ELABORATION OF SAFETY REQUIREMENTS

Kristina Forsberg, Eva Mårbring Isaksson, Saab AB, Jönköping, Sweden
Barbara Gallina, Kristina Lundqvist, Achille Penna, Mälardalen University, Västerås, Sweden

Abstract

According to the aircraft standard ARP4754A, requirements should be carefully traced and validated. A systematic methodology for safety requirements elaboration (refinement/decomposition as well as allocation management) is lacking. To overcome this lack, an ARP-aligned and DOORS implementable approach called RAP (Requirements Allocation Process) is proposed. RAP offers a textual as well as graphical means for managing safety requirements. Besides supporting requirements decomposition and allocation, RAP also supports design decisions. The usefulness of RAP is illustrated by an example, applying the approach to a High Lift System.

Introduction

The increasing number of requirements, coming from the possibility to integrate and combine functions, has increased the complexity level of avionics systems, to a point where it is troublesome to validate the requirements. In particular integration and increased complexity is problematic regarding safety requirements since integrated software intensive systems have made it harder to assess dependencies and fault propagation.

ARP4754A [1] plays a crucial role when developing avionics systems. This is a preferred guideline and recognized by FAA as an acceptable method for establishing a development assurance process [2]. ARP4754A prescribes close interactions between the safety assessment process and the system development process (see Figure 1), in order to capture safety requirements imposed on the design. However, more "What" than "How" is discussed in ARP4754A. Therefore, in this paper, we elaborate on hands-on methods to refine, decompose and validate safety requirements. The lack of a systematic method to refine system safety requirements is also recognized by EADS Germany in [3] where an approach that explicitly specifies the transition from abstract system requirements to concrete item requirements is presented. This transition is based on domain knowledge and properties, while our method is based on safety assessment methods targeting fault tolerance concepts and properties. The gray box in Figure 1 shows the focus of this paper.

Safety objectives are defined at aircraft level and often expressed as "loss of function" or "erroneous function". This high-level abstraction makes safety requirements differ from functional requirements in the sense that they are not measurable or testable. Since safety is a property of a system in use the safety objectives are not easily visible at the item (hardware/software) level. However, safety requirements are met by making the functional architecture evolve, by incorporating and combining fault-tolerant redundant items aimed at achieving the objectives. This incorporation is highlighted in the development process described in ARP4754A.

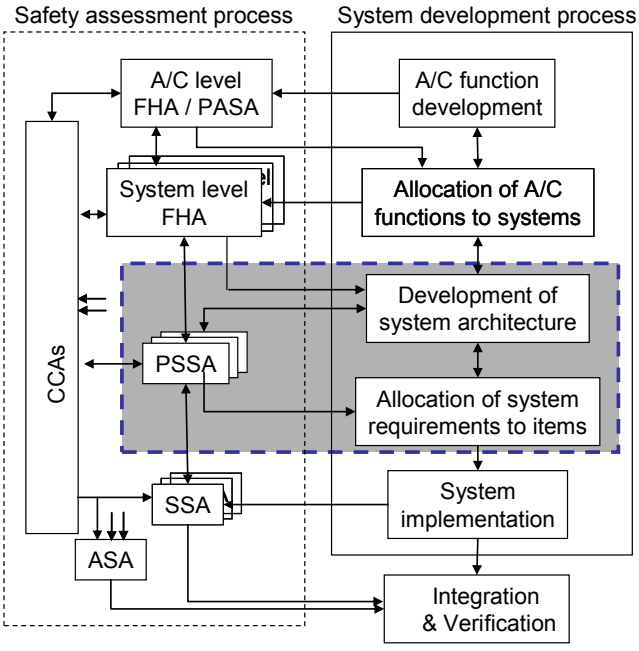

Figure 1. Safety assessment and development processes

The alignment between the safety assessment process and the system development process can be

seen in Figure 1. The left side (dotted) comprises the safety assessment processes:

- FHA (Functional Hazard Assessment): Examines aircraft and system functions to identify potential functional failures and classifies the hazards.
- PASA/PSSA (Preliminary Aircraft Safety Assessment/Preliminary System Safety Assessment): Establishes the aircraft (A/C) or specific system safety requirements and provides a preliminary indication that the anticipated aircraft or system architectures can meet those safety requirements.
- ASA/SSA (Aircraft Safety Assessment/System Safety Assessment): Evaluates systematically and comprehensively the implemented A/C and systems to show that relevant safety requirements are satisfied.
- CCA (Common Cause Analysis): Provides the tools to verify required independence, or to identify specific dependencies.

These safety assessment processes should be interleaved with the development phases, see Figure 1. However, operational guidelines to support safety managers during the evolution (carried out via e.g., decomposition/refinement) of safety requirements are missing. The contribution of this paper is a proposed method that details the interaction between the safety assessment process and the system development process. This alignment is called RAP (Requirements Allocation Process).

To develop RAP, we use the lessons learned from a previously developed safety critical system where refinement and decomposition of safety requirements was done late in the program. That is, the alignment between the two processes (see Figure 1) was not fully in place. As a result validation of assumptions on software integrity was overseen in the PSSA phase. This lead to a common mode problem, which was captured later on during the SSA process by safety engineers.

The rest of the paper is organized as follows. In the background section, some fundamental information onto which presented work is based. The core sections present the proposed method RAP for tracing and validating safety requirements; followed by showing RAP in usage by applying it to a high lift system and discussing its usefulness. Finally, some concluding remarks and suggestions for future work are presented.

Background

This section recalls the essential background on which the presented work is based: typical characteristics of safety-critical avionics systems, requirements traceability and main characteristics of a high lift system.

Safety-critical Avionics systems

Airplane safety is based on principles and techniques of the fail-safe design concept, which considers the effects of failures and combinations of failures in defining a safe design. Critical systems must be designed such that they will not fail in unknown ways. Thus, designers are expected to take into account all types of faults, e.g., random hardware faults and design faults in both hardware and software, together with the causality relationships that may lead to severe failures.

For avionics systems, an aircraft-level safety assessment establishes how critical a given function is depending on how severe a failure would be. From that criticality safety objectives are derived taking into account the aircraft-level architecture implementing the function. The key safety objectives placed on any equipment or system installed in an aeroplane is derived from CS-25 [4]. The paragraph 25.1309(b) in [4] requires that the aeroplane systems and associated components considered separately and in relation to other systems, must be designed so that:

1. Any catastrophic failure condition (i) is extremely improbable; and (ii) does not result from a single failure; and
2. any hazardous failure condition is extremely remote; and
3. any major failure condition is remote.

The corresponding numerical values of extremely improbable and/or remote are also found in [4], e.g.,

"Extremely Improbable Failure Conditions are those so unlikely that they are not anticipated to occur during the entire operational life of all aeroplanes of one type, and have a probability of the order of 1 x 10^{-9} or less. Catastrophic Failure

Conditions must be shown to be Extremely Improbable." [4].

The reliability objective of one computing lane can reach the order of 10^{-5} failures per flight hour, for loss of function, thus at least two redundant lanes are required to achieve this high safety objective. However, for integrity level there is no known methodology to calculate failure rate of design faults or to mathematically prove the absence of design faults thus fail-safe behavior is demonstrated through a combination of development assurance and independence and dissimilarity of redundant items/functions.

The safety objectives for Software and Complex Electronic Hardware (CEH) are not addressed by quantitative failure rate analyses so a development assurance approach is used in accordance with RTCA/DO-178C [5] and RTCA/DO-254 [6]. The development assurance approach is fundamentally different than hardware failure rate analysis. The applicable Development Assurance Level (DAL) for a given function, as determined by the worst case failure effect, are not considered independent and must be maintained throughout all elements of the system that contribute to that function. Five levels are used, A to E. These levels span from negligible (DAL E) to catastrophic (DAL A) hazards. For example, the display of attitude is a critical function. The loss of all attitude information is considered a catastrophic event. The safety objective for a catastrophic event is 10^{-9}. In a system that includes three similar attitude sensors, each with a hardware probability of 10^{-3} for loss of attitude data, the probabilities may be logically AND'd as the hardware failures of each sensor are considered independent. Therefore the safety objective can be achieved. Failures are considered common cause for both software and CEH, and the DAL for each of the three sensors must be Level A as determined by the catastrophic effect of the worst-case failure.

Safety requirements drive the architecture where several options are available for combining redundancy and diversity. It must be noted that software diversity designed by using N-version programming, as discussed in [7,8], cannot be proved to guarantee statistical independence and thus its claimed effectiveness for increasing reliability should be taken with caution and considered as an unproved hypothesis.

Requirements traceability

To keep track of the requirements evolution (e.g., decomposition and refinement), various approaches can be adopted (e.g., tabular formats and tree structures) and several tools, which implement those approaches, are at disposal. DOORS (Dynamic Object-Oriented Requirements System) [9] is one of them. DOORS is a commercial tool for requirements management and it offers a rather powerful support for tracing requirements as well as filtering. Requirements may be categorized as well as linked showing one-to-one, one-to-many relationships. Categories may be used to filter desired subsets.

Our approach is based on DOORS. Figure 2 shows a typical requirement tree where arrows (inlinks and outlinks) illustrate trace between DOORS modules. A requirement trace, for instance, relates the safety requirement SRz with its decomposed requirements SRz' and SRz''.

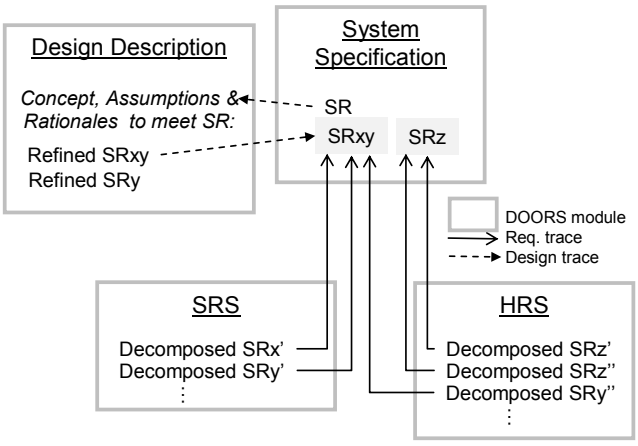

Figure 2. Requirement Tree System to Item level

How a system level safety requirement is decomposed/met by a design is shown in the requirements tree, characterizing requirements traceability.

In this paper we use *refinement* to denote the evolution of a high-level safety requirement from system to sub-system level. As the design develops, the safety requirement needs to be refined to address the design. Refined requirements are fed back to the system specification.

We use *decomposition* if a requirement is broken down from higher to lower level, e.g., sub-system to item SRS/HRS (SW/HW Requirements Specifications).

Refinement or decomposition is needed to split a complex requirement into a set of simpler more manageable, readable and more suitable requirements e.g., for allocation, implementation as well as verification (testability) purposes.

In the context of ARP4754A, safety requirements exist at the aircraft, system, and item level. Moreover, within this de-facto standard only decomposition is mentioned. Thus refinement is treated as a special case of decomposition, i.e., a requirement is decomposed into a set of simpler requirements.

High Lift System

The system under examination in this paper is a High Lift System (HLS). HLS extends and retracts trailing edge flap used to increase lift during slow flight. A flap is a movable portion of the wing that can be lowered into the airflow to produce extra lift to limit take-off and landing speeds. High lift control systems can be driven hydraulically, electrically or with hybrid technology. HLS consists of a set of components that are installed along the wing (shaft, actuators, sensors) in cockpit (Flap Lever) and avionic bays (control and monitoring computers, CMC). The main HLS functions are:

- Position the flap surfaces according to the pilot's command.
- Automatic control. Modify the input command for aircraft safety reasons (to protect the structure against outsized loads and avoid structural damage).
- Ensure fail-safe operation.
- Provide actual flap surface angle of the HLS to other A/C systems via output buses.

The pilot's control commands are sent from a single flaps lever located on the center console in the cockpit. The CMCs receive and process the commanded flap position and output corresponding control signals to a power control unit. A mechanical shaft transmission transmits the output torque to actuators, illustrated in Figure 3. Brakes are located at each motor output shaft and between drive stations of the transmission system.

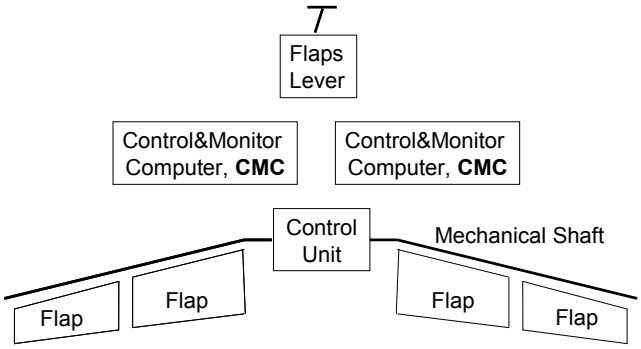

Figure 3. Overview of the High Lift System

The HLS in our example incorporates two CMCs for availability only ("loss of function" is classified as major). The high integrity safety requirements are allocated on each CMC.

Requirements Allocation Process

As mentioned in the introduction, ARP4754A prescribes the interaction between the safety assessment and system development processes. Here we present our method Requirements Allocation Process, called RAP, for implementing the interaction between PSSA work and the system design development see Figure 4 (extracted from Figure 1). RAP is performed in four steps comprising the evolution (step-1 and step-3) and validation (step-2 and step-4) of safety requirements. The Fault Tree Analysis (FTA) [10] method is used to provide a systematic support for requirements break down combined with a requirement traceability tool such as DOORS for safety requirements traceability and validation.

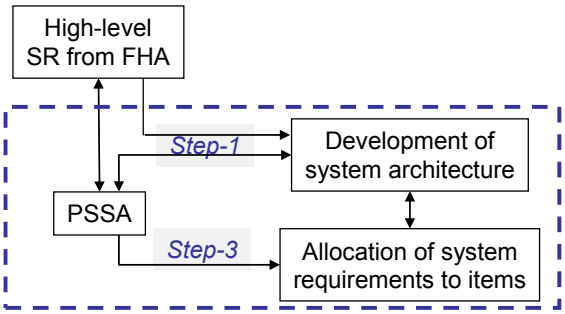

Figure 4. RAP Process Interaction Focus

This alignment process can easily be managed if implemented in a DOORS requirement tree, see a layout example in Table 1.

Table 1. RAP Usage for System Specification

ID	Object text	Object Type	Link Design Descr.	Allocation	Validation
SR001	"High-level requirement"	Safety Req.	IN	System Specification (RAP Step-1)	(RAP Step-2)
SR001.1	"Fed-back refined SR"	Safety Req.	OUT	HRS / SRS (RAP Step-3)	(RAP Step-4)
SR001.2	"Fed-back refined SR"	Safety Req.	OUT	HRS / SRS (RAP Step-3)	(RAP Step-4)
SR001.3	"Fed-back refined SR"	Safety Req.	OUT	HRS / SRS (RAP Step-3)	(RAP Step-4)
…					
SR001.X	"Fed-back refined SR"	Safety Req.	OUT	HRS / SRS (RAP Step-3)	(RAP Step-4)

RAP consists of four steps. The first step is to refine the high-level safety requirement (SR) allocated to a system (black box view), in parallel with the system architecture development (white box view). For the refinement step we use *Design trees* to capture and identify the refined requirements. The new set of safety requirements (refined requirements) shall then be fed back into the requirement tree (pictured in Figure 2 and Figure 5). The second step is to validate the set of refined requirements against the high-level safety requirement by using design trace, links between requirement module and design module. The third step is then to decompose (system-to-item level) the set of safety requirements and allocate to HW and/or SW items. For the decomposition step we use *Allocation trees*. The fourth step is to validate that the requirements have been correctly decomposed and allocated.

Step-1 Refinement

In the design description module architectural principles and design choices are described. As stated previously we use the FTA method to refine the high-level safety requirements imposed on the design.

Following symbols are used in the FTA diagrams:

The OR-gate indicates that the output occurs if and only if at least one of the input events occur.

The AND-gate indicates that the output occurs if and only if all the input events occur.

The Undeveloped event is used to terminate the tree without reaching the lowest level.

The design trees developed in this step give a graphical view of how the top event might occur, and which parts/items of the design that might contribute. The gates in the tree clearly show where independence is required or assumed.

Step-2 Validation of High-Level Safety Requirements

The enhanced set of detailed architectural safety requirements shall implement the higher-level safety requirement, see Figure 5. The set of refined requirements shall capture dependencies, assumptions, safety objectives imposed on the architecture. Validation is performed by assessing the design concept and the corresponding gates in the design tree which should be covered by the refined requirements.

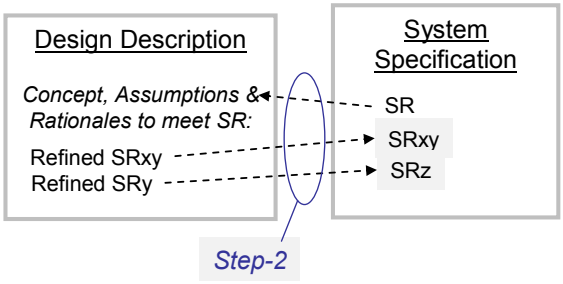

Figure 5. Design trace set for validation

Validation includes correctness and completeness. To check the correctness we propose to assess that the requirement is unambiguous and verifiable. For completeness we propose to assess that the resulting requirements (in the system specification module) capture dependencies and assumptions expressed in the design tree and the design concept (in the design description module). Equally important is to capture the rationales of the design choices. Here *Rationale* is a suggested key-textual means to support safety managers by forcing them to state the rationale behind their choices.

Step-3 Decomposition System-to-Item

The resulting safety requirements from step-1 express which constraints and fault tolerance mechanisms should be implemented. The system level requirements need to be decomposed (broken down) such that allocation to HW and SW items can be performed. Decomposed requirements are linked to the parent requirement. Also here we use FTA by constructing allocation trees. Developing the tree is done by combining the preliminary SW and HW items using AND and OR gates to illustrate how the components should interact to prevent or mitigate the top event.

Assessing the tree structure is an easy and straightforward way to identify and pin-point independence requirements imposed on specific hardware and software items.

Step-4 Validation of the refined Safety Requirements

This step validates that all safety requirements are properly broken down and allocated by checking that the decomposed requirements capture the connection between items seen in the allocation trees (see example in Figure 7). The decomposed requirements must be added into applicable HRS and/or SRS, traced via DOORS links that enable validation, see Figure 6.

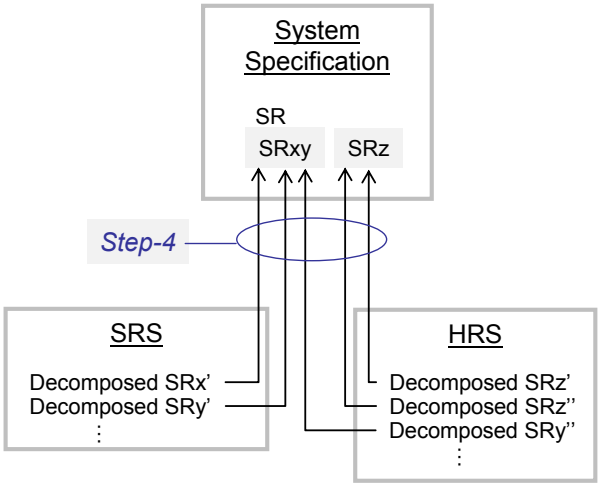

Figure 6. Traced requirements for validation

Applying RAP to HLS

The purpose of this section is to show how RAP is applied to HLS (which was briefly introduced in the background section). HLS is a safety critical system and is classified as DAL A.

When applying RAP we start with the safety requirements with highest criticality, since these typically drive the design. In our example we start with the safety requirements classified as CAT. HLS has 5 CAT events with required safety objective of less than 10^{-10} failures/flight hour, e.g.:

Wrong flap control function

Output not switched as required

Wrong auto-function

As stated earlier, the high integrity level of 10^{-10} requires (at least) two independent processing lanes, which must agree on correct output in order to move the Flaps. Two design principles of the processing lanes are: one command lane and one monitor lane (COM-MON) or two command lanes (COM-COM).

We now limit our attention to the first CAT event and we apply RAP, by executing its four steps.

Example Step-1

Refinement of the safety requirement, labeled SR001-Wrong flap control function.

Design concept: In our example the COM-MON design was chosen and the following design constraints /assumptions were made:

- COM and MON shall be developed to DAL A
- No CEH (FPGA, CPU) dissimilarity
- No multiple version dissimilar software.
- No resource sharing between the COM and MON processors

Integrity and independence principles used in the design process are:

I. Experience data w.r.t. CEH
II. Allocation of functions, monitors, sensor interfaces etc, to achieve independence between functions and their monitors (the COM and MON lanes).
III. Functional diversity between COM and MON.

Design Tree: The tree in Figure 7 shows the COM-MON design concept. The AND-gate imposes independence between COM and MON lanes to mitigate CAT events i.e., prevent an erroneous command from propagating.

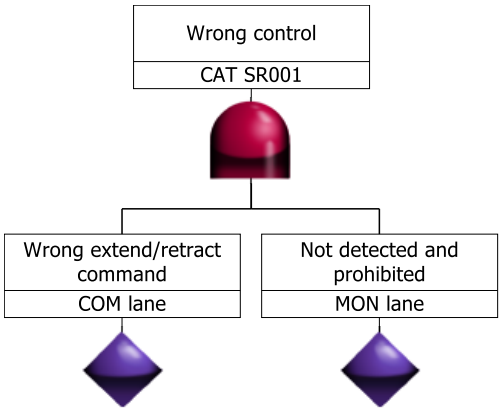

Figure 7. COM-MON concept

The CAT event "wrong flap control function" may occur when the command processing lane (COM) fails in a way that requires that the CMC shall be shut-down and both the COM and the monitoring processing lane (MON) fail to shut-down the servo command or arrest the flaps (deenergize brake solenoids). The possibility for wrong flap control function is a product between COM and MON, where the former one controls and the latter one monitors.

Resulting refined requirements:

1. SR001.1 Flap position shall be controlled by COM and MON independently. Rationale: to prevent undetected development errors and hardware failures leading to wrong Flap position.
2. SR001.2 COM shall be able to put CMC in failsafe state independent from MON. Rationale: The fail-safe behavior must be enforced by all independent members contributing to the undesired event.
3. SR001.3 MON shall be able to put CMC in failsafe state independent from COM. Rationale: The fail-safe behavior must be enforced by all independent members contributing to the undesired event.
4. SR001.4 Communication between COM and MON must not jeopardize required independence between COM and MON. Rationale: If communication is implemented (e.g., for fault detection) no single point of failure should be introduced.
5. SR001.5 COM lane shall be developed to DAL A.
6. SR001.6 MON lane shall be developed to DAL A.

Continue with next high-level "CAT" safety requirement and examine if the design concept and design tree capture the requirement or if the concept needs to be changed or modified.

Step-1 is finalized when all CAT-events are covered. After this step a handful of design trees are produced.

Example Step-2

Validation is performed using **all** the design trees from Step-1. These represent the most critical quantitative safety requirements and should be examined having the qualitative safety requirements in mind in order not to oversee constraints or assumptions imposed on the design. The refined requirements should capture both gate properties and design assumptions. Step-2 should be finalized before decomposition and allocation to HW and SW starts (Step-3).

Example Step-3

HW/SW Design concept: Software; Multiple version/dissimilar software is not anticipated Software integrity between COM and MON is assumed to be achieved by development assurance level A combined with functional diversity between COM and MON. This way COM and MON will not experience the same design fault simultaneously, thus the AND-gate is true. However, if the same function needs to be implemented in both COM and MON then correctness must be proven, i.e., the function must be fully testable or this "common" function must be dissimilar implemented between COM and MON.

Allocation Tree: Figure 8 pictures an early view of an allocation tree, not all events are developed.

Our example focuses on the software, thus the hardware specific events are left undeveloped.

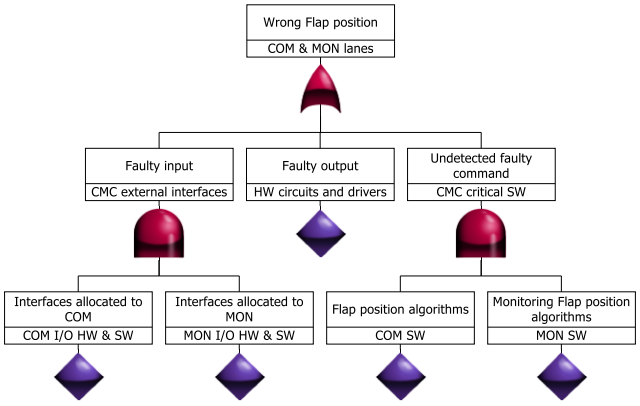

Figure 8. Decomposition of SR001.1

Resulting decomposed SW requirements:

Example of software requirements to be allocated to the applicable SRS from assessment of the AND-gates in Figure 8:

1. COM shall validate its input values. Rationale: To ensure that the Flap position command is based on correct values, taking into account that sensor inputs are sufficient w.r.t. their integrity and reliability.
2. MON shall validate its input values. Rationale: To ensure that the Flap position command is based on correct values, taking into account that sensor inputs are sufficient w.r.t. their integrity and reliability.
3. MON's input monitors shall be fully verifiable (testable) or dissimilar from COM's input monitors. Rationale: To mitigate SW development errors in COM or MON input monitors. Integrity based on DAL A SW alone is not sufficient since an error might propagate to a CAT event.
4. COM shall calculate the Flap position command based on its own data. Rationale: Important to enforce independence between COM and MON.
5. MON shall calculate the Flap position command based on its own data. Rationale: Important to enforce independence between COM and MON.
6. MON's calculation of Flap position command shall be fully verifiable (testable) or use dissimilar algorithms from COM. Rationale: To mitigate SW development errors in COM.

Note: this is just an extraction of safety-related requirements imposed on the software (from SR001.1). Requirements regarding e.g. deterministic behavior will evolve from required fail-safe concept (SR001.2 and SR001.3), such as watchdog monitoring of software execution.

Step-3 is finalized when all sub-system safety requirements are decomposed and allocated HW and/or SW items. Note that the allocation trees might generate redundant requirements, which must not be fed into the hardware or software requirement specifications.

Example Step-4

Similar to Step-2. Validation is performed using **all** the allocation trees from Step-3, this way overlapping parts will be caught.

Conclusion and future work

In this paper we have presented a new approach, called RAP, aimed at systematically guiding (safety) requirements engineers in managing requirements in compliance with ARP4754A. RAP guides the refinement, decomposition and allocation by providing a hands-on method to break down safety requirements based on FTA combined with a validation process, enforced by a series of tagging possibilities suitable for DOORS requirement structure.

In the future, we plan to enhance the method to include a decomposition step for high-level to low-level software/hardware requirements. Currently the scope of RAP stops at the item (HRS/SRS) level. Further we plan to apply RAP to a wider set of requirements, in this paper we elaborate on those safety requirements that are classified as catastrophic (CAT).

References

[1] ARP4754A, 2010, Guidelines for Development of Civil Aircraft and Systems, SAE International.

[2] AC20-174, September 30, 2011, Development of Civil Aircraft and Systems, U.S. Department of Transportation Federal Aviation Administration.

[3] Kondeva, Antoaneta, Martin Wassmuth, Andreas Mitschke, 2012, A Systematic Elaboration of Safety Requirements in the Avionic Domain, SAFECOMP Workshops, LNCS 7613, pp. 400–408.

[4] Certification Specifications for Large Aeroplanes CS25, 27 December 2007 - European Aviation Safety Agency.

[5] DO-178C, November 2011, Software Considerations in Airborne Systems and Equipment Certification, RTCA.

[6] DO-254, 2000, Design Assurance Guidance for Airborne Electronic Hardware, RTCA.

[7] J.C. Knight and N.G. Leveson, 1986, An Experimental Evaluation of the Assumption of Independence in Multi-version Programming, IEEE Transactions on Software Engineering, Vol. SE-12, No. 1, pp. 96-109.

[8] J.C. Knight and N.G. Leveson, 1985, A Large Scale Experiment In N-Version Programming, Digest of Papers FTCS-15: Fifteenth International Symposium on Fault-Tolerant Computing, Ann Arbor, MI. pp. 135-139.

[9] Telelogic DOORS.

http://www.telelogic.com/products/doorsers/doors/.

[10] ARP-4761, 1996, Guidlines and Methods for Conducting the Safety Assessment process on Civil Airbone Systems And Equipement.

[11] SYNOPSIS-SSF-RIT10-0070: Safety Analysis for Predictable Software Intensive Systems. Swedish Foundation for Strategic Research.

Acknowledgements

This work has been partially supported by the Swedish SSF SYNOPSIS project [11].