SUBSTANCE

Research article

AEROSPACE, INFORMATION AND COMMUNICATIONS TECHNOLOGIES

Developing Avionics Software

Andres Paz Ghizlane El Boussaidi

SUMMARY

Aircraft are increasingly relying on software for the control of their avionics systems' behaviour. Hence, research into more effective software engineering technologies is needed to reduce software and development complexities and to support airworthiness certifications. However, there is no detailed documentation on open source avionics software development readily available for research that can act as a benchmark to support the evaluation of proposed engineering approaches. To address this issue, we have developed the specification and design of a landing gear control software (LGCS, see Figure 1) with our own proposed methodology. In this article, we summarize the methodology and give an overview of the LGCS specification and design. We also discuss some of the issues we faced.

Requirement Specification and Design Methodology for Avionics Software

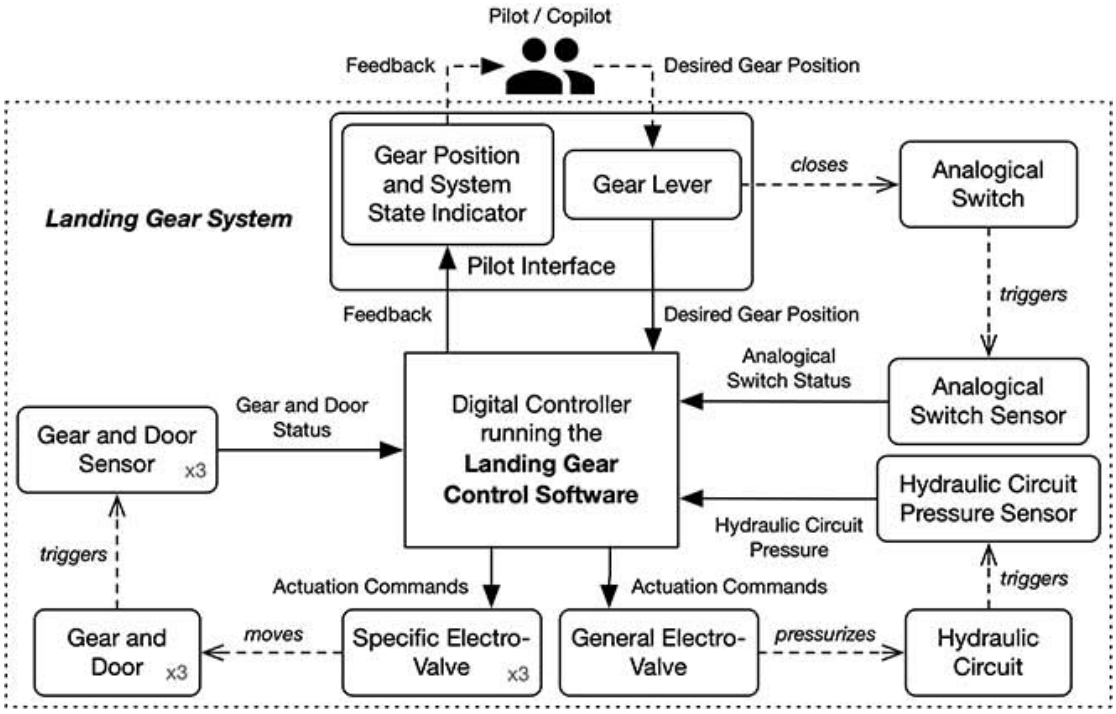

Figure 1. The LGCS and its operational context

Avionics software development is largely constrained by the DO-178C [1] (and its European equivalent ED-12C). To the best of our knowledge, no methodology in building software requirements specification and design under DO-178C has been reported in the literature. Therefore, we have defined one. We give a simplified, high-level view of this methodology in Figure 2. The process begins when a set of high-level system requirements and system safety requirements is allocated to software (SRATS). The requirements specification process covers the development (refinement and decomposition) of these SRATS into high-level software requirements (HLRs). A review of the HLRs validates these against the SRATS to check that their intent is being accurately captured by the HLRs in a way that is suitable in directing the software design process. The software design process covers the software architecture definition and the specification of low-level software requirements (LLRs). LLRs, together with the software architecture, are developed from the specified HLRs and are used to guide the source code implementation.

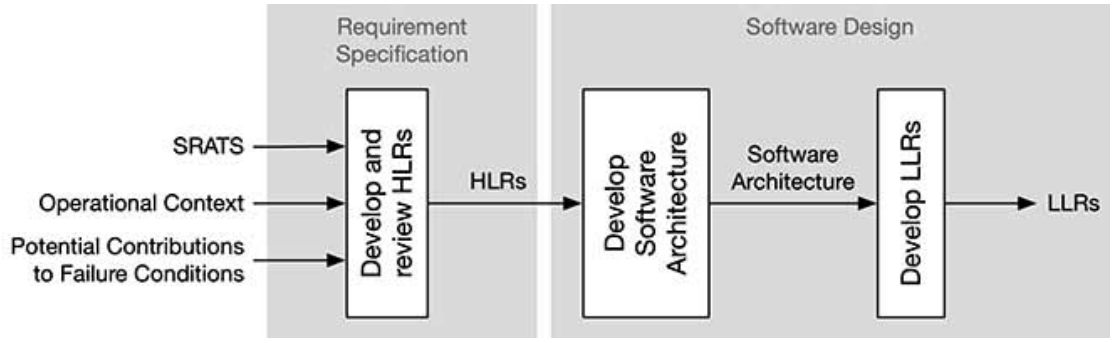

Figure 2. General flow of the proposed methodology

The activities may be sequential but DO-178C promotes this as a way of favouring meticulous engineering to build a safe product. Nevertheless, we have organized the actions within the activities to form iterative and incremental cycles that gradually build their outputs until they yield a software requirement specification and design that is suitable in directing the following phases in the development process: coding and verification.

Developing HLRs, LLRs and Software Architecture

Avionics systems are routinely specified in natural language [2–4]. Thus, our methodology considers SRATS and HLRs to be captured using natural language. However, natural language is not a suitable form of specification to support requirements-based analyses and verification due to its inherent ambiguities [5]. Therefore, before refining and decomposing SRATS, it is necessary to perform a manual review for ambiguities, inconsistencies and undefined conditions. This was the case with the LGCS specification. For this specification, we used as SRATS descriptions for a conventional landing gear arranged in a tricycle configuration, presented in [6]. These descriptions were given in natural language and contained several issues. Thus, we started by correcting and improving them before developing the HLRs for the LGCS. In the interest of avoiding similar problems in the HLRs specification, we employed a form of controlled natural language. This involved specifying them in a consistent manner as statements of how the software will change a set of *controllable variables* (*i.e.* variables the software can directly affect) in response to changes in a set of *monitorable variables* (*i.e.* variables the software responds to). Table 1 shows a subset of the HLRs developed from the SRATS.

Table 1. Examples of HLRs for the LGCS

ID	Description	Rationale	Traces	Precluded CFCs
HLR-4	When the LGCS is currently executing a retraction sequence and a **Down** value is received for the **Desired Gear Position**, the LGCS shall halt the current retraction sequence and revert all the actions that were executed. Likewise, when the LGCS is currently executing an extension sequence and an **Up** value is received for the **Desired Gear Position**, the LGCS shall halt the current extension sequence and revert all the actions that were executed.	A gear motion can be canceled by the pilot or copilot at any step of an ongoing sequence. Any action executed of the ongoing sequence has to be reverted.	SRATS-4 LLR-35	
HLR-6	Once the overall value of the **Hydraulic Circuit Pressure** is greater than or equal to 30,000 kPa and less than 35,000 kPa after the **General EV Actuation Command** is set to **Open**, the LGCS can set to **Open** the necessary specific EV.	This is the specified oil pressure needed in the hydraulic circuit for operating the specific electro-valves.	SRATS-6 LLR-14 LLR-43 LLR-44 LLR-45	
HLR-12	Once 2 seconds have elapsed since the **General EV Actuation Command** was set to **Open** and the overall value of the **Hydraulic Circuit Pressure** is still less than 30,000 kPa, the LGCS shall detect a failure of the general hydraulic electro-valve and halt the currently executing sequence.	The hydraulic circuit should be pressurized in less than 2 seconds after the general electro-valve has been stimulated. The hydraulic circuit being unpressurized after this time is an indication of its failure. If a failure is detected the system should not be not be trusted to perform correctly and, therefore, the LGCS shall halt its operation.	SRATS-12 LLR-44 LLR-56 LLR-57	CFC-3

DO-178C highlights the importance of avoiding the introduction of complexities during the design process. Therefore, we incorporated principles of software engineering that help minimize complexities and promote verifiability, such as modularity and encapsulation [7]. The result was the UML component diagram in Figure 3. We started by identifying an architectural style in line with the expected functionality of the LGCS. The closest match was the *Process Control* architectural style [8], where a system uses a set of inputs to determine a set of outputs that will produce a new state of the environment. Based on this architectural style we then defined the software components that would realize the functionality expressed in the HLRs. Each HLR was apportioned to a subset of the components, as seen in Figure 3.

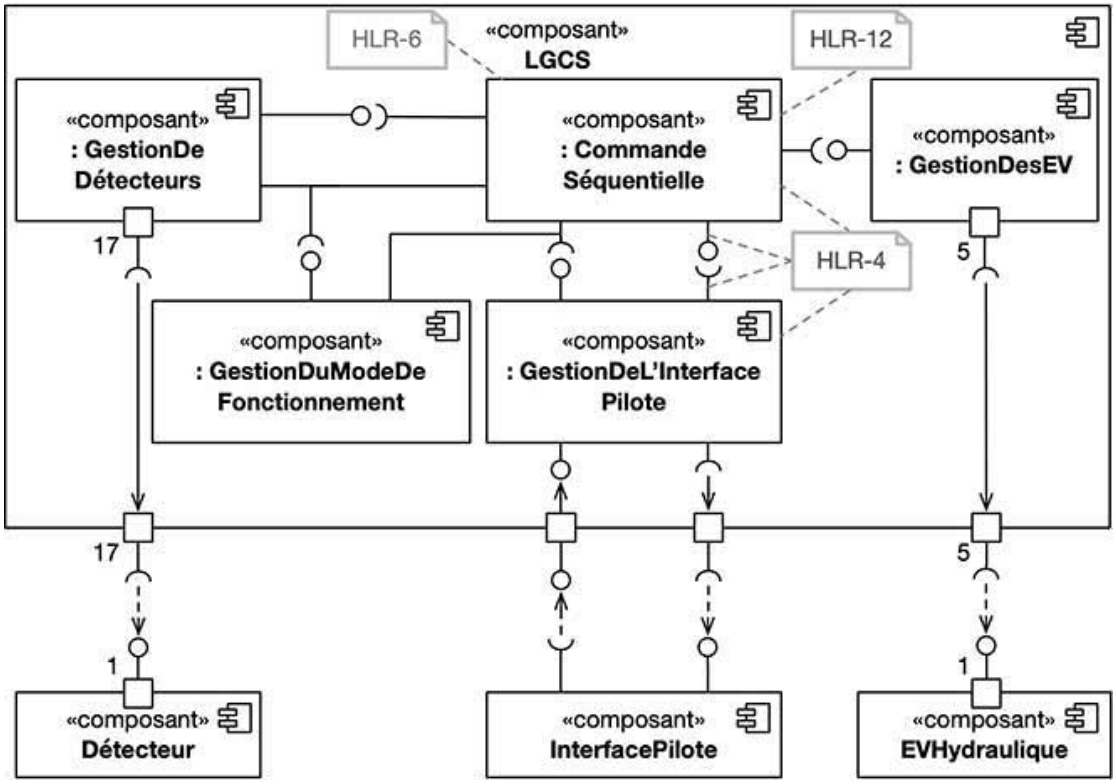

Figure 3. LGCS Architecture

Two parallel LLRs specifications were created: a textual specification using natural language (since they are requirements) and a model-based specification using UML state machines (since they can be regarded as the detailed software design). For the textual specification we used the same controlled natural language strategy employed with the HLRs. Table 2 shows a subset of the textual LLRs.

Table 2. Examples of LLRs for the LGCS

ID	Description	Traces	Component
LLR-35	If the waitForHydraulicPressure method is active and the RevertEvent is raised, all the actions that were previously executed shall be reverted.	HLR-4	SequenceController
LLR-43	If the General EV Actuation Command is set to Open, the waitForHydraulicPressure method shall be activated.	HLR-6	SequenceController
LLR-44	If the waitForHydraulicPressure method is active and the overall value of the Hydraulic Circuit Pressure monitorable variable is less than 30,000 kPa, the waitForHydraulicPressure method shall remain active until the PressurizationTimeoutEvent is raised.	HLR-6 HLR-12	SequenceController
LLR-45	If the waitForHydraulicPressure method is active and the overall value of the Hydraulic Circuit Pressure is greater than or equal to 30,000 kPa and less than 35,000 kPa, the waitForHydraulicPressure method shall end and the PressurizationEvent shall be raised.	HLR-6	SequenceController
LLR-56	If the waitForHydraulicPressure method is active and 2 seconds have elapsed since the General EV Actuation Command was set to Open, the PressurizationTimeoutEvent shall be raised.	HLR-12	SequenceController
LLR-57	If the waitForHydraulicPressure method is active, the PressurizationTimeoutEvent is raised and the overall value of the Hydraulic Circuit Pressure monitorable variable is less than 30,000 kPa, the FailureEvent shall be raised.	HLR-12	SequenceController

Figure 4 shows an excerpt of an equivalent model-based specification as a UML state machine of the LLRs in Table 2.

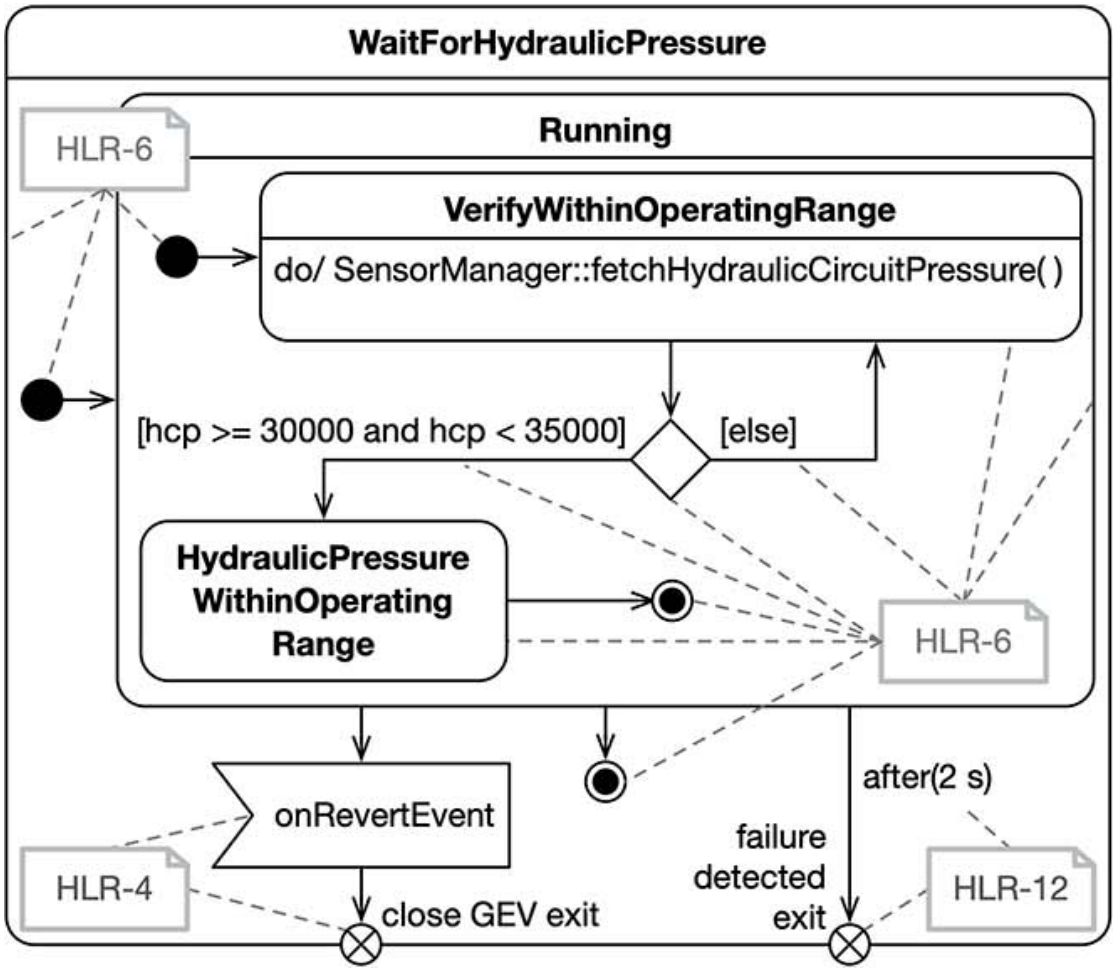

Figure 4. Excerpt of the UML state machine for the SequenceController component

Discussion

We developed the LGCS specification and design in compliance with the current avionics software regulation (DO-178C). We built them iteratively, having them verified by several domain experts. We summarize our observations throughout the process in four points:

1. *Quality and granularity of SRATS.* We started the process with SRATS containing several inconsistencies, ambiguities and confusing wording. Hence, they had to be corrected and clarified before proceeding with the development of HLRs. Therefore, the more clear, precise and complete the SRATS are, the easier the job of the software development team becomes.

2. *Language used for requirements specification.* We used a form of controlled natural language for both HLRs and LLRs to avoid ambiguities. LLRs were also specified using design models. Specifying HLRs using a formal language is still a challenge and is part of future work in order to support requirements-based verification.

3. *Granularity of LLRs.* DO-178C expects LLRs to be very detailed in order to enable the implementation of source code. This may be misleading when wanting to express conditions as close as possible to the code. However, DO-178C requires a greater separation between LLRs and code than pseudocode or action languages can provide. Thus, developing the LLRs for the LGCS with an appropriate level of granularity was challenging.

4. *Bi-directional traceability.* We used comment blocks in the design models to achieve backward traceability from software architecture and LLRs to their source HLRs (see Figures 3 and 4). However, forward traceability, *e.* linking HLRs to their developed LLRs, which is also required by DO-178C, was not possible with design models.

Although this is a work in progress, we have made the LGCS specification and design open and available to researchers. In doing so, we are making it possible to use it as a benchmark avionics software specification that will be used in the analysis of different engineering problems across the avionics industry and the evaluation of their proposed solutions.

Additional information

For more information on this research, please refer to the following conference paper:

Andrés Paz and Ghizlane El Boussaidi. Building a Software Requirements Specification and Design for an Avionics System: An Experience Report. In Proceedings of SAC 2018: Symposium on Applied Computing, Pau, France, April 9–13, 2018 (SAC 2018). DOI: 10.1145/3167132.3167268

Andres Paz

Andrés Paz is a PhD student in the Department of Software Engineering and IT at ÉTS. He is working on a DO-178C-compliant model-driven approach to support the development and certification of safety-critical avionics software.

Program : Software Engineering

Research laboratories : LASI – Computer System Architecture Research Laboratory

Ghizlane El Boussaidi

Ghizlane El Boussaidi is a professor in the Department of Software Engineering and IT at ÉTS. She specializes in model-based software engineering, and software architecture and design.

Program : Software Engineering Information Technology Engineering

Research laboratories : LASI – Computer System Architecture Research Laboratory

References

1. Software Considerations in Airborne Systems and Equipment Certification. Standard DO-178C, Radio Technical Commission for Aeronautics (RTCA), Inc., 2011.

2. Blouin. Modeling languages for requirements engineering and quantitative analysis of embedded systems. PhD thesis, Université de Bretagne-Sud, Dec. 2013.

3. Potter. Complying with DO-178C and DO-331 using Model-Based Design. Technical report, MathWorks, 2012.

4. Schamai, L. Buffoni, N. Albarello, P. Fontes De Miranda, and P. Fritzson. An aeronautic case study for requirement formalization and automated model composition in Modelica. In Proceedings of the 11th Int. Modelica Conf., Versailles, France, September 21-23, 2015, number 118, pages 911-920, 2015.

5. Moy, E. Ledinot, H. Delseny, V. Wiels, and B. Monate. Testing or Formal Verification: DO-178C Alternatives and Industrial Experience. IEEE Software, 30(3):50-57, May 2013.

6. Boniol and V. Wiels. The landing gear system case study. In ABZ 2014: The Landing Gear Case Study. Springer, 2014.

7. Bialy, V. Pantelic, J. Jaskolka, A. Schaap, L. Patcas, M. Lawford, and A. Wassyng. Software engineering for model-based development by domain experts. In Handbook of System Safety and Security. Syngress, 1st edition, 2017.

8. S. Levine. The control handbook. The electrical engineering handbook series. CRC Press New York, 1996.

Images references

The feature image was bought from Istock.com and is protected by copyrights.