Iterative Scenario-Based Testing in an Operational Design Domain for Artificial Intelligence Based Systems in Aviation

Bojan Lukic[1*], Jasper Sprockhoff[1], Alexander Ahlbrecht[1], Siddhartha Gupta[1],

Umut Durak[1]

[1]German Aerospace Center (DLR), Institute of Flight Systems, Lilienthalplatz 7, 38108 Braunschweig, Germany
[*]*bojan.lukic@dlr.de*

Abstract. The use and development of Artificial Intelligence (AI) based systems is becoming increasingly prominent in different industries. The aviation industry is also gradually adopting AI-based systems, for instance, with Machine Learning algorithms for flight assistance. There are several reasons why adopting these technologies poses additional obstacles in aviation compared to other industries. One reason are the strong safety requirements which lead to obligatory and thorough assurance activities such as testing to obtain certification. Therefore, a systematic approach is needed for developing, deploying, and assessing test cases for AI-based systems in aviation. This paper proposes a method for iterative scenario-based testing for AI-based systems. The method contains three major parts: First, a high-level description of test scenarios; second, the generation and execution of these scenarios; and last, monitoring of parameters during scenario execution. Parameters are refined, and the steps are repeated iteratively. The method forms a basis for developing iterative scenario-based testing solutions. As a domain-specific example, a practical implementation of this method is illustrated. For an object detection application used on an airplane, flight scenarios, including multiple airplanes are generated from a descriptive scenario model and executed in a simulation environment. The parameters are monitored using a custom Operational Design Domain monitoring tool and refined in the process of iterative scenario generation and execution. The proposed iterative scenario-based testing method helps in generating precise test cases for AI-based systems while having a high potential for automation.

Introduction

The practical use of Machine Learning (ML) applications for Artificial Intelligence (AI) based systems in aviation is still in an early stage. One reason is the premature nature of guidelines illustrating the proper implementation of those applications. Specifically, the additional and strict requirements and constraints for introducing new systems in the aviation industry pose an obstacle. This makes the implementation and certification of ML algorithms for autonomy challenging. Recently, the European Union Aviation Safety Agency (EASA) [1] and Society of Automotive Engineers (SAE) [2] each published early versions of fundamental guidelines, discussing the implementation of ML applications in aeronautical systems. These guidelines provide guidance for implementing level 1 ML applications, which can assist humans. Due to the premature nature of these guidelines, the certifiability of ML applications in aviation, especially for fully AI-based systems, is not yet given. Yet, similar to traditional software, it is certain that specific verification artifacts need to be provided to increase trust. Typical artifacts include the results of conducted tests. As defined in the EASA guidance, implementing AI-based systems requires the exact definition of their Operational Design Domain (ODD).

The ODD defines the conditions under which a system operates correctly. In the domain of AI-based systems, the ODD defines the execution boundaries under which the AI-based system is designed and defines the parameters which need to be satisfied for the system to properly operate [3]. The definition of parameter boundaries for the correct behavior of an AI-based system becomes especially important when working in safety-critical domains such as aviation. For instance, the ODD of an aviation system can help with the definition of design assurance levels [4]. In this context, the definition of the system's ODD guarantees the generation of precise test cases for high test coverage. One systematic approach for developing test cases for AI-based systems in their operational domain is model-

based testing using the Model-Based Systems Engineering (MBSE) methodology. Due to its highly descriptive nature and model-centric approach [5], MBSE is an appropriate methodology to model systems on all levels of abstraction, making it useful in the development process of test cases for ML applications [6]. The iterative scenario testing concept presented in this work is amongst others exemplified with methods from MBSE.

This paper discusses the generation of test scenarios for an AI-based system. The use case is about using a computer vision algorithm to perform object detection and determine the distance to other aircraft to predict dangerous situations. The scenarios represent different situations with foreign aircraft used for testing. The detailed use case is explained in [7]. A method for iterative scenario-based testing of AI-based systems is presented in the scope of this work. Three essential parts of the method are defined: A high-level description of the scenarios to be executed, the testing environment in which test scenarios are executed, and a monitoring tool for narrowing down the parameter boundaries for the ODD of the respective system. A domain-specific implementation of this methodology is also presented. For modeling the systems involved and developing test cases, the MBSE tool Cameo[1] is used. The simulation is executed in FlightGear, a highly customizable open-source software for flight simulation[2]. The scenarios are generated in a model-based approach in Cameo and then executed in a FlightGear instance. Parameters are monitored using a custom Python library. The findings show that the presented iterative scenario-based testing method facilitates the definition and refinement of test scenarios for AI-based applications.

The remaining paper is structured as follows: In Section 1, related work and the status quo of scenario-based testing with a model-based approach are discussed. Section 2 presents the development of a domain-independent method for iterative scenario-based testing. The implementation of this methodology is presented in Section 3 with tools used for defining scenarios, executing them, and monitoring them.

[1]Dassault Systemes, 2022, Cameo Systems Modeler, available at https://www.3ds.com/products-services/catia/products/no-magic/cameo-systems-modeler/.

[2]FlightGear developers & contributors, 2021. FlightGear, Available at https://www.flightgear.org/.

1 Related Work

In [8], Jafer and Durak discuss the complexity of simulation scenario development in aviation. They propose ontology-based approaches to develop an aviation scenario definition language (ASDL). According to the authors, ontologies provide invaluable possibilities to tackle the complexity of simulation scenario development. Durak presents a model-driven engineering perspective for scenario development in [9]. The use of metamodels for generating executable scenarios is demonstrated with a sample implementation. Durak's work is closely related to the research presented in the work at hand, specifically the development of conceptual metamodels for generating executable scenarios.

Simulation-based data and scenario generation for AI-based airborne systems is discussed by Gupta in [10]. In the work, the authors aim to answer the question of what needs to be simulated for synthetic data and scenario generation in the simulation engineering process of an AI-based system. The used methods are a simulation-based data generation process adapted from EASA's first usable guidance for Level 1 machine learning applications and the scenario-based approach using SES, which is explained more thoroughly in the publications of Durak [11], [12] as well as Karmokar [13]. The work in [10] is succeeded with [14], which discusses behavioral modeling for scenario-based testing in aviation and introduces an enhanced approach for scenario-based testing called Operational Domain Driven Testing.

Closely related, [15] demonstrates the testing of black box systems, such as AI-based applications for autonomous road vehicles, in their ODD. The framework introduced by the authors is used to learn monitors in a feature space and prevent the system from using critical components when exiting its ODD. Scenario-based testing of autonomous road vehicles is discussed in [16] and [17]. The authors present an automated scenario-based testing methodology for vehicles using advanced AI-based applications. The work shows that the presented formal simulation approach effectively finds relevant tests for track testing with a real autonomous vehicle.

In [18], Hungar presents scenario-based testing for automated road vehicles. The outcome of the work is the PEGASUS method, which is used to assess highly automated driving functions. According to the author, the most important steps for scenario-based testing involve capturing all evolutions, i.e. variants, of func-

tional scenarios, formalization of them, systematic testing, the analysis of critical regions, and finally, the development of a risk chart.

Closely related to [9], the work presented in this paper discusses model-driven scenario development. In addition to the methodologies discussed in the related work, an iterative scenario parameter adjustment and generation process is introduced, forming the iterative scenario-based testing method. The method is illustrated with an exemplary generation of test scenarios for an AI-based demonstrator. In the next section, the methodology for this domain- and tool-independent iterative scenario-based testing method is presented.

2 Iterative Scenario-Based Testing Concept

The related work shows that there are many ways to realize scenario-based testing for AI-based systems. Especially when talking about domain-specific tools, a variety of testing strategies are possible. A generalization of these testing strategies can help with defining universal testing methods. To achieve that, a fundamental, tool-independent method is needed to describe the basic methodology for iterative scenario-based testing on a high level of abstraction. This method can then be used to build some domain-specific testing tools. For such iterative scenario-based testing, three fundamental components have been identified:

Scenario Model

First, a high-level description of the testing scenarios needs to be defined. This high-level model can be achieved by describing the scenarios' fundamental components. Modeling tools or formalized methods can for instance be used to formulate the scenarios and derive all required scenario variations from the high-level model. The method shall be capable of generating an arbitrary number of scenarios with high parameter variation from the high-level description to achieve satisfactory test coverage for the application to be verified.

Environment for Scenario Execution

Second, an environment for executing the derived scenarios should be selected. The environment can be of different types, such as simulated, real system, or a mix of both, e.g. real systems extended with elements from augmented reality. These environments have different advantages and disadvantages. A simulated system can be deployed quickly, offers consistent conditions, and is cost-effective. The biggest drawback of simulated environments is their sim-to-real gap. The gap refers to the applicability of simulations to real-life environments, as many simulated environments cannot fully offer all relevant conditions as a real system. The biggest advantage of a real system is its closeness to the real-life environment in which the tested application is designed to operate in. Real systems are hard to deploy and costly. Especially when talking about automated and accelerated testing, real systems can pose a financial and temporal bottleneck in the testing process.

ODD Monitoring

Last, a monitoring tool is required for verification and for tracking all parameters that are necessary for and can have some variance on scenarios. By tracking these parameters and verifying the application to be tested, a precise ODD can be defined for the system. With feedback from the monitoring tool, parameters can be adjusted, or new parameters can be chosen for a new iteration of scenario generation. The tools for monitoring in the chain of scenario-based testing can be arbitrarily chosen as long as they are capable of monitoring parameters in real-time for synchronization purposes.

The described method is of an iterative nature. Each component feeds the next with some information. This loop is depicted in Figure 1.

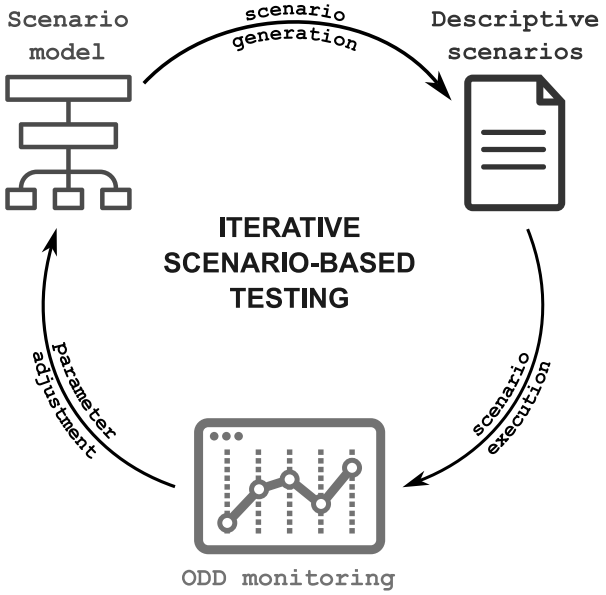

Figure 1: Iterative scenario-based testing

The execution of test scenarios can be accomplished in a simulated environment as well as a real system. Although both approaches are important to consider, the method depicted in Figure 1 is tailored towards testing in simulated environments. For generating application-readable scenario descriptions with the scenario modeling tool, some application, e.g. script, is needed. Similarly, after scenario execution and monitoring, some application is needed which feeds the result logs to the scenario modeling tool, decides on parameter adjustment, and triggers new scenario generation. The use of such intermediate applications and scripts enables high automation and optimization of the method. In ideal circumstances, the iterative scenario-based testing method forms a closed loop with automated test scenario generation, execution, and real-time monitoring of parameters.

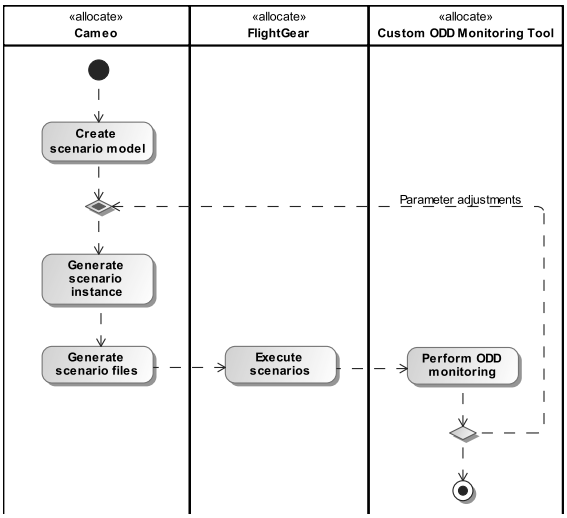

Figure 2: Flow of information and steps for iterative scenario-based testing used in this work

3 Exemplary Implementation

This section explains an exemplary implementation to demonstrate the derived method. For the implementation, domain-specific tools were selected that can be exchanged depending on the use case. The exemplary implementation of discussed method can be divided into three components: First, the MBSE-based scenario description and generation using Cameo; second, the execution of scenarios defined in generated XML files with the flight simulator FlightGear; and last, the monitoring of parameters during scenario execution with a custom ODD monitoring tool. The basic flow of information and steps are illustrated in Figure 2.

The high-level model of the scenarios is described with a profile diagram in Cameo. Profile diagrams are defined in the System Modeling Language. Additionally, extensions are used to increase the modeling capabilities with profile diagrams. One configuration of a specific scenario is generated with a block definition diagram, which can be transformed and exported into the desired XML scenario files with the help of scripts. XML files are generated for the use case on hand, since FlightGear uses a XML format for the scenario execution. However, other domain-specific formats can be used as well. The scenarios are executed within an instance of FlightGear and the parameters are monitored with a custom ODD monitoring tool. A more detailed description of the implementation is shown in the following subsections.

3.1 Scenario Format and MBSE-Based Scenario Generation

A high-level description of the necessary files for scenario execution can be observed in Figure 3. Along with scenario files, flight plan files are needed for scenario execution, as will be explained shortly.

The scenario files include various tags which define the inputs, objects, and attributes when executing them in FlightGear. An important tag is the <entry> tag which defines objects used in a scenario and can include the following additional tags: <*callsign*> – the identification of the aircraft, <*type*> and <*model*> – the type and model of the aircraft, <*flightplan*> – the flight plan which the scenario refers to, and <*repeat*> – a Boolean flag that indicates whether the scenario shall be repeated once or infinitely often.

The flight plans, which the scenario files refer to, are also in XML format. The most important tag in the flight plan is the <*wpt*> tag, which can include the following additional tags: <*name*> – the name of the waypoint, <*lat*> – the latitude of the entry that refers to the flight plan, <*lon*> – the longitude, <*alt*> – the altitude, <*ktas*> – the *knots true airspeed*, <*on-ground*> – if the specified object starts from the ground or not, <*gear-down*> – if the landing gear is retracted or extended, and <*flaps-down*> – if the flaps are retracted or extended. FlightGear offers many more configuration files which can be adjusted to change environmental parameters as

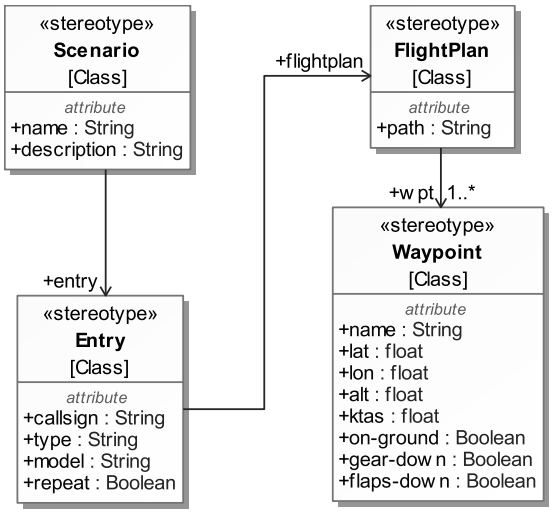

Figure 3: High-level description of the configuration files for FlightGear

well as parameters of entities and other components of interest for scenario-based testing. For simplicity, only the scenario and flight plan files along with their parameters are discussed here. Some high-level description, i.e. metamodel, of the scenario and flight plan files is needed to generate arbitrary test scenarios.

Figure 4 depicts one instance of the high-level description of the scenario and flight plan files.

3.2 Scenario Execution

The scenarios are executed within FlightGear. The respective XML files can be executed manually in a FlightGear instance, or referred to as parameters for automatic execution with startup of FlightGear. For automation purposes, we chose the latter. As explained in the previous subsection, one or more entries, e.g. planes, can be defined in a scenario file, with each flying according to a predefined route. In this instance, one passenger airplane is defined, which narrowly passes the user's plane. Figure 5 shows a screenshot of the scenario during execution in FlightGear.

3.3 ODD Definition and Monitoring

The ODD defines the conditions under which a system operates properly. Several parameters can have a variance on the scenarios executed in FlightGear, some of which were defined above. Additional parameters, such

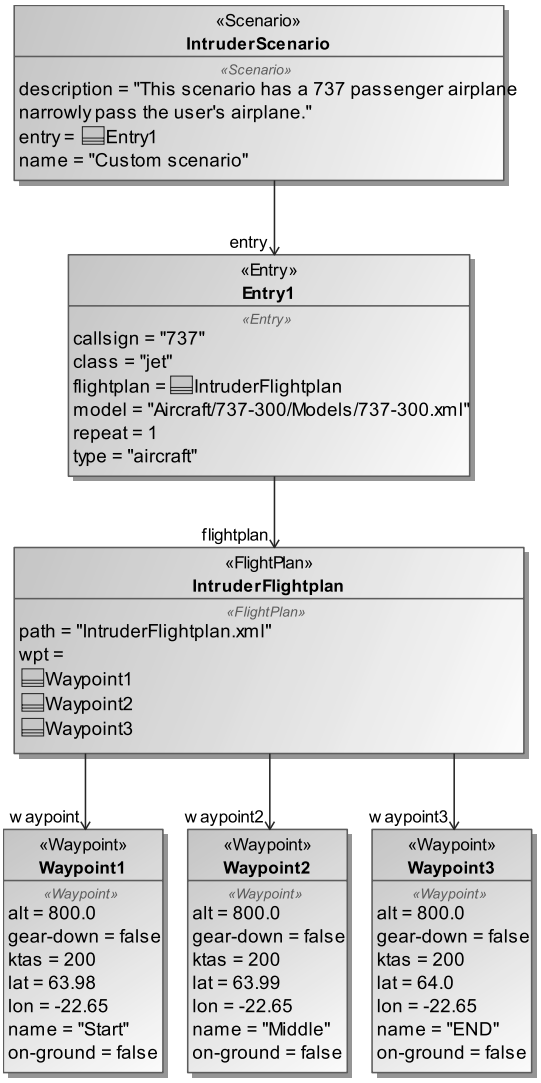

Figure 4: Block definition diagram of one scenario and flight plan configuration

Figure 5: Passenger airplane narrowly passing user's Cessna

as weather conditions, need to be considered. A high-level description of the domain model for the ODD of the AI-based system used on an airplane is depicted in Figure 6.

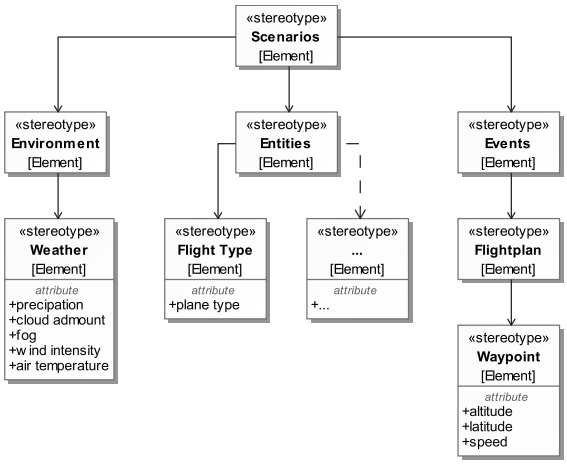

Figure 6: Domain model for the ODD of the scenario-based testing method

The parameter boundaries for the use case of object detection during scenario execution can be determined in an iterative process. Due to the high number of parameters to be considered, a manual exhaustive search for parameter boundaries is highly time-consuming. Therefore, some tool is needed which can track the necessary parameters during scenario execution and give feedback on the result of the tested application.

For monitoring these parameters in FlightGear, a public Python library[3] for fetching parameters from FlightGear's property tree is used. The AI-based system tested in this example is an object detection application. The result of the domain model's object detection and desired parameters can be logged using the monitoring tool. The feedback generated from the tool can then be used to adjust the values of selected parameters in Cameo, generate new scenarios, and narrow down the parameters to fit the ODD of the application. This iterative process can be used to narrow down the ODD boundaries of each parameter with every iteration.

A simplified ODD for the system with the two parameters altitude and speed can be defined as follows: "The application performs correct object detection of intruding airplanes of type Boeing 737 within following parameter boundaries:

- Altitude of intruding airplane relative to own airplane in feet, Δ alt: -100 to 100.
- Cumulative speed of intruding airplane as well as own airplane in knots true airspeed, Σ ktas: 0 to 500."

Table 1 shows an exemplary log recorded during execution of a scenario in FlightGear. For completeness and to reflect other relevant parameters currently covered in the scenario model, the latitude and longitude of the intruding airplane are logged along with the aforementioned altitude and speed.

Log #	lat	lon	Δ alt	Σ ktas	detect
1	63.970	-22.65	100	400	**no**
2	63.974	-22.65	100	399	**no**
3	63.978	-22.65	100	400	**no**
4	63.982	-22.65	100	399	**yes**
5	63.986	-22.65	100	400	**yes**
6	63.990	-22.65	100	403	**yes**
7	63.994	-22.65	100	399	**no**
8	63.998	-22.65	100	400	**no**

Table 1: Exemplary log of parameters monitored during scenario execution in FlightGear.

The table shows that a successful object detection is on hand for logs four to six. Therefore, the predefined ODD holds for the combination of parameters on hand. Now, single parameters can be adjusted for a potential revaluation of the predefined ODD. In this case, the altitude of the intruding airplane relative to the own airplane is increased by 100 feet. First, a scenario with a new configuration of attributes needs to be generated, similar to the one depicted in Figure 4. In this case, the altitude is adjusted to reflect the definition for the new test case. Lastly, the necessary XML files are generated from the configuration model. Now, scenario execution in FlightGear and parameter monitoring can be performed. Table 2 shows the log for the second iteration of parameter monitoring.

As shown in the second table, the object detection is successful for logs three to five. The predefined ODD still holds for the combination of parameters on hand. However, the ODD can now be adjusted and phrased

[3] Munyakabera Jean Claude, 2022. flightgear_interface, Available at: https://github.com/ironmann250/flightgear_interface

Log #	lat	lon	Δ alt	Σ ktas	detect
1	63.970	-22.65	200	400	**no**
2	63.974	-22.65	200	399	**no**
3	63.978	-22.65	200	401	**yes**
4	63.982	-22.65	200	400	**yes**
5	63.986	-22.65	200	400	**yes**
6	63.990	-22.65	200	400	**no**
7	63.994	-22.65	200	400	**no**
8	63.998	-22.65	200	400	**no**

Table 2: Exemplary log of parameters monitored during scenario execution in FlightGear.

more precisely in line with the altered parameter. The ODD for the application can therefore be rephrased as follows:
"[...] within following parameter boundaries:

- Altitude of intruding airplane relative to own airplane in feet, Δ alt: -100 to 200.
- [...]"

The loop of parameter adjustment, scenario generation, execution, and monitoring can be repeated until the changes in detection results fall below some predefined value and an ODD with some desired precision has been determined. The example for ODD monitoring and parameter adjustment presented above is simplistic and, for instance, does not consider constraints. Many more parameters can be and need to be considered when defining a precise ODD for the underlying application. Also, the granularity for testing parameter boundaries of the ODD needs to be determined accurately. For instance, a higher logging frequency of parameters can be chosen, which makes the tests more precise but also increases the testing effort. Also, instead of a Boolean for the result of the object detection, the more granular confidence of the object detector from the machine learning application can be used as a metric. The framework in itself requires fine-tuning and more testing to provide the right conditions for successful iterative scenario-based testing of various systems.

The exemplary implementation of model-based scenario generation and ODD monitoring in this section follows the method presented in Figure 1. Domain-specific tools such as Cameo, XML files, and a Python application were used to build a framework for iterative scenario-based testing. The implementation can be seen as a minimal working example, demonstrating the iterative scenario-based testing method explained in Section 2. The implementation can be developed further to allow for closed-loop scenario-based testing with automated scenario generation, execution, and monitoring.

4 Conclusion and Discussion

The use of ML applications in AI-based systems such as airplanes is steadily increasing. The thorough testing of these systems is a fundamental part of their development process. Certain industries, such as aviation, impose strict requirements and constraints for the use of AI-based applications, increasing the testing efforts required to certify and use these applications. Additionally, ML applications are often considered a black box. Therefore, black box testing methods need to be put in place that are as rigorous as current testing methods for common software systems.

This work depicts a method for iteratively testing an AI-based system which performs object detection in an airplane. For this purpose, a scenario-based testing loop was developed, including the three steps of generating application-readable scenario descriptions from models, execution of these scenarios, and parameter monitoring with model parameter adjustments. In addition to generating arbitrary test cases, the presented method illustrates the approximation of boundaries for the ODD of the ML application with iterative parameter adjustments.

This method can be further optimized by connecting its components, i.e. the high-level scenario description, scenario execution, and ODD monitoring, and creating a closed loop with automated scenario generation, execution, and parameter adjustment. Additionally, test oracles that determine the success or failure of individual tests should be investigated. The granularity of test cases, i.e. only success and failure evaluation or more finely grained evaluations, is important. These findings will be investigated in future research.

Acknowledgement

The presented research is financed by and part of the project *Model-Based Systems Engineering for Artificial Intelligence* (MBSE4AI). We thank everyone who provided support, insight, and expertise that greatly assisted the research.

References

[1] EASA. *EASA Concept Paper: First usable guidance for Level 1 machine learning applications*. Tech. rep. Apr. 2021. URL: https://www.easa.europa.eu/en/downloads/126648/en.

[2] EUROCAE WG-114/SAE and G-34 Artificial Intelligence Working Group. *Artificial Intelligence in Aeronautical Systems: Statement of Concerns*. SAE International. Apr. 2021. DOI: https://doi.org/10.4271/AIR6988.

[3] On-Road Automated Driving (ORAD) Committee. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. Standard. SAE International, Apr. 2021. DOI: https://doi.org/10.4271/J3016_202104.

[4] *DO-178C - Software Considerations in Airborne Systems and Equipment Certification*. Standard. RTCA, Dec. 2011. URL: https://www.do178.org/.

[5] A. Wayne Wymore. *Model-Based Systems Engineering*. 1993. ISBN: 9780203746936. DOI: 10.1201/9780203746936.

[6] Azad M. Madni. "MBSE Testbed for Rapid, Cost-Effective Prototyping and Evaluation of System Modeling Approaches". In: *Applied Sciences* 11.5 (2021). ISSN: 2076-3417. DOI: 10.3390/app11052321.

[7] Jasper Sprockhoff et al. "Model-Based Systems Engineering for AI-Based Systems". In: *AIAA SCITECH 2023 Forum*. Jan. 2023. DOI: 10.2514/6.2023-2587.

[8] Shafagh Jafer and Umut Durak. "Tackling the Complexity of Simulation Scenario Development in Aviation". In: *Proceedings of the Symposium on Modeling and Simulation of Complexity in Intelligent, Adaptive and Autonomous Systems*. Society for Computer Simulation International, 2017. ISBN: 9781510840300.

[9] Umut Durak et al. "Scenario Development: A Model-Driven Engineering Perspective". In: *SIMULTECH 2014 - 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. SCITEPRESS – Science and Technology Publications, 2014, pp. 117–124. URL: https://elib.dlr.de/94626/.

[10] Siddhartha Gupta et al. "From Operational Scenarios to Synthetic Data: Simulation-Based Data Generation for AI-Based Airborne Systems". In: *AIAA SCITECH 2022 Forum*. Jan. 2022. DOI: 10.2514/6.2022-2103.

[11] Umut Durak et al. "Using System Entity Structures to Model the Elements of a Scenario in a Research Flight Simulator". In: *AIAA Modeling and Simulation Technologies Conference*. 2017. URL: https://elib.dlr.de/112664/.

[12] Umut Durak et al. "Computational Representation for a Simulation Scenario Definition Language". In: *2018 AIAA Modeling and Simulation Technologies Conference*. DOI: 10.2514/6.2018-1398.

[13] Bikash Chandra Karmokar et al. "Tools for Scenario Development Using System Entity Structures". In: Jan. 2019. DOI: 10.2514/6.2019-1712.

[14] Siddhartha Gupta and Umut Durak. "Behavioural Modeling for Scenario-based Testing in Aviation". In: *AIAA SCITECH 2023 Forum (not yet published)*. Jan. 2023.

[15] Hazem Torfah et al. "Learning Monitorable Operational Design Domains for Assured Autonomy". In: *Proceedings of the International Symposium on Automated Technology for Verification and Analysis (ATVA)*. Oct. 2022.

[16] Daniel Fremont et al. "Formal Scenario-Based Testing of Autonomous Vehicles: From Simulation to the Real World". In: Sept. 2020, pp. 1–8. DOI: 10.1109/ITSC45102.2020.9294368.

[17] Francis Indaheng et al. "A Scenario-Based Platform for Testing Autonomous Vehicle Behavior Prediction Models in Simulation". In: *ArXiv* abs/2110.14870 (2021).

[18] Hardi Hungar. "Scenario space exploration for establishing the safety of automated vehicles". In: *3rd China Autonomous Driving Testing Technology Innovation Conference, 2020*. Dec. 2020. URL: https://elib.dlr.de/139626/.