Iris Groher, Thomas Vogel (Hrsg.): Software Engineering 2023 Workshops,
150 Digital Library, Gesellschaft für Informatik, 2023

On the Design and Model-Based Validation of Flight Control System Automation for an Unmanned Coaxial Helicopter

Hannes Hofsäß[1], Barzin Hosseini[2], Julian Rhein[3], Florian Holzapfel[4]

Abstract: An existing coaxial helicopter with a maximum takeoff weight of 600 kg has been transformed into a unmanned aerial system. To enable unmanned experimental validation of novel flight control-laws in flight tests, the helicopter's flight control system has been extended by digital components. As the loss of the helicopter platform due to a failure in the experimental digital flight control system has been identified a risk for the project, contingency procedures have been considered in the system design. In this paper, we suggest a model-based and lean development approach for system automation development, which to the best of the authors knowledge, is new. Within the process we suggest to introduce Operational Sequence Diagrams that only represent procedural system behavior to describe the Design Reference Scenarios of the Concept of Operations. Thereby we enable the total exploitation of a System Architecture Behavioral model within the design process, considering the definition of procedures, the validation of architecture design and automated code generation. To validate the suggested approach, we present its application to the design of the system automation for the flight control system of the unmanned helicopter. The system has been successfully integrated and tested in caged flight tests in November 2022.

Keywords: Aircraft Automation, Contingency Procedures, System Operation, Large Scale Drones, UAS, Model-Based Development

1 Introduction

For large experimental or prototype unmanned aerial system (UAS), the correctness of novel functions provided by software cannot be guaranteed and hence, other contingency approaches that ensure safe flight testing operations like tethering need to be applied [CF09]. While these approaches ensure safety, severe damage to the experimental system after a failure is usually accepted and thus, the risk for the project remains high. It follows that there is a need for contingency procedures to reduce the risk of a loss of the system, if the reproduction of the system is not in the scope of the project's budget, like it is for large scale drones in research and prototype projects. In a cooperation of the Technical University of

[1] Technical University of Munich, Institute of Flight System Dynamics, D-85748 Garching, Germany hannes.hofsaess@tum.de
[2] Technical University of Munich, Institute of Flight System Dynamics, D-85748 Garching, Germany barzin.hosseini@tum.de
[3] Technical University of Munich, Institute of Flight System Dynamics, D-85748 Garching, Germany julian.rhein@tum.de
[4] Technical University of Munich, Institute of Flight System Dynamics, D-85748 Garching, Germany florian.holzapfel@tum.de

doi:10.18420/se2023-ws-16

Munich, edm-aerotec GmbH and ZF Luftfahrttechnik GmbH, an existing coaxial helicopter with a maximum take-off weight of 600 kg is transformed into a unmanned aerial system. The applicability of flight control algorithms with a high level of complexity and task reduction for the remote pilot shall be demonstrated for the given flight mechanics of the coaxial helicopter. For that purpose, a first flight test campaign has been successfully conducted in 2022 with the aim of testing the digital sensors utilized for the unmanned system on the target helicopter platform and furthermore to identify the helicopters response dynamics [Ma22] [Ho22]. Furthermore, the helicopter's flight control system has been extended by digital components. As the test plan for the flight control law algorithms foresees an incremental extension of the flight control laws' functionality, a simple and robust contingency procedure is introduced, that ensures the shift of command authority to a beforehand tested flight control law in case of a failure of the untested law. In this paper we describe the development of the system automation function that supports both, the flight test operational procedures under nominal conditions as well as the afore mentioned contingency procedure in case of an emergency. We thereby suggest a holistic approach that leverages the use of an executable model to simulate the behavior of the system for purposes as procedural validation as well as automatic code generation. The approach, that includes the specification of operational procedures based on the trace of the finite-state machine that models the architectural behavior is, to the best of the authors knowledge, new. It allowed us to introduce the desired contingency procedures into the flight test operations while remaining within the limited budget of a research project. In section 2 we describe the development process and point out its unique features. Section 3 gives an introduction to our use-case, describing the subset of the system architecture that is relevant for this work and present the contingency procedures that the system automation needs to support. The system specification is given in section 4 followed by a description of the design of the state machines for system behavior modelling and their transition logics in section 5. Finally, we present the results of numerical simulations with the system model for validation of the integrated design in section 6.

Fig. 1: Test Aircraft utilized for the initial flight test campaign. An aircraft of the same type has been transformed into an unmanned system.

2 Development Process

In figure 2, we embed the herein suggested development steps into a "Vmodel process that is well known in the systems engineering domain. It includes model-based aspects

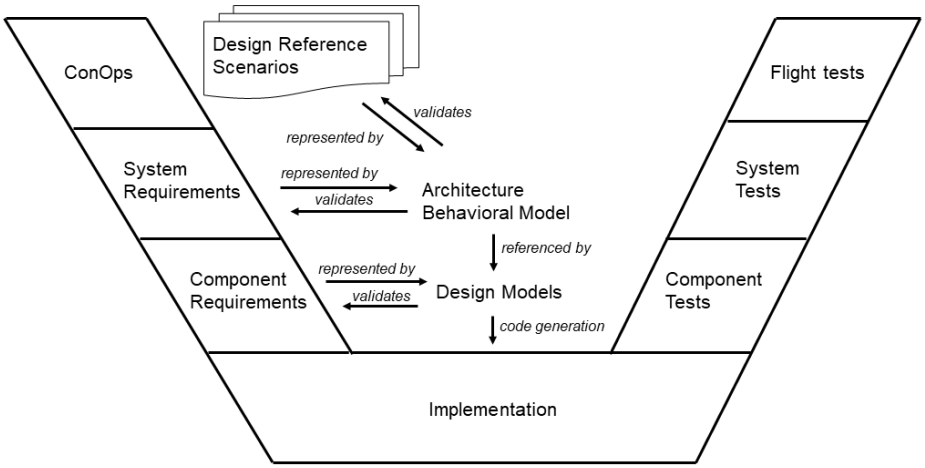

Fig. 2: Suggested development process for system automation functions that exploits an executable Architecture Behavioral Model.

suggested in [An23], like the usage of design models for both, automated code generation and requirements validation. Whereas the author focused on the process from a perspective of developing systems with a desired closed-loop control behavior, our work is complementary in a sense that we focus on the introduction of nominal and abnormal operational procedures by means of automation functionality that explicitly considers the system architecture development activity. The process begins with the capturing of stakeholder expectations in the Concept of Operations (ConOps). The ConOps contains the operational aspects of the system from a user perspective. According to [Ka10], it should consider all aspects of operations including nominal and off-nominal operations during integration and testing. It includes a description of operational scenarios, fault management strategies, description of human interaction and critical events. The operational scenarios describe the dynamic view of the systems operations and include how the system is perceived to function throughout the various modes and mode transitions, including interactions with external interfaces, response to anticipated hazards and faults and during failure mitigation. The authors in [Na20] suggest to include the operational procedures in a tabular way into the ConOps document, which builds up a basis for the automation development. In the following, we refer to the operational procedures as the Design Reference Scenarios.

After an initial ConOps is set up, the system architecture is derived by means of functional analysis. In [Ha21], the architecture is defined as the allocation of functions to elements that

are arranged in a structure and the definition of their interfaces. The authors in [Gi21] have extended common methods like functional decomposition by a Procedure-based Function Derivation for systematic identification of all functions related to the procedures. To provide a basis for the design and development of automation functions, the concept of a Function Life Cycle that provides a description of functional behavior is introduced in [Gi21], with the goal of adding capability to simulate the procedures and associated functions. The concept of behavioral modelling for architecture validation is also introduced in the system development process defined in [Ka10]. No statement is made on modelling concepts or utilized modelling languages in both works. In [Ka16] the authors use the Executable System Engineering Method to create a development environment that supports automated requirements verification using executable SysML. Requirements and executable behavior models are formulated in finite-state machines and integrated into one toolchain to create engineering products and ensure data consistency. The method covers a vast amount of development aspects, however, neither ConOps related aspects like operational procedures are considered nor is automatic code generation for integration with target hardware.

In order to mitigate hazards from failed functional elements in safety-critical systems, the allocation of redundancy is usually applied. If during operation, the desired functionality cannot be provided by the system, it is a common contingency procedure to reconfigure the system to a safe state that may provide a degraded set of functions. A popular example is the Airbus A320 law reconfiguration strategy after certain sensor failures [TLS06]. Other concepts, that are referred to as runtime assurance, introduce functional dissimilarity with the goal of reducing the rigour of verification for complex critical functions, by engaging less performant but fully verified backup functions before the system enters an unsafe state [MCC20]. It is thereby obvious, that the failure mitigation strategies, that are handled on architecture level through the introduction of redundancy and functional reconfiguration, includes an adaption in the behavior of the human operator due to the reduced functional performance or defined contingency plans. We can see that the design choice, that was taken during derivation of the systems architecture, influences the system operation and hence, needs to be reflected in the ConOps such that the stakeholders expectations from the operational domain can be consolidated with the contingency procedures. For that purpose, we suggest to support this development iteration by utilizing two development artifacts:

- An Architecture Behavioral Model that simulates the architectural behavior, such that failure mitigation strategies derived from functional analysis are validated. For modelling of the behavior, we use finite-state machines as they provide an executable and dynamic representation of a behavioral specification, such that validation can easily be performed by simulative assessment of the design reference scenarios. For the process of systematic state elicitation of the finite-state machines, the author in [Vo14] suggests to consider a structured requirements inspection, a function dependency analysis and knowledge from domain experts. Furthermore, various engineering tools support the design and execution of finite-state machines. Due to their formal definition they yield to powerful analysis techniques because it is possible

to explore all possible sequences of states [LV11]. Hence, the model can be exploited for formal analysis e.g. by model based safety assessment methods as suggested in [AW13]. Finally, the model can be setup in a modular way, such that the state charts can be referenced in other models used for code generation, which is supported e.g. by MATLAB/Simulink and Stateflow.

- The architecture definition activity is finished once consistency between system requirements, ConOps and the system architecture is achieved [Ka10]. As operational stakeholders of the system may not be from the technical domain, we suggest an abstracting view onto the architecture behavioral model that helps to include all domains into the discussion. We suggest to use the Operational Sequence Diagram described below to validate the behavior captured by the Architecture Behavioral Model with respect to the Design Reference Scenarios in a graphical representation, that can be reviewed by all domains.

For realizing the design models that fulfill functions that are allocated to specific target hardware elements, the state machines of the Architectural Behavioral Models can be referenced if supported by the applied toolchain. At the Institute of Fight System Dynamics of Technical University of Munich we use Mathworks Simulink and Stateflow Models to integrate the state machines of the Architecture Behavioral Model into the Design Models. For the purpose of code generation, we use a workflow that ensures certain objectives derived from the foundational standards for safety critical software DO-178C/ED-12 and DO-331/ED-216, that has been set up at the Institute of Flight System Dynamics. The details of this workflow are out of the scope of this work and can be found in [Di20].

Finally, the right hand side of the V-Modell follows the well-known process of iterative integration and verification of components and systems with the final goal of validation of the integrated system with respect to stakeholder expectations.

2.1 Operational Sequence Diagram

The architecture definition is finalized once the ConOps, the system requirements and the system architecture are consistent, which is an iterative activity. For that purpose, multiple domains need to convert towards an agreed upon set of Design Reference Scenarios. The usually utilized graphical representation of finite-state machines, the state transition diagram, which is also suggested in [Gi21] by means of a Functional Life Cycle, contains all states with respective transitions. For complex systems, this representation might quickly overwhelm non-developers from other domains. The trace of a finite-state machine represents the state response together with the input sequence that triggers it [LV11]. Thereby it captures the behavior of the finite-state machine and hence suites our purpose of providing a simplified representation of the behavioral system architecture model. In the following, we refer to the trace of the Behavioral System Architecture model as an Operational Sequence Diagram, which fulfills two tasks in our suggested process:

- It provides a graphical format to validate the system behavior in reviews with project members from other domains and to implement the Design Reference Scenarios in the ConOps.
- It supports the definition of initial draft versions of the Architecture Behavioral Model. The Operational Sequence Diagrams may not necessarily need be derived *after* an initial Architecture Behavioral Model has been designed by means of a finite-state machine. The method of abstract state machines seems to be well suited to fit into the herein suggested process, as it allows to define so called "ground models"that constitute a blueprint of the implemented piece of real world, thereby providing means for model validation by simulation and property verification via mathematical proof [Bö10].

3 System Description

Due to the iterative process of architecture design and ConOps definition, we here assume an initial architecture design has been derived and step into the process that validates the Design Reference Scenarios. The core components of the digital flight control system are given in figure 3. Pilot inputs are received by the remote control (RC) through the available

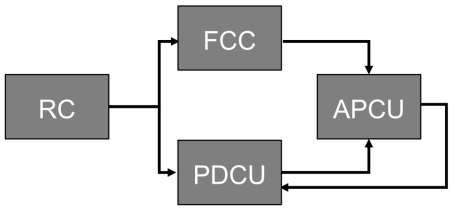

Fig. 3: Block diagram of the core components of the digital flight control system and the functional communication channels utilized for system automation.

human-machine interface (HMI) elements like sticks or switches and transmitted to the onboard receivers, that are omitted here for simplicity. The Flight Control Computer (FCC) is the primary source of actuator commands, that has enough computational resources available to host complex experimental flight control laws in its application layer. The system architecture is equipped with an air data, attitude and heading reference system and multiple backup inertial measurement units that provide the required feedback measurements like body rates and euler angles. As they play a secondary role for the automation considered in this paper, we omit them in the figure. The Pilot Data Concentrator Unit (PDCU) is an additional digital unit with less computational resources that hosts the backup law, which is a rigid direct law (DL) that maps the stick deflections to the control variables of the actuator position control unit (APCU). FCC and PDCU provide their computed actuator control commands to the APCU via dedicated communication channels. The APCU merges the received commands based on a simple selection logic, that prioritizes PDCU commands as

soon as they are received, and provides control of the actuators, which are as well omitted in the figure for simplicity. Finally, the PDCU receives feedback of the APCU about the availability of signals, transmitted by the FCC. The system architecture furthermore features redundant power supplies, onboard receivers and feedback sensors for inertial data. The APCU has been planned to host a high level of internal redundancy such that the system could also fulfill reliability-related requirements. The details of these activities are out of the scope of this paper and we consider the architecture design as given. Two contingency procedures drove the design of the automation. They can be formulated by the folowing system requirements:

- During flight, the pilot shall be capable of engaging the direct law upon a single request.
- In case of a loss of the FCC, the PDCU shall take over command authority.

The first procedure is driven by the experimental nature of the flight control laws that shall be tested. As the probability of unintended behavior is higher for flight law algorithms that are yet untested in flight, the desired contingency procedure is the degradation to a beforehand tested direct law. Obviously, the direct law needs to be tested first without an available backup, but that risk has been accepted within the project. The second procedure has been defined to mitigate the risk of severe system damage due to hardware failures of the FCC. Even though the developed system has been tested to operate on the target plattform in grounded tests - high onboard vibrations have been identified as a risk as they cause physical loads on hardware-communication and power-supply interfaces, as well as the hardware-internal circuits. Finally, it was declared an aim of the project to increase the experience in the domain of fault-tolerant flight control system design at the Institute of Flight System Dynamics of the Technical University of Munich.

4 Specification

In the following, we present the definition of the system states, the pilot interfaces and present an exemplary Design Reference Scenario in an Operational Sequence Diagram in this section.

4.1 Automation States

The design approach of the system states widely follows the hierarchical decomposition structure that is suggested in [KH18]. Figure 4 provides an overview of the states allocated

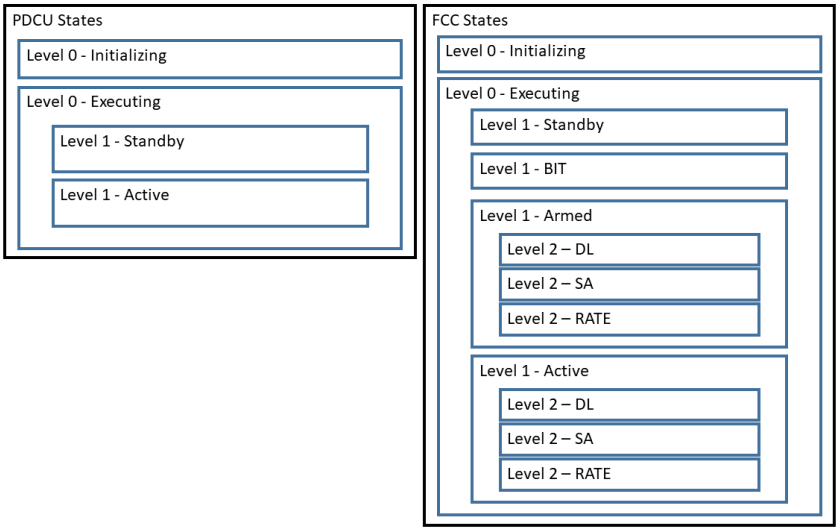

Fig. 4: Hirarchical decomposition structure of component states for PDCU (left) and FCC (right).

to the PDCU and FCC which includes the hirarchies of the states that are defined by several levels:

- Level 0: This Level has two states, Initializing and Executing. After boot-up of the component, it will start in the Initializing state, waiting for confirmation timers in the input data handing of the application layer to timeout, such that a valid integrity state for available inputs can be defined. In Initializing state, the component remains silent on its communication output channels linked with the APCU. Transition to Executing is triggered after all timeouts have passed.

- Level 1: This Level defines the component behavior with respect to digital output communication channels that are linked with the APCU.
  - Standby: The component remains silent at its communication channels that are linked with the APCU, no data is transmitted.
  - Build-In-Test (BIT): The component executes a build-in-test to verify the correct APCU response to predefined output signals.
  - Armed: Pre-conditions are fulfilled, such that the component is ready to transmit actuator command outputs of a flight law to the APCU, no data is transmitted.
  - Active: The component transmits actuator command outputs of a flight law on the APCU communication channel.

- Level 2: This level defines the flight law that produces the command outputs of the component transmitted to the APCU.
  - Direct Law (DL): A rigid feed forward law that maps the stick deflections of the RC to the actuator control variables.
  - Stability Augmentation (SA): An augmented direct law that utilizes body rate feedback to increase the damping of the helicopter response.
  - Rate Law (RATE): This law interprets stick deflections of the RC as desired rotational rates and considers body rotational rate feedback to shape the desired helicopter response to stick deflections.

The details of the flight laws are out of the scope of this work, in future it is furthermore planned to include higher modes of feedback control such as attitude and translational rate response types.

4.2 Pilot Interface

Besides the definition of the system states, we need to define the pilot's inclusion into the operation of the system. From the first system level requirement above, we can already derive, that the pilot needs to perform a dedicated action to engage the direct law. Hence, the system needs to provide suitable interfaces for all procedures that include an action of the pilot. According to [Bi96] a main challenge in designing system automation for flight control systems is complexity. Increasing automation complexity inherently increases task complexity for the operator, which again increases the opportunities that the system is used beyond its capabilities or without regard to its constraints or rules. Therefore, our design approach aims at minimizing task complexity by clearly assigning input signals to state transition requests that are easily interpretable. The following set of logical input signals has been identified:

- DL engage and disengage request: The signal that engages the direct law.
- BIT request: The signal that triggers the transition from Standby to BIT state in the level 1 FCC automation.
- FCC engage and disengage request: The signal that triggers the transition from Armed to Active in the level 1 FCC automation.

- Flightlaw request: The signal that triggers the transitions between the flight law states in the level 2 FCC automation

After their definition, we allocated the signals above to switches on the RC. All switches are two-position toggle switches except for the switch that hosts the law requests, which has three positions. We decided to use switches instead of pushbuttons, especially for the engage and disengage requests, as switches give a direct visual feedback of the current requests to the pilot, thereby easing the pilot's mode awareness. Besides the pilot HMI via the RC, the system includes a ground station that receives data from all system components for monitoring of all relevant internal logical states and output signals.

4.3 Operational Sequences

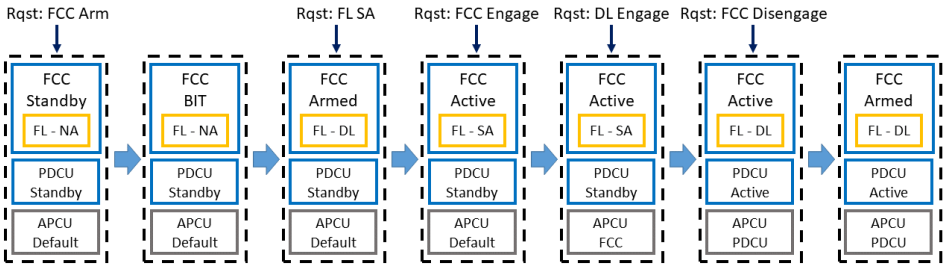

Fig. 5: Operational Sequence Diagram for an exemplary Design Reference Scenario: Switch of command authority to the DL hosted on the PDCU by a single pilot request while the system has started up with the FCC in command which is defined as the nominal system startup procedure.

An example for an Operational Sequence Diagrams that captures a Design Reference Scenario is depicted in figure 5. The illustrated procedure shows a take-over of command authority with the PDCU upon a single pilot request for the direct law. If the pilot triggers the dedicated request for the direct law on the remote control, the PDCU level 1 state transitions to Active, as the reason for the pilots request is likely to be a failure on the FCC, which hosts the experimental laws. Hence, a conservative provision of the direct law is performed by shifting the command authority to the PDCU. Note that the FCC also degrades its internal mode on level 2 to the direct law, in case it operates nominally. By this procedure, the intended take-over capability can furthermore be verified in a flight test without a failure of the FCC actually being required. Thereby we reduce the risk during the test as the pilot is prepared for a degradation in the control law and no pilot reaction delay needs to be considered. The sequences are plotted event-based, such that each of the blue arrows indicates a transition in at least one component state. This corresponds to the trace of the finite-state machine that implements the Architectural Behavior Model that is presented in section 5. Transitions can therefore only be triggered by external events like

pilot requests depicted by vertical arrows, or by time delayed transitions that occure after entry into certain states, like the transition from BIT to Armed state in the level 1 states of the FCC. Failure events are also considered as external event inputs.

5 Design

The design of the Architecture Behavioral Model has been carried out in MATLAB Simulink and Stateflow models. After their simulative validation based on the reference scenarios as presented in section 6, we integrated the state machines into the Simulink application layer Design Models for the FCC and the PDCU, from which C-code was automatically generated. Beneath the state machines, these models contained the functionality of the control laws as well as further input and output handling functionalities. In this section we describe the design of the logics that trigger the transitions between the states defined in section 4.1. Similar to the procedure in [Kr20], we present the state machines and summarize its input and output signals in interface tables. Due to their simplicity we omit the presentation of the level 0 logics, which are the same on FCC and PDCU. After boot-up, the *Initializing* state is entered and a transition to the *Executing* state is triggered several cycles later. Initialization is defined to be the time interval until confirmation timers at the application layers input handling module, which is not discussed in this work, have timed out, such that system health data that is provided to the system automation can be considered to be valid. The

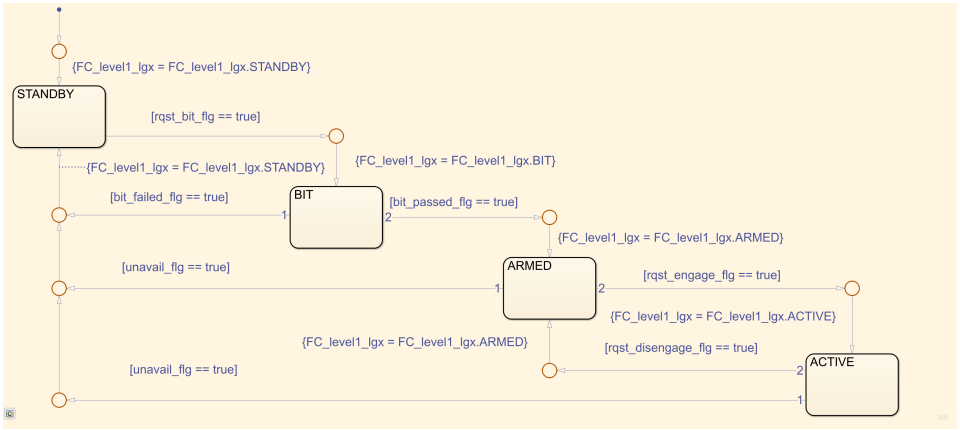

Fig. 6: FCC Level 1 state machine.

FCC level 1 state machine is given in figure 6. The transition action in the initial transition is necessary, as the FCC level 1 state machine is disabled as long as the FCC level 0 state machine is in the initializing state. The respective structure of state machines in the Simulink design is given in [KH18]. Hence, the output `FC_level1_lgx` is set to its initial value, which is defined to be *unused*. The request-based transitions must be triggered by the pilot via the HMI inputs defined in the previous section. When entering the *BIT* state, an automatic

Tab. 1: FCC Level 1 Interface

Name	Direction	Datatype
`rqst_bit_flg`	input	boolean
`unavail_flg`	input	boolean
`rqst_engage_flg`	input	boolean
`rqst_disengage_flg`	input	boolean
`bit_passed_flg`	input	boolean
`bit_failed_flg`	input	boolean
`FC_level1_lgx`	output	enumerated

build in test can be executed, to verify the correct response of the aircraft to predefined sequences of FCC actuator command signals. The detailed design of the built-in test is out of the scope of this work. The `bit_passed_flg` and `bit_failed_flg` are set according to the automatic assessment of the built-in test and hence the operation of the system with the FCC is forbidden in case of a built-in test failure. The transitions to the *Standby* state from the *Armed* and *Active* states are coupled to the `unavail_flg`. This signal is set to true based on the functional capability that the FCC can provide. As the main task of the FCC is to host flight control laws, a minimum set of valid input data needs to be received by the application layer, such that any flight law can provide valid outputs. In our case, all flight laws hosted by the FCC need at least valid data transmitted by the RC as it contains the stick deflections as required inputs of all flight control laws and the aforementioned pilot discretes. Hence, the `unavail_flg` is set to true if no valid RC data is received in the input handling section of the FCC application layer. Figure 7 illustrates the state machine of the FCC

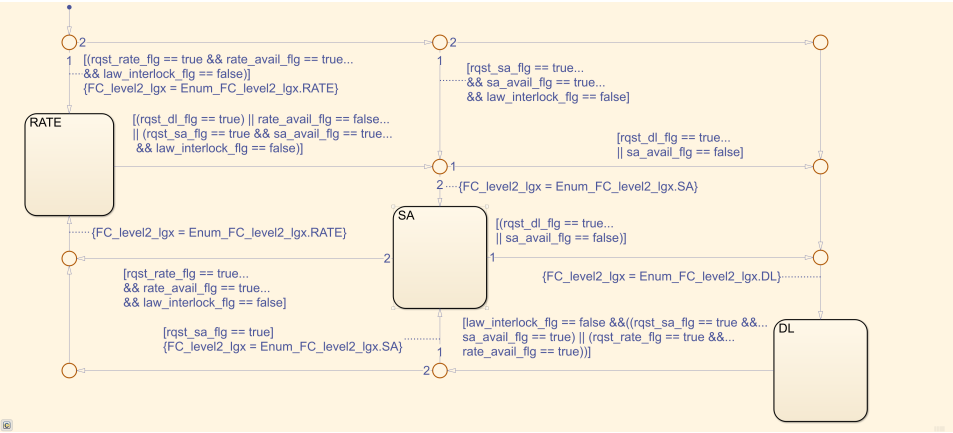

Fig. 7: FCC Level 2 state machine.

level 2 states. Similar as above, the Simulink design ensures that the state machine's output signal `FC_level2_lgx` is set to *unused* as long as the level 1 logics is not in the *Active* or the *Armed* state. For the initial flight tests, the opportunities of triggering a mode transition in the FCC level 2 logics should have been reduced to the minimum possible to reduce the

Tab. 2: FCC Level 2 Interface

Name	Direction	Datatype
`law_interlock_flg`	input	boolean
`rqst_rate_flg`	input	boolean
`rqst_sa_flg`	input	boolean
`rqst_dl_flg`	input	boolean
`rate_avail_flg`	input	boolean
`sa_avail_flg`	input	boolean
`FC_level2_lgx`	output	enumerated

occurrence of transients in the FCC actuator command outputs to a small set of predefined system states. Hence, the system should forbid the pilot to trigger state transitions to *SA* or *RATE* when controlling the helicopter in flight. We accounted for this functionality by introducing the `law_interlock_flg` and set it true if the FCC level 1 logics is not in *Armed* state or if the DL is requested by the dedicated switch for the DL request on the RC. Note that the input signal `rqst_dl_flg` is defined as a logical "or "operation of the DL engage request that is triggered via its dedicated switch and the Flight law request being set to DL which is equal to the switch being in the down position. Furthermore, we allow automatic degradations of the active flight law due to missing resources required to obtain a valid output of the respective flight law. For example, if body rate measurements are not received by the FCC application layer, both rate and stability augmentation cannot produce valid outputs, while the direct law does not require these measurements as it is designed in a feed forward manner. Hence, an automatic transition to the direct law would be triggered as the `rate_avail_flg` and the `sa_avail_flg` would both be set to false. Finally, figure 8 shows

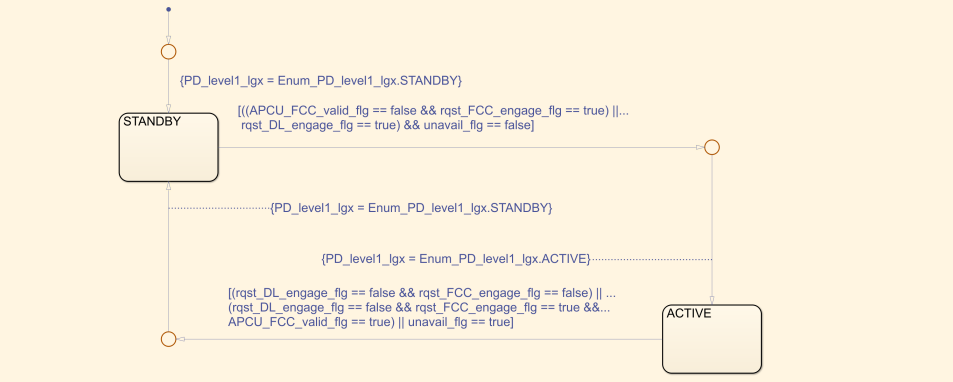

Fig. 8: PDCU Level 1 state machine.

the state machine for the PDCU level 1 states. We designed the initialization of the output the same way as for the FCC level 1 states. The transition from *Standby* to *Active* state considers the `dl_rqst_engage_flg` as well as the `fc_rqst_engage_flg` that are both commanded by the pilot discrete signals on the RC. As long as all conditions for the PDCU are met to

Tab. 3: PDCU Level 1 Interface

Name	Direction	Datatype
APCU_FCC_valid_flg	input	boolean
unavail_flg	input	boolean
rqst_dl_engage_flg	input	boolean
rqst_fcc_engage_flg	input	boolean
PD_level1_lgx	output	enumerated

provide valid direct law outputs, it will either start transmitting commands to the APCU if the pilot requests the direct law via the dedicated switch or if the pilot requests the FCC to be engaged, but the APCU does not receive the FCC commands. This could be the case if the FCC loses power supply during flight or if an internal monitoring of the FCC prevents the transmission of commands to the APCU after the detection of a failure to maintain behavioral integrity throughout the system. Hence, this transition logics implements the second contingency procedure defined in section 3 as it ensures the take-over of command authority by the PDCU if the FCC has failed. Still, the pilot can operate the system without an available FCC by only commanding the dedicated DL engage and disengage request.

6 Validation

In this section, we validate the design described above in Simulink simulations. For that purpose, we set up a Simulink test harness model and integrated the described state logics of FCC and PDCU that we also use for code generation into the model. This simple integration model serves as a behavioral model of the integrated real-world-system and hence, we use it for model-based validation of the system behavior with respect to the specification given by the operational sequences defined in diagrams like the one in figure 5. The state transitions can be triggered by sequences of simulated pilot HMI input signals as defined by the interfaces of the state machines in tables 1, 2 and 3. The correctness of the state transitions can be assessed by the state machine outputs, that are designed to describe the currently active state, and by a respective `cmd_transmit_flg` of the FCC and the PDCU application layers. These signals trigger the command transmission in the respective driver software that was designed by hand-written C-code and hence is out of the scope of this work. The input sequence on the left triggers the state logics on the right in the order specified by the operational sequence diagram. Validation is performed by assessing the state transitions provided by the simulation with respect to the transitions specified in the diagram. The results for the simulated operational sequence that is specified in figure 5 are depicted in figure 9. After boot-up, we simulate the pilot to start the build-in-test by operating the mechanically centered switch at t = 1s and releasing it at t =2s. The FCC transitions to the BIT state which is shown in the plot for `FCC_level1_lgx` and transitions to the *Armed* state after the built-in test has passed at t = 3s. Next, we set the `rqst_FL_lgx` to the stability augmentation law at t = 5s. As this request is allocated to the mechanically

latched three-position-switch, this action is equal to setting the switch to its middle position. As a result, the `FCC_level2_lgx` transitions to the *SA* state. Flipping the switch for the FCC engage request to the up position at t = 6s causes the `rqst_FC_engage_flg` to switch to true and hence, `FCC_level1_lgx` transitions to *Active* and the `FCC_cmd_transmit_flg` is set to true, indicating that the component now starts transmitting its command outputs to the APCU. After t = 8sec we simulate to flip the switch to engage the DL by setting the `rqst_DL_engage_flg` to true. Thus, `PDCU_level1_lgx` transitions to *Active* state and the PDCU starts transmitting direct law commands as a result of the `FCC_cmd_transmit_flg` switching to true. As the APCU component hosts a simple source selection that selects PDCU commands if command data is received and else selects FCC commands if received, a switch of command authority from FCC to PDCU has now been accomplished. Furthermore also the `FCC_level2_lgx` transitions to *DL* and hence direct law commands are transmitted also by the FCC. Finally, at t = 9s, we simulate to flip the FCC engage request back to the down position by resetting the `rqst_FC_engage_flg` to false and the `FCC_level1_lgx` transitions to *Armed* state, the `FCC_cmd_transmit_flg` settles to false, indicating that the FCC stops transmitting commands to the APCU. A switch in the `APCU_FCC_valid_flg` is not used for this specific scenario. Still, we can validate the behavior of the PDCU as it remains silent as long as the `APCU_FCC_valid_flg` and the `rqst_FC_engage_flg` remain true and the `rqst_DL_engage_flg` remains false. The design has been validated in multiple further scenarios like the one presented here, using the abstracted integration model. Also the automatic take-over of command authority in case of a loss of the FCC has been validated by means of the scenario specification.

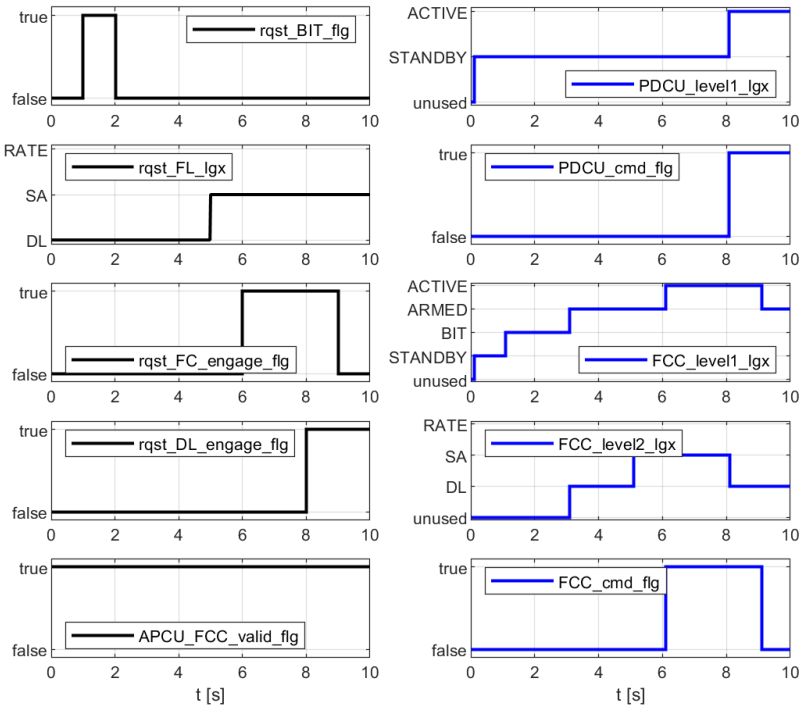

Fig. 9: Result of a numerical simulation with the Architecture Behavioral Model that integrates the state logics of PDCU and FCC.

7 Conclusion

In this work, we present the model-based design and validation of the system automation for the digital flight control system of an unmanned coaxial helicopter with a maximum take-off weight of 600 kg that is developed in a cooperation of the Technical University of Munich, edm-aerotec GmbH and ZF Luftfahrttechnik GmbH. We follow a holistic approach that centers an executable model of the system architectures behavior as the source for operational validation activities and code generation. We propose the novel definition of Operational Sequence Diagrams that provide a simplified view onto the Architecture Behavioral Model by means of the trace of its finite-state machines for specification of Design Reference Scenarios in the ConOps. Furthermore, we suggest to integrate the Architecture Behavioral Model in the Design Models used for automated code generation by means of model references. Thereby we ensure consistency throughout all design activities with a lean process. To the best of the authors knowledge, this development approach is new. It is

applicable for development projects that consider the loss of a large demonstrator aircraft as a key project risk. Leveraging the idea to validate design assumptions by an abstracted finite-state machine before designing an Architecture Behavioral Model is content of future work. We present the application of the process for an exemplary function that realizes the switch of command authority between two sources of actuator commands. Our approach allowed us to efficiently introduce the desired contingency procedures, has contributed to a seamless hardware integration phase and thereby reduced the overall project risk. The system has been tested in caged flight tests in November 2022 and performed as expected, tests in free flights are planned for early 2023.

Acknowledgments

The author H. Hofsaess has contributed the definition of the design process, the design of the Architecture Behavioral Model presented in the application use case and its behavioral representation in Operational Sequence Diagrams. The requirements and HMI Interface definition was joined work. Regarding realization of the caged flight tests J. Rhein has contributed largely in interface management and system integration testing at the Institute of Flight System Dynamics. Barzin Hosseini has largely contributed in the integration of the system into the demonstrator aircraft, as well as in the execution of the caged flight tests. The authors thank the team that has performed the caged flight tests, including Franz Sax (TUM - FSD), Lukas Maier, Aaron Barth (both TUM - HT), Thomas Petzold (edm aerotec GmbH) and the edm aerotec GmbH, who provided facilities and support during the test campaign. This work was funded by the Federal Government of Germany as part of the LuFo program (funding ID: 20Y1705C).

Supported by:

on the basis of a decision
by the German Bundestag

References

[An23] Angelov, J. N.: Model-Based Systems Engineering of Flight Control for VTOL Transition Aircraft, Diss., Institute of Flight System Dynamics, Technical University of Munich, 2023.

[AW13] Abdulkhaleq, A.; Wagner, S.: Integrating state machine analysis with system-theoretic process analysis. In: Lecture Notes in Informatics (LNI) - Proceedings, Volume P-215. 2013.

[Bi96] Billings, C. E.: Human-Centered Aviation Automation: Principles and Guidelines, Technical Memorandum, National Aeronautics und Space Administration, 1996.

[Bö10] Börger, E.: The Abstract State Machines Method for High-Level System Design and Analysis. Springer London, London, 2010, ISBN: 978-1-84882-736-3.

[CF09] Cooke, A.; Fitzpatrick, E.: Helicopter Test and Evaluation. Jhon Whiley und Sons, 2009.

[Di20] Dimitriev, K.; Zafar, S. A.; Schmiechen, K.; Lai, Y.; Saleab, M.; Nagarajan, P.; Dollinger, D.; Hochstrasser, M.; Holzapfel, F.; Myschik, S.: A Lean and Highly-automated Model-Base Software Development Process Based on DO-178C/DO-331. In: AIAA/IEEE 89th Digital Avionics Systems Conference (DASC). 2020.

[Gi21] Gierszewski, D. M.; Nagarajan, P.; Jaisle, J.; Krammer, C.; Maly, M.; Hozapfel, F.: Demonstration of a Procedure-Based Approach to Functional Analysis for an Optionally Piloted Vehicle. In: AIAA Scitech 2021 Form. 2021.

[Ha21] Haberfellner, R.; de Weck, O.; Fricke, E.; Voessner, S.: Systems Engineering. Springer Nature Switzerland AG, 2021.

[Ho22] Hosseini, B.; Sax, F.; Rhein, J.; Holzapfel, F.; Maer, L.; Barth, A.; Hajek, M.: Global Model Identification for a Coaxial Helicopter. In: Vertical Flight Society's 78th Forum and Technology Display. 2022.

[Ka10] Kapurch, S. J.: NASA Systems Engineering Handbook. DIANE Publishing, 2010.

[Ka16] Karban, R.; Dekens, F.; Herzig, S.; Elaasar, M.; Jankevicius, N.: Creating system engineering products with executable models in a model-based engineering environment. In: SPIE Astronomical Telescopes + Instrumentation. 2016.

[KH18] Krause, C.; Holzapfel, F.: Implementing a Multi-Level Finite State Machine with MATLAB Simulink and Stateflow in the Environment of High-Integrity Aircraft Controller Software. In: International Conference on Control, Automation and Robotics. 2018.

[Kr20] Krause, C.: Safe and Robust Automation of Aircraft and System Operation, Diss., Institute of Flight System Dynamics, Technical University of Munich, 2020.

[LV11] Lee, E. A.; Varaiya, P.: Structure and Interpretation of Signals and Systems. Addison-Wesley - Pearson Education Inc., 2011.

[Ma22] Maier, L.; Hosseini, B.; Barth, A.; Holzapfel, F.; Hajek, M.: On the Flight Test Campaign of a Coaxial Helicopter for the Development of an Unmanned Aerial System. In: AIAA AviationForum. 2022.

[MCC20] ad Michael DeVore, E. M. P.; Cooper, J.; Carr, G.: Run Time Assurance as an Alternate Concept to Contemporary Development Assurance Processes, Technical Report, NASA, 2020.

[Na20] Nagarajan, P.; Maly, M.; Jaisle, J.; Gesting, P.; andMaximilian Wechner, R. S.; Gierszewski, D.; Krammer, C.; Holzapfel, F.: Taking Autonomy Out-of-the-Loop - Novel Methodoogy for the Development and Automated Operation of UAS in Integrated Airspace. In: 2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC). 2020.

[TLS06] Traverse, P.; Lacaze, I.; Souyris, J.: Airbus Fly-By-Wire: A Process Toward Total Dependability. In: 25th International Congress of the Aeronautocal Science. 2006.

[Vo14] Vogelsang, A.: Model-based Requirements Engineering for Multifunctional Systems, Diss., Faculty for Informatics, Technical University of Munich, 2014.