

Article

Certification-Oriented Requirements and Model Verification Methodology for UAV Systems

Jan M. Kelner ^{1,*} , Sławomir Klimaszewski ² , Sławomir Brzózka ³  and Wojciech Stecz ^{4,*} 

¹ Institute of Communications Systems, Faculty of Electronics, Military University of Technology, 00-980 Warsaw, Poland

² Air Force Institute of Technology, 01-494 Warsaw, Poland; slawomir.klimaszewski@itwl.pl

³ PIT-RADWAR S.A., 05-054 Warsaw, Poland; slawomir.brzozka@pitradwar.com

⁴ Institute of Computer Sciences, Cardinal Stefan Wyszyński University, 01-938 Warsaw, Poland

* Correspondence: jan.kelner@wat.edu.pl (J.M.K.); w.stecz@uksw.edu.pl (W.S.)

Highlights

What are the main findings?

- A certification-oriented methodology integrating requirements management, functional hazard analysis (FHA), and formal model verification for unmanned aerial vehicle (UAV) systems was developed.
- Unified Modeling Language (UML) state-machine models verified with linear temporal logic (LTL), computation tree logic (CTL), and the NuSMV model checker enabled the detection of inconsistencies in safety-critical UAV operational scenarios.

What is the implication of the main finding?

- The proposed approach improves traceability between certification requirements, UAV operational scenarios, and safety-related system functions.
- Formal verification of UAV operational models can support the development of certification-ready and safety-critical UAV systems by enabling early detection of unsafe or incomplete behaviors.

Abstract

The paper presents a certification-oriented methodology for requirements management and model verification in the design of unmanned aerial vehicle (UAV) systems. The proposed approach addresses challenges in certifying safety-critical UAV platforms whose architectures include custom-developed hardware and software components. The methodology integrates requirements engineering, functional hazard analysis (FHA), and formal model verification into a unified design and validation workflow. Methods for categorizing, prioritizing, and decomposing system requirements are presented to support the development of reliable UAV software and operational procedures. A systematic approach for mapping certification requirements and FHA safety functions to UAV operational scenarios and system use cases is introduced, enabling traceability between safety requirements, system behavior, and verification artifacts. The proposed framework employs Unified Modeling Language (UML) state-machine models, validated using linear temporal logic (LTL) and computation tree logic (CTL). Model verification is performed using the NuSMV symbolic model checker to assess the correctness, completeness, consistency, and safety properties of operational scenarios. A practical case study concerning UAV handover between ground control stations (GCSs) demonstrates the applicability of the proposed method. The analysis identifies inconsistencies in the operational model and shows how formal verification supports the early detection of unsafe or incomplete system behaviors.



Academic Editor: Octavio Garcia-Salazar

Received: 26 May 2026

Revised: 16 July 2026

Accepted: 21 July 2026

Published: 22 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

The presented approach supports the development of certification-ready UAV systems by improving requirements traceability, reducing verification ambiguities, and facilitating the validation of safety-critical flight-control procedures.

Keywords: unmanned aerial vehicle (UAV) certification; requirements management; airworthiness; functional hazard analysis (FHA); safety-critical systems; model-based design; formal verification; linear temporal logic (LTL); computation tree logic (CTL); NuSMV

1. Introduction

Unmanned aerial vehicle (UAV) platforms are among the fastest-growing sectors of modern aerospace technology. Currently, the development of unmanned aircraft capabilities has far outstripped the process of creating a coherent legal and certification framework governing their operation in civilian airspace, particularly in populated areas and near critical infrastructure. Designing larger UAV platforms using commercially available components remains challenging because such components are rarely accompanied by documentation supporting certification and system integration. Although the concept of open UAV architectures has gained popularity, it does not guarantee that individual components are developed in accordance with certification requirements. Furthermore, UAV certification and airworthiness regulations are still evolving, and currently there is no fully coherent certification framework for unmanned aerial systems (UASs) within the European Union (EU).

1.1. Background and Motivation

Kovacova et al. [1] examined the state of unmanned aerial vehicle (UAV) operations and explored future operational concepts, focusing specifically on safety risk management. Their research addresses the existing regulatory framework concerning safety investigations and procedures for UAV missions. Additionally, they reference the classification of UAV categories established by the European Union Aviation Safety Agency (EASA) in Regulation 2019/945. The authors emphasize that UAV platforms operating in all categories (open, specific, and certified) must comply with airspace operational regulations. Abdulov [2] proposed a concept for UAV safety management based on a cyclic process that includes risk identification, risk assessment, threat detection, information exchange with air traffic control authorities, decision-making, and post-operation analysis. However, the presented model does not explicitly conform to formal certification methodologies for safety-critical airborne systems. Henderson [3] investigated the New Zealand UAV regulatory framework from the perspective of UAV users, identifying areas where regulations could be relaxed or strengthened. Given its alignment with the International Civil Aviation Organization (ICAO) regulations, the proposed framework serves as an important reference point for future UAV certification processes.

Certification becomes particularly challenging when UAV platforms incorporate newly developed research and development (R&D) components. In such cases, the certification process must verify not only the correctness of individual hardware and software components but also the safety and reliability of the integrated system under operational conditions. This includes validation of onboard software, flight control procedures, communication mechanisms, and ground control station (GCS) functionalities. Certification authorities assess the UAV system as an integrated whole rather than as a collection of individual subsystems. Consequently, certification requires comprehensive verification of operational scenarios, including abnormal and emergency conditions.

Figure 1 presents the overall certification-oriented workflow proposed in this paper. The framework integrates requirements management, functional hazard analysis (FHA), operational scenario modeling, and formal verification techniques into a unified UAV system validation process [4,5].

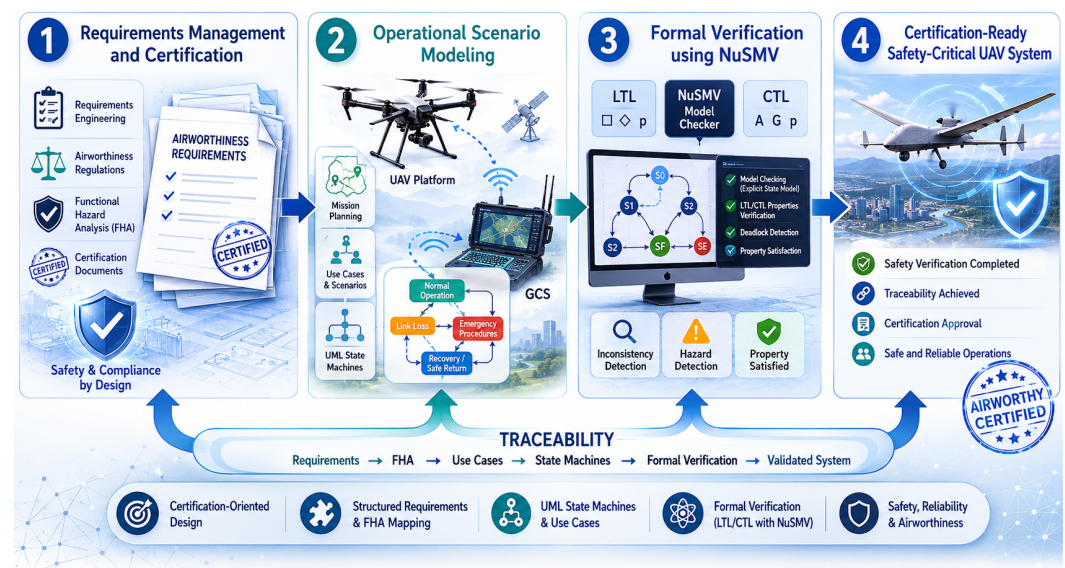


Figure 1. Certification-oriented workflow for UAV systems integrating requirements management, FHA, operational scenario modeling, and formal verification using UML state machines, LTL, CTL, and the NuSMV symbolic model checker. The graphical layout was prepared with the assistance of generative AI based on the authors' design and subsequently reviewed, edited, and approved by the authors.

Modern UAV architectures are highly complex (see [6]). A detailed discussion of UAV platform architectures is beyond the scope of this paper; interested readers are referred to [6], which presents an example MINI-class UAV platform together with aerodynamic and mission-planning models. For the purposes of this paper, it is sufficient to note that a UAV system typically consists of two major components: the airborne platform and the GCS. The airborne platform typically includes a flight control computer (FCC), a mission computer (MC), and additional onboard computing units, such as a data computer (DC) responsible for mission-data processing. It also incorporates the wireless management system (WMS) and communication subsystem. As standard, each UAV is equipped with a navigation system, which typically consists of a global navigation satellite system (GNSS) receiver (e.g., Global Positioning System (GPS)) and an inertial navigation system (INS) unit. The FCC is responsible for flight stabilization and platform dynamics, typically using embedded proportional-integral-derivative (PID) controllers. The MC supervises mission execution and emergency-response procedures. Depending on the UAV operational class, missions can be performed autonomously without continuous communication with the GCS. However, fully autonomous operations in urban environments remain highly restricted by current regulations. In larger drones, the flight management system (FMS) is a set of technologically advanced computers.

The proposed methodology is applied to a typical UAV system architecture consisting of airborne and ground-control subsystems. Figure 2 presents the general structure of the analyzed UAV platform, including the FCC, MC, onboard communication subsystems, and safety-critical flight-management components.

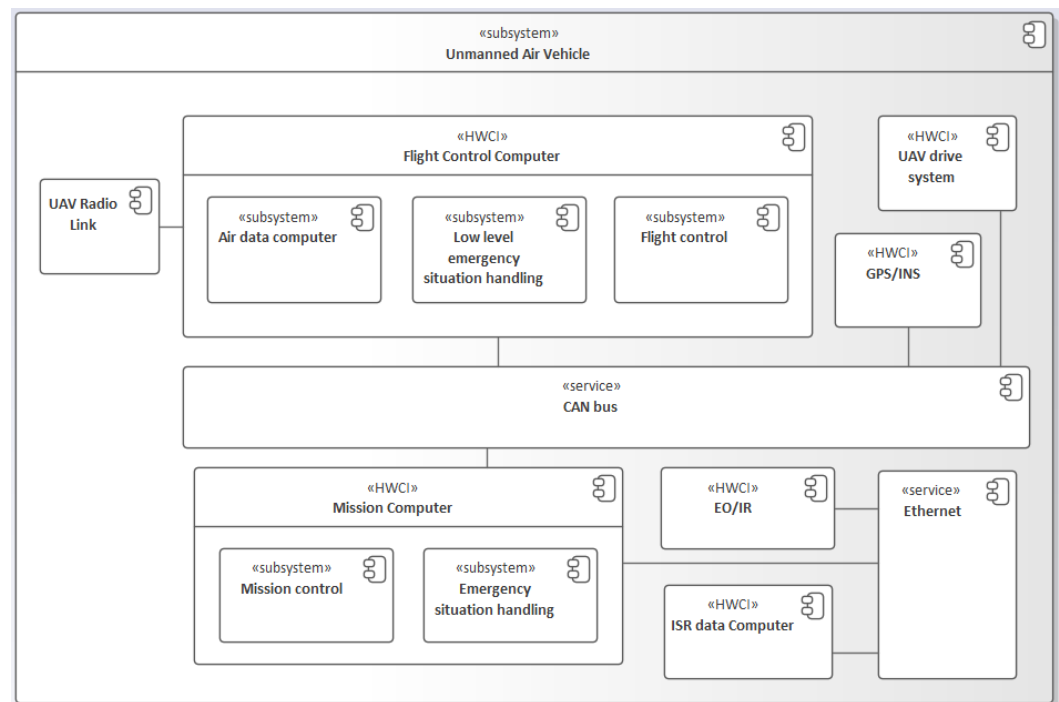


Figure 2. General architecture of a UAV system including the FCC and MC, which together implement safety-critical flight-management functions subject to certification.

Particular attention is paid during certification of the FCC and MC components because they directly affect flight safety. Certification authorities require evidence that catastrophic failure conditions are extremely improbable, in accordance with established airworthiness regulations [7]. For this purpose, FHA is commonly employed to identify safety-critical system functions and associated failure conditions.

From the certification authority's perspective, system testing is inherently multi-level and applies to the entire UAV ecosystem rather than isolated components. The primary objective of certification is not only to demonstrate functional correctness but also to verify the system's resilience to identified operational hazards that may occur during mission preparation, flight execution, communication loss, adverse weather conditions, or landing procedures. Evaluating components developed within R&D projects is particularly challenging because their operational reliability must be demonstrated despite limited prior deployment experience. Certification authorities also define their level of involvement (LOI) throughout the certification process and evaluate the UAV manufacturer's development, testing, and production procedures.

This paper addresses problems related to UAV certification from the perspective of requirements management and formal verification of safety-related system functions. Two major groups of certification artifacts are considered. The first group includes formal certification requirements that define the constraints of the entire system and its software and hardware components. The second group consists of FHA documents that decompose safety-critical system functions and identify potential failure conditions associated with them.

System designers must demonstrate that the UAV platform is resilient against identified threats and responds appropriately, even when communication with the GCS is lost. To achieve this objective, UAV subsystem behavior is modeled using UML state machines. These models describe normal system operation while explicitly incorporating emergency situations and fault-response mechanisms. To support the software validation and verification process, this paper proposes a model validation methodology based on state machines and modal logic (LTL and CTL). In the first step, UML state-machine models

are transformed into the input language accepted by the NuSMV model checker [8–11]. In the second step, the resulting models are formally verified using modal logic (LTL and CTL) within the NuSMV environment.

These techniques employ linear temporal logic (LTL) and computation tree logic (CTL), enabling the detection of inconsistencies in behavioral models [12]. The UAV operational procedures are modeled using state machines and formally verified with the NuSMV symbolic model checker [8,9]. Formal validation ensures model consistency and verifies that all operational paths terminate in properly defined states before detailed implementation and testing activities begin.

Modal logic has been widely applied for the formal verification of behavioral models. Selected representative studies are briefly reviewed below.

Neyzov et al. [13] present a method for developing and verifying control logic programs in which the behavior of the program's execution environment is modeled as constraints on the behavior of its input signals. This approach eliminates the requirement for explicit modeling of the processes implemented in the environment. The developed model operates in a closed-loop "program-environment" model, allowing for simplified modeling and a reduced state space.

Rashid et al. [14] describe a formal way to model and verify the reference implementation of epoch processing in the Beacon Chain. The LTL model aims to ensure that the epoch mechanism in the Beacon Chain is correct and minimizes the risk of errors. The authors argue that model verification techniques based on formal methods ensure the required correct operation of safety-critical systems.

A study by Zhou et al. [15] proposed a formal verification method for investigating the operation algorithms of autonomous ships. Timed automata were adopted as the formal language, and model verification based on the UPPAAL verification tool was used for automatic liveness verification. UPPAAL is based on CTL logic, or more precisely, on its simplified version. UPPAAL does not use full CTL, but rather a set of path and state queries that test models in real time.

Nemouchi et al. [16] present a formal modeling framework that independently models an electric vehicle model using multi-parameter transition systems and the dynamic evolution of the environment in which these vehicles operate. This framework was validated in two representative use cases: a dynamic urban delivery zone and an autonomous electric delivery vehicle. Safety and survivability properties are specified in CTL and verified using a model validation tool.

Ray et al.'s paper [17] presents Hardcaml, a domain-oriented programming language (DSL) for embedded hardware design, implemented in the OCaml programming language. Hardcaml enables rapid circuit validation, reducing the risk of user-input errors and miswiring. The language is integrated with NuSMV for LTL assertion proofing.

Pakonen et al. [18] explain how formal properties related to failures can be determined using LTL and CTL. They also demonstrate how model verification can be used to verify the logic designs of measurement and control applications based on function block diagrams. This method has been used for over a decade in projects in the Finnish nuclear industry.

Fritzsche et al. [19] present their experience with model verification for vehicle control systems. They compare various model verification techniques. Detailed tests are performed using the NuSMV model verification tool. They describe an architecture composed of dozens of state variables. To handle large-scale models, they propose a model verification procedure for use on multi-core processors.

1.2. Research Gap, Novelty, and Contribution

Although requirements engineering, FHA, UML state-machine modeling, and formal verification using LTL/CTL and NuSMV have each been extensively investigated in the literature, they are typically applied as independent engineering activities or within isolated stages of the system-development lifecycle.

Existing studies usually focus on one of the following aspects:

- Certification regulations and safety assessment,
- Formal verification techniques,
- Model transformation,
- Software testing,
- Requirements engineering.

To the best of the authors' knowledge, no previous work has presented a unified certification-oriented methodology that explicitly integrates certification requirements, FHA safety functions, operational scenarios, UML behavioral models, model transformation, and formal verification into a single engineering workflow. Consequently, end-to-end traceability between these development artifacts remains insufficiently addressed in the current UAV certification literature. The proposed methodology addresses this gap by integrating certification requirements, FHA safety functions, operational scenarios, model transformation, and formal verification into a unified certification-oriented engineering workflow with end-to-end traceability.

The primary objective of this paper is to develop a certification-oriented methodology for requirements management and formal verification of UAV operational models. The proposed methodology integrates requirements engineering, FHA, operational scenario decomposition, model transformation, and formal verification into a unified engineering workflow supporting UAV certification. The key contributions of the paper are summarized as follows:

- A certification-oriented engineering workflow that systematically transforms certification requirements into formally verifiable UAV operational models by integrating requirements engineering, FHA, operational scenarios, and formal verification within a single development process.
- A traceability framework linking certification requirements, FHA safety functions, operational scenarios, UML state-machine models, and formal verification artifacts, ensuring consistency throughout the certification-oriented development lifecycle.
- A model transformation methodology that converts UML state-machine models into NuSMV-interpretable representations while preserving behavioral semantics required for formal verification.
- An integrated certification-oriented formal verification framework based on LTL, CTL, and the NuSMV symbolic model checker, enabling systematic assessment of the correctness, consistency, completeness, and safety properties of UAV operational scenarios.
- A systematic methodology for mapping certification requirements and FHA safety functions onto UAV operational scenarios and UML use cases, enabling structured development of safety-critical operational models.
- A comprehensive case study demonstrating the complete certification-oriented workflow, from requirements decomposition and safety-function mapping to formal verification, inconsistency detection, and model refinement.

Unlike previous studies, which typically address certification, requirements engineering, model transformation, or formal verification as largely independent engineering activities, the proposed methodology does not introduce a new verification algorithm. Instead, it introduces a certification-oriented engineering process that systematically integrates

these established techniques into a unified workflow supporting end-to-end traceability from certification requirements and FHA safety functions to formal verification evidence. This integration reduces ambiguities in requirements verification and facilitates the development of certification-ready UAV systems while maintaining operational safety and software quality.

The remainder of the paper is organized as follows. Section 2 discusses requirements management, certification-oriented design methodologies, and the decomposition of UAV operational scenarios. Section 3 presents the proposed certification-oriented model-verification framework, including formal verification with LTL/CTL, UML-to-NuSMV model transformation, and model-validation techniques. Section 4 demonstrates the proposed methodology using a practical UAV case study. Finally, Section 5 concludes the paper and outlines future research directions.

2. Certification-Oriented Methodology

2.1. System Requirements

UAV systems are designed to support mission planning, data acquisition, processing, and information distribution tasks (i.e., reconnaissance cycle). In practice, many operational functionalities implemented by UAV platforms do not directly affect flight safety and therefore are not subject to formal certification procedures. Certification activities primarily focus on verifying that the airborne system as a whole meets operational safety and airworthiness requirements.

During system development, designers may apply different architectural and software-engineering methodologies, including NATO Architecture Framework (NAF) [20] or certification-oriented standards such as DO-178C [21]. DO-178C defines processes for the software lifecycle of airborne systems, including software testing, quality assurance, configuration management, and verification procedures. The standard is commonly applied in the development of software systems for airborne safety-critical applications.

Systems Modeling LanguageTM (SysML) [22] provides a convenient framework for modeling software components, firmware interfaces, and operational workflows. Although the implementation and testing of UAV systems generally follow standard industrial practices, certification-oriented systems require additional verification activities associated with operational reliability and fault tolerance. Designers must demonstrate that the implemented mechanisms effectively prevent or mitigate unsafe operating conditions, including communication loss, GPS signal disturbances, or propulsion-system failures.

The level of formalization during the design process may be increased using model-based system engineering [23] and Model-Based Development (MBD) approaches described in DO-331 [24], object-oriented methodologies defined in DO-332 [25], and formal mathematical methods specified in DO-333 [26]. These methodologies support the development of verifiable software models and improve traceability between system requirements, operational scenarios, and safety-related functions.

For certified systems, various sources of requirements can be identified. These include market and operational demands defined by end users, along with airworthiness guidelines set by regulations and certification authorities.

Certification procedures also require continuous interaction between system designers, end users, and certification authorities throughout the development lifecycle. This aspect is frequently underestimated during UAV system development and may lead to inconsistencies in operational assumptions, incomplete safety analyses, or errors in the certification process.

2.2. Certification of UAV Operational Scenarios

From the perspective of the system designer, the product being developed can execute missions that significantly influence how tasks are performed. Two types of missions are possible.

The first mission type involves reconnaissance flights conducted while maintaining continuous communication with the ground control station (GCS).

The second mission type consists of autonomous UAV operations performed without continuous communication with the GCS.

The pilot can carry out missions by operating the platform in various flight modes. Typical operational modes include autonomous navigation, waypoint tracking, loitering, landing procedures, and emergency recovery operations. The flight operations also encompass landing and parachute ejection.

The certification-oriented process requires identification of the scenarios that need to be certified. For instance, a general scenario (see Figure 3) can be defined for the implementation of the mission in communication with the GCS. This includes modes such as takeoff, transitioning between flight modes, utilizing the available payload for reconnaissance, and landing. The UAV must be capable of managing hazardous situations during flight, specifically: uncontrolled deviation from the mission area, collisions with terrain or other aircraft, loss of radio communication, and loss of GPS signal.

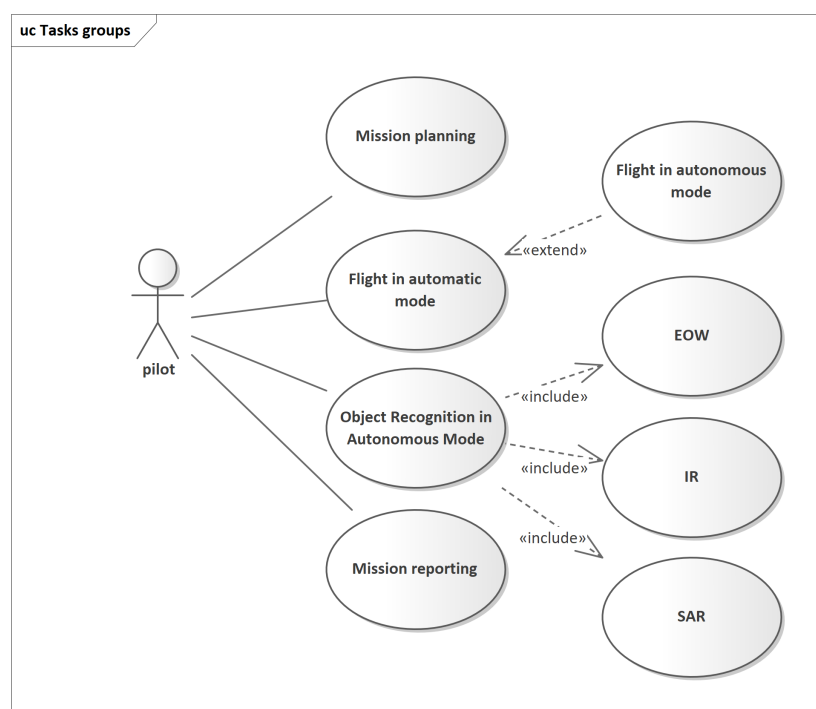


Figure 3. Task groups in the UAV management system.

In the event of a dangerous situation for the UAV and a loss of contact with the GCS, the algorithms are programmed to return the UAV to its landing site (see, for example [27]). If a safe return is not feasible, the UAV will enter emergency landing mode by deploying a parachute at the allowable altitude.

The certification process requires a formally defined set of identified emergency conditions that can be reviewed and validated by the certification authority. These scenarios may include typical applications during rescue operations, which often remain uncertified.

2.3. Combining Scenarios with Flight Safety Functions

Unmanned aircraft operate under various flight scenarios defined by their flight modes. Key functions that impact flight safety are emphasized in this context. FHA documentation defines safety-critical operational functions together with associated hazardous conditions and mitigation procedures. Both the aircraft manufacturer and the certifying authority identify these emergencies, categorizing them as in-flight emergencies. The document must not only describe possible hazardous events but also provide strategies for resolving them. Additionally, the system designer creates a comprehensive list of emergencies, which is then verified on a case-by-case basis by specialists involved in the certification process.

2.4. System Use Cases

The structure and granularity of UAV operational scenarios modeled in UML [22] depend on the platform class, mission objectives, and certification constraints imposed on the system (see [28]). To improve model traceability and verification consistency, UAV operational behaviors were organized into modular functional groups representing reusable operational patterns associated with different mission phases and safety constraints. These functional groups describe UAV flight-management activities performed in different operating modes and during emergency situations (see Figure 4).

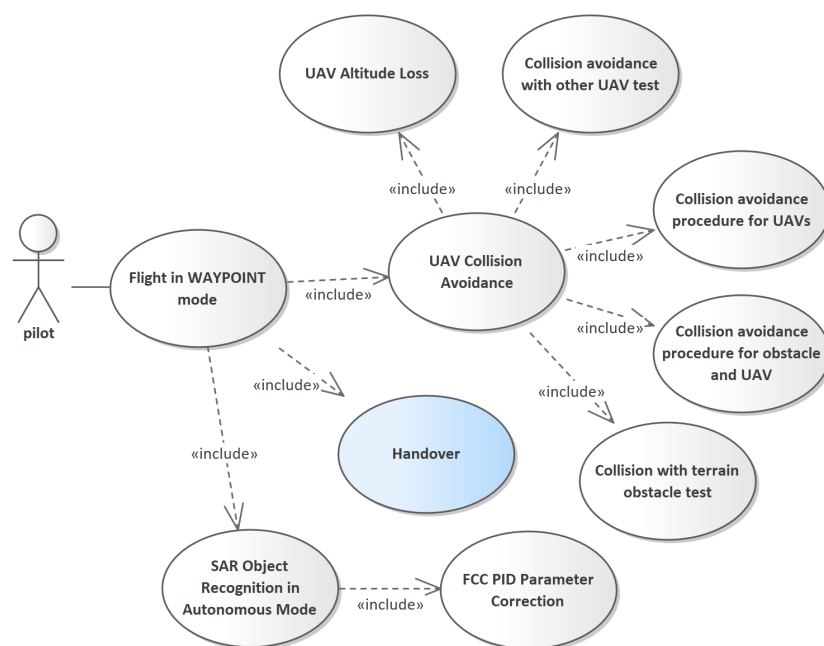


Figure 4. A use case diagram showing the three parallel actions of the system: flight modes, payload operations, and associated emergencies. Handover—a use case describing the pilot’s action during the transfer of a UAV to the control of another GCS.

Decomposed system use cases (basic groups of system functionality) are shown in Figure 5. They can be divided into cases describing mission planning activities and cases describing the implementation of subsequent mission stages. The resulting use-case structure provides a basis for designing a wide range of UAV missions (see Figure 5).

System use cases are utilized to break down the functions performed by the UAV, particularly in handling hazardous situations. Each decomposed use case is linked to specific functions from the FHA. Additionally, specific failure states are assigned to each FHA function, and a state may correspond to one or more use cases. To ensure the accuracy of the proposed solution, it is usually verified that each failure state is correctly associated with the corresponding specific situation, as illustrated in Figure 5. This approach facilitates

the development of certification evidence while reducing the effort required during external compliance assessment.

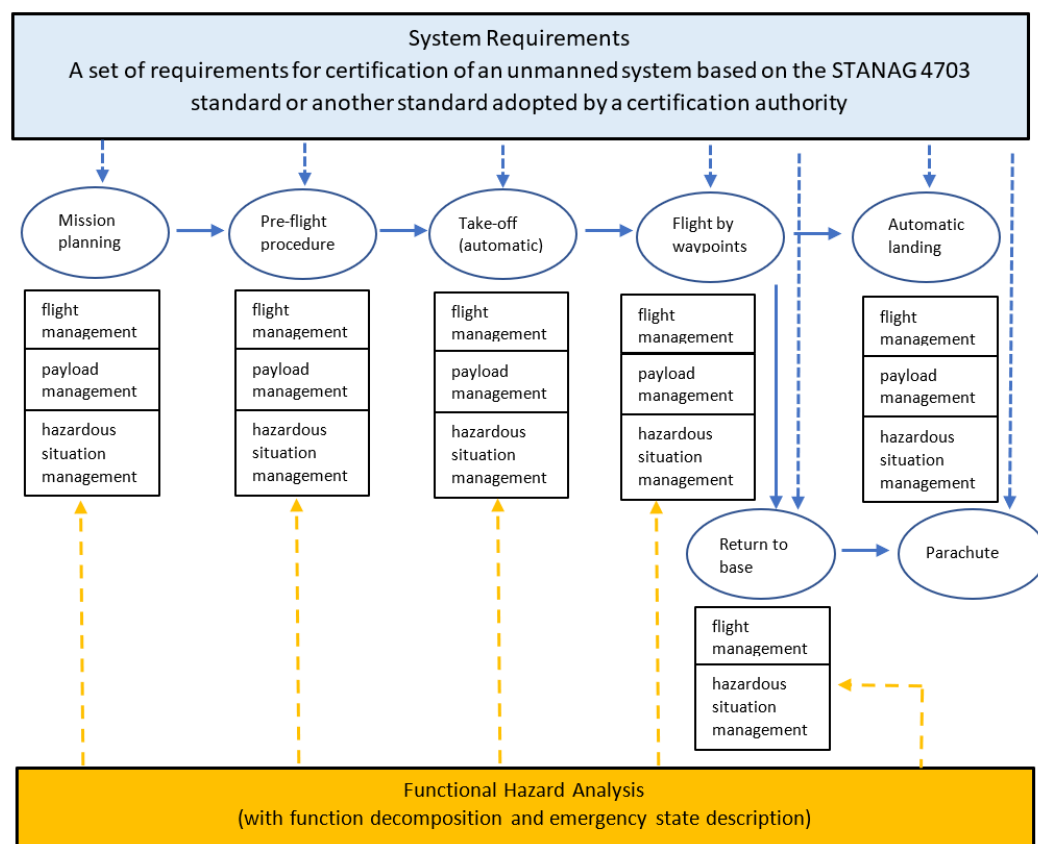


Figure 5. Example scenarios for UAVs that must meet certification requirements (see [29]). Oval shapes represent the functions executed by the UAV during the mission phase. The diagram shows a mission with two possible endings: landing in a designated area or an emergency parachute landing.

Figure 6 illustrates the functions related to UAV flight safety and emergencies for the indicated flight modes. A method for verifying the occurrence of emergencies is indicated.

ID	Function	Emergency state	Flight phase	Expected effect on: UAVs, humans, missions, other systems	Emergency classification	Verification method
100.1	Ensuring stability and control					
100.1.1	Determination of spatial orientation and speed of UAV	Inability to determine spatial orientation in the range, roll, pitch and yaw of the air platform	all phases except PARACHUTE	UAV loss; a threat to humans	II	tests in Hardware in the Loop simulator

Figure 6. Emergency state description. Each state is assigned to an FHA function. This state indicates the priority the system should give to handling the problem.

Each identified hazardous operational condition should be characterized using a structured safety-analysis and traceability framework that includes:

- Names the situation;
- Describes the situation unambiguously;
- Defines the risks on humans, facilities, and systems;
- Assigns a priority;
- Describes methods of avoiding dangerous situations;
- Describes the pilot's response procedures or the operation of the UAV's safety systems in the event of a dangerous situation.

In the process of building an unmanned system, it is crucial to define the UAV's behavior in the event of a dangerous situation during flight. The description of the response to a problem that occurs during flight must include detailed procedures for eliminating or bypassing it. In the case of non-redundant equipment, this problem is particularly important. If a propulsion system failure is isolated, the UAV can become unstable, making communication with the GCS impossible. In such a case, the unmanned platform must make autonomous decisions. Such situations require the design of procedures during the system design phase, when the emergency states shown in Figure 6 are defined.

The development of solutions for numerous risk management problems is currently supported by artificial intelligence mechanisms. These include popular techniques such as convolutional neural networks for classifying input data, as well as various heuristic algorithms. Despite this advancement, classical approaches based on static optimization methods or geometric solutions continue to be utilized. For instance, a geometric approach to securing a flight platform against collisions with other platforms is described in the algorithm by Park et al. [30].

In practical certification projects, the proposed methodology is intended to support the preparation of comprehensive requirements traceability matrices linking certification requirements, FHA safety functions, behavioral models, and formal verification artifacts. Since such matrices are project-specific and typically comprise numerous requirements and verification records, their detailed representation is beyond the scope of this paper.

3. Formal Verification Methodology

Quality assurance in certification-oriented development involves two complementary activities: validation and verification of requirements and system functions. Functions are created based on requirements. According to a commonly accepted definition, validation is the process of verifying the correctness and acceptability of requirements and the models derived from them. The purpose of function verification is to confirm the completeness and consistency of models developed from requirements. The process is inherently complex because it requires balancing user expectations with technical implementation constraints.

The model verification method presented in this paper involves mapping requirements transformed into use cases to certification requirements and system security functions. In this paper, we use the UML modeling language and state machines, which is not the only possible approach. The UML language, or its technical extension (SysML), allows for precise modeling of functions. State machine models implemented in UML can be checked for correctness and uniqueness.

A requirement or function is complete when its description allows for design work to be conducted at an appropriate level of detail. The required level of completeness varies depending on the methodology and the lifecycle stage at which the requirement is being explored or represented. Applying the DO-178 design methodology to the development of an unmanned system ensures that no essential requirements are omitted. A requirement or function is described precisely when it does not contain unnecessary or repetitive content. A requirement or function is feasible when it does not violate acceptable risk limits. A requirement or function is unambiguous when it can be verified that the solution meets the user's needs. Creating an FHA document allows the designer to verify the models' unambiguity. A model is considered testable when it can be verified through simulation tests or UAV flights that the safety boundaries have not been violated and the function achieves the specified objectives (see [31]).

The method used in our UAV projects for validating and verifying the correctness of the constructed model, which is the basis for developing software for safety-critical systems, consists of the following steps:

1. Developing system requirements in the form of UML use cases.
2. Developing a description of the system dynamics in the form of UML state machines for each functional requirement, by the Object Management Group (OMG) concept presented in the UML 2.5.1 specification in section 18.1.4 (see [32]).
3. Verifying the completeness and consistency of the model represented by the state machine based on CTL and LTL modal logic (Section 3.1).
4. Determining the minimum test coverage of the model represented by the state machine (see [33]).
5. Implementing the model and conducting tests of each module according to the designated test paths.

The proposed method described in Section 3.1 facilitates testing at both the module and unit levels, as the operations defined in each state of the state machine model are generally represented as UML sequence diagrams. The next section of this article presents a method for verifying the completeness and consistency of a model represented as a state machine using CTL and LTL modal logics. Although this verification step is frequently omitted in practice, it enables early detection of inconsistent, incomplete, or unreachable model states. Failure to perform such an analysis results in incomplete test coverage of individual algorithms or incompleteness of the algorithms.

Table 1 lists selected works describing an integrated approach to software model verification and test case generation. Villani et al. [34] present a model verification method using UPPAAL-based testing and CoFI (Conformance and Fault Injection). This integrated approach allows for additional model verification. This project has been implemented in the industry. It is worth noting that algorithms developed in MATLAB ver. R2023 and Simulink are used to verify the models (see [35,36]).

Table 1. Summary of selected methods for verifying the correctness of models and generating tests. Important works describing the combination of several software testing and verification methods are listed.

No	Model Verification	Test Generation	Reference	Remarks
1	LTL state machines with FeaVer	Model-cased test generator	[37]	FeaVer (Bell Lab. generator)
2	MCMAS+ (CTL extension)	State, transition coverage with multi agent system	[38]	
3	UPPAAL (CTL)	CoFI conformance and fault injection ConData test generator	[34]	
4	SMT solvers (Simulink)	Astree (code analyzer) MaTeLo test generator (Markow chains)	[35]	Project MBAT
5	ReMOTGen (Matlab and Doors)	SmartTestGen (Simulink state flows)	[36]	In-house GM solutions
6	NuSMV (CTL/LTL)	DAG's (UML state machines)	This paper	In-house solution

3.1. Software Testing with LTL and CTL

In this section, we describe how to utilize LTL and CTL techniques for model safety and liveness checking.

Initially, modal logics were developed to formally describe concepts that were subject to philosophical exploration, such as necessity and possibility. These concepts could not be effectively formalized within classical logic, which relies on binary semantics and the principle of extensionality. To address this issue, Kripke proposed [39] semantics for these operators based on the idea of possible worlds. These semantics were later adapted to logics that express properties of information systems, which have been utilized in the field of software security research, and Kripke models became a standard tool for modeling the behavior of such systems.

The logic LTL was defined in 1977 in Pnueli's paper [40]. He introduces new modal operators that express properties of infinite computations. These are interpreted on a fixed, infinite computation path.

The language of LTL is a language of propositional logic enriched with a few modal operators, for example: $F\varphi$ with the meaning that φ is true in some future state of a computation, or $G\varphi$ meaning that φ is true always in the future.

Let $M = (W, R, v)$ be a Kripke model, with a fixed valuation v , such that $v(p) \subseteq W$ is a set of worlds, or states of a computation, in which p holds. Then, for $a \in W$ and a formula φ , we write $M, a \models \varphi$ to state that φ is true in M in a world a . In our case, the model M represents a state machine, but it must be defined in the language of the model checker software, NuSMV [10,11]. This language facilitates the translation of the program into the Kripke model symbolically represented in NuSMV.

The logic CTL was introduced by Clarke and Emerson [41]. This logic allows a designer to express properties that are inexpressible in LTL, but on the other hand, some properties that are expressible in LTL are not expressible in CTL. Each operator in CTL logic has two versions, existential and universal, for example:

- $AG\varphi - \varphi$ is true always in the future on any computational path,
- $EF\varphi - \varphi$ is true in a future computational state in some state on some computational path.

The semantics of the logic CTL are defined concerning all possible paths starting from a given state, not a single fixed one.

The logics LTL and CTL are incomparable. There are properties expressible in one of these logics but not in the other. For example, the LTL formula $GFp \rightarrow GFq$, which states that on every path where p is true an infinite number of times, q is also true an infinite number of times, is not expressible in CTL. This follows from the fact that CTL cannot refer to a single, fixed path. On the other hand, the CTL formula $AG EFp$, which states that a state which satisfies p is reachable from any state of a computation is not expressible in LTL.

3.2. Testing with LTL and CTL

The truth of the formulas of both logics in a given model is decidable in the class of computations in polynomial space. This makes this problem difficult, but there exist sufficiently good methods for determining the truth of these formulas for sufficiently large systems. These logics allow us to express many important properties of finite systems.

The following is a list of the most typical properties.

1. Liveness—if the system requests a resource, it will eventually get it. Liveness can be expressed in both LTL and CTL. In LTL, liveness is expressed as:

$$G(\text{req} \rightarrow F \text{ granted}).$$

In CTL, liveness can be expressed as:

$$AG(\text{req} \rightarrow AF \text{ granted}).$$

2. Safety—two processes will never have access to critical resources at the same time. In LTL logic, safety is expressed as:

$$G(\neg(\text{crit}_1 \wedge \text{crit}_2)).$$

And in CTL as:

$$AG(\neg(\text{crit}_1 \wedge \text{crit}_2)).$$

3. The possibility of reaching a final state from any reachable state. In CTL, we express this property as

$$AG \, EF \, \text{end}.$$

In LTL, this property is not expressible. However, we can express in LTL the stronger property that every computation terminates in a final state

$$F \, \text{end}$$

This paper primarily investigated the possibilities of achieving a final state using state machines that represent data processing processes in the system managing an unmanned platform.

Implementing model verification mechanisms based on LTL and CTL logic would be too complex without the tools available to examine IT system designs. One such tool is NuSMV (<https://nusmv.fbk.eu/> (accessed on 18 June 2026)). It is one of the most important tools providing methods for symbolic system verification. It is being developed by Carnegie Mellon University, the University of Genoa, the University of Trento, and the Bruno Kessler Foundation. It is under continuous development; the latest version (i.e., version 2.7.1) was released in September 2025. It is released under the GNU LESSER GENERAL PUBLIC LICENSE (<https://nusmv.fbk.eu/static/lgpl2.1.txt> (accessed on 18 June 2026)), which allows commercial use of the software.

We chose this tool because it has found many practical applications. NuSMV is used commercially, for example, in an FPGA design library [17]. It has also helped detect errors, including false positives, in nuclear facility management and monitoring systems [18]. It has been implemented on a large scale in a vehicle control system. The authors presented techniques to speed up formula verification, such as by parallelizing the process [19].

3.3. Model Transformation

To manage the complexity of UML state machines, a modular decomposition strategy is employed. Instead of representing the entire state machine as a single NuSMV process, the model is partitioned into several NuSMV modules. This approach mirrors the structure of orthogonal regions in the UML diagram, enabling localized verification and clearer mapping between the system design and the formal model.

Each NuSMV module is designed to encapsulate one orthogonal region of the UML state machine. Within each module, a primary variable, *state*, is defined, whose values correspond directly to the states present in the UML region. To capture the full semantics of the transitions, these modules also incorporate auxiliary variables. These variables are used to model transition guards (e.g., boolean flags, counters), internal actions (e.g., variable updates), and the necessary signals for synchronization between concurrent regions.

Communication between these modules is achieved through a parameter-passing mechanism. The *main* module serves as the central supervisor; it instantiates each regional module and manages the global variables that facilitate inter-module communication and synchronization.

The translation of a UML state machine into a modular NuSMV model follows a process as outlined below:

1. **Decomposition:** The UML state machine is analyzed to identify all top-level orthogonal regions that will be mapped to a unique NuSMV module.
2. **State Mapping:** For each identified region, a NuSMV module is created. Within these modules, the set of all possible states in the UML region is mapped to an enumerated type variable, *state*.
3. **Transition Logic:** Each transition in the UML diagram is translated into a `next(state)` assignment within a NuSMV `case` or `if` statement.
 - Guards are implemented as boolean conditions within the transition statements.
 - Actions are modeled by updating auxiliary variables during the state transition logic.
4. **Auxiliary Variables:** Any control logic not captured by the *state* variable (e.g., timers, sequence counters, or event flags) is an additional variable within the specific module.
5. **Communication Modeling:** The dependencies between orthogonal regions are identified. If a transition in Region A depends on a state change in Region B, a communication variable is defined. This variable is passed as an input parameter to the module representing Region A.
6. **Main Module:** A global *main* module is constructed to:
 - Declare the global variables used for inter-module synchronization.
 - Instantiate each regional module.
 - Link the sub-modules by passing the relevant global variables as parameters.
7. **Verification:** Finally, LTL or CTL properties are defined to verify that the integrated behavior of all modules satisfies requirements of the specification.

Details on how specific complex conditions were modeled are provided in the case study model, and the problems encountered during this translation process, along with descriptions of selected solutions, can be found in Section 3.2.

For those interested in transforming UML state machine models into NuSMV-interpretable models, we refer to Wu et al. [42]. In this paper, the authors present a detailed process for transforming UML models into NuSMV-interpretable models. Examples are based on state machines in the MATLAB/Simulink StateFlow package. The package is compliant with the UML notation defined in the OMG standard.

It is worth noting the transformation of models described by orthogonal state machines. In such a case, one can either model each orthogonal state or merge the states into a machine without orthogonal states. The latter approach is usually used for small models. It is described by Górski et al. [29].

The presented model transformation methodology establishes the general mapping principles adopted within the proposed certification-oriented engineering workflow and provides sufficient information to understand the transformation process used in the case study. Detailed implementation rules and transformation algorithms depend on the adopted UML modeling conventions and the target NuSMV implementation and are therefore beyond the scope of the present paper.

4. Case Study of Proposed Methodology

This section presents an approach to solving a model validation problem using the developed approach and was based on part of the bachelor's thesis of one of the authors, Sławomir Brzózka. Modified state machine diagrams, presented in the work of [29], are used as an example. To verify the correctness of the model, an orthogonal state machine (Figure 7) was used.

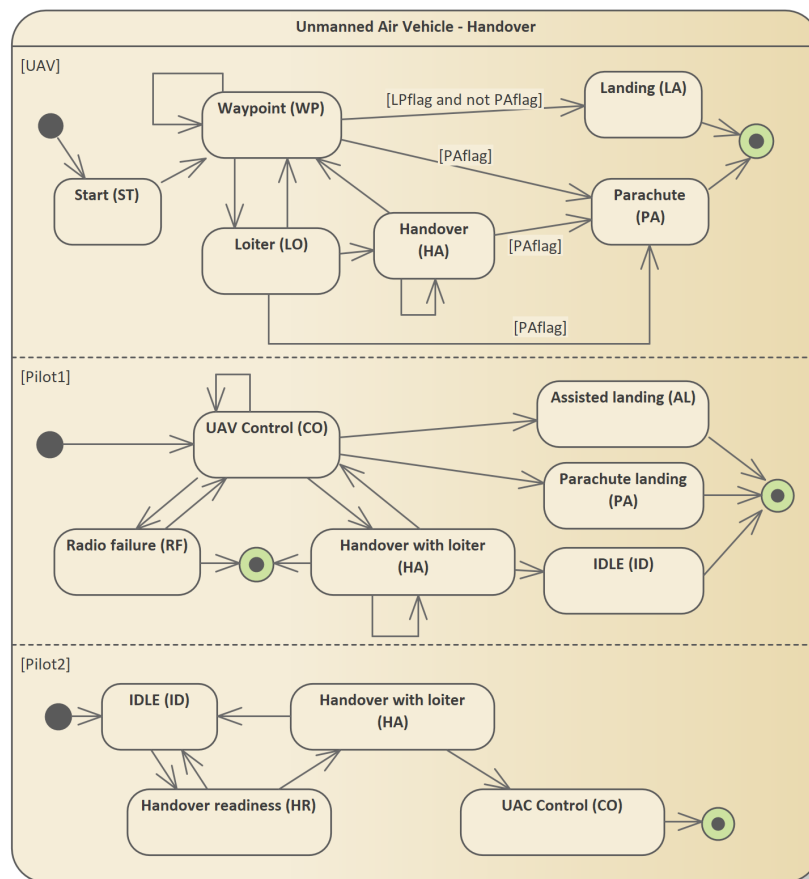


Figure 7. State machine with orthogonal states. Modified version of the state machine presented in [29]. The UAV model is marked with flags which, when set, cause the UAV to switch to emergency landing mode (PAflag) or return to base (LPflag).

NuSMV has its own domain-specific language for modeling. Within this language, the basic unit of a model is a module. Dividing the model into appropriate modules allows the representation of the orthogonality in the diagram, and the generation of a single, unified state machine is automated.

For modeling purposes in NuSMV, the following assumptions were made:

- The state machines for Pilot1 and Pilot2 (numbered according to the diagram) were modeled as a single module (hereinafter referred to as pilot) with different initial states. For Pilot1, the initial state is CO, and for Pilot2, it is ID.
- A set of commands sent from the pilot module to the UAV module was implemented. Communication is one-way and can be interrupted (state RF). Most state transitions, in the UAV state, are controlled by these commands.

The model consists of the following modules:

1. Main—The main module transmits information between the other modules.
2. UAV—UAV module that changes flight modes according to the diagram.
3. Pilot—The pilot module has states that correspond to the pilot's commands.
4. Timer—Support module used to count a specified number of ticks.

4.1. Main Module

This module serves as a coordinator for the other modules. This model includes two instances of the Pilot module (Pilot1 and Pilot2), one UAV, and the auxiliary variables described below. This module relays information between the active pilot and the UAV, as shown in the diagram. The first pilot starts in the active state, and after a successful handover, they switch roles.

1. SUCCEED—a predicate is true if the conditions necessary for the transfer to take place are met.
2. phase—represents the communication phase. Its initial value is BSP, followed by the Pilot and UAV phases in turn.
3. active—indicates which Pilot module is currently active (issuing UAV commands); initially, this variable is set to Pilot1. It changes only when the SUCCEED condition is met. A change in the value of this variable indicates that a successful BSP transfer has occurred between Pilot instances.
4. msg—A variable representing messages (commands) sent to the UAV module. The variable is set according to the internal variables of the Pilot modules, based on the value of the ‘active’ variable.

4.2. UAV Module

The UAV module simulates a UAV’s behavior by updating its state based on the ‘msg’ parameter. Its components include

1. RTB_flag—return-to-base flag.
2. state—the main state of the BSP module corresponding to the flight mode the BSP is currently in.
3. conn_timer—an instance of the Timer module that counts the number of ticks during which the UAV has received no message.
4. TIMEOUT—the number of ticks the BSP waits for a message. Parameter for the Timer module.

Let t be the current tick, and then the state transition function defining $state_{t+1}$ is given by:

$$\begin{aligned}
 & ERR \rightarrow ERR \\
 & END \rightarrow END \\
 & state_t \mid RTB_flag_{t+1} \wedge state_t \neq LA \rightarrow LA \\
 & \quad ST \rightarrow WP \\
 & \quad LA \rightarrow END \\
 & \quad PA \rightarrow END \\
 & state_t \mid msg_{t+1} = M_NONE \rightarrow state_t \\
 & \quad WP \mid msg_{t+1} = M_WP \rightarrow WP \\
 & \quad WP \mid msg_{t+1} = M_LO \rightarrow LO \\
 & \quad WP \mid msg_{t+1} = M_LA \rightarrow LA \\
 & \quad WP \mid msg_{t+1} = M_PA \rightarrow PA \\
 & \quad \quad WP \rightarrow ERR \\
 & \quad LO \mid msg_{t+1} = M_LO \rightarrow LO \\
 & \quad LO \mid msg_{t+1} = M_WP \rightarrow WP
 \end{aligned}$$

$$\begin{aligned}
& \text{LO} \mid \text{msg}_{t+1} = \text{M_HA} \rightarrow \text{HA} \\
& \text{LO} \mid \text{msg}_{t+1} = \text{M_LA} \rightarrow \text{LA} \\
& \text{LO} \mid \text{msg}_{t+1} = \text{M_PA} \rightarrow \text{PA} \\
& \text{LO} \rightarrow \text{ERR} \\
& \text{HA} \mid \text{msg}_{t+1} = \text{M_WP} \rightarrow \text{WP} \\
& \text{HA} \mid \text{msg}_{t+1} = \text{M_HA} \rightarrow \text{HA} \\
& \text{HA} \rightarrow \text{ERR}
\end{aligned}$$

where convention

$$\text{previous state} \mid \text{condition} \rightarrow \text{next state}$$

is used, and the first matching formula is assumed to be chosen.

Therefore, for example, if $\text{state}_t = \text{ERR}$ (system error) then $\text{state}_{t+1} = \text{ERR}$ else if $\text{state}_t = \text{END}$ then $\text{state}_{t+1} = \text{END}$ else if for any state_t condition $\text{RTB_flag}_{t+1} \wedge \text{state}_t \neq \text{LA}$ is true then $\text{state}_{t+1} = \text{LA}$ and so one.

The main point of interest of this module is the ‘RTB_flag’ that should be raised according to the fail conditions specified in the original diagram presented in [29]. Properties examined would focus on its behavior.

4.3. Pilot Module

The Pilot module represents one of the two pilots identified by a PID. The module also has access to the active and phase parameters, as well as to the success condition.

1. TIMEOUT—the value of the timeout parameter in Timer module instances.
2. state—the state of Pilot. Most states are related to commands sent to UAV, but it also includes additional states such as END and ID.
3. set_msg—the message that Pilot is attempting to send.
4. RF_timer—the timer indicating how long Pilot has been in the RF state.
5. HA_timer—the timer indicating how long Pilot has been in the HA state.

Let us define

1. r_t : RF timer expiration flag.
2. h_t : HA timer expiration flag.

Then, the (nondeterministic) transition function for the state is given by:

$$\begin{aligned}
& \text{state}_t \mid \text{phase}_t \neq \text{PILOT} \rightarrow \{\text{state}_t\} \\
& \text{CO_LO} \rightarrow \{\text{CO_LO}, \text{CO_WP}, \text{HA}, \text{AL}, \text{PA}, \text{RF}\} \\
& \text{CO_WP} \rightarrow \{\text{CO_LO}, \text{CO_WP}, \text{AL}, \text{PA}, \text{RF}\} \\
& \text{RF} \mid r_t \rightarrow \{\text{END}\} \\
& \text{RF} \rightarrow \{\text{CO_LO}, \text{CO_WP}, \text{RF}\} \\
& \text{HA} \mid \text{succed}_t \rightarrow \{\text{ID}\} \\
& \text{HA} \mid h_t \rightarrow \{\text{CO_WP}\} \\
& \text{HA} \rightarrow \{\text{HA}\} \\
& \text{ID} \rightarrow \{\text{ID}, \text{HR}\} \\
& \text{HR} \mid \text{succed}_t \rightarrow \{\text{CO_WP}\} \\
& \text{HR} \rightarrow \{\text{HR}, \text{ID}\} \\
& \text{AL} \rightarrow \{\text{END}\} \\
& \text{PA} \rightarrow \{\text{END}\} \\
& \text{END} \rightarrow \{\text{END}\}
\end{aligned}$$

We model the pilot selection by transitioning to one of the possible states.

For example if for any state $\text{phase}_t \neq \text{PILOT}$ then $\text{state}_{t+1} \in \{\text{state}_t\}$ else if $\text{state}_t = \text{CO_LO}$ then $\text{state}_{t+1} \in \{\text{CO_LO}, \text{CO_WP}, \text{HA}, \text{AL}, \text{PA}, \text{RF}\}$

4.4. Verified Properties

In formulas we would use predicate FINISHED defined by:

$\text{FINISHED} := \text{uav.state} = \text{END} \ \& \ (\text{pilot1.state} = \text{END} \mid \text{pilot2.state} = \text{END})$

Verified formulas are presented in Listing 1 and we will discuss each of them. We understand these formulas as:

1. On each path (in state machine), if **Uav** is in the **END** state, then in the next state exactly one **Uav** will also be in the **END** state.
2. On each path, if exactly one **Uav** is in the **END** state, then **Uav** will also be in the **END** state.
3. In each state, **msg** has the value **EXC** only if the formula FINISHED holds.
4. In each state, the state FINISHED is reachable.
5. There is a path where FINISHED never holds.
6. There is a path where **pilot1** is active until **pilot2** is active.
7. The state in which **pilot2** is active and active until **pilot1** is active is reachable.
8. There is a path where **HA_SUCCEED** is always reachable.
9. On every path, we will eventually reach states in which **HA_SUCCEED** never holds.
10. The error state (**ERR**) of the variable **bsp.state** is unreachable.

The purpose of the first two formulas is to justify the definition of the formula FINISHED. This is an equivalence between the states **END** on **Nsk** and one of **Uav** with precision of one tick. The presence of **X** in the formula is necessary due to the alternation of communication.

Formulas 3 through 5 check the properties of the formula FINISHED. Formula 3 ensures that **msg** does not throw an exception except after the system has terminated. Formula 4 ensures that the system does not enter a state where **Uav** cannot land. Formula 5 guarantees that there is an infinite path on which **Uav** does not land.

Formulas 6 and 7 check whether the *handover* procedure can proceed in both directions. Formula 6 only checks whether the value **P2** is reachable. Formula 7, on the other hand, aims to verify the possibility of passing from **pilot2** to **pilot1**. Note that it is necessary to guarantee that **pilot2** is active. Otherwise, the formula would trivially be satisfied in the first state (since **active = P1** holds in it).

Formulas 8 and 9 verify the possibility of passing **Uav** indefinitely. The weaker formula 8 only guarantees that there exists a path that has the corresponding state deeper in the tree at each branch, but this state need not be on the found path. The formula that would actually be desirable is $\exists G F \text{HA_SUCCEED}$, where $\exists$ quantifies over all paths in a state machine. All the formulas in NuSMV contain an implicit universal quantifier over initial states. However, this problem can be solved by negating the desired formula; we have

$$\neg \exists G F \text{HA_SUCCEED} \equiv \forall F G \neg \text{HA_SUCCEED}$$

and hence the form of Formula 9, which we expect to be false, because then $\exists G F \text{HA_SUCCEED}$ holds. This approach is also constructive; to prove the falsity of formula 10, NuSMV constructs a counterexample in the form of a path on which **HA_SUCCEED** holds infinitely many times.

The last formula is obvious, it checks whether the transition function for **Uav** handles all situations occurring in the model without ending up in the **ERR** state.

Listing 1. Verified properties.

```

1 G (uav.state = END
2   -> X (pilot1.state = END xor pilot2.state = END)) -- (1)
3 G ((pilot1.state = END xor pilot2.state = END) -> uav.state = END) -- (2)
4
5 AG (msg = EXC -> FINISHED) -- (3)
6 AG EF FINISHED -- (4)
7 EG !FINISHED -- (5)
8
9 E [active = P1 U active = P2] -- (6)
10 EF (active = P2 & E [active = P2 U active = P1]) -- (7)
11
12 EG EF HA_SUCCEED -- (8)
13 F G !HA_SUCCEED -- (9)
14
15 AG uav.state != ERR -- (10)

```

4.5. Problems Identified During Model Verification

The following issues were identified in the model regarding state transitions.

I. Radio Failure State (RF)

Problem description:

It is possible that after transitioning to the *RF* state in the pilot module, the UAV module will never reach the *END* state. This issue is caused by the current model, which assumes pilot control of the UAV, including landing and parachute deployment, and lacks a mechanism for responding to communication loss. In reality, the UAV is pre-loaded with a mission that requires a defined landing and parachute deployment procedure in unexpected situations.

Proposed solutions:

- Add a new state corresponding to Radio Failure in the UAV module. This is an implementable solution, but such a state could be achieved from almost any other state in the UAV module, which would significantly reduce readability and complicate model analysis. Handling priorities during parachute deployment is a particular problem.
- The *RTB* flag with a timer and communication confirmation—the solution most closely reflecting the reality of the situation under consideration. If the pilot module is in one of the following states: *RF*, *ID*, or *END*, the UAV receives a message with the value *NONE*. If the UAV does not receive a valid message for more than *n* cycles (assumed value of 5), it sets the *RTB* flag described above. The advantage of this approach over the previous one is greater flexibility, as the *RTB* flag can be set by factors other than *RF*, which can now be modeled and verified (including random factors).

II. UAV Control State (CO)

Problem description:

Figure 7 shows the *CO* → *HA* transition. If it is executed and the UAV is in state *WP*, which is possible, there is no transition to state *HA*. The current model does not provide verification of the correctness of mode changes in *CO*. This verification would require the ability to remember mode changes to verify transition correctness.

Proposed solutions:

- Create a new module in place of *CO*—then its internal state could remember and verify the correctness of mode changes. This is a good solution if the model contains more than *WP* and *LO* modes.
- Split the *CO* state into states *CO_WP* and *CO_LO*—this solves the problem of impossible transitions while only slightly changing the model. This solution was used for verification.

Performing model analysis allows corrections before starting the test path coverage generation process, significantly reducing the work required for subsequent model verification.

The presented case study illustrates the complete certification-oriented verification workflow, including the identification of model inconsistencies, analysis of NuSMV counterexamples, refinement of the behavioral model, and subsequent verification of the corrected model. The presented verification examples illustrate the typical outcomes of the proposed methodology by showing how formal verification identifies incomplete or inconsistent operational behaviors that may affect flight safety and therefore require model refinement before certification activities can proceed. The objective of the case study is to demonstrate the applicability of the proposed certification-oriented methodology rather than to provide an exhaustive analysis of all verification outputs generated by the NuSMV model checker. Verification examples are presented to illustrate the proposed methodology, whereas complete verification logs and intermediate counterexamples are implementation-specific and are therefore omitted for clarity.

5. Conclusions

The scope of verification and validation activities performed during the certification process of unmanned aerial platforms depends on the UAV class, operational environment, and system complexity. Nevertheless, the general approach to requirements management, safety analysis, and certification-oriented verification remains similar across different categories of UAV systems. Certification requirements should be treated as an extension of system and operational requirements, focusing not only on functional correctness but also on demonstrating that the designed system satisfies airworthiness and operational safety constraints.

The article discussed current challenges related to UAV certification and safety-oriented system validation. Particular attention was devoted to the problem of mapping operational requirements to safety-related system functions defined within the FHA. The proposed approach supports the decomposition of system requirements into certification-oriented operational scenarios and enables traceability between user requirements, safety functions, and emergency conditions.

The presented methodology also includes formal modeling of UAV operational procedures using state-machine representations. This requires further decomposition of system requirements and their association with FHA functions and operational use cases. Each atomic FHA function can then be linked to one or more operational requirements directly associated with specific system behaviors and implementations. Formal verification tools such as NuSMV can subsequently be used to validate the correctness and consistency of the developed models. This stage of the design process enables the detection of incomplete transitions, unreachable states, and inconsistencies in safety-critical operational procedures before implementation and flight testing.

Comprehensive verification of requirements and operational models significantly facilitates the certification process and supports the assessment of UAV airworthiness by certification authorities and external auditors.

Finally, it should be emphasized that the presented methodology is not limited exclusively to UAV platforms. Similar certification and validation challenges can also be observed in autonomous unmanned ground vehicles (UGVs), whose operational autonomy and system complexity may exceed those of airborne systems [43]. Consequently, the proposed requirements-management and formal-verification framework can also be adapted to other classes of autonomous safety-critical systems.

The direction for further development of the presented method is to develop mechanisms that enable the automation of transforming UML-compliant state machines into machines interpreted by tools such as NuSMV. Another important issue is the analysis of

the potential for innovation in formal semantics learning using software tools [44,45]. This type of approach could expand model validation capabilities.

Author Contributions: Conceptualization, W.S. and J.M.K.; methodology, W.S. and S.B.; software, S.B.; validation, W.S. and S.B.; formal analysis, S.K. and S.B.; resources, S.K.; writing—original draft preparation, S.K. and W.S.; writing—review and editing, W.S. and J.M.K.; visualization, W.S.; supervision, J.M.K.; funding acquisition, J.M.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported under grant no. UGB 531-000128-W300-22/2026 on “Analysis of the possibility of using modern technologies in communication systems and electronic warfare” by the Military University of Technology (WAT).

Institutional Review Board Statement: Not applicable.

Data Availability Statement: A sample model developed with the NuSMV (ver. 2.7.1) tool is available in the GitHub repository: <https://github.com/modelchecking26/handover> (accessed on 18 June 2026). The data used in the article and additional models can be made available by the authors upon request.

Conflicts of Interest: Sławomir Brzózka is affiliated with the commercial company PIT-RADWAR S.A., Warsaw, Poland. This affiliation did not involve research funding, data ownership, patents, or software ownership related to this study, and the affiliated company had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results. The authors declare no other conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AL	assisted landing
ATC	air traffic control
CLP	Constraint Logic Programming
CO	UAV control
CoFI	conformance and fault injection
CTL	computation tree logic
DC	data computer
EASA	European Union Aviation Safety Agency
EO	electro-optical
EU	European Union
FCC	flight control computer
FHA	functional hazard analysis
FMS	flight management system
GCS	ground control station
GM	General Motors
GNSS	global navigation satellite system
GPS	Global Positioning System
HA	handover
HR	handover readiness
ICAO	International Civil Aviation Organization
ID	idle
IFF	identification friend or foe
INS	inertial navigation system
IR	infrared
ISR	intelligence, surveillance, and reconnaissance

LA	landing
LO	loiter
LOI	level of Involvement
LTL	linear temporal logic
MC	mission computer
MCMAS	Model Checker for Multi-Agent Systems
NAF	NATO Architecture Framework
NATO	North Atlantic Treaty Organization
OMG	Object Management Group
PA	parachute
PID	proportional-integral-derivative
R&D	research and development
RF	radio failure
SAR	synthetic aperture radar
ST	start
SysML	Systems Modeling Language
UAS	unmanned aerial system
UAV	unmanned aerial vehicle
UGV	unmanned ground vehicle
UML	Unified Modeling Language
UPPAAL	Uppsala University and Aalborg University
WAT	Military University of Technology
WMS	wireless management system
WP	waypoint

References

1. Kovacova, M.; Gravio, G.; Patriarca, R. Unmanned Aerial Systems: Status and Forthcoming Challenges for Safety Risk Management. *Transp. Res. Procedia* **2022**, *65*, 329–338. <https://doi.org/10.1016/j.trpro.2022.11.037>.
2. Abdulov, A.; Abdulova, E. Towards Safety of Unmanned Vehicles in the Smart City Environment. *IFAC-PapersOnLine* **2021**, *54*, 959–964. <https://doi.org/10.1016/j.ifacol.2021.08.114>.
3. Henderson, I. Aviation Safety Regulations for Unmanned Aircraft Operations: Perspectives from Users. *Transp. Policy* **2022**, *125*, 192–206. <https://doi.org/10.1016/j.tranpol.2022.06.006>.
4. Kritzinger, D. Functional Hazard Analysis. In *Aircraft System Safety: Assessments for Initial Airworthiness Certification*; Woodhead Publishing: Duxford, UK, 2016; pp. 37–57.
5. Sheng, R. Systems Engineering Fundamentals. In *Systems Engineering for Aerospace: A Practical Approach*; Academic Press: London, UK, 2019; pp. 113–206.
6. Chodnicki, M.; Kubrynski, K.; Stecz, W. MATLAB-Based UAV Dynamics Configurator and Testbed for COTS Fighter UAV Platform. *Appl. Sci.* **2026**, *16*, 4906. <https://doi.org/10.3390/app16104906>.
7. Zhang, X.; Zhao, X. Architecture Design of Distributed Redundant Flight Control Computer Based on Time-Triggered Buses for UAVs. *IEEE Sens. J.* **2021**, *21*, 3944–3954. <https://doi.org/10.1109/JSEN.2020.3025551>.
8. Cimatti, A.; Clarke, E.; Giunchiglia, F.; Roveri, M. NuSMV: A New Symbolic Model Verifier. In Proceedings of the 1999 11th International Conference on Computer Aided Verification (CAV), Trento, Italy, 6–10 July 1999; pp. 495–499.
9. Norouzi, M.; Parsa, S. Verification of the Protection Services in Antivirus Systems by Using NuSMV Model Checker. *Int. J. Found. Comput. Sci. Technol.* **2014**, *4*, 57–67. <https://doi.org/10.5121/ijfcs.2014.4506>.
10. Cavada, R.; Cimatti, A.; Keighren, G.; Olivetti, E.; Pistore, M.; Roveri, M. *NuSMV 2.7 Tutorial*; Bruno Kessler Foundation: Trento, Italy, 2026.
11. Cavada, R.; Cimatti, A.; Jochim, C.; Keighren, G.; Olivetti, E.; Pistore, M.; Roveri, M.; Tchaltev, A. *NuSMV 2.7 User Manual*; Bruno Kessler Foundation: Trento, Italy, 2026.
12. Frappier, M.; Fraikin, B.; Chossart, R.; Chane-Yack-Fa, R.; Ouenzar, M. Comparison of Model Checking Tools for Information Systems. In Proceedings of the 2010 12th International Conference on Formal Engineering Methods (ICFEM), Shanghai, China, 17–19 November 2010; pp. 581–596.
13. Neyzov, M.; Kuzmin, E. LTL Specification for the Development and Verification of Logical Control Programs in Feedback Systems. *Autom. Control Comput. Sci.* **2025**, *59*, 951–984. <https://doi.org/10.3103/S0146411625700312>.

14. Rashid, M.; Rasool, I.; Afzaal, H.; Zafar, N. Formal Verification of Safety Properties of Epoch Processing in Beacon Chain. *Sci. Rep.* **2025**, *15*, 43522. <https://doi.org/10.1038/s41598-025-27396-w>.
15. Zhou, X.Y.; Jin, S.; Mei, Y.; Sun, X.; Yang, X.; Nie, S.; Zhang, W. Towards Hazard Analysis Result Verification for Autonomous Ships: A Formal Verification Method Based on Timed Automata. *J. Mar. Sci. Eng.* **2025**, *13*, 1058. <https://doi.org/10.3390/jmse13061058>.
16. Nemouchi, A.; Bouzenada, A.; Saidouni, D.; Díaz, G. Formal Modelling and Verification of Multi-Parameter Context and Agent Transition Systems: Application to Urban Delivery Zone and Autonomous Electric Vehicle. *World Electr. Veh. J.* **2025**, *16*, 494. <https://doi.org/10.3390/wevj16090494>.
17. Ray, A.; Devlin, B.; Quah, F.; Yesantharao, R. Hardcaml: An OCaml Hardware Domain-Specific Language for Efficient and Robust Design. *arXiv* **2023**, arXiv:2312.1503. <https://doi.org/10.48550/arXiv.2312.15035>.
18. Pakonen, A.; Buzhinsky, I.; Björkman, K. Model Checking Reveals Design Issues Leading to Spurious Actuation of Nuclear Instrumentation and Control Systems. *Reliab. Eng. Syst. Saf.* **2025**, *205*, 107237. <https://doi.org/10.1016/j.ress.2020.107237>.
19. Fritzsch, J.; Schmid, T.; Wagner, S. Experiences from Large-Scale Model Checking: Verifying a Vehicle Control System with NuSMV. In Proceedings of the 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST), Porto de Galinhas, Brazil, 12–16 April 2021; pp. 372–382.
20. Architecture Capability Team NATO Architecture Framework. *Version 4. Modeling Guidelines for Use of the UAF DMM*; NATO Consultation, Command & Control Board: Utrecht, The Netherlands, 2019.
21. Radio Technical Commission for Aeronautics (RTCA). *RTCA DO-178C. Software Considerations in Airborne Systems and Equipment Certification*; Radio Technical Commission for Aeronautics (RTCA): Washington, DC, USA, 2011.
22. Object Management Group (OMG). *OMG Systems Modeling Language™ (SysML®). Part 1: Language Specification, Version 2.0*; Object Management Group (OMG): Milford, MA, USA, 2026.
23. Gu, P.; Zhang, Y.; Chen, Z.; Zhao, C.; Xie, K.; Wu, Z.; Zhang, L. X-RMTV: An Integrated Approach for Requirement Modeling, Traceability Management, and Verification in MBSE. *Systems* **2024**, *12*, 443. <https://doi.org/10.3390/systems12100443>.
24. Radio Technical Commission for Aeronautics (RTCA). *RTCA DO-331. Model-Based Development and Verification. Supplement to DO-178C and DO-278A*; Radio Technical Commission for Aeronautics (RTCA): Washington, DC, USA, 2011.
25. Radio Technical Commission for Aeronautics (RTCA). *RTCA DO-332. Object-Oriented Technology and Related Techniques. Supplement to DO-178C and DO-278A*; Radio Technical Commission for Aeronautics (RTCA): Washington, DC, USA, 2011.
26. Radio Technical Commission for Aeronautics (RTCA). *RTCA DO-333. Formal Methods. Supplement to DO-178C and DO-278A*; Radio Technical Commission for Aeronautics (RTCA): Washington, DC, USA, 2011.
27. Gromada, K.; Stecz, W. Designing a Reliable UAV Architecture Operating in a Real Environment. *Appl. Sci.* **2021**, *12*, 294. <https://doi.org/10.3390/app12010294>.
28. Stecz, W.; Kowaleczko, P. Designing Operational Safety Procedures for UAV According to NATO Architecture Framework Designing Operational Safety Procedures for UAV According to NATO Architecture Framework. In Proceedings of the 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST), Online Streaming, 6–8 July 2021; pp. 135–142.
29. Górski, T.; Stecz, W. A Method for Modeling and Testing Near-Real-Time System Scenarios. *Appl. Sci.* **2024**, *14*, 2023. <https://doi.org/10.3390/app14052023>.
30. Park, J.W.; Oh, H.D.; Tahk, M.J. UAV Collision Avoidance Based on Geometric Approach. In Proceedings of the 2008 47th Annual Conference of the Society of Instrument and Control Engineers (SICE), Chofu, Japan, 20–22 August 2008; pp. 2122–2126.
31. Li, L.; Tiniakov, D. UAV Testing and Validation. In *Unmanned Aircraft Design: Features of Design Process*; Springer Nature Singapore: Singapore, 2026; pp. 239–260. https://doi.org/10.1007/978-981-95-7167-3_10.
32. Object Management Group (OMG). *OMG Unified Modeling Language (UML) Specification, Version 2.5.1*; Object Management Group (OMG): Milford, MA, USA, 2017.
33. Rocha, M.; Simão, A.; Sousa, T. Model-Based Test Case Generation from UML Sequence Diagrams Using Extended Finite State Machines. *Softw. Qual. J.* **2021**, *29*, 597–627. <https://doi.org/10.1007/s11219-020-09531-0>.
34. Villani, E.; Pontes, R.; Coracini, G.; Ambrósio, A. Integrating Model Checking and Model Based Testing for Industrial Software Development. *Comput. Ind.* **2019**, *104*, 88–102. <https://doi.org/10.1016/j.compind.2018.08.003>.
35. Dierkes, M. Combining Test and Proof in MBAT: An Aerospace Case Study. In Proceedings of the 2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD), Lisbon, Portugal, 7–9 January 2014; pp. 636–644.
36. Ramesh, S.; Gadkari, A. Combining Test and Proof in MBAT: An Aerospace Case Study. In Proceedings of the 2011 48th Annual Design Automation Conference (DAC), San Diego, CA, USA, 5–10 June 2011; pp. 13–16.
37. Chen, J.; Zhou, H.; Bruda, S. Combining Model Checking and Testing for Software Analysis. In Proceedings of the 2008 International Conference on Computer Science and Software Engineering (CSSE), Wuhan, China, 12–14 December 2008; Volume 2, pp. 206–209.

38. Elkholy, W.; El-Menshawey, M.; Bentahar, J.; Elqortobi, M.; Laarej, A.; Dssouli, R. Model Checking Intelligent Avionics Systems for Test Cases Generation Using Multi-Agent Systems. *Expert Syst. Appl.* **2020**, *156*, 113458. <https://doi.org/10.1016/j.eswa.2020.113458>.
39. Kripke, S. Semantical Considerations on Modal Logic. *Acta Philos. Fenn.* **1963**, *16*, 83–94.
40. Pnueli, A. The Temporal Logic of Programs. In Proceedings of the 1977 18th Annual Symposium on Foundations of Computer Science (SFCS), Providence, RI, USA, 31 October–2 November 1977; pp. 46–57.
41. Clarke, E.; Emerson, E. Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic. In Proceedings of the 1982 Logics of Programs Workshop, Yorktown Heights, NY, USA, May 1982; pp. 52–71.
42. Wu, R.; Du, Y.; Li, M. A Model Transformation Method Based on Simulink/Stateflow for Validation of UML Statechart Diagrams. *Mathematics* **2025**, *13*, 724. <https://doi.org/10.3390/math13050724>.
43. Chodnicki, M.; Nowakowski, M.; Pietruszewski, P.; Wesołowski, M.; Stępień, S. BIZON–UGV for Airport Pavement Testing: Mechanics and Control. *Appl. Sci.* **2024**, *14*, 2472. <https://doi.org/10.3390/app14062472>.
44. Steingartner, W. On Some Innovations in Teaching the Formal Semantics Using Software Tools. *Open Comput. Sci.* **2021**, *11*, 2–11. <https://doi.org/10.1515/comp-2020-0130>.
45. Schreiner, W.; Steingartner, W. Semantics-Based Rapid Prototyping of a Machine Controller Language. In Proceedings of the 2024 IEEE 17th International Scientific Conference on Informatics (Informatics), Poprad, Slovakia, 13–15 November 2024; pp. 337–342.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.