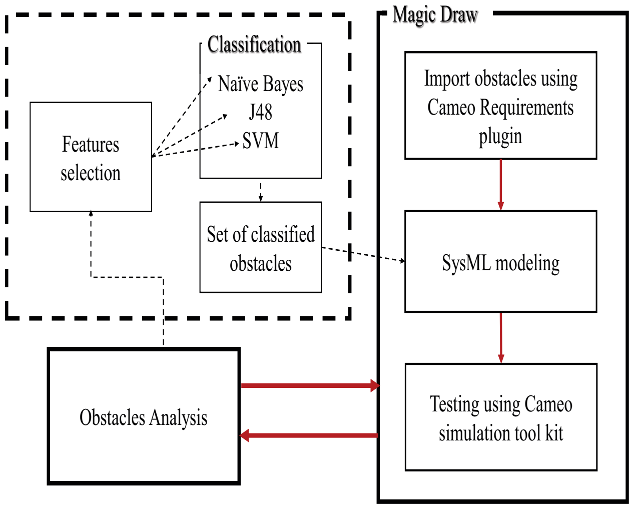


Fig. 1. Obstacles refinement for *Achieve[Full control of vehicle]*

TABLE II
SAMPLES OF MANUAL LABELING OF OBSTACLES INTO CYBER (C), PHYISCAL (P), CYBER-PHYSICAL (CP), AND UNCERTAIN (U).

Obstacles	Leaf obstacle	Non-satisfaction	Inaccuracy	Hazard	Non-information	Threat	Soft-Goal	Class-Label
ABS malfunction	0	0.14	1	1	0	0	0	CP
Wheel Speed sensor malfunction	1	0.02	1	1	0	0	0	P
ABS master controller failure	1	0.07	1	1	0	0	0	C
Failure related to road condition	0	0.14	1	0	0	0	0	U



Classification
Naïve Bayes
J48
SVM
Features selection
Set of classified obstacles
Magic Draw
Import obstacles using Cameo Requirements plugin
SysML modeling
Testing using Cameo simulation tool kit
Obstacles Analysis

Fig. 2. Overview of our approach: Classification of obstacles (dotted arrows) are shown in which classified obstacles are used in SysML modeling; Identified obstacles are imported to the SysML tool in which obstacles are validated during modeling and testing (solid arrows).

more reliant on machine learning algorithms which can learn and accumulate knowledge from historical data to support intelligent decision making [39]. Adapting machine learning algorithms into obstacle analysis will assist engineers of CPS in identifying obstacles that occur in the different components of CPS (i.e., *cyber*, *physical*, or *cyber-physical*). We added *uncertainty* as a fourth category for obstacles that cannot be determined at RE time, either because of lack of knowledge or if it is based on assumptions. This means to systematically classify and specify obstacles of CPS, especially the ones that might arise under uncertain conditions. Below, we present our feature selection for classifying obstacles, experimental setup, and results.

A. Feature Selection

We identified attributes for obstacles based on their classification in [19]. According to van Lamsweerde and Letier [19], obstacles could correspond to:

- **Non-satisfaction obstacles**: obstacles that obstruct the satisfaction of agent requests.
- **Inaccuracy obstacles**: obstacles that obstruct the consistency between the state of objects in the environment and its representation in the software.
- **Hazard obstacles**: obstacles that obstruct safety goals.
- **Non-information obstacles**: obstacles that obstruct the generic goal of making agents informed about object states.
- **Threat obstacles**: obstacles that obstruct security goals.

Other than these attributes, we derive two features based on the goal model of the system: **Softgoal obstacles**: obstacles that obstruct softgoals, and

Leaf obstacles: obstacles that obstruct an expectation or a requirement; assignable to an environment agent or to a software agent, respectively.

Based on these attributes, we develop an automated way of classifying obstacles using machine learning algorithms. Specifically, we derive 7 attributes for our classification, using different scales. Non-satisfaction obstacles are assessed in a range between (0–1), based on each obstacle's weight in the obstacles tree. For each parent obstacle, we give equal weight of its sub obstacle. For example, non-satisfactions weight for "loss of control obstacle" is 1 and all its child obstacles is 0.14. While the rest of attributes are assessed in a clear-cut sense (0 or 1). We built the training sets by manually labeling obstacles according to our features selection as being *cyber, physical, cyber-physical, or uncertainty*. In CPS, each obstacle could be assigned to one or more features. For example, the obstacle *Wheel Speed sensor malfunction* is assigned to: leaf obstacle (based on the obstacle tree), inaccuracy obstacle (obstructing consistency between vehicle speed in the environment and its representation in the software), and non-satisfaction obstacle (based on the obstacle's weight in the refinement tree). Table II shows an example of labeling obstacles of the ABS subject system. We made our datasets, together with other materials available in [40] for replications and cross validation purposes.

B. Experimental Setup

To evaluate our approach, we used the obstacles of the three subject systems as our datasets (24, 54 and 25 in ABS, bCMS, and TCM, respectively). We performed experiments on several classifiers, from different types of algorithms namely, SVM, Naive Bayes, and J48 decision-tree induction, which is the Weka implementation of C4.5. We evaluated all algorithms under a WEKA-based framework running under Java JDK 1.8 [2], an open-source machine learning package.

An appropriate validation method is necessary to measure the generalization error of the implemented model. Recent experimental and theoretical results show that for selecting a good classifier, the ten-fold cross-validation may perform better than more expensive leave-one out cross-validation [41]. Therefore, we used the 10-fold cross validation technique to identify the best model with the optimal parameters and to test its performance on new instances. The corrected paired t-test was run ten times for each pair of learning schemes and a 0.05 significance level was used.

In a cross validation, the dataset consists of samples partitioned into k equal size subsamples, where *k-1* are used in each iteration to train the model, and the remaining one is used for testing. This procedure is then repeated until all folds are used exactly once in the testing set and *k-1* in the training set. The classification accuracy is then evaluated as the mean of all obtained results in the different iterations. To identify the best classifier, we need to determine which one results in the highest accuracy (i.e., AUC).

[2]https://www.cs.waikato.ac.nz/ml/weka/

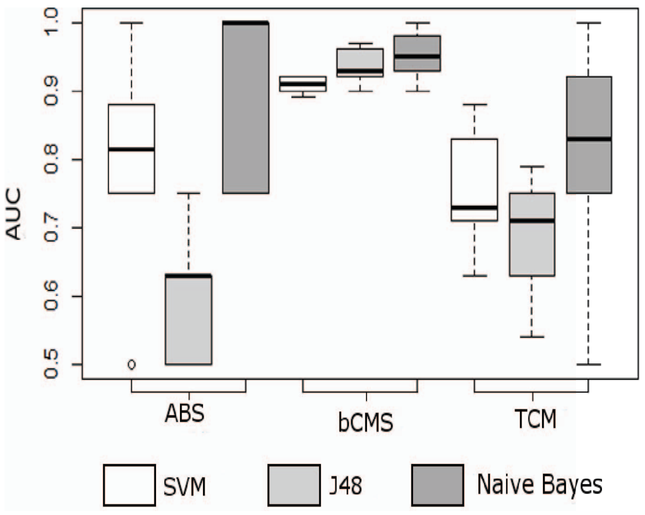


Fig. 3. Classifiers performance for the three datasets: AntiLock-Braking System (ABS), Transmission Control Module (TCM), and the barbados Car Crash Management System (bCMS), according to Area Under the Curve (AUC).

C. Results

The performance of the different classifiers is assessed by the area under the receiver operating characteristics curve (AUC). The AUC can be derived from the confusion matrix. AUC is a cut-off independent measure that accounts for the overall performance of a classification technique, since it considers all the possible cut-off values. The results show that Naive Bayes performs statistically better than J48 and SVM in all the three datasets. Specifically, the average AUC is 92%, 95%, and 82% for ABS, bCMS, and TCM, respectively. This may be reasonable since we only have a small instance in our datasets. It is possible that SVM and J48 do not have enough data for training the model, while Naive Bayes works better for such cases. While SVM performs better than J48 in two datasets, namely ABS and bCMS, J48 outperforms SVM in the TCM dataset. Figure 3 presents the average cross-validated results over the different datasets in terms of *AUC*. As more data we fit to the classifier, the performance gets better. For example, Figure 4 presents learning curves that clearly demonstrate error rate reduction of Naive Bayes classifier.

Next, we integrate the classified obstacles into model testing using a SysML tool, namely, Magic Draw[3], an industry leading cross-platform collaborative Model-Based Systems Engineering (MBSE) environment.

V. Obstacle-Driven Model Testing

In order to test the manifestation of these obstacles coming from different disciplines, SysML is considered as a stan-

[3]https://www.nomagic.com/products/cameo-systems-modeler#intro

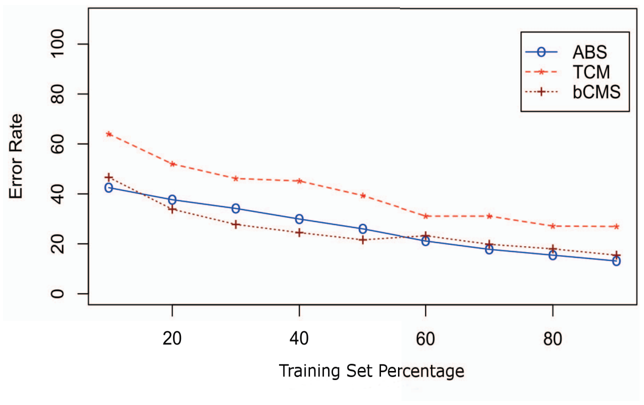


Fig. 4. Learning curve for Naive Bayes across the three datasets.

dard modeling language to be used. SysML is a graphical language for building models of large-scale, complex, and multi-disciplinary systems. SysML shows a great promise for creating object-oriented models of systems that incorporate not only software, but also hardware, devices, and other software control components [42], expressing both structure and behavior for complex systems.

In our work, we used multiple SysML diagrams. To express structure, we used Block Definition Diagram which describes the system hierarchy and system/component classifications, Internal Block Diagram which gives greater detail regarding the specific nature of the relationships between blocks (e.g., showing the details of flows or information transfer among blocks), and Parametric Diagram which represents constraints on system property values such as performance, reliability, and mass properties. To express behavior, we used State Machine Diagrams and Activity Diagrams. For documenting the requirements, we used Requirements Diagram. In this diagram, systems engineers document system requirements and show the relationships that exist between requirements and other system artifacts (e.g., satisfy, refine, derive, testcase). The built-in mechanism SysML has for modeling the requirements allows engineers to relate requirements and design elements/models described at different levels of abstraction.

Since Magic Draw allows for importing and updating requirements from other data sources using the capabilities provided by Cameo Requirements Modeler plugin, we imported the generated obstacles to the tool as extended requirements. Using extended requirements helps us capture the source of the requirement, associate risk level, and specify the verification method (i.e., test in our case). Then these obstacles are linked to the main requirement (i.e., leaf requirement in the goal model) by a *"deriveReq link"* or a *"containment link"*. Figure 5 shows a partial view of Requirements Diagram for ABS subject system. Since we view these obstacles as extended requirements to be tested, we derive a new requirement from each obstacle. For example, the requirement derived from the obstacle *"ABS malfunction"* is *"ABS shall prevent wheel lockup under all braking conditions"*. If this requirement is violated, answering the question *"what are the conditions that causes this violation?"* is our way of integrating obstacle analysis to model testing. In other words, testing is not only driven by requirements but also by obstacles.

Some of these obstacles must be formalized in order to derive a test case. As these obstacles are based on textual information, they present the same problems of an informal specification. Thus, we need to refine and formalize them so that the underlying natural language engine of Magic Draw can convert them to equations (e.g., the amount of brake fluid shall not fall below a certain minimum). Otherwise, we have to write the needed equations manually. Obstacles that cannot be refined to a level that we can derive a test case for, means they require a deep expertise knowledge to be tested (e.g., mechanical engineer). Formalizing and refining these requirements helps engineers in identifying the lower level of abstraction for the requirement.

A challenge in SysML is that it has to be tailored for use in a specific domain [43] (e.g., differentiating between the different components of CPS). Therefore, we have defined a UML requirements profile with four stereotypes, namely: cyber, physical, cyber-physical, and uncertain to stereotype each obstacle. The predicted labels from our classifier can also help in this step. After modeling the obstacles (i.e., extended requirements) in the Requirements Diagram and the structure model of the system, we can check Requirements Diagram model completeness by applying validation rules (e.g., constraints in a form of OCL expressions). For example, we define a validation rule to check that each obstacle has a name. This is useful especially for new derived obstacles (discussed below). We then derive test cases to automatically execute model testing. We used Cameo Simulation Toolkit [44], a Magic Draw plugin, for this purpose.

The Cameo Simulation Toolkit provides the first in the industry extendable model execution framework [44]. It allows engineers not only to model any subject at a higher level of abstraction but also to do so in sufficient detail that it can be executed and simulated. Cameo Simulation Toolkit also allows engineers to perform system testing in both manual and automated modes. Manual testing is performed in the executed models to verify expected results (e.g., value ranges), while automated testing is performed by modeling test case scenarios to test the desired behavior [44]. In fact, the derived test cases in our ABS subject system can be verified using Parametric Diagrams, Activity Diagrams, or State Machine Diagrams. For example, when ABS is running, stopping distance should not exceed a certain value (i.e., constraint). Parametric Diagram was used to verify this constraint. While performing simulation, we can test different values to check the satisfaction of the requirement and further do analysis in the system.

While creating the structure model of the system, and during the process of testing the ABS subject system, we realized that some of the obstacles are redundant. This is because some obstacles are linked to the same block system or the same activity action. For example, we found that the obstacle

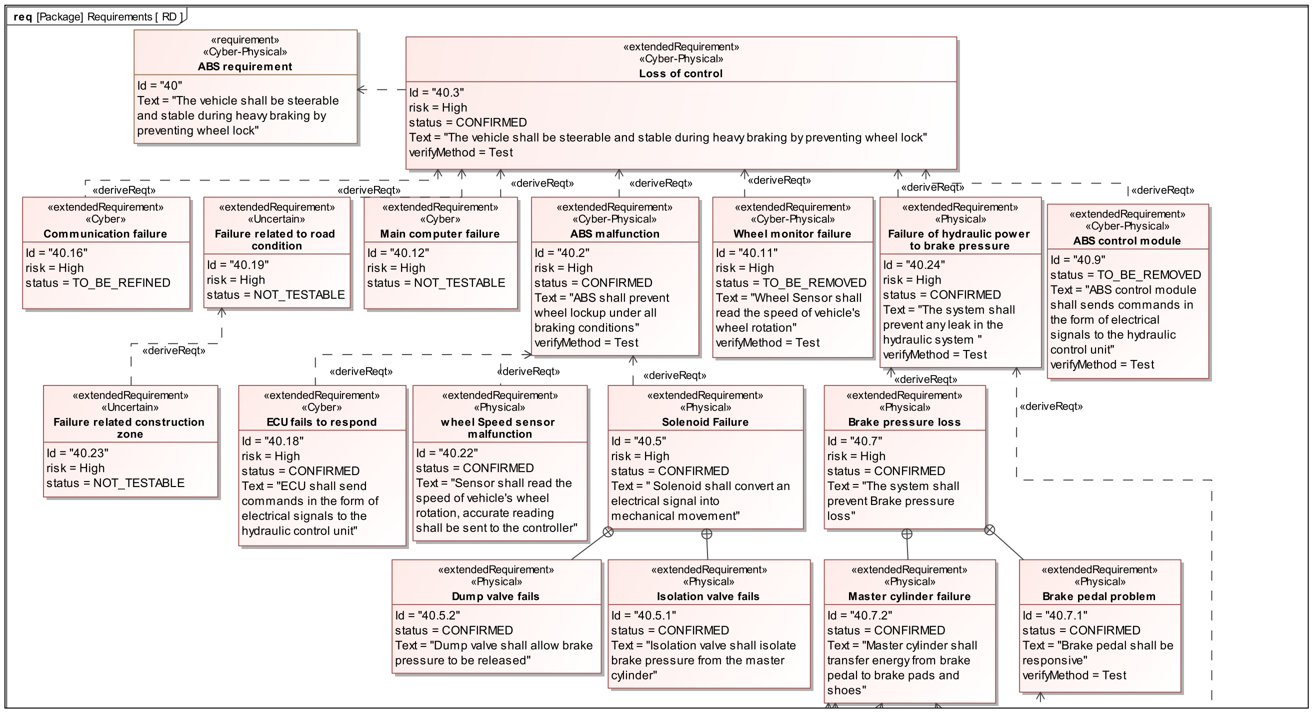

req [Package] Requirements [RD]
«requirement» «Cyber-Physical» ABS requirement
Id = "40"
Text = "The vehicle shall be steerable and stable during heavy braking by preventing wheel lock"
«extendedRequirement» «Cyber-Physical» Loss of control
Id = "40.3"
risk = High
status = CONFIRMED
Text = "The vehicle shall be steerable and stable during heavy braking by preventing wheel lock"
verifyMethod = Test
«deriveReqt»
«extendedRequirement» «Cyber» Communication failure
Id = "40.16"
risk = High
status = TO_BE_REFINED
«extendedRequirement» «Uncertain» Failure related to road condition
Id = "40.19"
risk = High
status = NOT_TESTABLE
«extendedRequirement» «Cyber» Main computer failure
Id = "40.12"
risk = High
status = NOT_TESTABLE
«extendedRequirement» «Cyber-Physical» ABS malfunction
Id = "40.2"
risk = High
status = CONFIRMED
Text = "ABS shall prevent wheel lockup under all braking conditions"
verifyMethod = Test
«extendedRequirement» «Cyber-Physical» Wheel monitor failure
Id = "40.11"
risk = High
status = TO_BE_REMOVED
Text = "Wheel Sensor shall read the speed of vehicle's wheel rotation"
verifyMethod = Test
«extendedRequirement» «Physical» Failure of hydraulic power to brake pressure
Id = "40.24"
risk = High
status = CONFIRMED
Text = "The system shall prevent any leak in the hydraulic system "
verifyMethod = Test
«extendedRequirement» «Cyber-Physical» ABS control module
Id = "40.9"
status = TO_BE_REMOVED
Text = "ABS control module shall sends commands in the form of electrical signals to the hydraulic control unit"
verifyMethod = Test
«extendedRequirement» «Uncertain» Failure related construction zone
Id = "40.23"
risk = High
status = NOT_TESTABLE
«extendedRequirement» «Cyber» ECU fails to respond
Id = "40.18"
risk = High
status = CONFIRMED
Text = "ECU shall send commands in the form of electrical signals to the hydraulic control unit"
«extendedRequirement» «Physical» wheel Speed sensor malfunction
Id = "40.22"
status = CONFIRMED
Text = "Sensor shall read the speed of vehicle's wheel rotation, accurate reading shall be sent to the controller"
«extendedRequirement» «Physical» Solenoid Failure
Id = "40.5"
risk = High
status = CONFIRMED
Text = " Solenoid shall convert an electrical signal into mechanical movement"
«extendedRequirement» «Physical» Brake pressure loss
Id = "40.7"
risk = High
status = CONFIRMED
Text = "The system shall prevent Brake pressure loss"
«extendedRequirement» «Physical» Dump valve fails
Id = "40.5.2"
status = CONFIRMED
Text = "Dump valve shall allow brake pressure to be released"
«extendedRequirement» «Physical» Isolation valve fails
Id = "40.5.1"
status = CONFIRMED
Text = "Isolation valve shall isolate brake pressure from the master cylinder"
«extendedRequirement» «Physical» Master cylinder failure
Id = "40.7.2"
status = CONFIRMED
Text = "Master cylinder shall transfer energy from brake pedal to brake pads and shoes"
«extendedRequirement» «Physical» Brake pedal problem
Id = "40.7.1"
status = CONFIRMED
Text = "Brake pedal shall be responsive"
verifyMethod = Test

Fig. 5. Requirements diagram for ABS case study (partial view)

"ECU fails to respond" is the same as "ABS control module failure". Similarly, the obstacle "Wheel monitor failure" is the same as "Wheel Speed sensor malfunction". We also found that some obstacles need more refinement (e.g., CAN communication failure must be refined to CAN bus failure and loose bus connector). As we analyze the system, we identify new obstacles in addition to the identified obstacles in the obstacle tree (e.g., "Brake line leak" as new derived obstacle from "Loss of brake fluid"), and we also create new links between already identified obstacles (e.g., a derive requirement link between "Loss of brake fluid" and "Master cylinder failure"), since one obstacle could be a condition for violating more than one requirement.

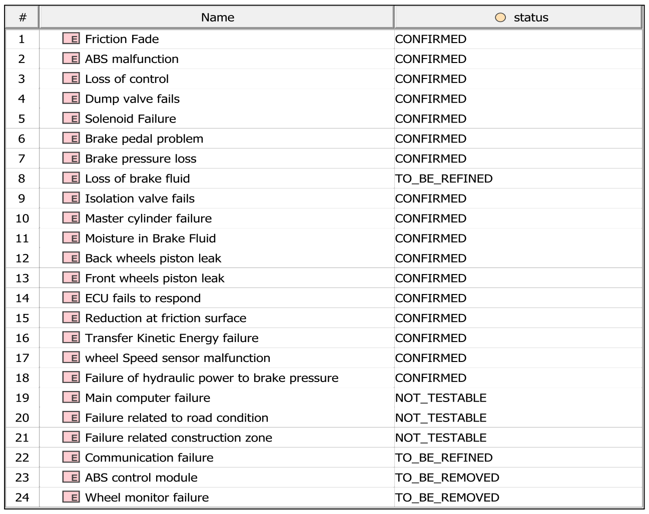
Thus, we created four properties to the requirements profile stereotype to check each obstacle *status*:

- **Confirmed**: to confirm that the obstacle is accurate and testable.
- **To-be-refined**: the obstacle needs a refinement.
- **To-be-removed**: the obstacle is redundant and thus must be removed.
- **Not-testable**: the obstacle cannot be tested within the SysML tool and requires other simulation tools (e.g., Modelica [45]) to describe the precise behavior of the system.

Our results for the ABS case study show that 2 obstacles are redundant, 2 obstacles need more refinements, 3 obstacles are not testable within the tool (1 of them was a cyber obstacle), and 17 obstacles are confirmed and can be model tested (mostly phyiscal obstacles). This shows that within SysML physical obstacles are addressed more readily than the cyber obstacles. Figure 6 shows the status of each obstacle after model testing, which can also be seen in the Requirements Diagram (Figure 5). We created a Verify Traceability Matrix in which we can visualize and verify that all obstacles are covered with test cases. The tool will automatically identify if there is any suspect links to be resolved. Figure 7 shows the traceability matrix for obstacles and their associated test cases. Our results indicate that while model testing is guided by our obstacles identification, it provides a mechanism for validating these obstacles during model testing via deriving new or identifying redundant obstacles. In other words, model testing is not only driven by the obstacles but also provides a mechanism for making obstacle analysis more robust at RE time.

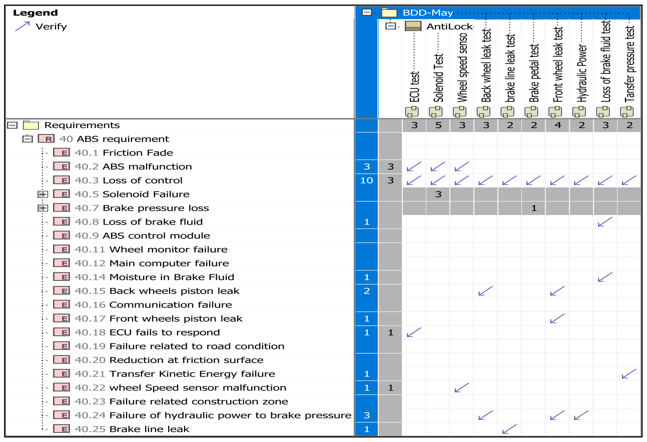
VI. Threats To Validity

In this section, we discuss the potential threats to the validity of our findings. Several factors can affect the validity of our study. Threats to external validity [46] impacts the generalizability of results. The primary threat to our study's external validity is the limited size of the datasets used in this experiment. However, we believe that the use of three datasets with different complexity helps to mitigate this threat. Another external validity could be our chosen tool. For example, Magic Draw allows executing SysML diagrams (e.g., parametric diagrams) and testing different values while simulation, which may not apply to other SysML tools.



#	Name	status
1	Friction Fade	CONFIRMED
2	ABS malfunction	CONFIRMED
3	Loss of control	CONFIRMED
4	Dump valve fails	CONFIRMED
5	Solenoid Failure	CONFIRMED
6	Brake pedal problem	CONFIRMED
7	Brake pressure loss	CONFIRMED
8	Loss of brake fluid	TO_BE_REFINED
9	Isolation valve fails	CONFIRMED
10	Master cylinder failure	CONFIRMED
11	Moisture in Brake Fluid	CONFIRMED
12	Back wheels piston leak	CONFIRMED
13	Front wheels piston leak	CONFIRMED
14	ECU fails to respond	CONFIRMED
15	Reduction at friction surface	CONFIRMED
16	Transfer Kinetic Energy failure	CONFIRMED
17	wheel Speed sensor malfunction	CONFIRMED
18	Failure of hydraulic power to brake pressure	CONFIRMED
19	Main computer failure	NOT_TESTABLE
20	Failure related to road condition	NOT_TESTABLE
21	Failure related construction zone	NOT_TESTABLE
22	Communication failure	TO_BE_REFINED
23	ABS control module	TO_BE_REMOVED
24	Wheel monitor failure	TO_BE_REMOVED

Fig. 6. Obstacles status using requirements profile



Legend
↗ Verify

			BDD-May									
			AntiLock									
			ECU test	Solenoid Test	Wheel speed senso	Back wheel leak test	brake line leak test	Brake pedal test	Front wheel leak test	Hydraulic Power	Loss of brake fluid test	Tansfer pressure test
Requirements			3	5	3	3	2	2	4	2	3	2
40 ABS requirement												
40.1 Friction Fade												
40.2 ABS malfunction	3	3	↙	↙	↙							
40.3 Loss of control	10	3	↙	↙	↙	↙	↙	↙	↙	↙	↙	↙
40.5 Solenoid Failure				3								
40.7 Brake pressure loss								1				
40.8 Loss of brake fluid	1										↙	
40.9 ABS control module												
40.11 Wheel monitor failure												
40.12 Main computer failure												
40.14 Moisture in Brake Fluid	1										↙	
40.15 Back wheels piston leak	2					↙			↙			
40.16 Communication failure												
40.17 Front wheels piston leak	1								↙			
40.18 ECU fails to respond	1	1	↙									
40.19 Failure related to road condition												
40.20 Reduction at friction surface												
40.21 Transfer Kinetic Energy failure	1											↙
40.22 wheel Speed sensor malfunction	1	1			↙							
40.23 Failure related construction zone												
40.24 Failure of hydraulic power to brake pressure	3					↙			↙	↙		
40.25 Brake line leak	1						↙					

Fig. 7. Verify traceability matrix where rows represent obstacles (i.e., extended requirements), and columns represent test cases.

Threats to internal validity are influences that can affect the independent variable with respect to causality [46]. Internal validity could be our selection of attributes. However, we follow the classification of obstacles according to KAOS model in [19], and also base on the elements of the KAOS goal model. Construct validity is the degree to which the variables accurately measure the concepts they claim to measure [46]. To mitigate the threats, we adopt AUC metric, which is extensively used in classification problems.

VII. Practical application

Integrating obstacles analysis techniques helps requirements engineers to identify obstacles at RE time. However, in practice, especially in the context of CPS, it is not clear whether the obstacles are due to cyber or physical components, or both. Thus, classifying and categorizing these obstacles via machine learning algorithms can provide an initial mechanism for: 1) identifying these components, especially for obstacles that might arise under uncertain conditions (i.e., obstacles due to uncertainty), and 2) successfully enabling specific domain modeling approach at earlier stages, (e.g., guide in identifying requirements stereotypes).

One of the challenges that modelers have when writing requirements in SysML is that there is little information on how to properly layout the Requirements Diagrams [47]. We observed that obstacle tree (also goal model) can be mapped to a Requirements Diagram in SysML, in which the negated goal is the main requirement (i.e., a general requirement in SysML), and the obstacles are extended requirements. This could guide modelers to better layout the Requirements Diagram.

As mentioned earlier, the deeper we trace the obstacle tree; the more finer obstacles are. Specifically, obstacles at the leaves should correspond to cyber, physical, but not cyber-physical components. SysML model testing then can be used for validating these obstacles, and further helps engineers during maintenance in identifying the responsible disciplines (i.e., software engineers, electrical engineers, and/or mechanical engineers) for handling these obstacles, which saves efforts and time.

VIII. Conclusion

In this paper, we investigated machine learning's capabilities of automatically classifying the obstacles generated from KAOS into four components (i.e., cyber, physical, cyber-physical, and uncertain). We then derived test cases based on the classified obstacles to automatically execute model testing within a state-of-practice SysML tool, namely, Magic Draw. Our results show that Naive Bayes are among the most stable obstacle classifiers. In particular, Naive Bayes outperforms J48 and SVM using three different datasets. Additionally, while model testing is guided by our obstacles identification, it provides a mechanism for validating these obstacles during model testing.

Our future work includes defining heuristics for deriving and testing obstacles in the context of CPS, and further integrating obstacle resolutions techniques of the derived obstacles in case of their occurrence. We also plan to identify specific properties and constraints for each of the created CPS stereotypes.

Acknowledgement

This research is partially supported by the U.S. National Science Foundation (Award CCF-1350487). Alenazi is supported from University of Hafr-ALBatin and University of Cincinnati Scholarships.

References

[1] S. C. Suh, U. J. Tanik, J. N. Carbone, and A. Eroglu, *Applied Cyber-Physical Systems*. Springer, 2014.

[2] Object Management Group, "Systems Modeling Language (SysML)," http://www.omgsysml.org, Last accessed: June 2018.

[3] W. Schäfer and H. Wehrheim, "The challenges of building advanced mechatronic systems," in *International Conference on the Future of Software Engineering (FOSE)*, Minneapolis, MN, USA, May 2007, pp. 72–84.

[4] M. Sabetzadeh, S. Nejati, L. C. Briand, and A.-H. E. Mills, "Using SysML for modeling of safety-critical software-hardware interfaces: guidelines and industry experience," in *International Symposium on High-Assurance Systems Engineering (HASE)*, Boca Raton, FL, USA, November 2011, pp. 193–201.

[5] No Magic, "Cameo Systems Modeler MagicDraw," Last accessed: June 2018. [Online]. Available: {https://www.nomagic.com/products/cameo-systems-modeler}

[6] L. C. Briand, D. Falessi, S. Nejati, M. Sabetzadeh, and T. Yue, "Traceability and SysML design slices to support safety inspections: A controlled experiment," *ACM Transactions on Software Engineering and Methodology*, vol. 23, no. 1, Article 9, February 2014.

[7] S. Nejati, M. Sabetzadeh, D. Falessi, L. C. Briand, and T. Coq, "A SysML-based approach to traceability management and design slicing in support of safety certification: framework, tool support, and case studies," *Information & Software Technology*, vol. 54, no. 6, pp. 569–590, June 2012.

[8] S. Nejati, M. Sabetzadeh, C. Arora, L. C. Briand, and F. Mandoux, "Automated change impact analysis between SysML models of requirements and design," in *International Symposium on Foundations of Software Engineering (FSE)*, Seattle, WA, USA, November 2016, pp. 242–253.

[9] M. Alenazi, D. Reddy, and N. Niu, "Assuring virtual-plc in the context of SysML models," in *New Opportunities for Software Reuse (ICSR)*, vol. 10826. Springer, 2018, pp. 1–16.

[10] O. Badreddin, A. Sturm, and T. C. Lethbridge, "Requirement traceability: A model-based approach," in *Model-Driven Requirements Engineering Workshop (MoDRE), 2014 IEEE 4th International*. IEEE, 2014, pp. 87–91.

[11] M. Alenazi, N. Niu, W. Wang, and A. Gupta, "Traceability for automated production systems: A position paper," in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2017, pp. 51–55.

[12] A. K. Chopra, F. Dalpiaz, F. B. Aydemir, P. Giorgini, J. Mylopoulos, and M. P. Singh, "Protos: foundations for engineering innovative sociotechnical systems," in *International Requirements Engineering Conference (RE)*, Karlskrona, Sweden, August 2014, pp. 53–62.

[13] A. K. Chopra and M. P. Singh, "From social machines to social protocols: Software engineering foundations for sociotechnical systems," in *International Conference on World Wide Web (WWW)*, Montreal, Canada, April 2016, pp. 903–914.

[14] L. C. Briand, S. Nejati, M. Sabetzadeh, and D. Bianculli, "Testing the untestable: model testing of complex software-intensive systems," in *International Conference on Software Engineering (ICSE)*, Austin, TX, USA, May 2016, pp. 789–792.

[15] S. Ali, H. Lu, S. Wang, T. Yue, and M. Zhang, "Uncertainty-wise testing of cyber-physical systems," *Advances in Computers*, vol. 106, pp. 23–94, 2017.

[16] M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz, and R. Norgren, "Understanding uncertainty in cyber-physical systems: a conceptual model," in *European Conference on Modelling Foundations and Applications (ECMFA)*, Vienna, Austria, July 2016, pp. 247–264.

[17] S. Ali and T. Yue, "U-Test: evolving, modelling and testing realistic uncertain behaviours of cyber-physical systems," in *International Conference on Software Testing, Verification and Validation (ICST)*, Graz, Austria, April 2015, pp. 1–2.

[18] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Science of Computer Programming*, vol. 20, no. 1-2, pp. 3–50, April 1993.

[19] A. van Lamsweerde and E. Letier, "Handling obstacles in goal-oriented requirements engineering," *IEEE Transactions on Software Engineering*, vol. 26, no. 10, pp. 978–1005, October 2000.

[20] E. A. Lee, "Cyber physical systems: design challenges," in *International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, Orlandao, FL, USA, May 2008, pp. 363–369.

[21] M. Waibel, L. Steenkamp, N. Moloko, and G. Oosthuizen, "Investigating the effects of smart production systems on sustainability elements," *Procedia Manufacturing*, vol. 8, pp. 731–737, 2017.

[22] N. Niu and S. Easterbrook, "Extracting and modeling product line functional requirements," in *International Requirements Engineering, 2008. RE'08. 16th IEEE*. IEEE, 2008, pp. 155–164.

[23] M. Sabetzadeh, S. Nejati, L. Briand, and A.-H. E. Mills, "Using SysML for modeling of safety-critical software-hardware interfaces: Guidelines and industry experience," in *High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on*. IEEE, 2011, pp. 193–201.

[24] S. Nejati, M. Sabetzadeh, D. Falessi, L. Briand, and T. Coq, "A SysML-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies," *Information and Software Technology*, vol. 54, no. 6, pp. 569–590, 2012.

[25] J. D. Musa, A. Iannino, and K. Okumoto, "Software reliability: Measurement, prediction, application. 1987," 1987.

[26] G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing*. John Wiley & Sons, 2011.

[27] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, no. 1, p. 6, 2013.

[28] M. Zhang, S. Ali, T. Yue, R. Norgren, and O. Okariz, "Uncertainty-wise cyber-physical system test modeling," *Software & Systems Modeling*, pp. 1–40, 2017.

[29] S. Mitra, T. Wongpiromsarn, and R. M. Murray, "Verifying cyber-physical interactions in safety-critical systems," *IEEE Security & Privacy*, vol. 11, no. 4, pp. 28–37, 2013.

[30] E. A. Lee and S. A. Seshia, *Introduction to embedded systems: A cyber-physical systems approach*. MIT Press, 2016.

[31] A. van Lamsweerde, "KAOS tutorial," *Cediti, September*, vol. 5, 2003.

[32] A. Cailliau, C. Damas, B. Lambeau, and A. van Lamsweerde, "Modeling car crash management with kaos," in *Comparing Requirements Modeling Approaches Workshop (CMA@ RE), 2013 International*. IEEE, 2013, pp. 19–24.

[33] Automative Electronics, "Transmission Control," http://cecas.clemson.edu/cvel/auto/systems/transmission-control.html, last accessed: June 2018.

[34] A. Capozucca, B. Cheng, G. Georg, N. Guelfi, P. Istoan, G. Mussbacher, A. Jensen, J.-M. Jézéquel, J. Kienzle, J. Klein *et al.*, "Requirements definition document for a software product line of car crash management systems," *ReMoDD repository, at http://www. cs. colostate. edu/remodd/v1/content/bcms-requirements-definition*, 2011.

[35] E. M. Clarke and P. Zuliani, "Statistical model checking for cyber-physical systems," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2011, pp. 1–12.

[36] P. Antsaklis, "Goals and challenges in cyber-physical systems research editorial of the editor in chief," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3117–3119, 2014.

[37] M. Lukasiewycz, S. Steinhorst, F. Sagstetter, W. Chang, P. Waszecki, M. Kauer, and S. Chakraborty, "Cyber-physical systems design for electric vehicles," in *Digital System Design (DSD), 2012 15th Euromicro Conference on*. IEEE, 2012, pp. 477–484.

[38] D. Alrajeh, J. Kramer, A. van Lamsweerde, A. Russo, and S. Uchitel, "Generating obstacle conditions for requirements completeness," in *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012, pp. 705–715.

[39] B. Tang, "Toward intelligent cyber-physical systems: Algorithms, architectures, and applications," 2016.

[40] M. Alenazi and N. Niu, "Subject systems dataset," Last accessed: June 2018. [Online]. Available: https://scholar.uc.edu/show/3f462575p

[41] D. M. Hawkins, S. C. Basak, and D. Mills, "Assessing model fit by cross-validation," *Journal of chemical information and computer sciences*, vol. 43, no. 2, pp. 579–586, 2003.

[42] E. Huang, R. Ramamurthy, and L. F. McGinnis, "System and simulation modeling using SysML," in *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come*. IEEE Press, 2007, pp. 796–803.

[43] Darius Silingas, "Enabling Domain-Specific Extensions to SysML," https://www.nomagic.com/events/webinars/item/webinar-7-2015-enabling-domain-specific-extensions-to-sysml, Last accessed: June 2018.

[44] No Magic, "Cameo Simulation Toolkit," Last accessed: June 2018. [Online]. Available: {https://www.nomagic.com/product-addons/magicdraw-addons/cameo-simulation-toolkit}

[45] Modelica, "Modelica and the Modelica Association," https://www.modelica.org/, last accessed: June 2018.

[46] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[47] Andrius Armonas, "Requirements Writing in SysML," https://www.nomagic.com/mbse/images/whitepapers/Requirements-Writing-in-SysML.pdf, Last accessed: June 2018.