# Modeling UAS Flight Procedures for SORA Safety Objectives

Charles Mathou, Kevin Delmas, Jean-Charles Chaudemar, Pierre de Saqui-Sannes

# Modeling UAS Flight Procedures for SORA Safety Objectives

Charles Mathou
*ISAE-SUPAERO, Université de Toulouse*
*ONERA, Toulouse*
Toulouse, France
charles.mathou@isae-supaero.fr

Kevin Delmas
*ONERA, Toulouse*
Toulouse, France
kevin.delmas@onera.fr

Jean-Charles Chaudemar
*ISAE-SUPAERO, Université de Toulouse*
Toulouse, France
jean-charles.chaudemar@isae-supaero.fr

Pierre de Saqui-Sannes
*ISAE-SUPAERO, Université de Toulouse*
Toulouse, France
pdss@isae-supaero.fr

***Abstract*—Development of unmanned aerial systems (UAS), made of an unmanned aerial vehicle (UAV) and equipment such as a ground station, has increased tremendously in recent years. This has made more pressing the need for new design methodologies that provide a reliable and thorough safety assessment throughout the entire design process. The European specific operations risk assessment (SORA) document provides recommended operational safety objectives (OSO) to achieve. The current paper lays groundwork to comply with OSOs pertaining to UAS flight procedures. Key criteria for modeling such procedures are identified and lead to the choice of the AltaRica DataFlow (ADF) language. The Cecilia Workshop is used to model three real-life UAS emergency flight procedures. Custom components developed for this model are presented while discussing the process of modeling a formal procedure from an informal text source. A safety analysis is performed on the resulting model by computing minimal cut sets on an undesired procedure outcome. The results are then reviewed, providing feedback to increase the procedures' safety gain.**

***Index Terms*—AltaRica, MBSA, OSO, SORA, UAS, UAV**

| | |
|---|---|
| ADF | AltaRica Dataflow |
| CC | Command & Control |
| EASA | European Union Aviation Safety Agency |
| F/CTL | Flight Controller |
| OSO | Operational Safety Objectives |
| RC | Radio Control |
| SORA | Specific Operations Risk Assessment |
| UAS | Unmanned Aerial System |
| UAV | Unmanned Aerial Vehicle |

## I. Introduction

Safety is a key issue in the field of aeronautics. In terms of regulations, the ARP4754 and ARP4761 are well-known reference documents for safety assessment process of large-scale airborne systems such as planes and helicopters. Closely related to safety is the domain of security. Security is an equally important aspect of a system's design and operation, notably to provide acceptable levels of risk management. However, addressing malicious interference or intrusion is a domain of its own, and our work is focused on managing safety rather than security. While the proposed methodology might be extended to address security concerns, we leave that path open for others to pursue. With the Specific Operations Risk Assessment (SORA), the European Union Authority for Air Safety (EASA) provides designers and safety experts with guidelines on UAS design, including safety assessment [1] for a limited range of UAS and their applications. Operational procedures describe the expected flow of actions to be taken by both the UAV and the ground crew under specific circumstances (*e.g.*, UAV preparations, take-off, loss of an engine, loss of communications). Modeling such procedures and verifying their validity is a recurring requirement of multiple safety objectives defined in the SORA. For UAS, these procedures are usually described in text form within a flight manual. The propagation of a failure from one procedure to another may be difficult to assess on such informal descriptions.

Safety assessment can benefit from the emergence of model-based safety analysis (MBSA) to solve this issue. The latter leverages the model-based paradigm in order to facilitate safety assessment processes that were historically relying on fault tree analyses (FTA) and failure modes and effect analyses (FMEA). Our goal is thus to lay the groundworks for new approaches leveraging model-based methodologies to facilitate the safety assessment of UAS emergency procedures.

The remainder of the current paper is organized as follows. Section II provides a concise overview of the SORA and a select few safety objectives from which stems the need for modeling and analyzing operational procedures. Section III presents our choice of the AltaRica language based on previously identified criteria. Section IV introduces our approach to modeling procedures on three simple examples, from which we draw lessons in section V. Section VI surveys related work. Section VII concludes the paper and outlines future work.

## II. The SORA

### A. General presentation

EASA classifies UAS operations into three categories. The larger and faster the drone, or the denser the area it will be flying near to, the more stringent the safety requirements.

In increasing order of requested safety, these categories are 'Open', 'Specific' and 'Certified.' 'Certified' UAS operations are considered to be risky enough to require safety processes derived from the ones applied to commercial aircrafts. 'Open' operations cover use-cases such as small-scale entertainment quadcopters. 'Specific' UAS operations fall short of both previous categories. They are too large or heavy to be authorized for use without a detailed safety analysis, but too small to justify the complex procedures entailed by a full-scale aeronautical certification process. It is for this category of operations that the SORA provides recommendations on how to perform safety assessments.

To this end, the SORA defines various Operational Safety Objectives (OSOs) to be taken into account by safety experts. The OSOs can be refined into several sub-criteria. For instance OSO#7 addresses UAS inspection and can be refined into two criteria, covering the inspection procedures and the training of the inspecting crew respectively. Each criterion can be complied with at up to three levels (denoted *High*, *Medium* and *Low*) of integrity (or safety gain) and assurance (or confidence in the claimed safety gain).

### B. The OSOs of interest

In order to ease comprehension, we propose an alpha-decimal notation to point to specific descriptions of an OSO. It uses the following pattern: O*i*-*X*-C*j*-*Y*-*(x)*, where:

- *i* is an integer referring the OSO number;
- *X* is either *I* for the integrity levels, or *A* for the assurance levels of the relevant OSO;
- *j* is an integer referring to OSO *i*'s criterion number;
- *Y* is either *L*, *M*, *H*, or a combination thereof, for the level of compliance of the considered OSO;
- *(x)* is an optional index referring to a further subdivision of this criterion as used in the SORA

The SORA defines specific OSOs covering UAS procedures. More precisely, the OSOs 8, 14 and 21 require that *Operational procedures are defined, validated and adhered to*, pertaining respectively to technical issues with the UAS, human error and adverse operating conditions. OSO 11 states that *Procedures are in-place to handle the deterioration of external systems supporting UAS operations* [1].

The integrity level of these objectives is evaluated against three distinct criteria.

- O8-I-C1: the procedures must cover a minimal set of scenarios (the reader is referred to [1] for the list of all covered scenarios).
- O8-I-C2: procedure simplicity, which in turn makes them less error-prone.
- O8-I-C3: human errors must be considered in the definition and execution of the procedures; a higher compliance level requires clear procedures with contingencies for human errors, as well as crew training.

The three levels of assurance of these OSOs are increasingly stringent in terms of procedure validation.

- O8-A-C1-L: no validation required except for emergency procedures.
- O8-A-C1-M: validation according to relevant standards, and proof of emergency and contingency procedures via test flights or simulation.
- O8-A-C1-H: flight tests are conservative, proofs are validated by a competent third-party.

### C. Definition of a procedure

This subsection presents the model of procedure considered in the article. While they do not explicitly detail their structure of a procedure, the authors of [2] model their procedure according to the following guidelines, which we have taken for our models.

***Procedure structure*** A procedure begins with an *event* (*e.g.*, the occurrence of a fault, or an actor's intervention such as the pilot deciding to take off). This event is followed by a directed acyclic graph of *tasks*, representing the execution of *actions* by *actors* of the procedure. We consider that procedures must end in one of at least two *outcomes*. These represent at a strict minimum the impact of the procedure's success or failure on the global scenario.

***Procedure behavior*** Starting from the event, tasks can be performed in a sequence, branch off of each other in a parallel or conditional way or merge into a single branch.

### D. Relevant criteria

Not all the criteria described in section II-B are immediately applicable to our study. Since our approach is to lay some groundworks for the modeling and safety assessment of operational UAS procedures, concepts pertaining to crew training (O8-I-C3-H) or checklists (O8-I-C3-L-(b)) are outside the scope of this paper. While the concept of procedure complexity (O8-I-C2-L,M,H) is certainly relevant in the global safety assessment, we keep this avenue open for future work. The core of our study is for the modeling of operational procedures, thus on description O8-I-C1-(a), with a focus on flight procedures, *i.e.* listings O8-I-C1-(a)-(4) to O8-I-C1-(a)-(7). In terms of assurance, we chose to concentrate on the simulation of operation procedures as a mean to verify their efficiency, as per O8-A-C1-M-(b)-(2).

As a consequence, we propose the following criteria a suitable procedure modeling tool should have:

***Procedure structure***: a procedure can be modeled according to the structure defined in II-C. It must be possible to model several procedures within the same model (*i.e.* several distinct events, task and outcome sections) (O8-I-C1-L-(a)). Procedure models should have the ability to allocate tasks to the various actors in a procedure so that later work might be done on that topic (O8-I-C3-L-(a)).

***Procedure behavior***: a procedure can be modeled according to the behavior defined in II-C. It must be possible to account for the dependency between various events. Events other than the initiating events must fireable to alter the execution state of the procedure tasks.

***Error propagation***: the modeling framework should be able to represent the occurrence, propagation and possibly mitigation of errors within a procedure. Note that an error

must be understood as an incorrect output of a procedure task that differs from procedure-triggering events (O8-I-C3-M).

***Safety assessment***: finally, the modeling framework should offer a formalism that may be leveraged to perform safety analyses, namely computing minimal cut sets for undesired final states (O8-A-C1-M-(b)-(2)).

## III. AltaRica Dataflow

There are various languages that are able to model procedures. Business process diagrams (BPD), of which the international standard BPMN [3] is the most known, are such an example. BPMN offers a wide variety of components to model diverse procedure structures and behaviors. However, [4] highlights how the lack of semantic verification tools can lead to multiple errors given the shift in the application domain from business processes to flight procedures. BPMN also provides means to represent the propagation of messages between procedures, and provides visual cues to separate actors of a procedure. A limitation of these constructions is that they have no underlying formalism [5]. Attempts have been made to associate BPMN with more formal languages in order to enable formal safety analyses based on Petri nets [4], [6]. However, these transformations are still largely unproven and suffer from combinatorial explosion [6].

Another option is the formal language AltaRica Data-Flow (ADF) that can be compiled to Petri nets or finite state machines [7]. It has been implemented in various industrially mature tools such as Cecilia Workshop (Satodev), Simfia (EADS Apsys) and Safety Designer (Dassault System) [8]. ADF has been instrumental in genuine industrial applications as presented in [9], [10].

Modeling procedures falls within ADF's capabilities. Leveraging the Cecilia Workshop's possibility of creating custom components, complex procedure structure and behavior can be modeled. Mechanisms such as tags can be used in order to allocate ownership while event dependency can be implemented using the built-in synchronization. The Cecilia Workshop also offers users the possibility to perform safety analyses in the form of fault tree, minimal cut sets or sequence generation.

Based on an early draft of SORA, the authors of [2] propose a model of a multi-actor collision avoidance scenario. They define and connect ADF components that represent the various tasks. The three-tiered execution performance of a given task is determined by that of the previous task, as well as by the state of a required resource component. Observers connected to specific task components represent undesired events that might lead to an unsafe situation. These observers are then used to compute minimal cut sets and provide safety feedback.

The authors of [2] plan to further their work in other concept of operations. However they only model a single procedure, and do not consider modeling several concurrent procedures. As such they do not tackle the issue of error propagation across multiple procedures. The current paper proposes a complementary approach where multi-procedure models are handled to assess the safety impact of procedures on one another and the global scenario. To our knowledge, no published work uses ADF to model procedures in such a way.

## IV. Modeling real-life UAS procedures

Our use case consists in a few operational procedures lifted from a fixed wing UAS's flight manual [11]. These procedures have been modeled using Cecilia Workshop. The following subsections present the use case and three procedures used for illustration purposes. The ADF components we created and the rationale for modeling procedures are also presented.

### A. Use case

*1) General presentation:* The procedures we modeled have been designed for a small, fixed-wing UAV typically flying at 80 $km.h^{-1}$. The drone can be controlled manually by the operator, via Radio Contol (RC) or Command & Control (CC) communication channels. If CC are available, the drone's autopilot may be manually changed by the pilot to one of the following six flight modes:

- *TAKEOFF*: the UAV is hand-launched and flies to the predefined flight altitude;
- *FPLN*: the UAV flies the predefined flight plan;
- *SELECTED*: the UAV flies according to the speed, altitude and bearing settings specified by the pilot;
- *GOHOME*: the UAV flies to a predefined home point and circles it when arrived;
- *LANDING*: the UAV flies the landing approach (it does not land on its own);
- *SAFEIMPACT*: the UAV flies a low-speed descending geo-centered spiral in order to hit the ground with minimal energy.

*2) Ordering outcomes:* To assess the contribution of errors to the global risk, we need to formalize the outcome of a procedure. Based on our use case's flight manual, we have identified five possible outcomes for a flight procedure:

- *Resume* (*R*): flight operations may resume as they were before the procedure was initiated;
- *Go home* (*G*): the UAV goes to and circles the home point where the pilot may decide whether to resume or terminate the flight;
- *Land* (L): the pilot performs a forced landing;
- *Safe impact* (S): the UAV performs a low-energy ground collision;
- *Crash* (C): the UAV crashes uncontrollably.

We define a severity order denoted $l < r$ standing for *outcome* $r$ *is more severe than* $l$. In our example, a previous safety assessment concluded that $R < G < L < S < C$. The minimal outcome is $R$, as no meaningful safety reduction occurs. The maximal outcome is $C$ as it represents the most severe outcome identified by the SORA, *i.e.* a crash outside of a predefined ground buffer or a fly-away.

*3) Example procedures:* The extract of our model presented in this paper covers three of our flight manual's emergency procedures. In our flight manual, operational procedures are referred to by the name of their triggering fault or event.

- *AP Fault*: the auto-pilot is faulty; it reboots automatically while the UAV's on-board flight manager engages the *SAFEIMPACT* flight mode. If the auto-pilot recovers, the flight manager gives the control of the UAV back to the auto-pilot and engages the *GOHOME* flight mode, ending the procedure.
- *CC COMM LOSS*: command and control communications are lost (radio controls are not affected); the UAV enters *GOHOME* at an altitude no lower than 50 meters, climbing if necessary. The pilot is expected to monitor the trajectory by video. Once overflying the *HOME* point, the UAV sends a landing warning to the pilot. Once the pilot can see his UAV, he switches to manual control and lands the UAV himself.
- *F/CTL FAULT (RUDDER)*: the rudder controller encounters a fault; the UAV sends an emergency landing warning to the pilot, who switches to manual control, and corrects the trajectory for the missing control surface using elevators as an alternative control surface. He decides whether to attempt an emergency landing on a suitable area or to engage the *SAFEIMPACT* flight mode.

### B. Quick reminder on Altarica Dataflow

A system modeled with ADF is a set of interconnected components. The behavior of these components is based on the following elements : *flow variables*, the inputs and outputs that are used to interface the component with its environment; *state variables*, the internal variables that can encode the component's functional or dysfunctional state (*e.g.*, failure modes); *events*, the elements used to trigger the transitions between the component's states.

The component's functional and/or dysfunctional behavior is defined by the following relations between states, flows and events. *Transitions* encode the possible state evolution. A given transition is fired when its guard (a condition over the values of the state and flow variables) is true and its associated event is triggered. Upon being fired, the associated action is performed, consisting in an assignment of state variables (*e.g.*, a *failure* transition might change a component's state variable from *nominal* to *failed*). *Assertions* encode the constraints between the possible values of flow and state variables. This is typically used to compute the value of an output flow variable based on the values of state and input flow variables.

### C. Modeling the elements of a procedure

*1) Structure of a procedure:* Our procedure models can be divided into the following sections:

- Initiating event: a single event component initiating the procedure. This component is described in IV-C3.
- Task graph: a directed acyclic graph of task and logic components, representing the execution of procedure tasks. These components are described in IV-C4 and IV-C5.
- Outcome determination: this section determines the outcome of individual procedures, and then determines the global outcome of all relevant procedures. This process is explained in IV-C6.

(a) Event component. (b) Inactive event (c) Active event

Fig. 1: Visuals of the event component.

*2) Enumerated types:* In order to model varying levels of performance and different procedure outcomes, we defined two enumerated types. Both of those enumerations have an empty element called *na* (for *Not Applicable*).

The ***component status*** (***CS***) type: {*nom, part, fail*}. It represents the level of performance of a given component or task, in descending order of adequacy. *nom* stands for nominal, *part* for partial loss and *fail* for total loss. This enumeration is lifted from [2], we have only renamed the enumeration values and the type for easier reference.

The ***procedure status*** (***PS***) type: {*resume, resume_part, gohome, gohome_part, land, land_part, safeimpact, safeimpact_part, crash*}. We propose this enumeration in order to describe the possible outcomes to an operational flight procedure. It is based on the outcomes defined in IV-A2.

Four additional outcomes (suffixed with *_part*) represent a degraded version of the original suffixed outcome. *resume_part* might represent a scenario where the operation is resumed, albeit in a degraded mode (*e.g.*, reduced maximum speed) where *resume* would have optimistic and *gohome* pessimistic. The *crash_part* outcome does not exist as *crash* already is the worst possible outcome.

The order relation using the previous $<$ operator becomes $R < R_d < G < G_d < L < L_d < S < S_d < C$, where the subscript $d$ points to the degraded alternative of the relevant outcome.

*3) Event component:* Event components represent a specific event that may occur in the system. The presented model only contains emergency procedures, although event components may also represent nominal events. It is important to note that these components can be activated *during* the simulation. By default, they are deactivated (the event does not occur), and can each be activated (thus injecting the event in the system) independently from other event components. Event components have one state variable and one output flow variable:

- The *Boolean* state variable *event_triggered* represents whether the event has been triggered.
- The *CS* output $O$ is equal to *nom* when the event is triggered, and *na* when it is not (this allows task components to compute the appropriate outcome).

Event components (figure 1a) have only one transition, *trigger_event*, that changes *event_triggered* from *false* to *true*. The component is grayed out when not triggered (figure 1b), and red when it is triggered (figure 1c).

*4) Task component:* Task components represent an action performed by an actor. We extended the task component

defined in [2] to handle the additional *CS* value *na* and to which we added a third transition. Task components have one state variable, two input flow variables and one output flow variable:

- The *CS* state variable *perf_status* represents how well the task can be performed internally.
- The *CS* input $I$ represents the status of the previous execution of a task component.
- The *CS* input $R$ represents the status of the resource the task component depends on for its own execution.
- The *CS* output $O$ represents the current task's execution status.

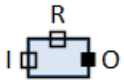


Fig. 2: Task component (flow variable names added for convenience).

The component task's output is based on its state variable, the status of previous executions and its required resource.

- If $I$ is *na*, so is $O$ (this represents the non-activation).
- Otherwise, if either $I$, $R$ or *perf_status* are *fail*, so is $O$.
- Otherwise, if both the component's internal state and its resource are *nom*, then $O$ is equal to $I$.
- Otherwise, $O$ is equal to *part*.

Task components have three transitions :

- *partial_fail*: partial failure of the component, *perf_status* changes from *nom* to *part*
- *total_fail*: total failure of the component, *perf_status* changes from *nom* to *fail* or from *part* to *fail*

Figures 3 and 4 show the visuals of our task components during simulation. The gray background indicates that the task is inactive. Green, orange and red backgrounds indicate an active task whose output is respectively *nom*, *part* and *fail*. A single diagonal slash indicates that the component's *perf_status* is *part*, while a cross means it is *fail*. No slash indicates *perf_status* is *nom*.

Some tasks described in our flight manual appear in multiple procedures. As a result, our models include several components representing a same task, but that are formally distinct from ADF's point of view. In order to ensure that all the components representing a same task fail simultaneously and in the same way, we used ADF's *synchronization* feature. Our model thus includes several synchronization events, that trigger all component events modeling the same task.

*Resource components* have no input, and output a *CS* type that connects to task components' *R* input. Resource components have the same transition types as task components. Those transitions determine the output value of the resource components.

(a) *nom* (b) *part* (c) *fail*

Fig. 3: Simulation visuals of the inactive task component with decreasing *perf_status*.

(a) $O$ = *nom* (b) $O$ = *part* (c) $O$ = *fail*

Fig. 4: Simulation visuals of the task component where *perf_status* = *nom*, with a decreasing output variable.

*5) Conditional activation components:* Similarly to [2], we have added components to represent the conditional activation of a branch of our model.

The diamond-shaped *condition component* represents a condition to be evaluated. This component has an input $I$ connected to a previous task, and an input $A$ connected to a Boolean value. If $A$ is *true*, $I$'s value is transmitted to the condition component's first output, while its second output receives the value *na*, and conversely.

The value of $A$ is controlled by a *Boolean switch* component which users can alter during the simulation to represent different scenarios. A green switch outputs *true* while a red switch outputs *false*. These components can be used to represent mutually exclusive branches in a procedure such as in figure 5.

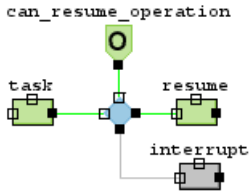


Fig. 5: Conditional structure representing whether an operation can be resumed or should be interrupted.

The *conditional merge* component (logical *OR* gate symbol) is designed to be used with a condition component. It takes as inputs the two branches that stemmed from a condition component (figure 6). The output it produces is equal to the execution state of the branch that was activated (*na* if the procedure is not active).

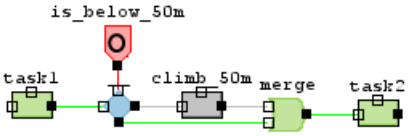


Fig. 6: Conditional structure representing the UAV ensuring it is at a proper altitude.

*6) Outcome determination components:* Some task components in our model represent the *R*, *G*, *L*, *S* or *C* outcomes of IV-A2 (called *outcome task components*). As such, they carry additional information compared to other task components since they introduce a possible outcome for the considered

(a) Minimum (b) Maximum

Fig. 7: *PS* type comparison components.

procedure. A converter component converts the *CS* output of such components to the *PS* type, formally carrying information about both the type of outcome and its status.

In order to process those *PS* type values, it was necessary to have components computing minimum and maximum values of a *PS* pair. Both have two *PS* type inputs, $I_1$ and $I_2$, and one *PS* type output $O$. For both components, if $I_1$ is *na*, then $O = I_2$, and conversely. Otherwise, our components respectively output the minimum and maximum of $I_1$ and $I_2$ according to the order relation of IV-C2.

### D. Modeling procedures

We here present the ADF models of the three procedures detailed in IV-A3.

*1) AP Fault (figure 8):* In this procedure, there are two main branches being performed in parallel.

In the top branch, the auto-pilot performs an auto-reset immediately after the fault occurs. Meanwhile, the UAV automatically engages the *SAFEIMPACT* flight mode (*S* component), hence the parallel branching that occurs in the procedure.

Should the auto-pilot recover successfully, our flight manual indicates that the UAV sends a warning message to the ground pilot (*Warning* component), switches to the automatic control mode (*CM_auto*) and engages the *GOHOME* flight mode (*G*). Even though the flight manual describes those three actions in a sequence, we chose to branch the warning task away from the control and flight mode tasks. It seemed to us that a simple warning task should not be blocking for such higher priority tasks since they were performed by the UAV itself and not the pilot, whose awareness thus is not critical to the execution of the procedure.

In this procedure, we have two tasks that can lead to a procedure outcome: *G* and *S*. As shown in figure 8, those components are connected to converters so that the procedure's outcome may be computed.

*2) CC COMM Loss (figure 9):* The trigger event for this procedure is the loss of CC communications. Here, the procedure branches off conditionally based on whether the UAV is below 50 meters. If it is, a corrective task is performed to climb, otherwise nothing specific to this branch happens. The conditional branches then merge back together. The UAV engages the *GOHOME* flight mode and warns the pilot, who then manually lands the UAV. Once again, we modeled the warning task on a parallel branch as we considered it non blocking. Losing CC is immediately noticeable by the pilot (no more flight data on the console), and once the UAV flies around the home point, the pilot may decide on his own to take manual control of the UAV.

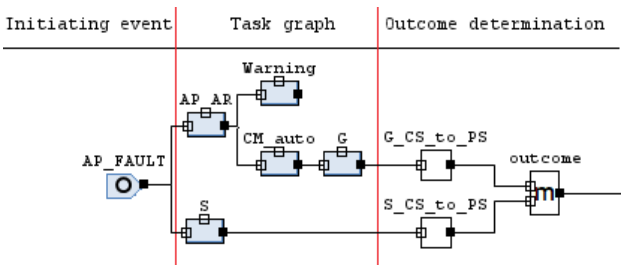


Fig. 8: ADF model of the AP Fault procedure.

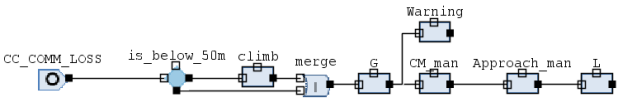


Fig. 9: ADF model of the CC Comm Loss procedure.

*3) F/CTL Fault (rudder) (figure 10):* When the rudder controller encounters a fault, the UAV sends an emergency landing warning to the pilot. Unlike in the previous two procedures, this warning has been considered blocking as the detection of a rudder fault by the pilot from the ground while the UAV is in auto-pilot might be non-trivial. The pilot switches to manual control, and corrects the trajectory for the missing control surface using the elevators. The pilot's decision to attempt a forced landing or engage the *SAFEIMPACT* mode is represented by a conditional branching in the model.

*4) Determination of outcomes:* In order to perform safety analyses, we must first determine the outcome of the procedures individually and then determine how all the activated procedure's outcomes merge together.

We consider that the outcome of an individual procedure is determined by the successfully or partially executed outcome task of least severity. Should there be no outcome task whose execution was at least partially successful, we consider the procedure results in a crash. This behavior was modeled using the minimum component, as can be seen in figure 11.

To determine the global outcome of our scenario, *i.e.* of all activated procedures, we have adopted a pessimistic approach. Namely, among all activated procedures (those whose *PS* output is different from *na*), we chose the one of highest severity as the global outcome. This behavior was modeled using the maximum component as seen in figure 12. The top procedure successfully executes both the *GOHOME* and *SAFEIMPACT* outcome tasks, which we determined ends the procedure in the *GOHOME* outcome. The bottom procedure only engages (and successfully executes) the *LAND* outcome task. Since $G < L$, the scenario ends with the $L$ outcome.

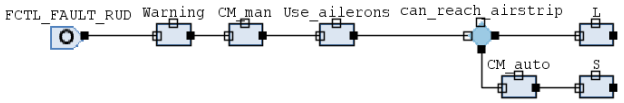


Fig. 10: ADF model of the F/CTL Fault (rudder) procedure.

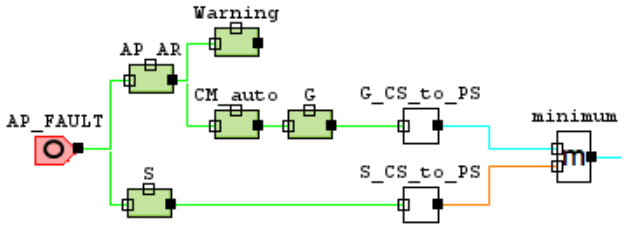


Fig. 11: Computing the outcome of the AP Fault procedure.

## V. Discussion

### A. *Computing minimal cut sets for the* Crash *outcome*

Using the Cecilia, we computed minimal cut sets of orders 1 to 5 for the undesired *Crash* outcome. Results indicate there are 19 cut sets of the $2^{nd}$ order and 13 of the $3^{rd}$ order. There are no $1^{st}$, $4^{th}$ or $5^{th}$ order cut sets. There are trivially no $1^{st}$ order cut sets as any procedure requires at least one fault to be activated and another fault to fail completely.

Among our minimal sequences, some near duplicates appear, where the *partial* and *total* losses of a resource are found with another fault in two distinct sequences. For instance, we have the following:

```
{'AP_Fault_partial', 'FM_Manager.total_failure'}
{'AP_Fault_total', 'FM_Manager.total_failure'}
```

The loss of the auto-pilot, event partial, triggers the *AP_Fault* procedure. Since that procedure does not take into account the severity of the auto-pilot fault, the worst is assumed in both cases, leading to this pessimistic result. There are 8 such near-duplicates of the $2^{nd}$ order and 5 of the $3^{rd}$ order. This nearly systematic occurrence highlights the fact that the granularity of the model isn't adequately matched with the definition of the procedures. A more in-depth knowledge would allow safety experts to conclude on this specific point.

As shown in table I, the *AP Fault* procedure only has two $2^{nd}$ order minimal cut sets, due to its parallel nature. The benefits of this structure are made even clearer when compared to the *CC comm loss* and *F/CTL fault (rudder)* procedures' nine $2^{nd}$ order minimal sequences, due to the larger number of single points of failure in their structure.

As a consequence, most of *AP fault*'s minimal cut sets are of the $3^{rd}$ order, which *CC comm loss* and *F/CTL fault (rudder)* have significantly less of. Adding parallel branching

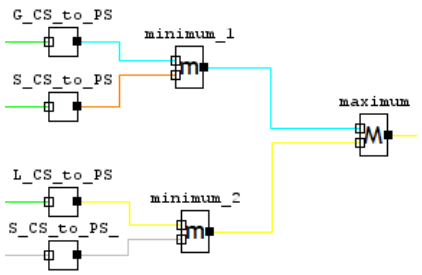


Fig. 12: Computing the outcome of the AP Fault procedure.

| | Order of cut set | | | | |
|---|---|---|---|---|---|
| Initiating fault | 1 | 2 | 3 | 4 | 5 |
| AP fault | 0 | 2 | 10 | 0 | 0 |
| CC comm loss | 0 | 9 | 0 | 0 | 0 |
| F/CTL fault (rudder) | 0 | 9 | 3 | 0 | 0 |

TABLE I: Number of minimal cut sets involving each initiating event.

in a procedure can thus reduce the amount of lower order minimal cut sets and shift them over to higher orders. This is however to be balanced with procedure complexity as adding too many parallel branches might prove counter-productive should one specific actor be overwhelmed.

The absence of higher order minimal cut sets in the analysis also originates in the limited amount of procedures it was performed on. Adding more procedures to the analysis would add more interactions that could lead to undesired outcomes.

The minimal cut set `{'CC_Comms_total', 'FCTL_FAULT_RUD.trigger_event'}` illustrates the need for procedures handling multiple faults. Namely, in the absence of CC communications, the pilot is unable to take control of the UAV (CC are a resource required by component *CM_man* in figure 10), despite RC communications possibly remaining available. In order to improve the *F/CTL fault (rudder)* procedure, a condition component should thus be added to ascertain whether CC communications are available. Should they not be, the relevant branch could make use of RC communications instead, before merging back with the now unused CC communications branch.

Through the modeling process, several instances of a 'Warning' task component have had to be placed. This task was not explicitly described as executed in parallel to others in the source material. We have however considered it as such on several occasions, based on the context of the procedure (namely, whether the pilot's awareness of an emergency depended solely on this task). When considered non-blocking, that task was placed on a parallel branch featuring no outcome task component, thus having no impact on the outcome. While we discuss this specific task below regarding future work, this does question the importance of procedure tasks that appear to have no safety impact on the global scenario. Other such tasks exists that are not presented in this article; for instance the pilot may be required to monitor the UAV's controlled crash via video link even though he has no means to control it. Pilot actions in such procedures are sometimes limited to non-existent (as it is in the *AP Fault* procedure), it would thus be beneficial for the UAS designers to more explicitly state the expected safety gain of such tasks.

### B. *Limitations*

The procedures model makes the assumption that individual task components are independent from one another. This is due to the high-level of abstraction of the approach that was used to model procedures. Largely ignorant of the lower level architectures of the UAV whose procedures were modeled, the procedures model does not tie in with genuine physical and

functional models. Future work will be done in order to: first, acquire a model of a UAV's architecture, and second, to create bridges between the physical and functional layers and the procedures model. This would increase both the consistency and verisimilitude of the procedure model.

Another implicit assumption of our model is that a fault is always detected. Our models thus include no section pertaining to the detection of the fault. This limitation is connected to a previous remark in V-A concerning the apparent lack of safety impact of the 'Warning' tasks. These tasks could actually be the first elements of a new section in our model dedicated to modeling the detection of a fault by actors of the system.

Other possible avenues of development are the modeling of undesired activation of task components, procedures with multiple initiating events or procedure complexity.

## VI. Related work on the SORA

The authors of [12] have developed a web-based questionnaire that allows users to get a safety report of their drone. The tool gathers system information directly from the user and matches it against SORA criteria, producing a report as output. This report takes the form of a table summarizing the highest level of compliance met by the UAS with regards to each OSO. The proposed approach allows to remove part of the tedium of going over the numerous safety criterion of all 24 OSOs. The tool is still in early development stages, and only covers multi-rotor drones over a limited range of compliance levels. Furthermore, the generated safety report provides feedback as to which objectives are met and which aren't, but in the latter case, it does not provide any improvement suggestions as can already be found in the corpus of the SORA.

In [13], a safety assessment is done for a UAS recording sport events, according to the methodology provided by the SORA. The authors conclude that their operation matches the lowest category of risk and that the SORA's OSOs are thus all only recommended as optional or with low levels of compliance. However, the authors suggest that further work is required on the SORA to cover multi-drones use cases, as those scenarios are getting more and more leverage. We also identified that had the risk level of the pair *drone-operation* been any higher in that article, the described means of mitigation and associated assurance would not have been enough to qualify the operation under the recommended SORA levels of compliance. This highlights the need for proper procedure analysis methodologies (*e.g.,* safety assessment processes) if more risky drone operations are to be analyzed.

## VII. Conclusion

The significant increase in demand for UAS and their massive deployment in closer proximity to humans and areas of risk put in sharp relief the need for increased safety. SORA provides an early framework to ease safety analyses. However, the SORA standard does not define the methodologies and processes required to model and analyze the procedures.

To tackle this problem, this paper provides a methodology based on the ADF formal language to model and analyze emergency procedures. The Cecilia Workshop tool was used to compute minimal cut sets leading to undesired procedure outcomes, and to identify shortcomings in the procedures.

Future work stills needs to be done in order to formalize the separation of actors in a procedure so that one actor's specific safety impact can be assessed. We will also expand the processes with a fault detection as it really occurs, which will contrast with the current ADF model where a fault in the system is immediately noticed by the relevant actors.

## Acknowledgments

This work was supported by the Defense Innovation Agency (AID) of the French Ministry of Defense (research project CONCORDE N° 2019 65 0090004707501).

## References

[1] EASA, "Easy access rules for unmanned aircraft systems (regulations (eu) 2019/947 and (eu) 2019/945)," 2020.

[2] P. Bieber, C. Seguin, V. Louis, and F. Many, "Model based safety assessment of concept of operations for drones," in *Congrès Lambda Mu 20 de Maîtrise des Risques et de Sûreté de Fonctionnement*, St Malo, France, 10 2016.

[3] "BPMN omg homepage," https://www.omg.org/spec/BPMN, accessed: 2022-10-03.

[4] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in BPMN," *Information and Software Technology*, vol. 50, no. 12, pp. 1281–1294, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584908000323

[5] G. Penna, R. Sordo, I. Benedetto, N. Mezzopera, and P. Maria Teresa, "A lightweight formalism for the integration of BPMN models with domain ontologies," *AIBP@ AI* IA*, vol. 1101, pp. 11–20, 01 2013.

[6] S. Meghzili, A. Chaoui, M. Strecker, and E. Kerkouche, "Transformation and validation of BPMNmodels to petri nets models using groove," in *2016 International Conference on Advanced Aspects of Software Engineering (ICAASE)*, 2016, pp. 22–29.

[7] A. Rauzy, "Mode automata and their compilation into fault trees," *Reliability Engineering & System Safety*, vol. 78, no. 1, pp. 1–12, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095183200200042X

[8] T. Prosvirnova, M. Batteux, P.-A. Brameret, A. Cherfi, T. Friedlhuber, J.-M. Roussel, and A. Rauzy, "The altarica 3.0 project for model-based safety assessment," *IFAC Proceedings Volumes*, vol. 46, no. 22, pp. 127–132, 2013, 4th IFAC Workshop on Dependable Control of Discrete Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667015339999

[9] P. Bieber, J. P. Blanquart, G. Durrieu, D. Lesens, J. Lucotte, F. Tardy, M. Turin, C. Seguin, and E. Conquet, "Integration of formal fault analysis in ASSERT: Case studies and lessons learnt," in *Embedded Real Time Software and Systems (ERTS2008)*, Toulouse, France, Jan. 2008. [Online]. Available: https://hal.science/hal-02270317

[10] R. Bernard, J.-J. Aubert, P. Bieber, C. Merlini, and S. Metge, "Experiments in model based safety analysis: Flight controls," *IFAC Proceedings Volumes*, vol. 40, no. 6, pp. 43–48, 2007, 1st IFAC Workshop on Dependable Control of Discrete Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667015310934

[11] "Avem - aeromapper." [Online]. Available: https://www.aeromapper.com/avem-2-2/

[12] K. H. Terkildsen and K. Jensen, "Towards a tool for assessing uas compliance with the jarus sora guidelines," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2019, pp. 460–466.

[13] C. Capitan, J. Capitan, A. R. Castano, and A. Ollero, "Risk assessment based on sora methodology for a uas media production application," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2019, pp. 451–459.