doi: 10.82124/CEAS-GNC-2026-009 CEAS-GNC-2026-009

Toward a Modular Architecture for Reusable System Automation in Aerospace Applications

Mohamad Ammach Research Associate, Institute of Flight System Dynamics, Technical University of Munich, Garching, Germany. mohamad.ammach@tum.de

Evangelos Huber Research Associate, Institute of Flight System Dynamics, Technical University of Munich, Garching, Germany. evangelos.huber@tum.de

Johannes Bender Research Associate, Institute of Flight System Dynamics, Technical University of Munich, Garching, Germany. johannes.bender@tum.de

Florian Holzapfel Prof. Dr.-Ing, Institute of Flight System Dynamics, Technical University of Munich, Garching, Germany. florian.holzapfel@tum.de

ABSTRACT

Automation continues to play an increasingly significant role in aviation, taking over various pilot functions and alleviating pilot's workload. This growing reliance on automation has heightened interest in the development and advancement of automation and flight control systems. This paper presents the architecture of a System Automation (SA) designed for integration into flight control systems across different aircraft configurations, including multi-copters and coaxial helicopters. The proposed architecture is designed to be modular and reusable across various platforms. The SA evaluates and determines the operational state of onboard systems. Each system is modeled using a state machine governed by specific transition conditions. These conditions are generated independently by a separate module using Boolean logic and relational operators. The paper further details the data structures of the modules and the interconnections between them. The architecture is implemented and tested on the SA for both a coaxial helicopter and a multi-copter. Results demonstrate the system's functionality and validate its applicability across different configurations.

Keywords: system automation, vehicle management, state machine, reusability, modular design

Nomenclature

DA	=	Decision Atomics
DM	=	Decision Making
FCS	=	Flight Control System
FMS	=	Flight Management System
MMS	=	Mission Management System
SA	=	System Automation
VMS	=	Vehicle Management System

1 Introduction

Automation has been an integral part of aircraft systems for decades, aiming to reduce the pilot's workload and enhance operational safety. According to [1], more mistakes and mishaps are caused by the pilots than by the mechanical and environmental factors, making the usage of automation even more desirable. However, recent accidents have highlighted a troubling trend: pilots losing situational awareness or failing to understand automated systems adequately. This shift underscores the importance of designing automation that supports, rather than undermines, pilot performance [2].
To ensure safe and effective human-machine interaction, aircraft automation must meet several key design requirements. The first is understandability: the automation must be comprehensible to the pilot. If pilots do not have a clear grasp of the system's functions, they may struggle to respond appropriately. This is especially critical in time-sensitive or emergency situations. The second requirement is transparency: the automation should clearly show both its current actions and intended behavior. When pilots are unsure what the system is doing or planning, the interaction becomes opaque and confusing, even in routine situations [3]. Simplicity is also important: the design should avoid unnecessary complexity. While a certain level of sophistication is essential, too much complexity can lead to misinterpretation, errors, or delayed responses. Flexibility is a fourth requirement: automation should operate in abnormal scenarios and allow manual intervention. Overall, automation should enhance pilot performance by being predictable, intuitive, and supportive. It should not introduce ambiguity or cognitive overload.

1.1 State of the Art

Recent advancements in aircraft concepts, including Vertical Take-Off and Landing (VTOL) platforms, have created a need to redesign automation systems to address the novel requirements of these aircraft. In [4], the author introduces an approach that utilizes state machines within an automation module, referred to as system automation. This approach provides a method for discrete-logic decisions represented by state machines and their relationships. These relationships may exist at a single level or across multiple levels, thereby establishing a hierarchy among the state machines. Inputs to the state machines are processed in a separate module called the input processing module. However, the precise relationship between the processing module and the state machines is not shown.
This approach forms the foundation for subsequent implementations of system automation [5–7]. These implementations, however, differ significantly due to the absence of a standardized framework for system automation architecture. Each implementation presents a different design of the system. This work aims to bridge this gap.
A structure for the system automation (SA) is introduced in [8], dividing it into two primary components: Decision Atomics and Decision Making. In that context, the structure serves mainly as a display layout rather than as the central focus of the work. Moreover, the structure is not clearly defined, and the connections between components remain ambiguous. This paper addresses these ambiguities by presenting a clear and transparent architectural framework.
This work proposes a unified, modular architecture for SA, modeling each aircraft subsystem as an independent entity within the overall architecture. The proposed approach specifically addresses the unique requirements of emerging aircraft types, such as VTOL platforms. The main contributions are outlined below.

- The architecture is structured around two core modules: Decision Atomics, which manages discrete logic, and Decision Making, which integrates and synthesizes outputs from Decision Atomics into a state-based decision process. The modular and general design ensures adaptability and reusability across a wide range of aircraft platforms and operational scenarios.
- The interconnections and interactions between the Decision Atomics and Decision Making modules are thoroughly defined and explained, illustrating how data and signals flow between them, and

establishing standardized protocols for module communication, thereby addressing the ambiguities observed in prior implementations.
- The internal structure of each module is thoroughly described, specifying key components and processing logic to provide a clear blueprint for implementing robust and scalable SA systems.

1.2 Outline

The remainder of this paper is organized as follows. The structure of the flight control system is discussed more in Section 2. In Section 3, the proposed method for unifying the architecture of the system automation is introduced with all its details. The implementation of this method is shown in Section 4, and the results are discussed in Section 5. In the last section, Section 6, a summary of the presented work is given.

2 Flight Control System Overview

The flight control system consists of the vehicle management system (VMS), the mission management system (MMS), and the flight management system (FMS). Figure 1 depicts the structure of the flight control system. Both the MMS and FMS are out of the scope of this paper. The MMS controls all mission-related functionalities and subsystems. The FMS handles the aircraft's control law. It is divided into the manual pilot, the Auto-Flight System (AFS) and maneuver pilot, the inner loop controller, and the trajectory controller. This management system is responsible for controlling the aircraft's flight. The flight control moding module decides how the aircraft is to be controlled. It switches between the different control modes based on the inputs from the VMS.
System Automation (SA), the focus of this work, is a component of the Vehicle Management System (VMS) that determines the functional state of various aircraft systems—such as the landing gear, actuators, and thrust units—based on their behavior. SA does not replicate or interfere with the internal logic of these individual systems. Instead, it operates independently, using feedback from aircraft sensors and pilot inputs (seen in the sensor suite) to infer the current state of each system and contribute to overall system awareness. The SA output is mapped to the modules in the MMS and FMS via an output handler (not shown in the figure).
The Signal Integrity Check & Voting module is responsible for checking the integrity and validity of all the inputs. It then forwards the valid inputs to the system's logic along with a validity status indicating whether the input signals are valid. This module then processes these inputs and decides the current state of the different systems. The details of the working of this module are out of the scope of this paper.

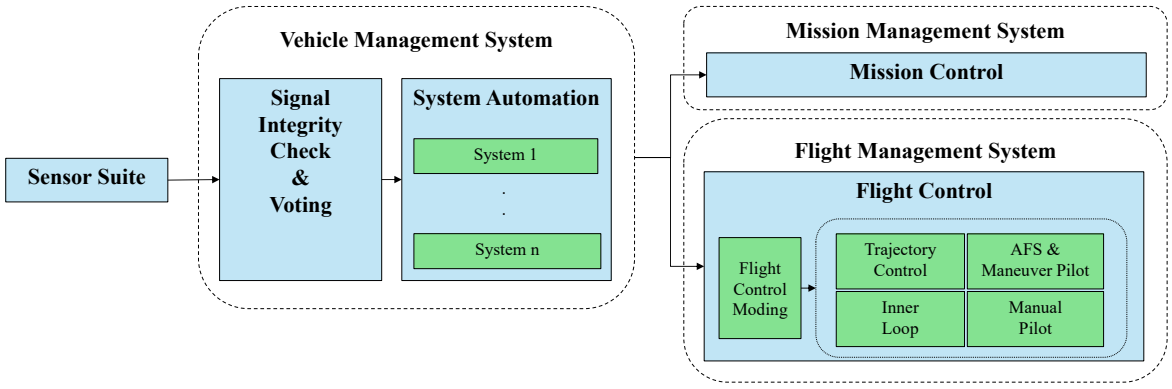

Fig. 1 Structure of the Flight Control System

3 Methodology

The purpose of the presented method is to demonstrate a clear architecture for the system automation, with the aircraft systems and their logic clearly displayed. The architecture is general and reusable, applicable to different aircraft types.
Each system is processed independently, resulting in a modular design for all the systems. Such a design is preferable as it creates independence in design and testing. As seen in Figure 2, the method splits the system into Decision Atomics (DA) and Decision Making (DM). The system's inputs go into the DA, where all signals related to the system are processed. The DA generates the signals used by the DM. The DM contains the state machines depicting the different states of the system.
A system can be divided into subsystems, each with its own DA and DM modules. This creates a clear hierarchy and distribution of the several systems.

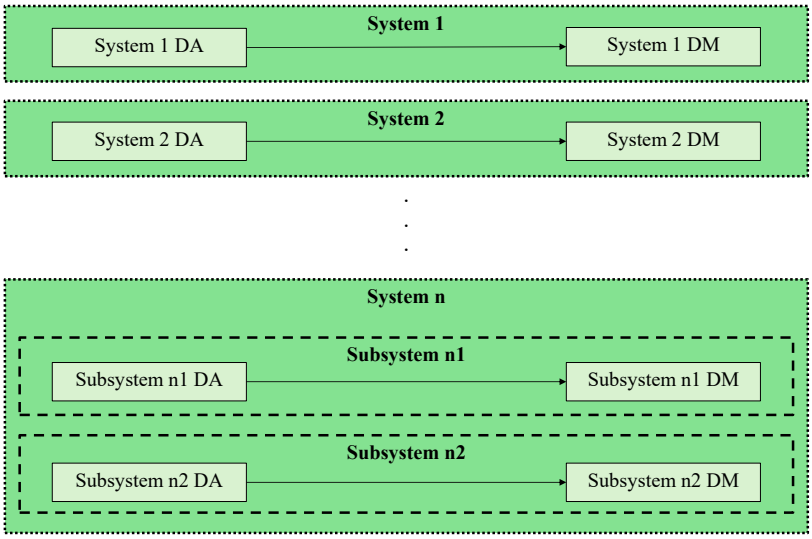

Fig. 2 Structure of the Systems Level with Decision Atomics(DA) and Decision Making(DM)

3.1 Decision Atomics

The Decision Atomics module receives all inputs relevant to the system. These inputs can include feedback from system-specific sensors, aircraft state information, and pilot commands. For example, the landing gear system relies on sensor feedback from the landing gear itself, aircraft state data such as speed, and pilot input given via the landing gear lever.
Within the DA, processing is performed mainly using Boolean logic and relational operators. Relational operators compare signals, such as checking if airspeed exceeds a certain threshold. The result of these operations is a Boolean signal, which is either true or false. The Boolean signals are referred to as flags, which are then used in the DM. It is essential to generate one flag for each transition condition in the state machines to keep the structure as simple as possible. Each flag is used only once in the state machine for a single transition. This way, no feedback from the state machine for the current state is needed. All the processing happens in the DA. The generation of each flag demonstrates the conditions and requirements in order for a transition in the state machine to occur. Such a design shifts the complexity from the state machine to the DA, thereby improving the state machine's readability. The mentioned complexity is more manageable within the DA.
In a state machine, logic is implemented through explicit states, transitions, and conditional actions, which can quickly lead to increased model complexity as the number of conditions or inputs grows. This is known as the state explosion problem, in which each combination of input conditions may require a separate state or transition, leading to a rapid increase in the total number of elements in the model. As

noted in [9], the number of states can grow exponentially with the number of conditional paths, making verification and maintenance increasingly difficult. As logic becomes more detailed, the state machine often requires nested states, parallel states, and numerous conditional transitions, making the diagram harder to follow, debug, and maintain.
In contrast, the same logic can often be expressed more compactly using Boolean logic and relational operators. These allow the designer to implement condition-based logic using logical expressions rather than explicitly creating a separate state for every possible condition combination. For example, instead of defining individual states for each input combination, Boolean and relational operators let you directly express the desired behavior through conditions like "if signal 1 is TRUE and signal 2 is FALSE," which keeps the model flatter and more readable without enumerating all possible states. As a result, Boolean logic scales more gracefully with complexity, avoiding the visual and structural overhead inherent in large state machines. By separating the processing of transition conditions into the DA, with only state changes handled in the DM, the proposed method actively limits the growth of state machine complexity and directly mitigates the state explosion problem.
Figure 3 illustrates an example DA. In this scenario, five input signals are processed. The first two signals are combined using an AND operation to generate the first flag, which triggers the transition from state 1 (S1) to state 2 (S2). The third and fourth signals are compared to check if the third is greater than the fourth. The result is then combined with the negation of the fifth signal using an OR operation, producing the second flag. This second flag represents the condition for transitioning from state 2 (S2) to state 1 (S1). Both flags are then sent to the DM module.

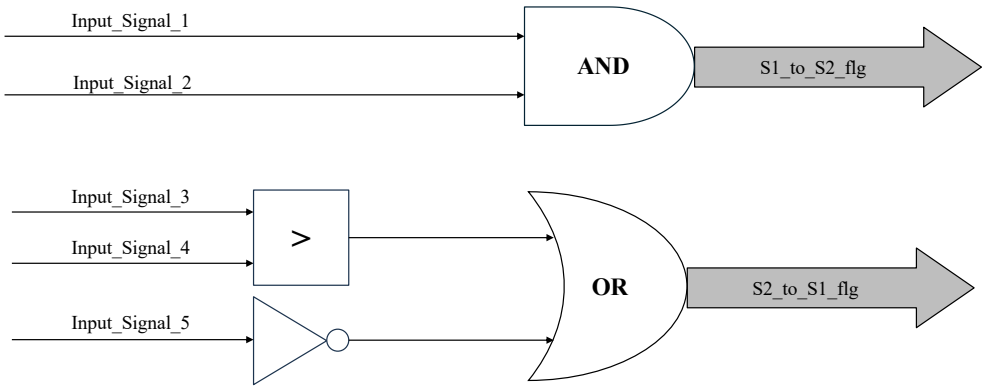

Fig. 3 Exemplary Structure of the Decision Atomics Module

3.2 Decision Making

The DM module consists of a state machine that represents the system's possible states. Transitions between these states are controlled by the flags generated in the DA module. Each transition is associated with exactly one flag; when that flag is true, the transition is triggered. This clear separation ensures that the transition logic remains in the DA, while the DM is solely responsible for managing states.
This design makes the state machine more readable, less prone to errors, and easier to review and test. For example, as shown in Figure 4, the *S1_to_S2_flg* and *S2_to_S1_flg* flags—both created in the DA—directly determine the conditions for transitioning from *State1* to *State2* and vice versa in the DM. By assigning each transition a single, well-defined flag, the functions of the DA and DM are clearly divided, improving modularity and maintainability.
The state machine outputs the system's current state as an enumeration. Enumeration is a data type that declares a series of named constants representing integer values. This can be seen in the *State_Enum* output.

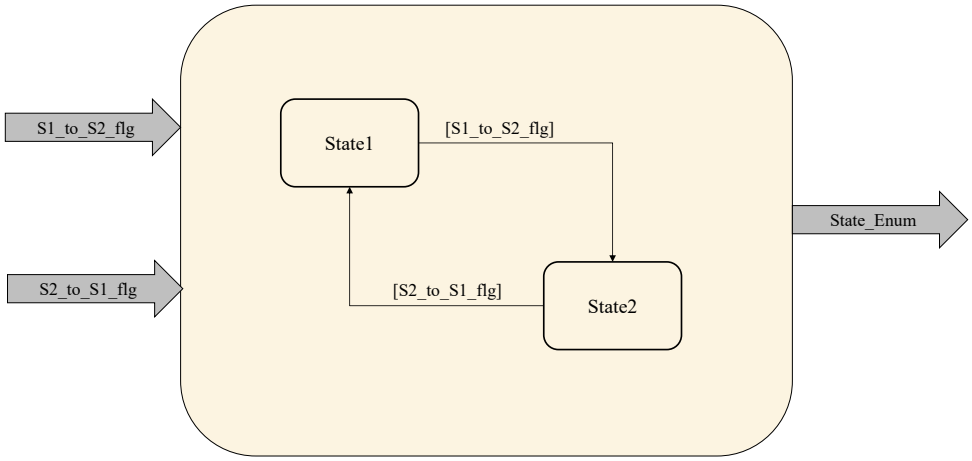

Fig. 4 Exemplary Structure of the Decision Making Module

4 Implementation

The method is implemented in MATLAB Simulink and Stateflow [10–12]. Stateflow is used for the state machines in the DM, while Simulink is employed in the development of other parts of the system. The created models conform with the modeling guidelines developed at the Institute of Flight System Dynamics [13].

4.1 Decision Atomics

The logic for the Decision Atomics (DA) is implemented within a Simulink model, with each system utilizing a dedicated DA model. Boolean algebra and relational operator blocks in Simulink are used to generate flag signals that represent the transition conditions for the Decision Module (DM). All signals are consolidated onto a Simulink Bus that serves as the DA's output. The structure of this bus is defined in a data dictionary associated with the model.

Figure 5 illustrates an example of this implementation. The depicted logic and signals demonstrate the core concept of the DA. The generated flags, *to_UP* and *to_DOWN*, correspond to the transition conditions for the gear states UP and DOWN, respectively. For both transitions, the system verifies that the landing gear compartment door is open using sensor feedback. The gear is raised once the takeoff stage is complete, and it can be lowered during the landing stage when the aircraft's airspeed is below the permissible threshold to prevent structural damage. An emergency condition allows the pilot to override the speed restriction if necessary.

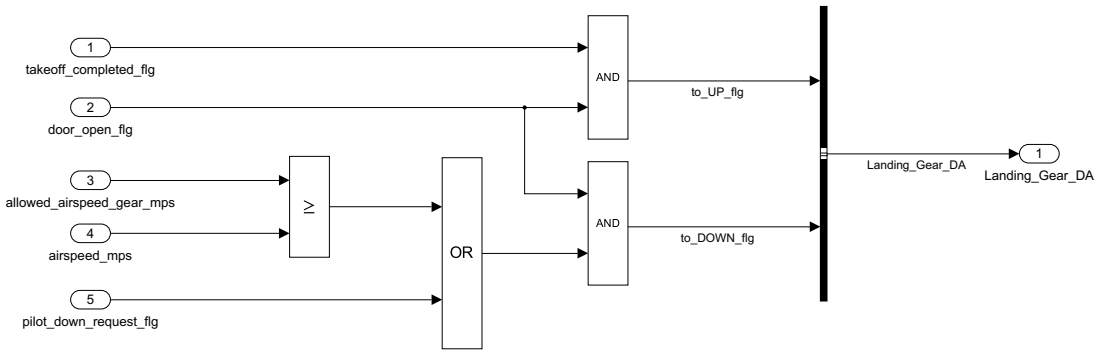

Fig. 5 Implementation of the Decision Atomics Module for a Landing Gear System

4.2 Decision Making

Similar to the DA, the DM is implemented in a Simulink model, that contains the state machine. The state machine is a Mealy machine, whose output depends on both the current state and the input. The Mealy approach is preferred here because it allows the outputs to respond immediately to changes in input conditions, resulting in faster system response times.
There are two states: *DOWN* and *UP*. The default state is *DOWN*, as the system initializes with the aircraft on the ground. The flags generated in the DA are used for state transitions. This structure simplifies the state machine, making it more readable and facilitating debugging and testing. The state machine's output is an Enumeration data type, *LG_State_Enum*, which contains the two values *DOWN* and *UP*, representing the two states.
The DM model is linked to its own data dictionary, which contains the Enumeration output data type. This data dictionary also references the DA's data dictionary to access the DA Bus, thereby linking the DA and DM. Each system contains only the signals and values required for its operation. Cross-referencing between systems is not permitted to ensure architectural independence.

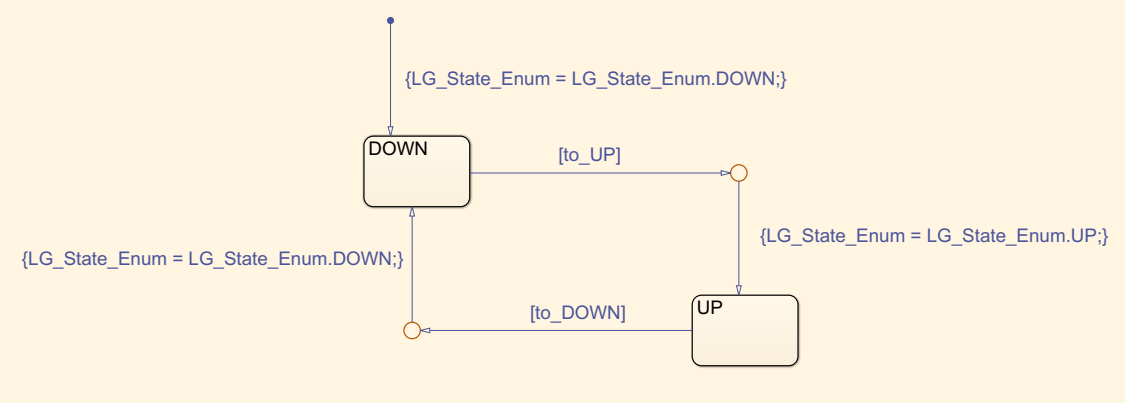

Fig. 6 Implementation of the Decision Making Module for a Landing Gear System

5 Results

The presented architecture is implemented as part of the flight control system for a multicopter and a coaxial helicopter and tested in the MATLAB/Simulink environment. The testing procedure is designed to comprehensively verify system behavior across a wide range of operational scenarios and ensure that outputs match the expected results defined by system requirements.
Testing extensively uses Simulink's built-in Test Harness and Test Manager tools, which provide a controlled framework for automated scenario execution and results validation. The structure of each test case directly reflects the implemented logic—whether realized through Boolean/relational operators in Simulink or modeled as state machines in Stateflow—thereby ensuring precise and targeted evaluation.
A dual testing strategy is employed to systematically address both unit-level and integration-level verification. To maximize objectivity and coverage, these two approaches are conducted independently by separate testers. The first approach leverages an external verification tool introduced in [14]. The second approach applies reusable library blocks, each representing a specific test scenario, to stimulate the system and verify its behavior under clearly defined conditions.
Testing on unit and integration levels ensures that individual systems and models function correctly both in isolation and as part of a larger integrated system. Unit-level testing focuses on evaluating each module independently, which is feasible given the modular architecture and the absence of interdependencies. The Decision Atomics and Decision Making modules are tested together using various input combinations to achieve comprehensive coverage, confirming that all relevant parts of the system are invoked during testing. In contrast, integration testing focuses on validating the correct interaction between

these modules, with success defined by components functioning properly together in coordinated system scenarios. Both layers are indispensable: unit testing builds confidence in each building block, while integration testing ensures reliable performance and communication within the complete system.

5.1 Unit-Level Testing

Unit-level testing is performed to verify the correct operation of each system component independently, with a focus on the Decision Atomics and Decision Making modules. The process employs two complementary methods to ensure thorough and reliable verification.

In the first method, each module is tested within a Simulink test harness, using an automated verification tool as described in [14]. This tool systematically verifies that all requirements are met by executing the module across a comprehensive set of input scenarios and comparing the outputs against expected results. The tool ensures consistency, repeatability, and completeness in requirement coverage, while also enabling rapid identification of discrepancies. Details of the tool's internal operation are beyond the scope of this paper.

The second method centers on reusable library blocks, each representing a specific test scenario. For example, in the scenario where *to_UP_flg* becomes true (as shown in Figure 5), both *takeoff_completed_flg* and *door_open_flg* must also be true. These library blocks act as standardized input generators for the Decision Atomics module within the Simulink test harness. The output from the Decision Making module is then evaluated with an assertion block, also implemented as a reusable library component. This modular approach streamlines test creation, supports easy updates, and ensures consistency across test cases—since changes to a library block instantly propagate to all scenarios that use it.

For each system, all defined test scenarios are executed. Each scenario is implemented as a dedicated test harness, and corresponding test manager files are used to organize and run the tests. All unit tests were successful, with every module producing the correct outputs for every tested input and scenario. This demonstrates comprehensive requirement coverage and confirms the reliability of the individual components. This framework ensures that all specified behaviors are assessed, confirming that the modules consistently produce correct outputs across all tested conditions.

By rigorously testing the Decision Atomics and Decision Making modules in isolation, the approach enables flexible reuse of these modules in different system contexts. For instance, the Decision Atomics module can be tailored for unique requirements—such as distinct landing gear conditions—while still leveraging the verified Decision Making module. This modular testing strategy both simplifies future adaptations and strengthens overall system reliability.

5.2 Integration-Level Testing

Integration-Level testing is performed to validate the complete System Automation (SA) and ensure that all subsystems and modules work together as a unified whole. This phase simulates realistic operational environments by applying a comprehensive set of input values that drive the system through all significant operational states. The aim is to assess how well the different modules—previously tested in isolation—interact and cooperate when combined, ensuring that system-level requirements are satisfied under varied and realistic scenarios.

Both previously described testing methods are employed at this stage. The first method uses automated tools to systematically verify that integrated modules respond correctly to all inputs and produce the expected outputs, enabling rapid identification of integration issues or requirement mismatches. The second method, which leverages reusable library blocks for test scenario construction, proves especially effective at the integration level. These blocks serve as modular input generators and output validators, making it easy to assemble complex, multi-module test scenarios. Modifications to centralized library blocks are instantly reflected across all affected tests, greatly simplifying maintenance and enabling rapid adaptation as system requirements evolve.

Successful integration tests confirm that the fully assembled SA system operates correctly and robustly

in a wide range of situations, verifying not only individual module performance but also their seamless collaboration. This approach highlights the scalability and flexibility of the modular architecture: once modules have been validated in unit testing, they can be confidently reused in new system configurations, with only their integrated behavior requiring retesting. As a result, integration testing both ensures reliable system performance and significantly reduces long-term testing overhead by minimizing duplication of effort.

6 Conclusion

This paper introduced a modular and reusable architecture for a System Automation (SA) within the Vehicle Management System (VMS) tailored for flight control applications in different aircraft configurations, including multi-copters and coaxial helicopters. By modeling each system independently through state machines and generating transition conditions using Boolean logic and relational operators, the proposed approach ensures both clarity and adaptability. The separation between condition generation and state evaluation supports system generality and facilitates reuse in varied applications.
The implementation in MATLAB/Simulink and Stateflow demonstrated the practical applicability of the architecture, and the testing validated its core functionality across multiple configurations. These results confirm the potential of the proposed method to serve as a reusable foundation for managing system states in flight control systems.
Future work will focus on the signal integrity check, which was introduced in Section 2 but not discussed in detail. A key objective will be to illustrate its working principles and highlight its significance within the overall system architecture. In addition, further exploration of testing strategies—particularly through the use of tools based on formal methods—will help to assess the robustness of the SA and its readiness for deployment in operational environments.

Declaration of Use of Artificial Intelligence

During the preparation of this work the author used AI tools in order to proofread the text language. The author reviewed and edited the content afterwards as needed and takes full responsibility for the content of the publication.

References

[1] Scott A. Shappell and Douglas A. Wiegmann. U.s. naval aviation mishaps, 1977-92: Differences between single- and dual-piloted aircraft. *Aviation, Space, and Environmental Medicine*, 67, 1996. ISSN: 2375-6322.

[2] Antonio Chialastri. Automation in aviation. In Florian Kongoli, editor, *Automation*, chapter 5. IntechOpen, Rijeka, 2012. doi: 10.5772/49949.

[3] C. E. Billings. *Aviation Automation: The Search for A Human-centered Approach*. CRC Press, 1st edition, 1997. doi: 10.1201/9781315137995.

[4] Christoph Krause and Florian Holzapfel. Implementing a multi-level finite state machine with matlab simulink and stateflow in the environment of high-integrity aircraft controller software. In *2018 4th International Conference on Control, Automation and Robotics (ICCAR)*, pages 147–151, 2018. doi: 10.1109/ICCAR.2018.8384660.

[5] Christoph Krause and Florian Holzapfel. Designing a system automation for a novel uav demonstrator. In *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1–6, 2016. doi: 10.1109/ICARCV.2016.7838813.

[6] Christoph Krause and Florian Holzapfel. *System Automation of a DA42 General Aviation Aircraft*. 2018. doi: 10.2514/6.2018-3984.

[7] Evangelos Huber, Johannes Bröcker, Tim Rupprecht, and Florian Holzapfel. *Design of the Nominal System Automation of a Cascaded Flight Control Architecture for a Multi-Crew eVTOL Aircraft*. 2024. doi: 10.2514/6.2024-4422.

[8] Valentin Adamov Marvakov. *Automating the Transition of Lift-to-Cruise eVTOL Aircraft*. Phd thesis, Technische Universität München, Garching, DE, February 2023. Available at `https://mediatum.ub.tum.de/doc/1699602/1699602.pdf`.

[9] Edmund M Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, London, Cambridge, 1999. ISBN: 0-262-03270-8.

[10] The MathWorks Inc. "MathWorks". `https://www.mathworks.com`. Accessed: 2026-03-17.

[11] The MathWorks Inc. “Simulink”. `https://www.mathworks.com/products/simulink.html`. Accessed: 2026-03-17.

[12] The MathWorks Inc. “Stateflow”. `https://www.mathworks.com/products/stateflow.html`. Accessed: 2026-03-17.

[13] Konstantin Dmitriev, Shanza Ali Zafar, Kevin Schmiechen, Yi Lai, Micheal Saleab, Pranav Nagarajan, Daniel Dollinger, Markus Hochstrasser, Florian Holzapfel, and Stephan Myschik. A lean and highly-automated model-based software development process based on do-178c/do-331. In *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, pages 1–10, 2020. doi: 10.1109/DASC50938.2020.9256576.

[14] Johannes Bröcker, Tim A. Rupprecht, Evangelos Huber, and Florian Holzapfel. *Automated Verification of State Machines in eVTOL Aircraft: A Lean Development Approach*. doi: 10.2514/6.2024-4426.