From Operational Design Domain to Runtime Monitoring of AI-based Aviation Systems

1st Christoph Torens
Institute of Flight Systems
German Aerospace Center (DLR)
Braunschweig, Germany
christoph.torens@dlr.de

2nd Siddhartha Gupta
Institute of Informatics
Clausthal University of Technology
Clausthal-Zellerfeld, Germany
siddhartha.gupta@tu-clausthal.de

3rd Nirmal Roy
Institute of Informatics
Clausthal University of Technology
Clausthal-Zellerfeld, Germany
nirmal.roy@tu-clausthal.de

4th Jasper Sprockhoff
Institute of Flight Systems
German Aerospace Center (DLR)
Braunschweig, Germany
jasper.sprockhoff@dlr.de

5th Umut Durak
Institute of Informatics
Clausthal University of Technology
Clausthal-Zellerfeld, Germany
umut.durak@tu-clausthal.de

***Abstract*—For the integration of autonomy and machine learning in the next generation of systems for urban air mobility and unmanned aircraft, it needs to be shown that these functions can be integrated safely. Moreover, it must be shown that these systems can operate safely with these active machine-learning functions. One prerequisite is that the machine-learning function is only utilized when it is expected to operate safely within the specified environmental conditions. This paper shows a model-based approach for the definition of the Operational Design Domain (ODD). The ODD enables the formalisation of the expected environmental conditions during the operation and the system states. With the formal model, the ODD specification can then be transformed into a specification of a runtime monitoring language called RTLola. This enables the utilization of the RTLola runtime monitoring framework to check log files against violations of the ODD and, later, supervise the ODD during operation and in flight. The goal is to automate the supervision and make it more user-friendly. This is shown in two separate use cases, where an ODD model is created and then exported and transformed into a monitoring specification. The approach is validated with a set of log files from the use cases.**

***Index Terms*—machine learning (ML), trustworthiness, operational design domain (ODD), unmanned aircraft system (UAS), innovative air mobility (IAM)**

I. Introduction

There is a huge interest in extending autonomy by utilizing Machine Learning (ML) for Unmanned Aircraft System (UAS). Only with the use of these technologies it is possible to enable object detection, i.e., humans on a landing pad, detecting and avoiding intruder traffic. For the use of ML in the next generation of UAS, it must be shown that these systems can operate safely. However, the challenge is to show that the ML system is trustworthy and can give safe results for any possible input during operation. This is nontrivial since the operation and its environment can only be controlled to a certain degree. As a result, an input might be fed into the ML system that is not trained, not expected, and where the ML system is not guaranteed to operate safely.

To define and describe the problem of inputs during operation, the Operational Design Domain (ODD) concept was introduced by EASA [1] as the operational conditions on the ML constituent level. This concept was adapted from the definition used in the automotive industry, which is based on [2]. Furthermore, the operational domain (OD) was introduced in [3] as the operational conditions at the system level. By defining these conditions, it is possible to separate the operation into several domains. The OD defines all operating conditions in which the system can operate safely. The aircraft, flight envelope, and flight performance may limit this. The ODD defines all operating conditions in which the ML constituent can safely operate. These may be limited by the ML constituent, the training data, or the validation performance. Ideally, the ODD matches the OD. In that case, the ML constituent can be utilized safely in the complete operational domain and, thus, in all operations. However, especially with the introduction of ML as a new technology into systems, the ODD can be a reduced set compared to the OD, e.g. operation only at daytime and not at night. This allows us to define specific, selected operating conditions as the ODD that the ML component can be trained to a good performance with sufficient training data. As such, an ODD specific development of the ML component can significantly reduce the efforts for training and validation. On the other hand, this introduces a new problem: the ML component can now only be utilized in this reduced set of environmental conditions of the ODD. The supervision of this valid utilization can be done by the pilot. However, this adds to the existing workload of the pilot. A better way would be to have an automated system that is specifically designed for determining whether the current situation during operation is inside or outside the ODD. The idea of this paper is to define, model, transform, and supervise the ODD, thus automating the problem of ODD determination.

To assure trustworthiness, the ODD is used in conjunction with scenarios for a safety argument of an aviation system

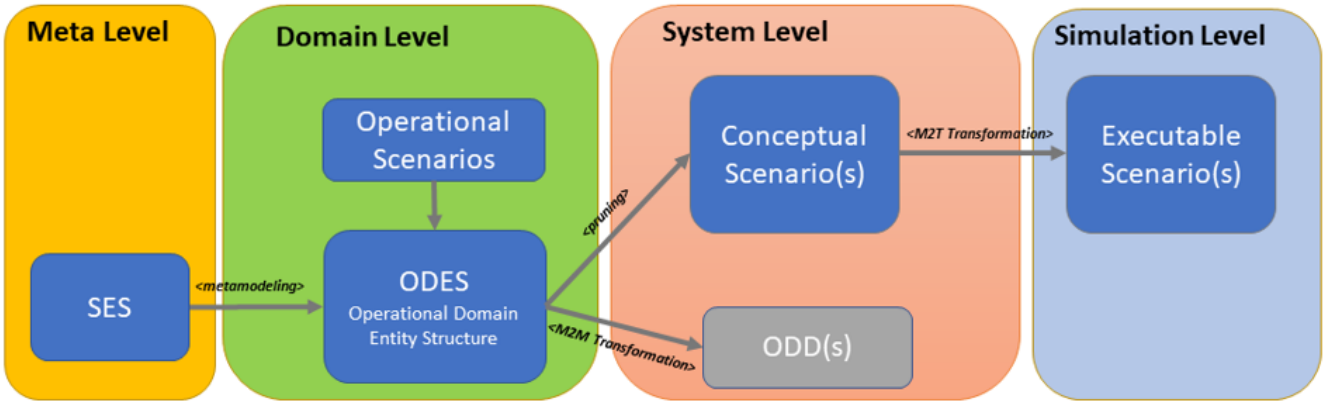

Fig. 1. Generation of Scenarios and ODD from Domain Model

under verification. A scenario describes the system with the major entities, their initial conditions, a timeline of significant events, and the environment [4]. A systematically generated set of scenarios can reduce the effort of testing a complex system by selecting essential test cases [5]. In our previous works [6] [7], we showed an approach for generating scenarios and ODD from the same domain model shown in Fig. 1. A domain model captures all the elements (with the operating parameters and relationships among each other) required for testing the AI system in the simulation environment. The operational scenarios derived from the Concept of Operations determine the elements for verification. [6] also showed how a typical domain model structure is similar to the ODD structure from BSI: PAS 1883 [8]. The results were then shown in a human-readable table.

The contributions of this paper have two aspects. The first one is to define a computational representation for both ODD and OD based on our study of EASA Concept Paper: First Usable Guidance for Level 1 and 2 Machine Learning Applications [1], ISO 34503 standard [9], and OpenODD concept paper [10]. The format should be generated in conjunction with our existing scenario-based approach. Its output will be scenarios and ODD in computational formats, with scenarios already having machine-readable format defined in [11]. The ODD gets exported in YAML. The YAML format hierarchically describes the properties and lists each parameter's boundary values. The export as a machine-readable is essential for further tooling and configuration of the aircraft and systems. The second aspect is a transformation of the text format ODD into a monitor specification. Here, we use RTLola as specification language [12]. The transformation from the ODD text format to an RTLola specification can be interpreted as a transformation from one domain-specific language (DSL) to another. The idea is to automate the supervision of ODD parameters from the model of the ODD by supporting and transforming the ODD into an RTLola monitoring specification. The challenge for the transformation is to map the hierarchical YAML format into the modular monitoring specification language and to represent ODD limitations, conditions and boundaries. This can be used to validate the simulation scenarios and safeguard the autonomous operation by explicitly stating implicit assumptions.

Fig. 2 shows the complete workflow of our contribution. The boxes within the dotted line show the modes in the ODME tool. The operational scenarios are defined in the system's Concept of Operations and help to model all the constituents imported for the system in the domain model. The domain model contains all possible scenarios and OD. Different scenarios or ODDs can be defined using the tool. The generated RTLola specification supervises the ODD to validate the scenarios and the simulation runs. The green arrow in the diagram represents the transformation to the RTLola specification. The executable RTLola module will monitor and read the specifications and verify the log files for any violation of the ODD. The goal is also to utilize the ODD Monitor in the final system. Supervising the ODD, even in operational scenarios, is necessary to safeguard any autonomy or machine learning components.

II. Related Work

A. AI Standards

The overall goal is to achieve trustworthy autonomy in the context of the safety critical aviation domain, with a specific focus on innovative air mobility. There is a lot of work on general verification of machine learning, and establishing trustworthiness. However, the question always remains on what is required for authority approval and certification of this technology. To address this question, EASA published an AI roadmap and subsequently first guidance documents for the certification of ML in aviation [1] [3]. With the automotive domain, ISO 34503 standard [9] and ASAM OpenODD concept paper [10] and the Safety of the Intended Functionality standard (SOTIF) [13] define and utilize the ODD concept to define the safety aspects, safety cases [14] and operating conditions for an automated driving task.

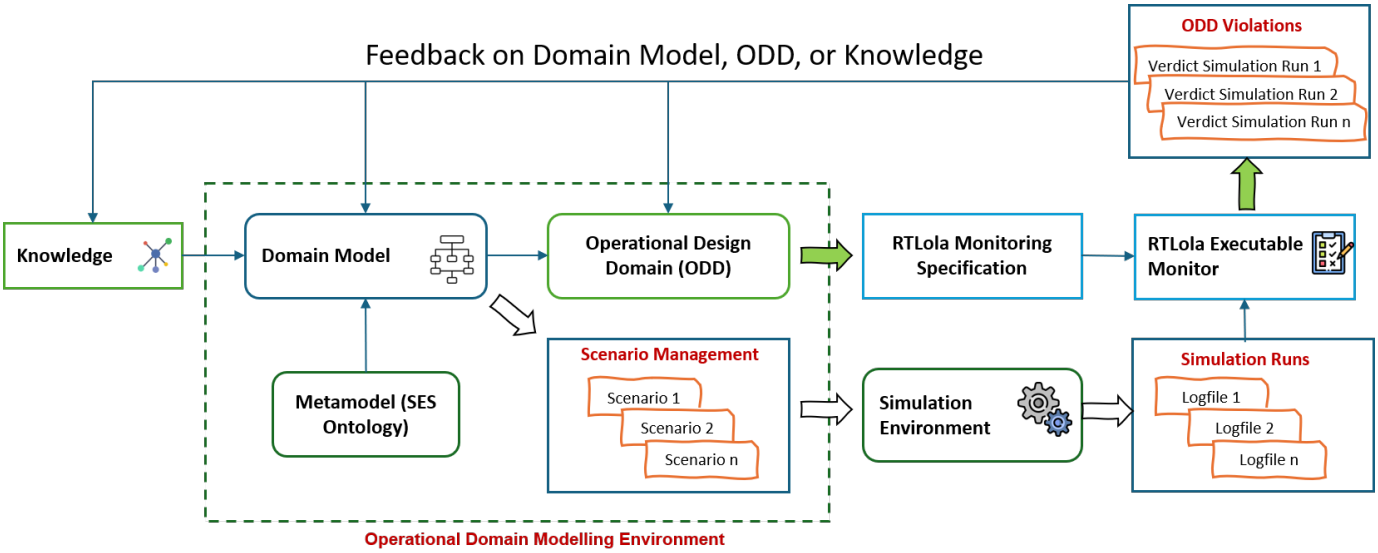

Fig. 2. The Operation Domain Modelling Environment Workflow for scenario generation and ODD definition and transformation to executable monitors

B. ODD

SAE J3016 [2] states that Operational Design Domain (ODD) is *"Operating conditions under which a given driving automation system or feature thereof is specifically designed to function, including, but not limited to, environmental, geographical, and time-of-day restrictions, and/or the requisite presence or absence of certain traffic or roadway characteristics"*. This definition of ODD is by far the most popular one used in the ODD literature, and all other definitions of ODD are a variation of the original, including the EASA concept paper: First Usable Guidance for Level 1 and 2 Machine Learning Applications [1] and the OpenODD concept paper [10]. An additional concept associated with ODD is called Operational Domain (OD).

ODD is an important concept in developing and deploying autonomous systems, ensuring they operate within well-defined and suitable conditions. ODD helps manage the expectations and limitations of autonomous systems by considering factors like weather, road types, speed limits, traffic density etc. By defining these conditions, the ODD ensures that the operation of the autonomous system is safe, appropriate, and optimal. ODD is used to define scenarios that accurately represent real-world operating conditions for testing and validation. ODD aids in safety assessment and risk analysis, helping identify potential hazards and prioritize risk mitigation strategies [15]. It also facilitates regulatory compliance by aligning with safety and operational requirements. ODD can promote effective communication and standardization among stakeholders, establishing a shared understanding of system behaviour across scenarios [16].

There is a need for a proper ODD format and associated tooling due to the complexity and diversity of autonomous systems. [10] states that the language should be machine-readable and human-readable, be able to be parsed using any text parsers, and it should enable queries for attributes of the ODD. This will help achieve all the benefits outlined previously. Using associated tooling facilitates automated processing, easier analysis, and compliance checking, improving productivity and reducing errors. The tool should clearly distinguish between ODD and OD definitions.

[17] discusses the development of the ODD language from a conceptual perspective. A structured natural language format that is human-readable is introduced. Specifically, the domain model with hierarchy and attributes is described. The attributes are further detailed regarding metrics, datatypes, and units. Additionally, qualifiers for inclusion, exclusion and conditionals or dependencies for inclusion and exclusion utilizing logical operators. With these formal model blocks, it is possible to build complex ODDs. A second publication from the authors [18] discusses that the ODD should also support a formal representation and a conversion between the two. It should be noted that while the language is highly adaptable and hierarchical, it is possible to model a large variety of complex ODDs. However, whether these ODDs can be measured or checked is a different question. For example, it is possible to model the road type "cobblestone," but it might be difficult to have a sensor distinguish this from "concrete." One of the key concepts from the second paper is the statement "DETERMINE".

```
DETERMINE light\_rain WHEN
  droplet\_size < one AND rain\_rate < 2
```

This enables the mapping of low-level sensor measurements to high-level modelling concepts.

C. Monitoring and Runtime Assurance

Monitoring is used as a technique to supervise and validate data streams. In this work, we use monitoring to automatically determine if we are inside or outside of the ODD. Furthermore, RTLola has been used in previous work to monitor, supervise and safeguard cyber-physical systems. RTLola is a real-time monitoring language and toolkit. RTLola can be used to monitor data streams of systems during flight to analyze,

safeguard and assure systems and operation [19] [20] [21]. The concept of runtime assurance can be utilized to safeguard the behavior of an UAS during flight or operation. A standard that describes a reference architecture for runtime assurance is the ASTM standard F3269 [22]. In this work the focus is on the monitoring and automated test for inside or outside of ODD.

III. ODD Use Case

we look at two ML use cases in the context of UAS and aviation. One use case is a UAS flight test performed in the context of a DLR project HorizonUAM [23]. This use case is validated with data from the original flight tests. The other use is from a simulation study performed utilizing the simulation engine FlightGear [7], and is validated with logs from simulated flights.

The HorizonUAM project researched several aspects of airtaxis, including the utilization of autonomy and specifically ML. For the project a demonstration and flight test was done with a drone to simulate flight and approach of an airtaxi at a vertiport landing spot. The use case was tested and demonstrated as part of a larger project at the DLR National Experimental Test Center for Unmanned Aircraft Systems in Cochstedt with a DJI M600 Pro drone. The flight controller is a Pixhawk 4. It uses the PX4 flight stack as autopilot software. The ML component is the detection of humans at the landing spot via the onboard camera. The person detection is done by an ML algorithm onboard the vehicle. The ODD for the ML component is defined as:

- flight altitude: 20m to 50m
- flight speed: 0 to 10 m//s
- pitch angle: -10 to +10 degree
- roll angle: -10 to +10 degree

This ODD describes limitations of the system and its operation, however the training data for the ML component was recorded with these conditions. As a result, the flight altitude submitted to these values, since higher altitudes would result in any persons appearing too small in the captured image. Any lower attitudes would result in presence appearing too big. Similar effects would apply for the flight speed with high speeds possibly resulting in lower image quality. The pitch or role angles are relevant because this is a huge impact on the image perspective. Additional properties of the ODD would be time of day and//or image brightness. However, in this work we do not focus on further properties.

For the second use case, we utilize an existing simulation set up in FlightGear. We define the use case to be the detection of intruder aircraft in order to avoid near mid-air collisions. An object detection model is trained on a synthetic image data set generated FlightGear using the approaches described in [24]. In these methods values for all parameters of the ODD are sampled from a data model. The values are used to render scenes in the simulator conforming to the ODD. Images of these scenes are then used as training data for the object detector. We have defined the following ODD for our specific application:

- flight altitude: 10000 to 15000 feet
- flight speed: 300 to 430 miles/h
- pitch angle: -10 to +10 degree
- roll angle: -10 to +10 degree

Since our model was trained exclusively on data within these ranges it is important for the aircraft to stay within the ODD during operation. Therefore, ODD monitoring during flights is needed.

IV. ODD Modeling

A. Rationale

The ODD modelling process extends the scenario modelling process established in earlier works [6], [7], [24], [25] in aviation. The scenario modelling process entails transforming stakeholder requirements into executable specifications. This transformation is done through three levels of abstraction: operational scenarios, conceptual scenarios, and executable scenarios. The operational scenarios are written in plain language based on the ConOps (a document describing the system, operation, environment, and procedures of UAS), elements and then modelled as conceptual scenarios. Conceptual scenarios are models that represent the elements of the scenarios, their relationships with each other, and their attributes with valid value ranges. The models can be reduced to characteristics of individual scenarios, which can be transformed into a machine-readable format that serves as executable scenarios. The EASA concept paper [3] uses the operational scenarios defined through ConOps to describe the system's OD. The OD should contain operating parameters classified as different types with permissible value ranges. An ODD is technically similar to the OD but has smaller sub-ranges. The OD is defined at the system level, and ODD is defined at the AI/ML constituent level.

B. ODME Tool

The ODME tool is a GUI-based tool developed to utilise scenarios and ODD modelling techniques for practical aviation research. It was initially developed in 2019 to realise the early stages of scenario modelling research by transforming operational scenarios into models, applying model transformations and exporting the scenarios in machine-executable form in XML format. The tool has been recently upgraded to include ODD modelling in conjunction with scenario modelling.

The tool consists of various modes correlating to various functionalities supported by it. The two modes relevant to the paper are the domain modelling mode and the Operational Design Domain mode. Domain modelling is the point of entry for any new project on the tool. It allows the domain expert to map relevant elements hierarchically, exhibiting the relationships between the components and their associated attributes. Technically, the attributes and their defined ranges in the domain model correspond to the OD. The Operational Design Domain Mode can automatically extract the relevant details from the domain model and represent them in a human-readable table as the OD. The user can add comments to the various values but can only edit the domain model to reflect the

changes in the OD. The mode also supports an ODD manager mode, which allows the user to define multiple ODDs within one OD. An ODD is defined by selecting a subset of the OD's permitted range. The ODD can be exported as XML or YAML form by the ODD manager, facilitating further computation for monitoring by specific programs.

C. Use Case Model

Fig. 3 showcases the model used for this research paper. It is defined by using generalised characteristics of ODDs as described by the ISO 34503 standard [9] catering to the specific needs of our use case. The ODD comprises three main sections – the *Environment*, *Static_Entities* and *Dynamic_Entities*. The environment has four components – *Weather*, *Particulates*, *Connectivity* and *Illumination*. The *Weather* entity defines the *Air_Temperature*, the *Wind*, *Rainfall* and *Snowfall*. Various strengths define the last three *Weather* conditions, e.g. *Rainfall* could range from *No* to *Cloudburst*, with each *Rainfall* type characterised by specific values of properties. The other aspect of the *Environment* is *Particulates*, defined by different sizes, intensities, and types. *Connectivity* describes the communication type and the positioning system used in the airborne system. *Illumination* defines the luminosity, the sun's position and the *Cloudiness* in *Weather*, ranging from *No* to *Overcast*.

Besides the *Environment*, the other two model components include the static and dynamic entities. For our use case, the *Static_Entities* consist of six elements. The first element is the *flight_Area*, the second is the *Landing_Pads*, and the third is *Geofencing*. The fourth and fifth elements are the *Trees* and *Buildings* in the system's surroundings, with the last element being specific parameters for the *Airspace*. The static elements cater, in general, to different types of airborne systems. The *Dynamic_Entities* describe the *Subject_Drone* and other systems referred to as *Intruder_Drone*. The model also defines the state variables and the payload of the subject aircraft. Finally, specific for the node *Drone_State* Fig. 4 shows a more detailed view of the editor function that enables to model the specific properties on the top right of the screenshot. This is where the boundary values of the ODD are modeled, as property of the leaf node.

V. ODD Transformation

A. YAML Export and Transformation to RTlola

The ODD from the Domain Model created using the ODME tool can be extracted and exported as a YAML file. The YAML output is hierarchical in structure, the structure is taken from the domain model structure that is shown in Fig. 3. For the ODD it is important that this output also includes the attributes and boundary values for each leaf node of the domain model. The modeling of these values for the leaf node *Drone_State* is shown in Fig. 4. The resulting YAML file is shown in Fig. V-A. The specification shows the modeling of simple boundary values, such as lower bound and upper bound for the altitude. Furthermore, it is possible to define additional layers of boundary values, such as warning bounds.

In the next step, the extracted YAML file is then transformed into RTLola format, Fig. 8. The transformation is written in Python using the library textX, a meta-language for building Domain-Specific Languages (DSLs). The tool uses meta-models that represent the structure of the YAML specifications. The created textX metamodel file is used as a blueprint for parsing YAML files and generating the model.

For each node, the RTLola input declaration is generated ("input") that defines the variable name and its corresponding data type. The function handles the translation of limits (lower bound, upper bound limits in ODD) specified in the YAML file into RTLola trigger specifications. In the RTLola specification language, a trigger defines the condition in which the monitoring algorithm determines a violation of the property. These triggers and expressions capture the constraints and operational specifications stated in the ODD, ensuring that the output RTLola code reflects the intended operational limit of the system. For each boundary value a trigger specification is generated, Fig. 8.

B. ODD Mapping to Log File

An ODD, such as the model in Fig. 3 describes parameters in an abstract manner, e.g., *Subject_Drone.Drone_State.Flight_Altitude*. In the first use case, a PX4 flight stack was used with corresponding log files. A practical problem arises when the available log files do not represent the abstract ODD 1-to-1. In this case, a mapping from the ODD to the log file must be done. For the HorizonUAM use case log files, we need four parameters as the ODD. The PX4 flight software produces dozens of different log files in CSV format. The corresponding parameters from the ODD are present in at least two different log files.

- vehicle_attitude_setpoint_0.csv and
- vehicle_gps_position_0.csv

The transformation from YAML to RTlola must generate a valid specification file. And for the RTlola framework to be able to read and parses the information from the trace, in this case the log file, it is necessary to declare all log file parameters as "input" variables. This is automated by a script that reads the log file and adds "input" declarations into the RTlola specification file for each parameter.

Finally, it is necessary to map the abstract ODD parameter to the specific log file parameter. This part of the mapping currently has to be done manually, since it is necessary to identify the parameter from the log file that matches the ODD parameter. We need to have a mapping for each of these cases and for each log file. Fig. 6 shows the mapping for vehicle_gps_position_0.csv, where the left part of the mapping is the parameter name in the ODD model (Fig. 3) and the right side is the parameter name of the log file (*"alt"*).

Fig. 7 shows the mapping for the FlightGear use case. In this case the log file contains all 3 parameters from the abstract ODD.

Specifying this mapping from the abstract parameter to the specific parameter in the log file enables the generation of

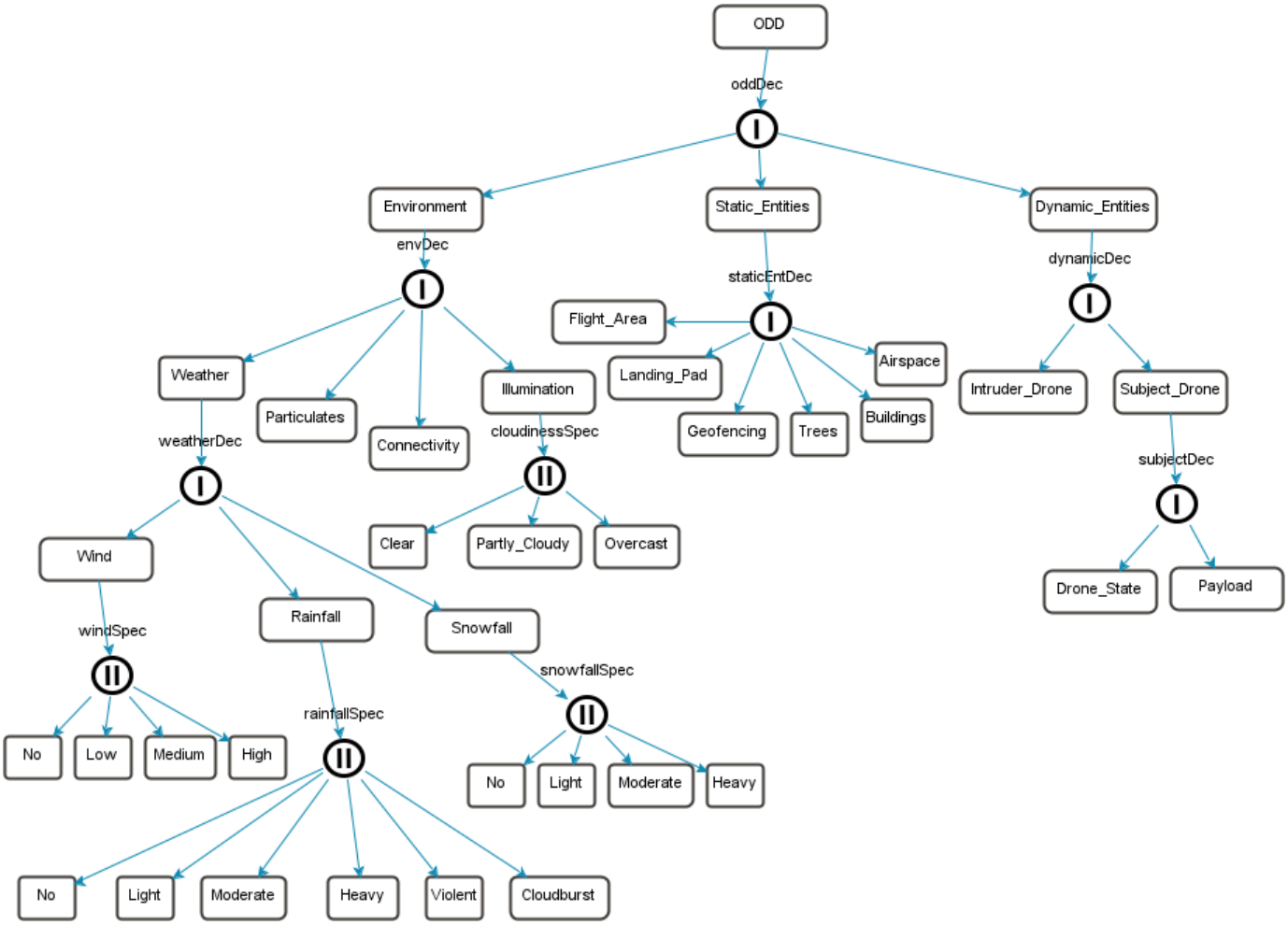

Fig. 3. The Use Case Domain Model based on ISO34503 Standard

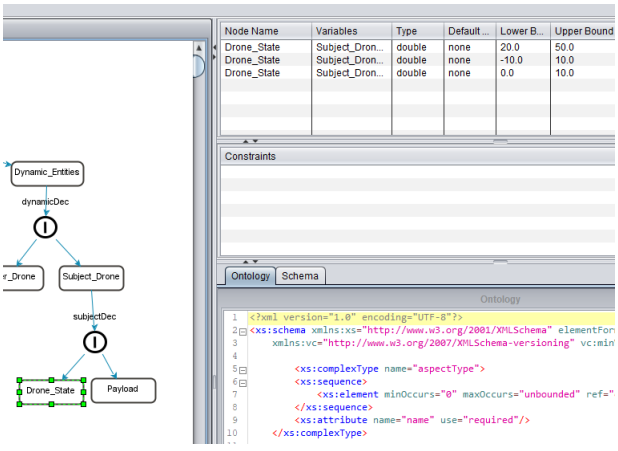

Fig. 4. Attributes and their defined ranges

```
Subject_Drone:
  Drone_State:
    -Altitude_m:
       type: double
       lower bound: 20.0
       lower warning bound: 25.0
       upper bound: 50.0
       upper warning bound: 45.0
       function: NONE

    -Drone_speed_ms:
       type: double
       lower bound: 0.0
       lower warning bound:
       upper bound: 10.0
       upper warning bound:
       function: NONE
    ...
```

Fig. 5. YAML Output

monitoring specification from the ODD parameter boundaries. However, there is an additional challenge. In the ODD model, the boundary values are again described on an abstract level. For example, the flight altitude is specified with 20m to 50m. In the PX4 log file, the altitude is specified in millimeter above NN. So in this case, the unit has to be adapted, as well as the relative offset of the altitude about sea level. This problem

```
Subject_Drone.Drone_State.Altitude_m:alt
Subject_Drone.Drone_State.Speed_ms:vel_m_s
```

Fig. 6. Mapping specification for the HorizonUAM use case

```
Subject_Drone.Drone_State.Altitude_m:
    Altitude_Aircraft
Subject_Drone.Drone_State.Speed_ms:
    Airspeed_Aircraft
Subject_Drone.Drone_State.Pitch:
    Pitch_Aircraft
Subject_Drone.Drone_State.Roll:
    Roll_Aircraft
```

Fig. 7. Mapping specification for the FlightGear use case

can be addressed in 2 ways. First, it is possible to manually convert the unit and add the corresponding offset manually, e.g. via an additional mapping file. As a result, the generated trigger specification must be adapted to the boundary value plus the relative offset.

Secondly, it would be possible to initialize the altitude with the first entry from the log file as relative zero altitude. All subsequent height measurements in the log file would be treated as offsets to that altitude automatically. This requires an adapted specification of the trigger in the RTlola output. The trigger needs to calculate the current altitude relative to the altitude from the initial start of the log file. However, RTlola is a very powerful specification language and thus it is very easy to do this.

```
input time_utc_usec: Float64
input lat: Float64
input lon: Float64
input alt: Float64
trigger alt < 200.0 "alt lower bound exceeded"
trigger alt > 230.0 "alt upper bound exceeded"
trigger alt < 205.0 "alt lower warning bound"
trigger alt > 225.0 "alt upper warning bound"
input alt_ellipsoid: Float64
input s_variance_m_s: Float64
...
```

Fig. 8. RTLola Output

VI. ODD Monitoring

For runtime monitoring, we use RTlola as the framework. RTlola is a framework that is developed by CISPA [1]. The benefit of RTlola is that it is a formal language that enables modular high level specifications of monitoring properties. An interpreter for the language exists, as well as a compiler for C++ or Rust. This means that it can be utilized to analyze log files, as in this example. Furthermore, it can also be utilized as an online monitoring integrated into the onboard system. As such, it can be utilized as a monitor for a run-time assurance architecture as standardized with RTCA F3269.

[1]https://finkbeiner.groups.cispa.de/tools/rtlola/

A. ODD Validation using Test Flight Logs

For this validation, we used test flight logs from our flight tests for the HorizonUAM project. During the project, we conducted several flight tests with our drone. For convenience, the logs are validated using the RTlola playground via a web interface [2]. A screenshot of the web interface is shown in Fig. 9. The web interface shows a text input for the specification, a text input for the log data, which is called trace, and then a text output console with output from RTlola on the trace. Furthermore, in the screenshot it can be identified that the trigger for altitude is listed in the console. This means, that the monitor found a violation of the altitude property boundary within the trace.

B. ODD Validation using Simulation

We execute flight simulations in FlightGear to create data logs to test our approach. Our simulations start by flying a 737-200 in the air within ODD boundaries. Utilizing a Python script, we change the target values of the aircraft's autopilot. By changing target speed, target altitude, pitch or heading, we create artificial violations of the defined ODD boundaries. We use the internal logging function provided by FlightGear to generate the data logs. We defined a log configuration to enter the aircraft's position, orientation, and speed each 200 ms into a .csv output file. Using the described approach, starting from the domain model, then YAML export, transformation to RTlola file, and mapping file, it was possible to validate the monitoring with the RTlola playground.

VII. ODD challenges

Several challenges exist with this approach. This approach assumes that the parameters of the ODD can be directly measured as sensor inputs or can be broken down and mapped to sensor inputs. However, this might not always be the case. For example, there might be limitations on high-level concepts, such as cloudiness, urban environment, or other aspects, that are not directly measurable. For weather information, it might be necessary to rely on external data rather than attempting to identify the degree of rain at the moment. In this case, the ODD develops into a complex hierarchical model that relies on a multitude of sensors, inputs, and online information. Existing gaps between ODD conceptual parameters and sensor inputs might be modelled, similar to the ODD framework from [18] with "DETERMINE" statements.

Another problem arises if sensory inputs cannot determine any aspect of the ODD. Either, if there is no sensor of the required type. In that case, the ODD cannot be determined, and thus, the ML component cannot be safely used. Furthermore, the determination of the ODD is dependent on the sensor inputs and cannot be established if the sensor fails. These conditions have to be checked as constraints for an automated utilization of the ODD concept.

[2]https://rtlola.cispa.de/playground/

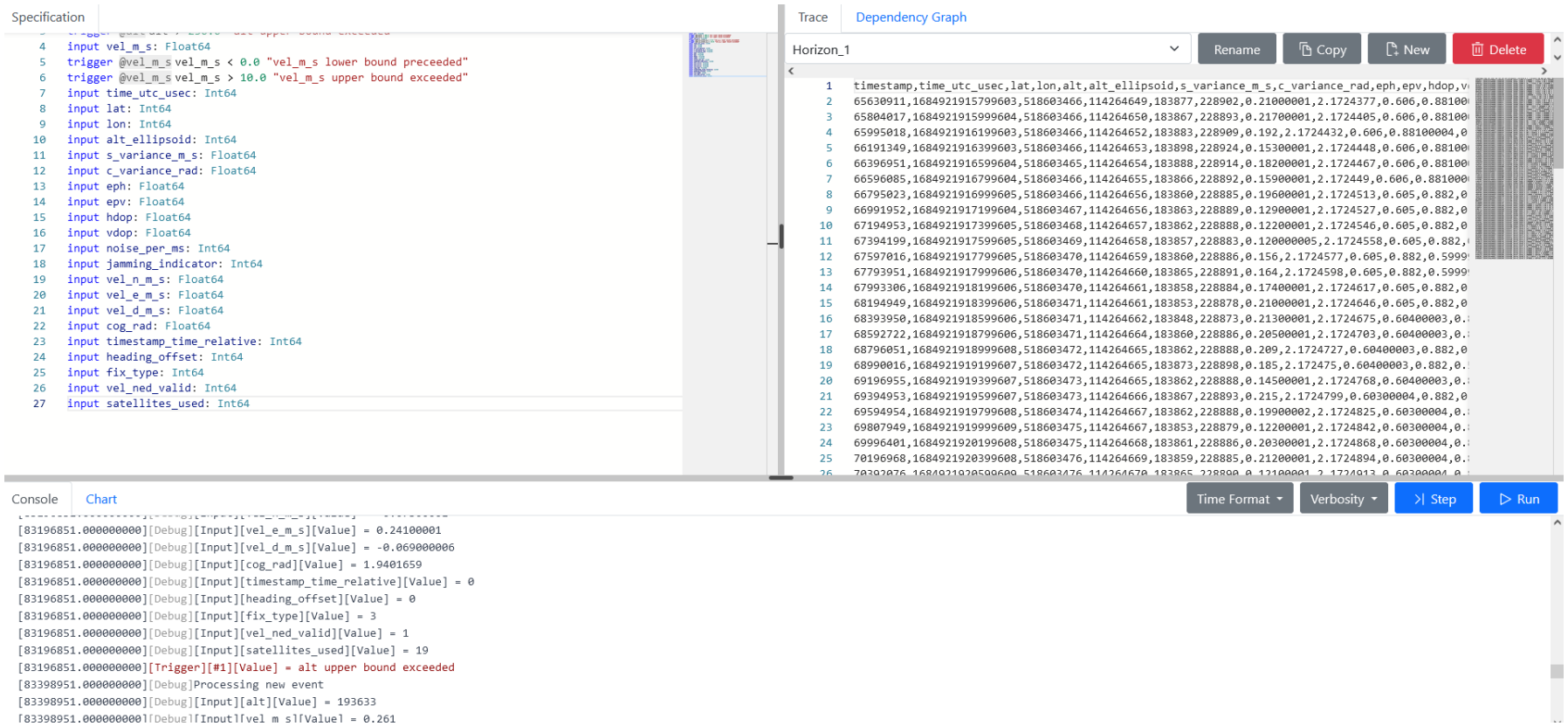

Fig. 9. Validation of ODD monitoring using RTlola playground

VIII. Conclusion

This paper discussed the modeling and definition of a computational representation for ODD and OD. Additionally, a transformation of ODD to a runtime monitoring specification was shown. It was shown which practical problems arise, when utilizing an ODD description with actual log files and how these can be solved. The transformed runtime monitoring specifications were validated using log files from two separate use cases. The mapping file from the ODD to the log file currently has to be created manually. Future work will research further aspects of automation, e.g., automated mapping using standardized log files or tagged log files.

References

[1] EASA, "Concept Paper First usable guidance for Level 1 and 2 machine learning applications," 2023.

[2] SAE, "J3016_202104 - taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," https://www.sae.org/standards/content/j3016_202104/, 2021.

[3] EASA, "Concept paper: guidance for level 1 & 2 machine learning applicationsproposed issue 02," 2024.

[4] O. Topçu, U. Durak, H. Oğuztüzün, and L. Yilmaz, *Distributed simulation: A model driven engineering approach*. Springer, 2016.

[5] J. Ma, X. Che, Y. Li, and E. M.-K. Lai, "Traffic scenarios for automated vehicle testing: A review of description languages and systems," *Machines*, vol. 9, no. 12, p. 342, 2021.

[6] S. Gupta, U. Durak, O. Ellis, and C. Torens, "From operational scenarios to synthetic data: Simulation-based data generation for ai-based airborne systems," in *AIAA SCITECH 2022 Forum*, 2022, p. 2103.

[7] S. Gupta and U. Durak, "Operational domain metamodel for testing ai systems in aviation," in *AIAA SCITECH 2023 Forum*, 2023, p. 2589.

[8] BSI, "Pas 1883:2020 - operational design domain (odd) taxonomy for an automated driving system (ads)," https://shop.bsigroup.com/products/operational-design-domain-odd-taxonomy-for-an-automated-driving-system-ads-specification, 2021.

[9] I. S. 33, "Iso 34503:2023 road vehicles, test scenarios for automated driving systems specification for operational design domain," 2023.

[10] ASAM, "Asam openodd: Concept paper," 2021.

[11] U. Durak, S. Jafer, R. Wittman, S. Mittal, S. Hartmann, and B. P. Zeigler, "Computational representation for a simulation scenario definition language," in *2018 AIAA Modeling and Simulation Technologies Conference*, 2018, p. 1398.

[12] J. Baumeister, B. Finkbeiner, S. Schirmer, M. Schwenger, and C. Torens, "RTLola Cleared for Take-Off: Monitoring Autonomous Aircraft," in *Computer Aided Verification*, S. K. Lahiri and C. Wang, Eds. Cham: Springer International Publishing, 2020, pp. 28–39.

[13] ISO, "Pas 21448 road vehicles safety of the intended functionality sotif," 2019.

[14] "The UL 4600 Guidebook." [Online]. Available: https://safeautonomy.blogspot.com/2022/11/blog-post.html

[15] A. Schnellbach and G. Griessnig, "Development of the iso 21448," in *Systems, Software and Services Process Improvement: 26th European Conference, EuroSPI 2019, Edinburgh, UK, September 18–20, 2019, Proceedings 26*. Springer, 2019, pp. 585–593.

[16] M. Gyllenhammar, R. Johansson, F. Warg, D. Chen, H.-M. Heyn, M. Sanfridson, J. Söderberg, A. Thorsén, and S. Ursing, "Towards an operational design domain that supports the safety argumentation of an automated driving system," in *10th European Congress on Embedded Real Time Systems (ERTS 2020)*, 2020, pp. 1–10.

[17] P. Irvine, X. Zhang, S. Khastgir, E. Schwalb, and P. Jennings, "A Two-Level Abstraction ODD Definition Language: Part I," in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct. 2021, pp. 2614–2621, iSSN: 2577-1655. [Online]. Available: https://ieeexplore.ieee.org/document/9658751

[18] E. Schwalb, P. Irvine, X. Zhang, S. Khastgir, and P. Jennings, "A Two-Level Abstraction ODD Definition Language: Part II," in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct. 2021, pp. 1669–1676, iSSN: 2577-1655. [Online]. Available: https://ieeexplore.ieee.org/document/9658812

[19] S. Schirmer, C. Torens, J. C. Dauer, J. Baumeister, B. Finkbeiner, and K. Y. Rozier, "A hierarchy of monitoring properties for autonomous systems," in *AIAA SciTech Forum*, 2023, pp. 1–13. [Online]. Available: https://elib.dlr.de/193868/

[20] C. Torens, F.-M. Adolf, P. Faymonville, and S. Schirmer, "Towards intelligent system health management using runtime monitoring," in *AIAA Infotech @ Aerospace, AIAA SciTech Forum*, Januar 2017, pp. 1–11. [Online]. Available: https://elib.dlr.de/111412/

[21] C. Torens, P. Nagarajan, S. Schirmer, J. Dauer, J. E. Baumeister, F. Kohn, B. Finkbeiner, G. Manfredi, and F. Löhr, *Certification Aspects of Runtime Assurance for Urban Air Mobility*. AIAA SCITECH 2024 Forum, 2024, ch. Session: Certification, Verification and Artificial Intelligence, pp. 1–17. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2024-1464

[22] P. Nagarajan, S. K. Kannan, C. Torens, M. E. Vukas, and G. F. Wilber, "ASTM F3269 - An Industry Standard on Run Time Assurance for Aircraft Systems," in *AIAA Scitech 2021 Forum*. VIRTUAL EVENT: American Institute of Aeronautics and Astronautics, Inc., Jan. 2021. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.2021-0525

[23] C. Torens, F. Juenger, S. Schirmer, S. Schopferer, D. Zhukov, and J. C. Dauer, *Ensuring Safety of Machine Learning Components Using Operational Design Domain*. AIAA SCITECH 2023 Forum, 2023, ch. Session: Software Platforms and Applications, pp. 1–14. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2023-1124

[24] J. Sprockhoff, S. Gupta, U. Durak, and T. Krueger, "Scenario-based synthetic data generation for an ai-based system using a flight simulator," in *AIAA SCITECH 2024 Forum*, 2024, p. 1462.

[25] J. Sprockhoff, B. Lukic, V. Janson, A. Ahlbrecht, U. Durak, S. Gupta, and T. Krueger, "Model-based systems engineering for ai-based systems," in *AIAA SCITECH 2023 Forum*, 2023, p. 2587.