A Methodology for the Development of Assurance Arguments for Unmanned Aircraft Systems

Ewen Denney, Ph.D.; SGT / NASA Ames Research Center; Moffett Field, California, USA

Ganesh Pai, Ph.D.; SGT / NASA Ames Research Center; Moffett Field, California, USA

Keywords: Safety case methodology, Assurance case, Argumentation, Unmanned aircraft systems, Aviation safety, Safety assurance

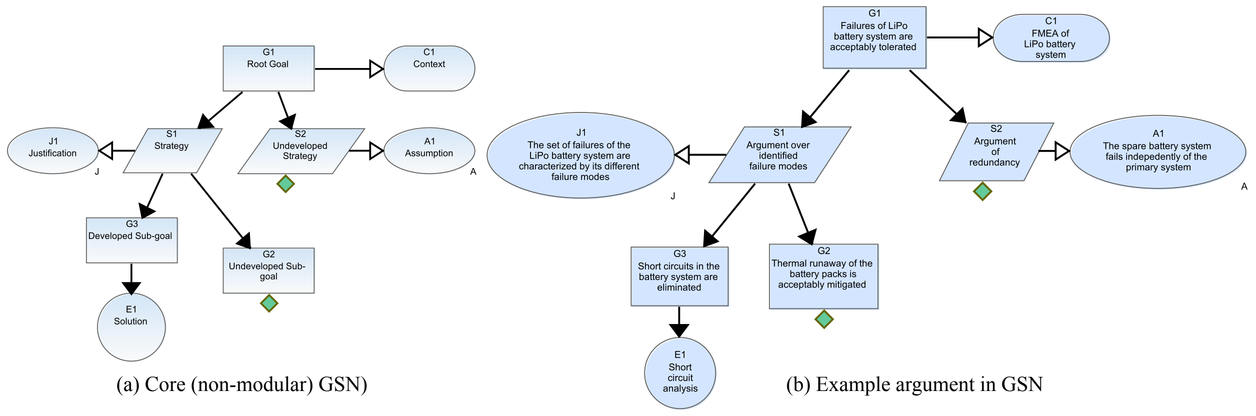
Abstract

A key requirement when obtaining regulatory authorization to conduct certain kinds of unmanned aircraft system (UAS) operations in civil airspace, e.g., beyond line-of-sight, and over congested areas, is to create and submit a safety case. Central to modern safety cases is a notion of *argument*, i.e., an explicit chain of reasoning linking the required safety substantiating evidence to the overall safety objectives and assertions. In this paper, we present a methodology for the principled development of structured arguments, supporting both top-down and bottom-up argument development approaches. We use the goal structuring notation (GSN) to present the arguments created, and leverage our toolset, AdvoCATE, for automation support. Our methodology focuses on the data flow between its six constituent activities building upon, and extending, our earlier work on a lightweight approach for assembling safety arguments from the artifacts of an integrated systems and safety engineering process for small UAS. We have applied some of the activities to create assurance arguments for real aviation systems, which in turn, has informed the development of the methodology. We give two examples of applying our methodology to UASs, addressing aspects of operational safety assurance in one example, and airworthiness in the other.

Introduction

Creating and submitting a safety case is both an accepted best practice and a regulatory requirement in many safety-critical industries. In the aviation sector, however, safety cases have been used largely by the military (ref. 1), where *safety targets* representing an acceptable level of risk are established, and the safety case provides the requisite assurance that those targets have been satisfied. In contrast, civil aviation uses a combination of highly prescriptive *normative regulations*, which mandate concrete product requirements and compliance processes, and so-called *performance-based regulations* specifying minimum operating performance standards (MOPS). Nevertheless, safety cases have been used for risk management and safety assurance of a variety of civil aviation systems, e.g., those supporting flight-crew operations in terminal-area airspace (ref. 2), and air-traffic management operations during en-route flight (ref. 3). With respect to unmanned aircraft systems (UASs), the requirement to produce a safety case ultimately depends on the governing regulatory authorities and applicable policies (refs. 4–6), besides the UAS type, configuration, and type of operations. Nonetheless, certain kinds of operations, e.g., beyond line-of-sight (BLOS), in congested airspace, or at night, currently do require a convincing safety case (refs. 5–7) for flight authorization to be granted. It has also been suggested that safety cases may be appropriate for airworthiness assurance of certain types of UAS (ref. 8), e.g., those possessing non-standard equipment, or novel design/safety features.

A safety case is a specialization of the more general notion of *assurance case*, which can be used to provide assurance of broad system concerns including dependability, safety, and security. In particular, a safety case is a comprehensive, defensible, and valid justification of the safety of a system for a given application in a defined operating environment. This concept is compatible with (and largely similar to) what is considered to be a safety case in many relevant aviation standards and guidance documents. Indeed, common to all are the requirements to explicitly state the overall safety assertions and objectives, i.e., the safety *claims*, and supply the safety substantiating *evidence*. However, each also supplies its own context/application-specific interpretation of the exact purpose and nature of the safety case (refs. 1, 5, 6, 9, 10), together with the required components, expected content and presentation format. Associated with safety cases is an additional notion of *argument*—i.e., a chain of reasoning connecting the claims and the evidence—although, depending on the standard/guidance document used, arguments may be either explicitly required (refs. 1, 5, 10), or implicitly created (ref. 6). The motivation to use structured arguments for UAS safety assurance is twofold: first, they enable the explicit tracing of safety and airworthiness considerations, from concept, to requirements, to evidence of risk mitigation and control; secondly, they are useful as a centralized organizing component of the diverse assurance information aggregated by a UAS safety case.


G1
Root Goal
C1
Context
J1
Justification
J
S1
Strategy
S2
Undeveloped Strategy
A1
Assumption
A
G3
Developed Sub-goal
G2
Undeveloped Sub-goal
E1
Solution
G1
Failures of LiPo battery system are acceptably tolerated
C1
FMEA of LiPo battery system
J1
The set of failures of the LiPo battery system are characterized by its different failure modes
J
S1
Argument over identified failure modes
S2
Argument of redundancy
A1
The spare battery system fails indepedently of the primary system
A
G3
Short circuits in the battery system are eliminated
G2
Thermal runaway of the battery packs is acceptably mitigated
E1
Short circuit analysis

(a) Core (non-modular) GSN) (b) Example argument in GSN

Figure 1 — Simple example of a structured argument and a notation for its presentation.

Additionally, arguments have been shown to make it easier to comprehend and critically review a safety case (ref. 11). To further improve clarity in presenting arguments, graphical notations such as the goal structuring notation (GSN) have emerged, which has not only seen some usage in both civil and military aviation (refs. 1, 2), but also has undergone standardization (ref. 12). Recently, GSN has been provided with formal foundations (refs. 13–15).

We have previously defined a lightweight approach for assembling safety arguments—using the artifacts of an integrated systems and safety engineering process for small UAS (refs. 16, 17)—which we extend in this paper. We first describe a generic methodology for the principled creation of (safety) assurance arguments, after which we describe its application to two real UAS examples: *i*) addressing the safety of a ground-based detect-and-avoid (GBDAA) capability, which was used in transit operations with fixed-wing UASs; and, *ii*) airworthiness assurance (in particular, type design assurance), of an unmanned rotorcraft system (URS) whose operational concept includes BLOS and night operations. Our methodology supports both *top-down*, and *bottom-up* development of arguments, uses the GSN to present the arguments created, and leverages automation support through our toolset AdvoCATE (ref. 18).

Structured Arguments

An *argument* is a connected series of propositions used in support of the truth of an overall proposition. The latter is usually termed as a *claim*, whereas the former represents a chain of reasoning connecting the claim and the *evidence*. Our vision of a safety case is a structured, and evolving, argument that comprises explicit *safety claims*, assimilates heterogeneous *safety-substantiating evidence*, and presents the reasoning required to conclude that a system will be safe for a defined application and operating environment. We present the elements of a safety case as an *argument structure*, i.e., a diagrammatic presentation of the underlying argument using the goal structuring notation (GSN). Figure 1a shows an argument as a directed acyclic graph of different GSN nodes and links; the different node types shown—i.e., *goal*, *strategy*, *assumption*, *justification*, and *solution*—represent the core argument elements, whereas the links specify *support* or *contextual* types[1] of relationships between the nodes. GSN also has, among other abstractions, notational extensions for modularity, e.g., *module references*, and *away* nodes, though we will not cover those here.[2]

In general, nodes refer to external items including *a*) artifacts such as hazard logs, requirements documents, design documents, various relevant models of the system, etc.; *b*) the results of engineering activities, e.g., safety, system, and software analyses, various inspections, reviews, simulations, and verification activities including different kinds of system, subsystem, and component-level testing, formal verification, etc.; and *c*) records from ongoing operations, as well as prior operations, if applicable. Nodes additionally contain *metadata* drawn from domain ontologies that provide supplementary and relevant domain-specific semantic information. Figure 1b gives a simple illustrative example of an argument structure in the core GSN (retaining the layout of Figure 1a, to aid understanding).

[1] Links with filled arrowheads show the *support* relationship; links with the hollow arrowhead show the *contextual* relationship.
[2] For more details on GSN and its extensions, see reference 12.

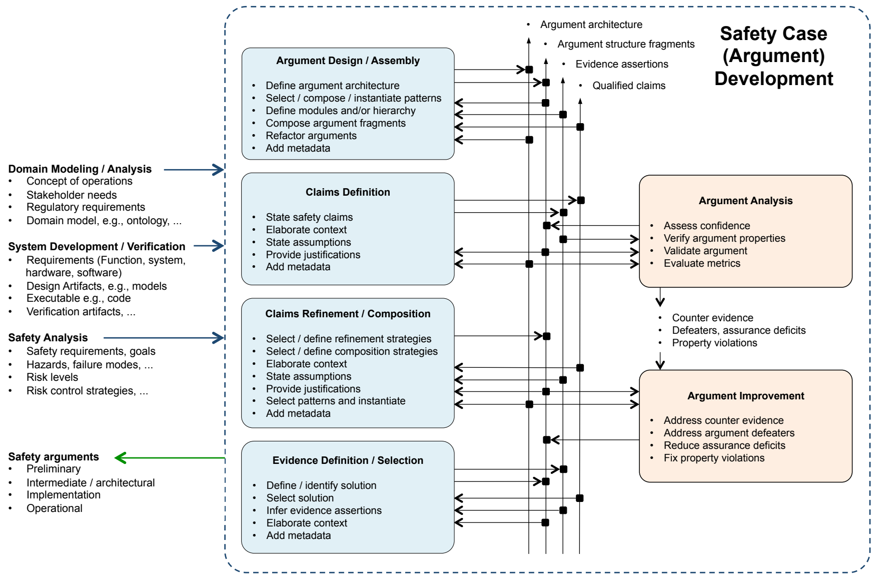

Safety Case (Argument) Development
Argument Design / Assembly
Define argument architecture
Select / compose / instantiate patterns
Define modules and/or hierarchy
Compose argument fragments
Refactor arguments
Add metadata
Claims Definition
State safety claims
Elaborate context
State assumptions
Provide justifications
Add metadata
Claims Refinement / Composition
Select / define refinement strategies
Select / define composition strategies
Elaborate context
State assumptions
Provide justifications
Select patterns and instantiate
Add metadata
Evidence Definition / Selection
Define / identify solution
Select solution
Infer evidence assertions
Elaborate context
Add metadata
Argument architecture
Argument structure fragments
Evidence assertions
Qualified claims
Argument Analysis
Assess confidence
Verify argument properties
Validate argument
Evaluate metrics
Counter evidence
Defeaters, assurance deficits
Property violations
Argument Improvement
Address counter evidence
Address argument defeaters
Reduce assurance deficits
Fix property violations
Domain Modeling / Analysis
Concept of operations
Stakeholder needs
Regulatory requirements
Domain model, e.g., ontology, ...
System Development / Verification
Requirements (Function, system, hardware, software)
Design Artifacts, e.g., models
Executable e.g., code
Verification artifacts, ...
Safety Analysis
Safety requirements, goals
Hazards, failure modes, ...
Risk levels
Risk control strategies, ...
Safety arguments
Preliminary
Intermediate / architectural
Implementation
Operational

Figure 2 — Methodology for developing (safety) assurance arguments, comprising six distinct activities.

In Figure 1b, we state the top-level claim—i.e., "Failures of the LiPo battery system are acceptably tolerated"—in the *goal node* G1, and in context of the failure modes and effects analysis (FMEA) of the LiPo battery system (shown as the *context node* C1). We decompose this claim using two *strategies*, S1 and S2, which provide complementary arguments, i.e., over the identified failure modes, and of redundancy, respectively. The latter relies on an assumption of independence in failures of the redundant batteries (*assumption node* A1), but has not been further developed (indicated by the '◇' node annotation). The use of the former has been justified (in the *justification node* J1), and results in two sub-goals: G2 (concerning the acceptable mitigation of thermal runaway), and G3 (concerning the elimination of short circuits in the battery system), respectively. The former remains to be developed, while the latter is addressed by the *evidence node* E1, i.e., short circuit analysis.

Argument Development Methodology

Overview: Our process for (safety) assurance argument development (Figure 2) is concurrent, and iterative, with the processes for domain modeling/analysis, safety analysis, and system development/verification (shown in Figure 2, as providing inputs to the argument development process). The processes are periodically synchronized at the milestones defined during the plan for system development and certification (ref. 16). At each milestone, we produce an argument structure that reflects (at that point in the system development process) the inclusion of the specific artifacts available, and the state of the safety argument (and, consequently, system safety). Safety argument evolution can, then, be characterized by the series of interrelated argument structures produced at the various milestones. Thus, a *preliminary* safety argument (shown in Figure 2, as an output of the argument development process) would correspond to a milestone at the end of the concept definition and requirements analysis phase, whereas an *architectural* safety argument would correspond to the milestones at the end of the phases of system architecture development and system design. As shown in Figure 2, our methodology for argument development comprises six distinct activities; namely: *i*) argument design/assembly, *ii*) claims definition, *iii*) claims refinement/composition, *iv*) evidence definition/selection, *v*) argument analysis, and *vi*) argument improvement. The activities *i*) – *iv*) can be considered as the *core* process for argument development, while the remaining two activities play a supporting role.

We focus mainly on the data flow, rather than control flow, since the order in which the activities are performed will be tailored to the specific needs of the project and domain. In general, argument development can be performed in a top-down, or a bottom-up manner. The former entails a definition of a high-level argument organization (and the constituent top-level claims), followed by successive refinement into lower-level details. In contrast, the latter is concerned with assembling an argument based upon the inferences that can be drawn from the available and existing lower-level details (i.e., the evidence). In practice, however, it is not uncommon to utilize a combination of both top-down and bottom-up argument development. Next, we describe each activity of the argument development process.

Argument Design/Assembly: In a top-down development of safety arguments, first we define an *argument architecture*, which specifies a high-level and abstract organization of the overall structure/elements of the argument. One approach to realize the argument architecture is to select and compose *argument patterns*—i.e., abstractions representing various styles of argument (ref. 13)—taking into account system assurance concerns, the types of claims requiring support, and argument design criteria such as compliance with safety principles, reducing the cost of re-certification, modular organization, maintainability, etc. We instantiate patterns using domain- and system-specific data (which may, itself, be generated using a tool, e.g., for formal verification), to produce fragments of *instance arguments*. We can directly compose instance arguments (to be consistent with pattern composition), or there may be a need to introduce manually-created, intermediate, *glue arguments*. Based on the project needs, we can then specify a modular organization of the argument architecture, which can be beneficial in a number of ways, e.g., to reflect the modularity inherent in the system, to manage safety argument size, to constrain the impact of changes during argument evolution, as well as to support distributed development. Furthermore, we can also introduce a hierarchical organization in argument fragments to reflect the natural hierarchy of claims, or the refinement of an abstract argument fragment into a more detailed argument.

During both top-down and bottom-up development, we use the argument fragments produced from pattern instantiation, along with those produced from the remaining activities (e.g., claims refinement/composition, and evidence definition/selection, as shown in Figure 2) to assemble an argument consistent with its architecture. As both the system and its assurance argument evolve, we refactor the argument to improve its comprehensibility, and the consistency with its architecture and/or argument design criteria (e.g., maintainability). Essentially, the activity of argument design/assembly is one wherein we combine top-down and/or bottom-up argument development, together with tasks for improving argument understandability and supporting argument evolution.

Claims Definition: We define safety claims based upon the system development phase and the available artifacts, e.g., using safety requirements identified through hazard analyses. We additionally state the context in which the claims can be interpreted and determined to be valid, along with any relevant assumptions and the necessary justifications. Note that the assumptions made can be both about the system for which assurance is sought, and the environment in which the system operates. We will refer to the combination of claim, its associated context, justifications and assumptions as a *qualified claim*. For example, in a safety argument, a qualified correctness claim about a software function would be accompanied with its specification as context, an assumption (or an auxiliary claim supported by evidence) that the specification is valid, and a justification of the bearing of the correctness of that function on system safety. The results of the claims definition activity are propositions—specified at an appropriate level of abstraction—concerning system, subsystem, and component properties, which have been determined through safety analysis to have a (direct or indirect) bearing on system safety, e.g., the reliability of a hardware fail-safe, the correctness of a software switch, etc. When a claim concerns low-level assertions about evidence items, we distinguish them as *evidence assertions*.

Claims Refinement/Composition: The core task of this activity is to define and/or select the appropriate strategies to link related claims. Then, we specify the associated rationale, i.e., the appropriate justifications, assumptions, as well as the relevant context related to a reasonable use of a strategy. Specifically, claims refinement is an iterative and successive decomposition of higher-level, abstract claims into their lower-level refinements, performed in top-down argument development. For example, to develop a claim of subsystem reliability into sub-claims about the reliability of the constituent components, we can use a strategy of reasoning over *cut-sets*, i.e., the unique combinations of components whose failure leads to subsystem failure; the subsystem architecture serves as associated context, and the subsystem failure analysis provides the requisite justification for using the strategy. Since argument patterns specify an abstraction of the argument obtained from applying these activities to specific types of claims, an additional approach to implement the activity of claims (definition and) refinement, is pattern instantiation.

In a bottom-up development, the strategies chosen must aggregate or compose, rather than refine, lower-level claims into higher-level claims. Typically, this is required when solutions (such as the result of a specific verification) are available from which evidence assertions can be inferred, and which must be linked to the higher-level claims requiring support. For example, low-level assertions of successful unit testing can be used to (partially) support a software fitness claim through a strategy of (unit) test-driven verification applied to the individual software units. That strategy would also be accompanied with a description of the appropriate context, and the justifications of the relevance of that strategy. In general, the result of claims refinement/composition is a collection of fragments of argument structures, whose leaf nodes are qualified claims, evidence assertions, or solutions.

Evidence Definition/Selection: The result of the evidence definition/selection activity is a collection of argument structure fragments in which claims have been supported by solutions, or solutions provide evidence assertions that must be composed to support higher-level claims. During top-down development, we specify evidence requirements in the early phases of system development and during safety analysis. As we develop the system, we refine the evidence requirements and choose/produce those solutions that provide the requisite degree of assurance. For example, a (higher-level) claim of software fitness may be supported by a proof of correctness, as well as through other verification evidence, such as testing, static analysis, etc. In general, the choice of the evidence to be used depends upon the evidence assertion obtained from claims refinement, and the degree of assurance required. Thus, a lower-level assertion of functional correctness would be better supported by a proof, whereas an assertion of reliability would necessitate software testing for a specified duration in an environment representative of actual operations. When solutions are available, from which evidence assertions can be inferred, then evidence selection also involves determining whether the available solutions meet the evidence requirements, are trustworthy, and are appropriate for the claims that require support.

Argument Analysis: Argument analysis involves a number of tasks whose goal is to analyze the soundness, or cogency, of the argument structures produced; namely:

a) *Property verification*: We specify (structural) argument properties and verify them to evaluate the *quality* of the argument structure. Properties can reflect concerns such as *well-formedness*, e.g., the argument structure contains no cyclic links, *internal completeness*, i.e., all paths from all claims in the argument structure lead to evidence, as well as more general structural constraints required by best practices, e.g., a specific type of claim has at least two or more independent paths to, and/or forms of, supporting evidence.

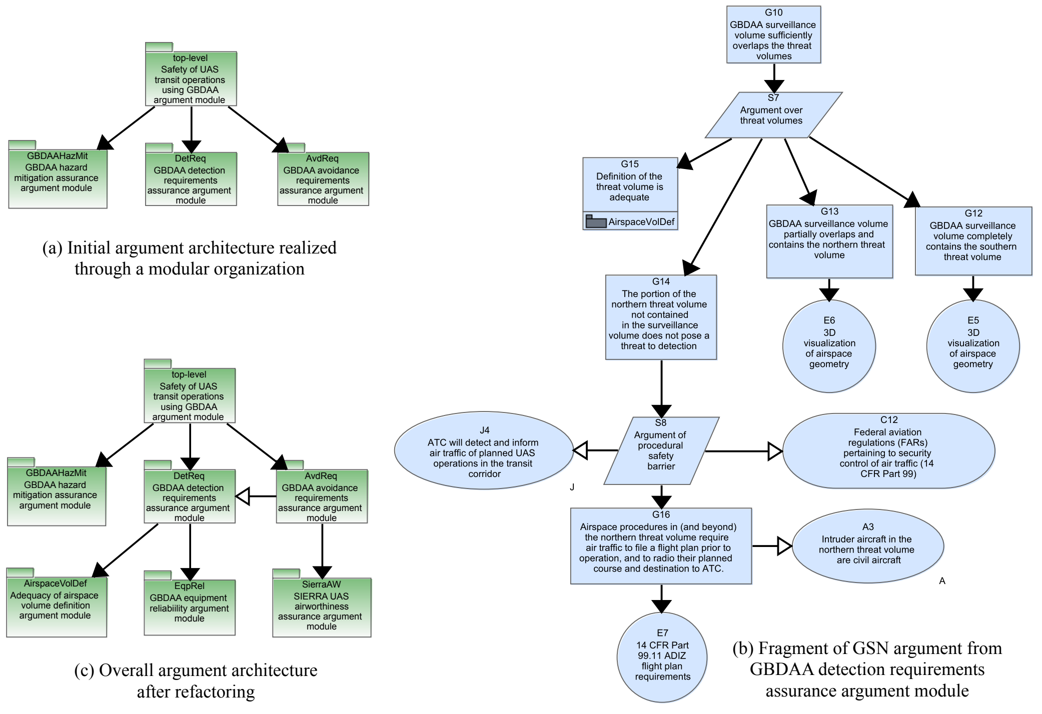
b) *Argument validation*: We evaluate the argument structure for *fallacies* (ref. 20), i.e., possible flaws in the reasoning; *assurance deficits* (ref. 21), i.e., knowledge gaps that lower confidence in claims; and *defeaters* (ref. 22), i.e., ways to attack an argument. Additionally, we examine the elements of the argument structure for relevance and consistency, following which we identify counter-evidence that may either undermine the solutions provided and/or reduce the strength of the argument.

c) *Confidence assessment*: This task supports safety-related decision making by characterizing the confidence that can be placed in an argument. The goal is to evaluate the credibility of safety claims on the basis of the strength of the argument and the veracity of the evidence supplied. Currently, a number of ways are available to evaluate confidence, e.g., qualitatively, by using so-called *confidence arguments* (ref. 21), through the use of probabilistic models—such as Bayesian networks—which specify a joint distribution over the underlying sources of argument uncertainty (ref. 23), or through belief combination (ref. 24).

d) *Metrics-based assessment*: Metrics computed on argument structures, such as *size*, or *coverage* (ref. 18), are useful to gauge the progress of the safety argument through its evolution. Specifically, metrics provide a convenient summary of the state of the argument development process and, thereby, a means to gauge whether synchronization with the remainder of the processes is feasible at a given milestone. For instance, to synchronize an architectural safety case with, say, a milestone of design review, we can compute metrics that measure the extent to which higher-level system safety claims have been developed into lower-level subsystem/component-level claims. Here, an underlying assumption is that decision models exist to interpret the computed values of a metric against the acceptability thresholds established for process control.

Argument Improvement: Whereas argument analysis is a retrospective activity, argument improvement uses the results of argument analysis for (proactively) improving the argument through the following sub-tasks, namely: *a*) including the counter-evidence identified, *b*) resolving the identified argument defeaters, *c*) reducing the identified assurance deficits, *d*) modifying the argument structure to address fallacies, and *e*) fixing property violations, if any. No specific order is imposed on the tasks, each of which represents a source of change to the argument structure. Thus, improving an argument amounts to determining the exact changes to be made, their nature, and their scope. In general, changing the argument amounts to editing the argument by adding, removing, or replacing argument fragments, e.g., as proposed in references 25 and 26. Sub-tasks *a*) and *b*) in particular may require the definition of notational extensions to GSN (refs. 21, 22, 26).

Through our toolset for assurance case automation, AdvoCATE (refs. 13–15, 18), automation support exists for a number of tasks during argument development, e.g., pattern instantiation, hierarchisation, modular organization, property verification, and metrics-based assessment. The latter two tasks in particular leverage, and extend, the *querying* capability of the tool (ref. 27) which, in turn, *i*) relies on enriching the argument with (domain-specific) *metadata,* and *ii*) permits the generation of *views*, i.e., fragments of an argument that satisfy a specific query.


top-level
Safety of UAS transit operations using GBDAA argument module
GBDAAHazMit
GBDAA hazard mitigation assurance argument module
DetReq
GBDAA detection requirements assurance argument module
AvdReq
GBDAA avoidance requirements assurance argument module
AirspaceVolDef
Adequacy of airspace volume definition argument module
EqpRel
GBDAA equipment reliability argument module
SierraAW
SIERRA UAS airworthiness assurance argument module
G10
GBDAA surveillance volume sufficiently overlaps the threat volumes
S7
Argument over threat volumes
G15
Definition of the threat volume is adequate
AirspaceVolDef
G13
GBDAA surveillance volume partially overlaps and contains the northern threat volume
G12
GBDAA surveillance volume completely contains the southern threat volume
G14
The portion of the northern threat volume not contained in the surveillance volume does not pose a threat to detection
E6
3D visualization of airspace geometry
E5
3D visualization of airspace geometry
J4
ATC will detect and inform air traffic of planned UAS operations in the transit corridor
J
S8
Argument of procedural safety barrier
C12
Federal aviation regulations (FARs) pertaining to security control of air traffic (14 CFR Part 99)
G16
Airspace procedures in (and beyond) the northern threat volume require air traffic to file a flight plan prior to operation, and to radio their planned course and destination to ATC.
A3
Intruder aircraft in the northern threat volume are civil aircraft
A
E7
14 CFR Part 99.11 ADIZ flight plan requirements

(a) Initial argument architecture realized through a modular organization

(b) Fragment of GSN argument from GBDAA detection requirements assurance argument module

(c) Overall argument architecture after refactoring

Figure 3 — Excerpt of modular argument architectures, and a fragment of the GBDAA safety argument.

Application to Unmanned Aircraft Systems

We briefly describe two examples where we have applied our methodology to create (safety) assurance arguments for real UASs and their operations. Due to space limits, we mainly focus on describing the core argument development process, i.e., the activities of argument design, claims definition, claims refinement/composition, and evidence definition/selection, whereas the supporting activities of argument analysis and improvement are not covered.

Safety of Ground-based Detect-and-Avoid: In late 2013, an earth science mission funded by the National Aeronautics and Space Administration (NASA) employed a ground-based detect-and-avoid (GBDAA) capability—i.e., an alternative means of compliance to the so-called *see-and-avoid* requirements[3] of the federal aviation regulations (FARs), through the use of ground-based, or airborne sensors, e.g., radar—during UAS transit operations. As such, FAA approval of UAS operations utilizing detect-and-avoid capabilities currently requires a safety case. We were part of the team involved in creating the associated safety case that the FAA approved (ref. 7). Though that safety case did not require (and therefore did not contain) explicit safety assurance arguments, we have retrospectively created them as an exercise in combining top-down and bottom-up argument development. In part, that exercise has informed the development of our methodology to created structured assurance arguments. In brief, the safety case provided assurance that UAS transit operations could be safely conducted by using GBDAA instead of visual observers, by asserting that *i*) the functional requirements of the GBDAA system, i.e., for detection and avoidance, were met through evidence comprising airspace analysis and flight testing; and that *ii*) any additional hazards introduced by the GBDAA system could be acceptably managed by defining (and executing) specific operational procedures, as well as by leveraging existing hazard controls afforded by the current air-traffic management system. Essentially, this reflects the (initial) high-level architecture of the argument underlying the safety case.

Argument Design (Define Argument Architecture): Figure 3a gives a modular view of this initial argument architecture. Here, the module node *top-level* contains the broader claim of the safety of UAS transit operations, in which the GBDAA capability represents *one* safety mechanism to address a mid-air collision (MAC) hazard. Other mechanisms include airspace deconfliction, and temporal and/or spatial separation of concurrent flight operations. The arguments within the modules *DetReq* and *AvdReq* provide assurance that the functional requirements on the

[3] Specifically, 14 CFR Part 91, Subpart B, §91.111, §91.113, and §91.115.

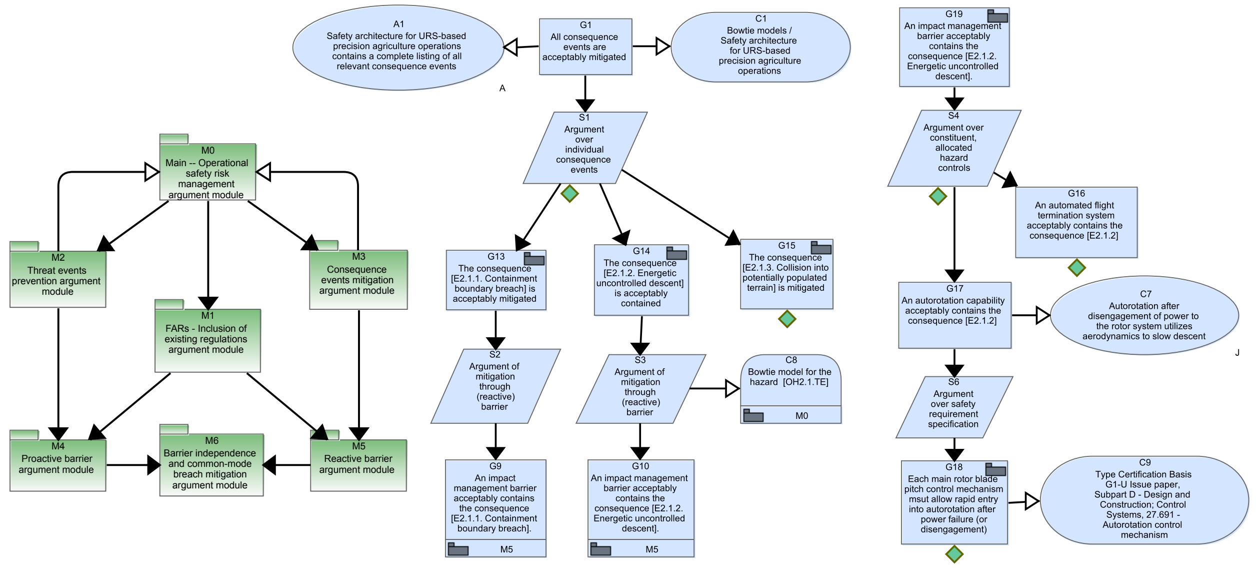
GBDAA system are met. Likewise, the argument module *GBDAAHazMit* contains the argument that addresses the claims of mitigating the hazards introduced by the GBDAA capability.

Applying Claims Definition, Claims Refinement/Composition, and Evidence Definition/Selection: Concurrently with argument architecture definition, we perform the activities of claims definition, and claims refinement (Figure 2), where we develop the concrete arguments in each module. For example, in the module *top-level,* we refine the objective of safe UAS transit operations into sub-claims concerning the operational requirements on detection and avoidance. In turn, we allocate those sub-claims to the GBDAA system, and further develop them in the argument modules *DetReq* and *AvdReq* respectively. Figure 3b gives a fragment of the overall assurance argument (not shown) in the module *DetReq*, whose main safety objective is the satisfaction of the detection requirements of UAS transit operations by the GBDAA sensor component (i.e., the ground-based radar). We apply, and automatically instantiate[4], the *requirements-breakdown* argument pattern (ref. 13) to develop that safety objective into lower-level claims concerning the individual detection requirements. For instance, one such claim concerns the requirement of sufficient surveillance coverage; refining that lower-level claim eventually results in a sub-claim (goal node G10 in Figure 3b) of sufficient overlap between the three-dimensional (3D) airspace volume covered by radar surveillance, i.e., the *surveillance volume*, and the airspace volume(s) where a credible threat of MACs exists, i.e., the *threat volume*(*s*). We support the claim of sufficient overlap between the surveillance and threat volumes, in part through an argument over individual threat volumes, and eventually through (analytical) evidence comprising 3D airspace modeling and visualization showing the degree of overlap between the relevant airspace volumes. In Figure 3b, this reasoning is conveyed by the argument *legs* containing the goal nodes G12 and G13 supported by the solution nodes E5 and E6 respectively. Through an examination of the FARs (as well as from individuals who were familiar with the operational airspace) it had also been determined that a procedural mitigation was already in place to address certain kinds of air traffic threats (solution node E7). The argument leg including the nodes S8, G16, E7, C12, J4 and A3 (Figure 3b) was then created in a bottom-up manner, and determined to provide an effective mitigation to intruder aircraft arising from an uncovered area of the threat volume (goal node G14).

Argument Design (Refactor Arguments): After performing the activities of claims definition, claims refinement/composition, and evidence definition/selection, the modules *DetReq* and *AvdReq* each contained claims (and complete arguments, where appropriate) for equipment reliability, adequacy of characterizing the different airspace volumes, and airworthiness. Additionally, there were contextual dependencies between the arguments across the two modules, e.g., the argument for assuring that a specific avoidance procedure is successful requires, as context, the reliable detection of threats in a specific flight phase. We refactored the arguments into modules addressing equipment reliability (*EqpRel*), airspace volume definition (*AirspaceVolDef*), and ownship airworthiness (*SierraAW*). Figure 3c shows the resulting refactored argument architecture identifying these additional modules and the contextual dependency. Thus, there is a refinement of both the argument and its (modular) architecture where changes or updates to one affect the other, reflecting a combination of both top-down, and bottom-up argument development.

<u>Airworthiness/Safety Assurance of an Unmanned Rotorcraft</u>: Airworthiness refers to the fitness of an air vehicle for safe operations and, in general, it can be considered to be a reliability barrier to (flight) safety hazards. A key aspect of UAS airworthiness assurance, which considers the entire system including the ground-based components, is to establish that the UAS configuration conforms to a *type design* (i.e., all relevant system design documentation), which is to be, itself, approved against a *certification basis*, i.e., the relevant airworthiness standards and regulations. As part of a NASA project, we are part of a team undertaking the development of a certification basis for a class of unmanned rotorcraft systems (URSs), where we are applying our methodology for developing structured arguments. Our goal is to provide a common framework to *i*) organize the preliminary safety case, *ii*) communicate the derivation of requirements on airworthiness and operational safety, in particular those relevant for type design assurance, and *iii*) eventually record how those requirements have been met by a specific URS design[5]. The initial concept considers daytime operations for applications such as precision agriculture. Future concepts extend this to include operations at nighttime, in reduced visibility conditions, and BLOS, for diverse applications including pipeline monitoring, etc. The rotorcraft characteristics—e.g., a turbine-powered tandem rotor configuration, a maximum takeoff weight (including payload) of 1000 lb., a physical envelope of 21 ft. x 13 ft. x 5.5 ft., a (payload dependent) endurance up to 5 h., and a maximum airspeed of 105 knots—warrant airworthiness determination, since operations over any populated areas pose an appreciable safety risk in the presence of in-flight failures.

[4] Using our toolset for assurance case automation, AdvoCATE (ref. 18).
[5] Although, this aspect is out of scope for the work presented here.


A1
Safety architecture for URS-based precision agriculture operations contains a complete listing of all relevant consequence events
A
G1
All consequence events are acceptably mitigated
C1
Bowtie models / Safety architecture for URS-based precision agriculture operations
S1
Argument over individual consequence events
G13
The consequence [E2.1.1. Containment boundary breach] is acceptably mitigated
G14
The consequence [E2.1.2. Energetic uncontrolled descent] is acceptably contained
G15
The consequence [E2.1.3. Collision into potentially populated terrain] is mitigated
S2
Argument of mitigation through (reactive) barrier
S3
Argument of mitigation through (reactive) barrier
C8
Bowtie model for the hazard [OH2.1.TE]
M0
G9
An impact management barrier acceptably contains the consequence [E2.1.1. Containment boundary breach].
M5
G10
An impact management barrier acceptably contains the consequence [E2.1.2. Energetic uncontrolled descent].
M5
M0
Main -- Operational safety risk management argument module
M2
Threat events prevention argument module
M3
Consequence events mitigation argument module
M1
FARs - Inclusion of existing regulations argument module
M4
Proactive barrier argument module
M6
Barrier independence and common-mode breach mitigation argument module
M5
Reactive barrier argument module
G19
An impact management barrier acceptably contains the consequence [E2.1.2. Energetic uncontrolled descent].
S4
Argument over constituent, allocated hazard controls
G16
An automated flight termination system acceptably contains the consequence [E2.1.2]
G17
An autorotation capability acceptably contains the consequence [E2.1.2]
C7
Autorotation after disengagement of power to the rotor system utilizes aerodynamics to slow descent
J
S6
Argument over safety requirement specification
G18
Each main rotor blade pitch control mechanism msut allow rapid entry into autorotation after power failure (or disengagement)
C9
Type Certification Basis G1-U Issue paper, Subpart D - Design and Construction; Control Systems, 27.691 - Autorotation control mechanism

(a) Fragment of modular argument architecture

(b) Fragment of argument structure in module M3 (consequence events mitigation)

(c) Fragment of argument structure in module M5 (reactive barriers)

Figure 4 — Excerpt of modular argument architecture for URS airworthiness/operational safety assurance

Argument Design: Using the URS concept of operations, we first conduct an operational hazard assessment to identify operational hazards, followed by a functional hazard analysis to identify a class of *threat events*, i.e., the events contributing to a hazard, including URS and operator functional failures/deviations. Thereafter, we define a *barrier model* specifying an abstract safety architecture, i.e., the combination of *proactive* and *reactive* hazard controls meant to manage operational hazards. The former are concerned with preventing the identified threat events, whereas the latter attempt to mitigate/contain *consequence events* when hazard prevention is unsuccessful. For example, during operations at higher altitudes, a '*loss of control with a deviation from the flight path*' is a hazard with potentially catastrophic worst-case consequences including an energetic uncontrolled descent and collision into populated terrain. The safety architecture to manage this hazard comprises *a*) proactive barriers, such as elements of airworthiness to minimize in-flight failures, vertical and lateral containment systems to reduce exposure by creating a *geo-fence* around the intended operational area, as well as *b*) reactive barriers, such as *autorotation* to contain energetic descent, or engine malfunctions, and *automated flight termination* to mitigate geo-fence breaches.

In general, we can use barriers to manage multiple hazards, threat events, and consequence events. We encode the safety reasoning underlying the barrier model in the form of a *tiered* argument architecture, where each tier addresses a specific assurance concern. In particular, the highest tier addresses the mitigation of operational hazards; the middle tier refines that concern into the prevention of threat events, the mitigation of consequence events, and the introduction of the relevant existing regulations, whereas the lowest tier addresses the relevant proactive/reactive barriers, common-mode barrier breaches, and eventually specifies particular safety objectives. Of those objectives, the subset that pertains to airworthiness, in particular to type design assurance, comprise the requirements to be included in the certification basis. Figure 4a shows a modular representation of the top-level argument architecture.

Claims Definition and Claims Refinement: Each module (Figure 4a) contains arguments whose claims reflect the concerns being addressed by the relevant tier. Thus, the *operational safety risk management argument module*, M0, (which represents the top-tier) addresses the overall claim that the intended URS operations have acceptable risk using an argument of acceptably managing the risk associated with the constituent hazardous activities. Subsequently, we use an argument of operational hazard mitigation to develop the top-level claim of module M0, into claims concerning *a*) the prevention of all identified threat events, e.g., functional hazards, and *b*) the mitigation of all known consequence events. We further refine each of those claims in the *threat events prevention argument module,* M2, and the *consequence events mitigation argument module,* M3, respectively (Figure 4a). Additionally, the argument structures contained in the *FARs – inclusion of existing regulations argument* module, M1, relates the claims derived from the existing FARs—in particular, 14 CFR Part 27, as well as parts 21, 23, 91, 135, and 137—to the identified operational hazards. Together the modules M1–M3 represent the middle tier of the argument architecture.

Likewise, the modules M4–M6 address the concerns of the lowest tier of the argument architecture, i.e., claims regarding proactive/reactive barriers, and the associated safety objectives. Figures 4b and 4c show, respectively, fragments of the arguments contained within the modules M3 and M5. As shown, the former addresses the mitigation of the identified consequence events, one of which is an energetic uncontrolled descent (goal node G14). We employ a mitigation barrier, in particular (rotorcraft) *impact management*, referring to the barrier model in context (shown as the *away* context node C8, indicating a reference to a context node C8 declared in module M0). We develop the resulting sub-claim (i.e., the *away* goal node G10 of Figure 4b) in the module M5 (shown in Figure 4c, as the *public* goal node G10) into lower-level claims concerning an automated flight termination system and an autorotation capability (goal nodes G16 and G17, respectively), by argument over the allocated hazard controls. Eventually, we specify a safety objective pertaining to autorotation (goal node G18 in Figure 4c), which is reflected in the URS certification basis (referenced through the associated context node C9). In general, the lowest tier of the argument architecture contains similar safety objectives which, upon aggregation, represent the certification basis for URS type design assurance.

Concluding Remarks

We have described a methodology for the principled construction of (safety) assurance arguments that takes, as input, the artifacts produced from domain modeling, safety analysis, and system development/verification to produce, as output, an evolving argument that aims to capture the safety state of the system. Our methodology is compatible with, but distinctly different from, other contemporary methodologies for safety argument development, such as the *six-step method* (refs. 12, 19, 28). In particular, our approach explicitly defines a high-level argument architecture (that may/may not be modular), and leverages automated pattern instantiation (ref. 13) along with pattern composition for argument assembly. Additionally, our approach combines top-down, and bottom-up argument development, focusing on the data flow, rather than control flow, of the activities for argument development. We have applied our methodology to two real example UAS, one of which has dealt with the safety assurance of transit operations (and, in the process, informed the development of our methodology); the other has addressed flight safety and airworthiness, in particular design assurance requirements to be included in a type certification basis.

Acknowledgements

This work has been supported, in part, by the UAS Integration in the NAS project of the Integrated Systems Research program, and the SASO project of the Airspace Operations and Safety Program, of NASA ARMD.

References

1. UK Ministry of Defence. "Safety management requirements for defence systems." *Defence Standard 00-56, Issue 4*, Jun. 2007.

2. European Organisation for the Safety of Air Navigation (EUROCONTROL). Preliminary safety case for airborne traffic situational awareness for enhanced visual separation on approach, v2.1. Aug. 2011.

3. International Civil Aviation Organization (ICAO). "AFI RVSM pre-implementation safety case core document." Feb. 2008.

4. Haddon, D., and C. Whittaker. "Aircraft airworthiness certification standards for civil UAVs." Safety Regulation Group, UK Civil Aviation Authority (CAA), 2002.

5. UK Civil Aviation Authority (CAA). "Small unmanned aircraft: congested areas operating safety case." Information Notice IN-2014/184, Nov. 2014.

6. US Department of Transportation. Federal Aviation Administration (FAA). "Order 8900.1, Flight standards information management system, Volume 16, Unmanned aircraft systems." Jun. 2014.

7. Berthold, R., E. Denney, M. Fladeland, G. Pai, B. Storms, and M. Sumich. "Assuring ground-based detect and avoid for UAS operations", In *Proc. 33rd IEEE/AIAA Digital Avionics Systems Conference*. Oct. 2014, 6A1-1–16.

8. Clothier, R., B. Williams, M. Wade, J. Coyne, and A. Washington. "Challenges to the development of an airworthiness regulatory framework for unmanned aircraft systems." In *Proc. 16th Australian International Aerospace Congress*. Feb. 2015.

9. ICAO. Guidance material on building a safety case for delivery of an ADS-B separation service, v1.0. Sep. 2011.

10. EUROCONTROL. "Safety Case Development Manual." *DAP/SSH/091*, edition 2.2. Nov. 2006.

11. Hawkins, R., I. Habli, T. Kelly, and J. McDermid. "Assurance cases and prescriptive software safety certification: A comparative study." *Safety Science* 59, (2013): 55–71.

12. Goal Structuring Notation Working Group. GSN community standard version 1. Nov. 2011. Accessed Apr. 9, 2015. http://www.goalstructuringnotation.info/.

13. Denney, E., and G. Pai. "A formal basis for safety case patterns." In *Proc. 32nd International Conference on Computer Safety, Reliability and Security*. LNCS 8153, Sep. 2013, 21–32.

14. Denney, E., G. Pai, and I. Whiteside. "Formal foundations for hierarchical safety cases." In *Proc. 16th IEEE International Symposium on High Assurance Systems Engineering*. Jan. 2015, 52–59.

15. Denney, E., and G. Pai. "Towards a formal basis for modular safety cases." In *Proc. 34th International Conference on Computer Safety, Reliability, and Security* (*to appear*). Sep. 2015.

16. Denney, E., C. Ippolito, R. Lee, and G. Pai. "An integrated safety and systems engineering methodology for small unmanned aircraft systems." In *Proc. InfoTech @ Aerospace Conference*, AIAA 2012-2572. Jun. 2012.

17. Denney, E., and G. Pai. "A lightweight methodology for safety case assembly." In *Proc. 31st International Conference on Computer Safety, Reliability and Security*. LNCS 7612, Sep. 2012, 1–12.

18. Denney, E., G. Pai, and J. Pohl. "AdvoCATE: An assurance case automation toolset." In *Proc. 31st International Conference on Computer Safety, Reliability and Security Workshops*. LNCS 7613, Sep. 2012, 8–21.

19. Kelly, T. "A systematic approach to safety case management." *SAE Technical Paper* 2004-01-1779, Mar. 2004.

20. Greenwell, W., J. Knight, C. Holloway, and J. Pease. "A taxonomy of fallacies in system safety arguments." *Paper presented at the 24th International System Safety Conference*, 2006.

21. Hawkins, R., T. Kelly, J. Knight, and P. Graydon. "A new approach to creating clear safety arguments." In *Proc. 19th Safety Critical Systems Symposium*, Feb. 2011, 3 – 23.

22. Goodenough, J., C. Weinstock, and A. Klein. "Eliminative induction: A basis for arguing system confidence." In *Proc. 35th International Conference on Software Engineering*, May 2013, 1161–1164.

23. Denney, E., I. Habli, and G. Pai. "Towards measurement of confidence in safety cases." In *Proc. 5th International Symposium on Empirical Software Engineering and Measurement*. Sep. 2011, 380–383.

24. Ayoub, A., J. Chang, O. Sokolsky, and I. Lee. "Assessing the overall sufficiency of safety arguments." In *Proc. 21st Safety-Critical Systems Symposium*. 2013, 127–144.

25. Felici, M. "Modeling safety case evolution – Examples from the air traffic management domain." In *Proc. 2nd International Workshop on Rapid Integration of Software Engineering Techniques*. LNCS 3943, 2006, 81–96.

26. Kelly, T., and J. McDermid. "A systematic approach to safety case maintenance." *Reliability Engineering and System Safety* 71, no. 3 (2001): 271–284.

27. Denney, E., D. Naylor, and G. Pai. "Querying safety cases." In *Proc. 33rd International Conference on Computer Safety, Reliability, and Security*. LNCS 8666, Sep. 2014, 294–309.

28. Bloomfield, R. and P. Bishop. "Safety and assurance cases: past, present and possible future – an Adelard perspective." In *Proc. 18th Safety-Critical Systems Symposium*. Feb. 2010, 51 – 67.

Biography

Ewen Denney, Ph.D., SGT, Inc., NASA Ames Research Center, Moffett Field, CA 94035, USA, telephone – (650) 604-2274, e-mail – ewen.denney@nasa.gov.

Ewen Denney is a Senior Computer Scientist in the Robust Software Engineering (RSE) technical area of the Intelligent Systems Division (Code TI), at NASA Ames Research Center, where he has worked on automated code generation and safety certification in the aerospace domain. He is a co-recipient of a 2014 NASA Group Achievement Award, and he holds a Ph.D. in Computer Science from the University of Edinburgh.

Dr. Ganesh Pai, Ph.D., SGT, Inc., NASA Ames Research Center, Moffett Field, CA 94035, USA, telephone – (650) 604-0760, e-mail – ganesh.pai@nasa.gov.

Ganesh Pai is a Research Engineer in the Robust Software Engineering (RSE) technical area of the Intelligent Systems Division (Code TI), at NASA Ames Research Center, where he works in the area of safety assurance as applied to unmanned aircraft systems, and aviation systems/software. Dr. Pai is a co-recipient of a 2014 NASA Group Achievement Award, and he holds a Ph.D. in Computer Engineering from the University of Virginia.