Dynamic Safety Cases for Through-life Safety Assurance

Ewen Denney and Ganesh Pai
SGT / NASA Ames Research Center
Moffett Field, CA 94035, USA
Email: {ewen.denney, ganesh.pai}@nasa.gov

Ibrahim Habli
Department of Computer Science
University of York, York YO10 5DD, UK
Email: ibrahim.habli@york.ac.uk

***Abstract*—We describe *dynamic safety cases*, a novel operationalization of the concept of *through-life safety assurance*, whose goal is to enable proactive safety management. Using an example from the aviation systems domain, we motivate our approach, its underlying principles, and a lifecycle. We then identify the key elements required to move towards a formalization of the associated framework.**

***Index Terms*—Dynamic safety case, Safety assurance, Lifecycle processes, Safety management**

I. Introduction

Safety standards not only heavily guide the development and usage of software systems in safety-critical applications, but also form the basis for their approval and certification. Historically, these standards have tended to be highly *prescriptive*, imposing specific certification requirements and also specifying the means of compliance [1].

Over the past two decades, however, the practice of safety certification has been migrating to a *goal-oriented* paradigm—placing greater emphasis on explicitly stating safety claims, and supplying an argument along with evidence that developers/operators have to generate—to satisfy *certification goals* that regulators define [2], as opposed to following a prescribed process. In general, such (structured) arguments and evidence are captured in the form of a *safety case* [3], i.e., a comprehensive, defensible, and valid justification of the safety of a system for a given application in a defined operating environment. The rationale for establishing this approach, in part, is to respond to the challenge of certifying systems that include novel technologies, e.g., integrated modular avionics (IMA), and comparatively new software engineering techniques, e.g., model-based development and code generation. In particular, the rate of change of such technologies has outpaced the ability to prescribe one-size-fits-all certification measures.

A safety case is usually defined prior to system deployment, and its validity relies on assumptions/predictions about system behavior (including its interactions with its environment). Safety-critical software systems are increasingly interconnected and dynamically reconfigurable (e.g., networked medical devices [4]), possessing greater authority and autonomy (e.g., driverless cars). As such, these systems exhibit emergent behavior along with the capability to learn and adapt through their usage. In our opinion, the appreciable degree of uncertainty about the actual operational system behavior renders existing safety certification approaches deficient. We believe this applies regardless of whether the underlying safety argument is implicit—as is the case, largely, in prescriptive certification—or explicit, where goal-based certification is concerned. In particular, as the system evolves after deployment, the mismatch between our understanding of the system (as documented in the safety case) and actual system operation, may potentially invalidate many of the prior assumptions made, undermine the evidence supplied and, thereby, defeat the safety claims made. Indeed, gaps between the documented safety reasoning and the actual safety of the system might lead to "a culture of 'paper safety' at the expense of real safety", as reported in the inquiry following the RAF Nimrod aircraft accident [5]. Despite significant improvements in operational safety monitoring, there is insufficient clarity on evolving the safety reasoning based on monitored data.

In short, there is a need for a new class of safety certification/assurance techniques that are continually assessing and evolving the safety reasoning, concurrently with the system, to provide *through-life safety assurance*. That is, safety assurance is provided not only during initial development and deployment, but also at runtime based on operational data. The intent is to transform safety assurance into a continuous, evolutionary activity. The *safety management system (SMS)* concept reflects the recognition of this need in many safety-critical sectors, e.g., (civil) aviation [6], and it aspires to similar properties, i.e., continuous safety assurance with predictive capabilities, but mainly considering in scope, business processes (including policies and procedures), and safety management capability.

This paper describes *dynamic safety cases (DSCs)*—along with a set of core principles and lifecycle activities—as an engineering solution for through-life safety assurance, where the data produced from an SMS can be utilized to create a continuously evolving assurance argument. In particular, we explore the hypothesis that DSCs can be rigorously implemented based on the following: (1) an explicit *safety argument*, annotated with *metadata* that provides tracing to relevant regulations, external concerns, requirements, individuals, system artifacts, and any other information deemed necessary; (2) a means of determining *confidence* in claims of the argument; (3) a collection of *monitors* providing the link between the system, argument elements and the confidence structure; and (4) a collection of *update rules* triggering actions in response

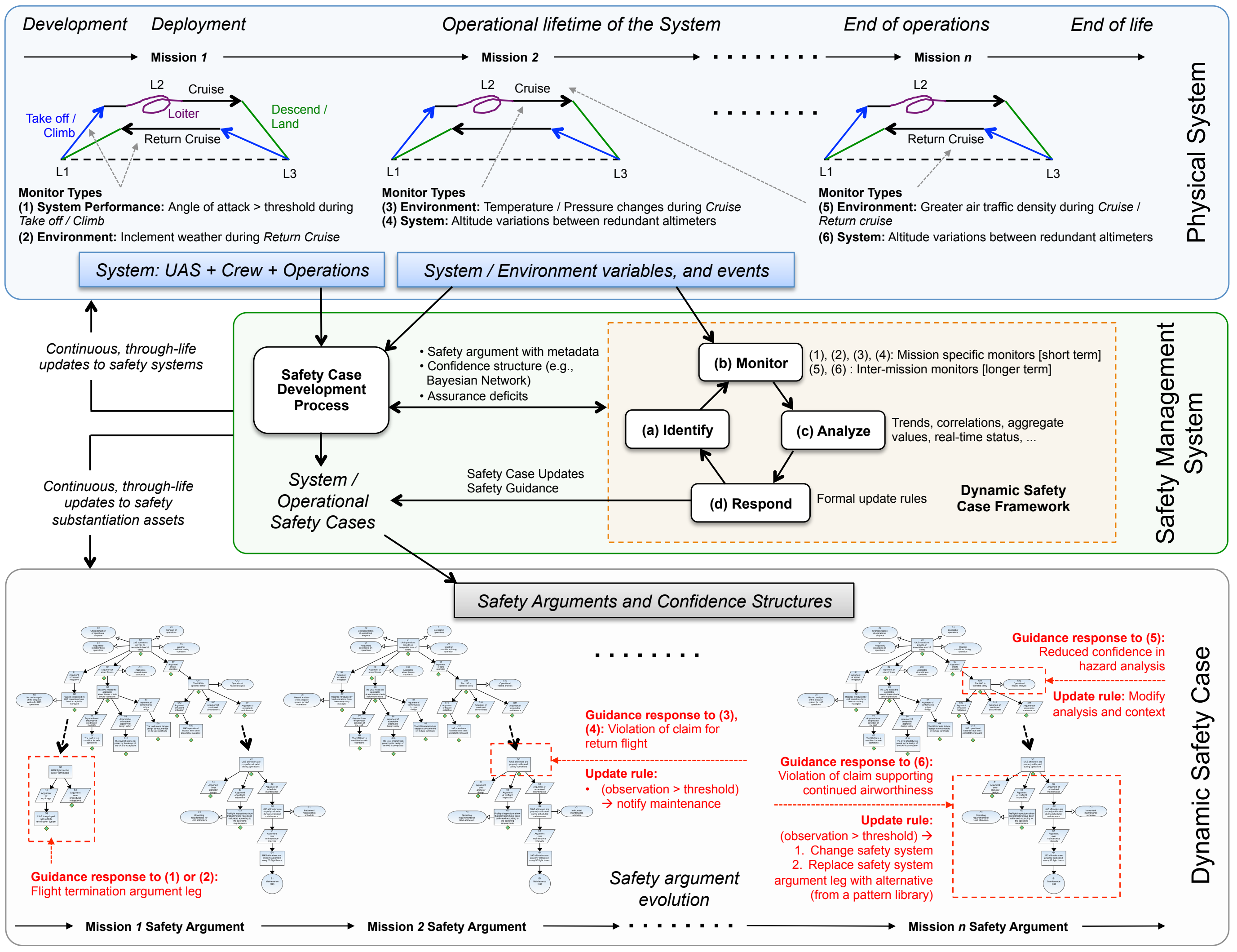
Development
Deployment
Operational lifetime of the System
End of operations
End of life
Mission 1
Mission 2
Mission n
Physical System
L2
Cruise
Loiter
Take off / Climb
Descend / Land
Return Cruise
L1
L3
Monitor Types
(1) System Performance: Angle of attack > threshold during Take off / Climb
(2) Environment: Inclement weather during Return Cruise
(3) Environment: Temperature / Pressure changes during Cruise
(4) System: Altitude variations between redundant altimeters
(5) Environment: Greater air traffic density during Cruise / Return cruise
(6) System: Altitude variations between redundant altimeters
System: UAS + Crew + Operations
System / Environment variables, and events
Continuous, through-life updates to safety systems
Safety Case Development Process
• Safety argument with metadata
• Confidence structure (e.g., Bayesian Network)
• Assurance deficits
(b) Monitor
(1), (2), (3), (4): Mission specific monitors [short term]
(5), (6) : Inter-mission monitors [longer term]
(a) Identify
(c) Analyze
Trends, correlations, aggregate values, real-time status, ...
(d) Respond
Formal update rules
Dynamic Safety Case Framework
Safety Case Updates
Safety Guidance
System / Operational Safety Cases
Continuous, through-life updates to safety substantiation assets
Safety Management System
Safety Arguments and Confidence Structures
Guidance response to (5): Reduced confidence in hazard analysis
Update rule: Modify analysis and context
Guidance response to (3), (4): Violation of claim for return flight
Update rule:
• (observation > threshold) → notify maintenance
Guidance response to (6): Violation of claim supporting continued airworthiness
Update rule: (observation > threshold) →
1. Change safety system
2. Replace safety system argument leg with alternative (from a pattern library)
Guidance response to (1) or (2): Flight termination argument leg
Safety argument evolution
Mission 1 Safety Argument
Mission 2 Safety Argument
Mission n Safety Argument
Dynamic Safety Case

Fig. 1. Framework for dynamic safety cases (DSCs): Extending the safety management system concept to provide *a*) continuous, through-life safety updates to the relevant systems, and *b*) management of the corresponding safety substantiation artifacts

to changes in the system, its safety argument, or the confidence in that argument.

II. Motivating Example

Typically, Unmanned Aircraft Systems (UASs) consist of one or more (unmanned) air-vehicles, ground control stations, flight crew, and communication infrastructure. An example mission scenario for such a system could be (Fig. 1, top): *a*) take-off from a *home* location L_1, *b*) transit to a *loiter* location L_2 following a defined heading, *c*) perform a specific set of operations at L_2, e.g., surveillance, measurement, payload delivery, etc., *d*) continue onward to a different landing location L_3, after which *e*) unload/swap the onboard payload/instruments. Thereafter, *f*) take off from location L_3, and *g*) return home, i.e., back to location L_1. Given a UAS category and its intended operations, flight operations may be authorized under certain rules that, in turn, are governed by weather conditions, e.g., visual flight rules (VFR) and visual meteorological conditions (VMC), in specific airspace classes, e.g., class G, no higher than a pre-determined altitude, e.g., 2000 ft., and may be restricted to specific airspace and/or geographic boundaries.

A. From Operational Monitoring to Safety Updates

During flight operations, both for an individual mission and over the lifespan of multiple missions (Fig. 1, top), events occurring in the operating environment and the UAS may necessitate systemic changes to maintain flight safety.

For example, in Fig. 1, during en route flight (i.e., between locations L_1 and L_3, and vice versa) the onset of inclement weather can change the parameters of both weather and instrument requirements for continued flight safety. Similarly, temperature and pressure changes beyond the allowed thresholds can potentially violate the operating constraints on aircraft performance, and affect critical instruments such as altimeters. Maintaining system safety would, then, require a combination of short-term changes (e.g., flight termination) together with longer term system modifications (e.g., replacing a sensor or actuator, thus requiring a modification to the relevant parts of the software and the corresponding verification artifacts) and/or procedural modifications (e.g., reducing the intervals between scheduled maintenance for instrument calibration).

Even when components of airborne systems function normally, as well as when there are deviations (from nominal operations) whose risk has been considered to be acceptable, there can be violations of the assumptions for continued flight safety. For instance, inclement weather in en route flight can affect the restriction of flight under VMC. In turn, a combination of system-wide changes may be required—such as the initiation of a flight termination procedure, notifying air traffic control, and/or updating flight plans—to maintain system safety. Similarly, variations observed in the density of air traffic over multiple missions may invalidate the assumptions made regarding the risk levels of the associated flight hazards. Consequently, existing hazard mitigation measures may turn out to be insufficient, necessitating a revision of the hazard analysis and, potentially, the introduction of new safety mechanisms.

B. *Updating the Safety Case*

We create the safety case for our example system in parallel with system development, and record it as a structured argument containing claims, reasoning and evidence that reflect the results of engineering and safety analyses/verification. We reflect the updates to the system in its safety argument (Fig. 1, bottom), based upon monitoring and analysis of the safety relevant-events/assurance variables, so as to make the argument consistent with the system, and to inform subsequent safety activities and system changes.

In general, *monitors* observe events and/or deviations of assurance variables from threshold values in a defined time interval, which can be symptomatic of the need for a safety related system change. For our example, we can use a variety of monitors including instruments, process metrics, etc., to observe: *a*) the *operational environment*, i.e., variables pertaining to weather conditions, air traffic and other airborne phenomena (e.g., birds); *b*) the *UAS*: in particular, performance parameters (e.g., airspeed), aerodynamic control surface variables (e.g., aileron positions), system health (e.g., of the powerplant), and maintenance status (e.g., maintenance log); *c*) *crew operations*, e.g., crew adherence to communication procedures; and *d*) the *safety culture*, e.g., effectiveness of safety reporting systems.

Based upon monitored data and the corresponding analysis, we capture the required safety modifications to the system (Section II-A), and the related modifications to the safety case by way of update rules (Section III-C). For instance, if greater than anticipated air traffic in a certain airspace sector during en route flight were to be observed over the next x flight hours, an update rule would be to notify the safety case team to revisit the hazard analysis within a certain time frame t. The eventual update to the safety case would be the argument that justifies the safety claims corresponding to any newly introduced safety mechanisms (resulting from the revised hazard analysis).

III. Framework

A. *Principles*

Based on our example, we believe that a framework for dynamic safety cases must support three fundamental principles:

1) *Proactively compute the confidence in, and update the reasoning about, the safety of ongoing operations*: DSCs go beyond conventional safety/health monitoring that largely deal with faults/failures to address *leading indicators* of, and *precursors* to, hazardous behaviors [7]. These relate to different types of uncertainty—i.e., aleatory (randomness in the system) and epistemic (incomplete knowledge about the system)—changes in which should challenge our confidence in system safety, prompting action on the indicators/precursors before they potentially develop into hazards. The ability to compute confidence from operational data is necessary to evolve safety reasoning and enable continuous safety assurance.

2) *Provide an increased level of formality in the safety infrastructure:* Here, the goal is to enable automated support for safety reasoning. To support automated analysis, e.g., determining dependencies between the system and corresponding argument fragments, and automatically updating the argument based on operational data, we need to move towards a more formal basis: not necessarily of the argument itself, but rather the infrastructure for creating the argument.

3) *Provide a mixed-automation framework*: The idea is to generalize from the notion of a *complete* standalone argument to a *partially-developed*, but well-formed, argument with open tasks assigned to various stakeholders. In general, the response to a change can be an automated update to the argument, or even the system itself, but may require human review/action depending on the nature and criticality of the change.

B. *Lifecycle*

Now, we suggest a lifecycle (Fig. 1, middle) comprising four continuous activities.

1) *Identify*: The sources of uncertainty in the safety case, i.e., so-called *assurance deficits* (ADs) [8], can weaken our confidence in safety. As the system and its safety argument change, so will the assurance deficits.

2) *Monitor*: We collect data at runtime related to both system and environment variables, events, and the ADs in the safety argument(s). To enable monitoring, we periodically interrogate both the argument and its links to external data, e.g., using argument querying [9].

3) *Analyze*: To understand the impact on safety reasoning, we analyze the operational data to examine whether the threshold defined for ADs are met, and to update the confidence in the associated claims.

4) *Respond*: Enabling a proactive response to operational events that affect safety assurance (by changing the system, environment, and the safety case, when necessary) is at the heart of DSCs. Deciding on the appropriate response depends on a combination of factors including the impact of confidence in new data, the available response options already planned, the level of automation provided, and the urgency with which certain stakeholders have to be alerted.

C. Towards a Formal Basis

We now sketch the key elements of a preliminary implementation of DSC, building on previous work on a rigorous approach to safety arguments [9]. In brief, we define an argument as a labeled tree, which is subject to various structural restrictions that describe well-formedness conditions. Nodes of the tree represent argument elements, and the labels give node descriptions, types, and *metadata*. We extend this notion for DSCs by making explicit the tight coupling of the argument with the system, i.e., we link the system and its argument via metadata (on argument nodes), and monitors to feed updated system information into confidence computations.

Thus, a DSC for a given system comprises:

1) A collection of *assurance variables* (AVs), including system, environment, and assurance artifacts—i.e., all safety-relevant artifacts, in general—as well as events (e.g., UAS landing);

2) An *argument structure*, with metadata relating its nodes to AVs, and the nodes of a *confidence structure*;

3) A *confidence structure*, e.g., a *Bayesian network* [10], whose nodes relate to argument fragments and monitor output;

4) A collection of *monitors* of type $(\mathsf{AVar}^* \rightarrow \mathsf{EnumVal} \mid \mathsf{ContinuousVal}) \times \mathsf{Period}$ which examine AVs and return either discrete (e.g., warning zones) or continuous output, with a given period. We use an abstract definition of monitor which, we assume, can also perform data analysis. Also, since events are considered as AVs, monitors can be triggered when a particular event occurs.

5) A collection of *update rules* of type $\mathsf{Condition} \rightarrow \mathsf{Action}^*$, with the general form $C[x] \Rightarrow \mathtt{forEach}(y :: Q \mid A[x, y])$. The rule condition C is a formula over the confidence structure and AVs; Q is a query in the AdvoCATE query language [9] that lets us determine those parts of the argument requiring changes (e.g., the evidence below some assumption); and A is a response action. Specific responses depend on the precondition and the queried nodes. For example,

a) Remove a branch of the argument depending on an invalidated assumption: $\mathtt{not}(\mathsf{trafficDensity} < n) \Rightarrow \mathtt{forEach}(y :: \mathsf{solves}^*\ \mathsf{Contextualizes} \mid \mathtt{replaceWith}(y, \mathtt{empty}))$.

b) Create a task for an engineer to inspect evidence items when the confidence in a particular branch drops below a threshold: $\mathtt{confidence}(\mathsf{NodeX}) < n \Rightarrow \mathtt{forEach}(E :: \mathsf{dependsOn}(E);\ \mathsf{traceTo}(\mathsf{NodeX})) \mid \mathtt{createTask}(\mathsf{engineer}, \mathsf{inspect}(E), \mathsf{urgent}))$.

IV. Concluding Remarks

We have described our vision of dynamic safety cases (DSCs), a novel approach to extend safety management systems. We are currently exploring different formalizations, in particular of the update rules, with a view to their implementation in our toolset, AdvoCATE [11], e.g., it might be useful to combine rules to define more complex responses. Although the current definitions do not allow it, monitors could alternatively feed directly into rule conditions. Rather than develop new monitors, our framework seeks to enable the integration of existing monitoring mechanisms [12] and analysis techniques [13] into safety management. DSCs can integrate with approaches for assuring open, adaptive systems, and provide a basis for realizing the concept of runtime certification [14]. Claims made in DSCs, e.g., about the expected system behavior under failure conditions, are interlinked with system requirements and the corresponding verification evidence. Thus, there is a notion of requirements update during runtime operations, similar to [15], and the opportunity to include evidence from runtime verification [16]. Intuitively, a dynamic safety case ought to have some notion of "dynamic robustness". For example, *safety integrity levels* (SILs) [17] reflect the idea that an increase in risk requires a corresponding improvement in mitigation. We could formulate this, and other such properties, as argument constraints and formally verify that a DSC actively meets those constraints.

Acknowledgment

This work was supported by the SASO project of the NASA ARMD Airspace Operations and Safety Program.

References

[1] R. Hawkins, I. Habli, T. Kelly, and J. McDermid, "Assurance Cases and Prescriptive Software Safety Certification: A Comparative Study," *Safety Science*, vol. 59, pp. 55–71, 2013.

[2] J. McDermid, "Software Safety: Where's the Evidence?" in *Australian Workshop on Safety Critical Systems and Software*, vol. 3, 2001, pp. 1–6.

[3] M. Sujan et al., *Evidence: Using Safety Cases in Industry and Healthcare*. The Health Foundation, Dec. 2012.

[4] N. Decker, F. Kühn, and D. Thoma, "Runtime Verification of Web Services for Interconnected Medical Devices," in *Proc. 25th Intl. Symp. Software Reliability Engineering (ISSRE 2014)*, Nov. 2014.

[5] C. Haddon-Cave, "The Nimrod Review: An Independent Review into the Broader Issues surrounding the Loss of the RAF Nimrod MR2 Aircraft XV230 in Afghanistan in 2006," Report, The Stationery Office, London, UK, Oct. 2009.

[6] International Civil Aviation Organization (ICAO), *Safety Management Manual (SMM)*, 3rd ed., 2013.

[7] T. Reiman and E. Pietikäinen, "Leading Indicators of System Safety – Monitoring and Driving the Organizational Safety Potential," *Safety Science*, vol. 50, no. 10, pp. 1993–2000, 2012.

[8] R. Hawkins, T. Kelly, J. Knight, and P. Graydon, "A New Approach to Creating Clear Safety Arguments," in *Proc. 19th Safety-Critical Systems Symposium (SSS' 11)*, Feb. 2011.

[9] E. Denney, D. Naylor, and G. Pai, "Querying Safety Cases," in *Computer Safety, Reliability and Security (SAFECOMP 2014)*, LNCS 8666, Sep. 2014, pp. 294–309.

[10] E. Denney, G. Pai, and I. Habli, "Towards Measurement of Confidence in Safety Cases," in *Proc. 5th Intl. Conf. Empirical Software Engineering and Measurement (ESEM 2011)*, Sept. 2011, pp. 380–383.

[11] E. Denney, G. Pai, and J. Pohl, "AdvoCATE: An Assurance Case Automation Toolset," in *Computer Safety, Reliability and Security: SAFECOMP 2012 Workshops*, LNCS 7613, Sep. 2012.

[12] M. Machin, et al., "Specifying Safety Monitors for Autonomous Systems using Model-checking," in *Computer Safety, Reliability and Security (SAFECOMP 2014)*, LNCS 8666, Sep. 2014, pp. 262–277.

[13] J. Krall, T. Menzies, and M. Davies, "Learning the Task Management Space of an Aircraft Approach Model," in *AAAI Spring Symp. Ser.*, 2014.

[14] J. Rushby, "Runtime Certification," in *Proc. 8th Intl. Workshop on Runtime Verification (RV)*, 2008, pp. 21–35.

[15] K. Welsh, P. Sawyer, and N. Bencomo, "Towards Requirements Aware Systems: Run-time Resolution of Design-time Assumptions," in *Proc. 26th Intl. Conf. Automated Software Engineering (ASE 2011)*, Nov. 2011, pp. 560–563.

[16] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive Software needs Quantitative Verification at Runtime," *Communications of the ACM*, vol. 55, no. 9, pp. 69–77, Sep. 2012.

[17] F. Redmill, "Safety Integrity Levels – Theory and Problems," in *Proc. 8th Safety-Critical Sys. Symp. (SSS '00)*, 2000, pp. 1–20.