A Lightweight Methodology for Safety Case Assembly

Ewen Denney and Ganesh Pai

SGT / NASA Ames Research Center
Moffett Field, CA 94035, USA.
{ewen.denney, ganesh.pai}@nasa.gov

Abstract. We describe a lightweight methodology to support the automatic assembly of safety cases from tabular requirements specifications. The resulting safety case fragments provide an alternative, graphical, view of the requirements. The safety cases can be modified and augmented with additional information. In turn, these modifications can be mapped back to extensions of the tabular requirements, with which they are kept consistent, thus avoiding the need for engineers to maintain an additional artifact. We formulate our approach on top of an idealized process, and illustrate the applicability of the methodology on excerpts of requirements specifications for an experimental Unmanned Aircraft System.

Keywords: Safety cases, Formal methods, Automation, Requirements, Unmanned Aircraft Systems.

1 Introduction

Evidence-based safety arguments, i.e., safety cases, are increasingly being considered in emerging standards [10] and guidelines [3], as an alternative means for showing that critical systems are acceptably safe. The current practice for demonstrating safety, largely, is rather to satisfy a set of objectives prescribed by standards and/or guidelines. Typically, these mandate the processes to be employed for safety assurance, and the artifacts to be produced, e.g., requirements, traceability matrices, etc., as evidence (that the mandated process was followed). However, the rationale connecting the recommended assurance processes, and the artifacts produced, to system safety is largely implicit [7]. Making this rationale explicit has been recognized as a desirable enhancement for "standards-based" assurance [14]; especially also in feedback received [4] during our own, ongoing, safety case development effort.

In effect, there is a need in practice to bridge the gap between the existing means, i.e., standards-based approaches, and the alternative means, i.e., argument-based approaches, for safety assurance. Due to the prevalence of standards-based approaches, conventional systems engineering processes place significant emphasis on producing a variety of artifacts to satisfy process objectives. These artifacts show an appreciable potential for reuse in evidence-based argumentation. Consequently we believe that automatically assembling a safety argument (or parts of it) from the artifacts, to the extent possible, is a potential way forward in bridging this gap.

In this paper, we describe a lightweight methodology to support the automatic assembly of (preliminary) safety cases. Specifically, the main contribution of our paper

is the definition of transformations from tabular requirements specifications to argument structures, which can be assembled into safety case fragments. We accomplish this, in part, by giving process idealizations and a formal, graph theoretic, definition of a safety case. Consequently, we provide a way towards integrating safety cases in existing (requirements) processes, and a basis for automation. We illustrate our approach by applying it to a small excerpt of requirements specifications for a real, experimental Unmanned Aircraft System (UAS).

2 Context

The experimental Swift UAS being developed at NASA Ames comprises a single airborne system, the electric Swift Unmanned Aerial Vehicle (UAV), with duplicated ground control stations and communication links. The development methodology used adopts NASA mandated systems engineering procedures [15], and is further constrained by other relevant standards and guidelines, e.g., for airworthiness and flight safety [13], which define some of the key requirements on UAS operations. To satisfy these requirements, the engineers for the Swift UAS produce artifacts (e.g., requirements specifications, design documents, results for a variety of analyses, tests, etc.) that are reviewed at predefined intervals during development. The overall systems engineering process also includes traditional safety assurance activities as well as *range safety* analysis.

3 Safety Argumentation Approach

Our general approach for safety assurance includes *argument development* and *uncertainty analysis*. Fig. 1 shows a data flow among the different processes/activities during the development and safety assurance of the Swift UAS, integrating our approach for safety argumentation.[1] As shown, the main activities in argument development are *claims definition*, *evidence definition/identification*, *evidence selection*, *evidence linking*, and *argument assembly*. Of these, the first four activities are adapted from the *six-step method* for safety case construction [8].

The main focus of this paper is argument development[2]; in particular, we consider the activity of argument assembly, which is where our approach deviates from existing methodologies [2], [8]. It reflects the notion of "stitching together" the data produced from the remaining activities to create a safety case (in our example, fragments of argument structures for the Swift UAS) containing goals, sub-goals, and evidence linked through an explicit chain of reasoning.

We distinguish this activity to account for (i) *argument design criteria* that are likely to affect the structure of the overall safety case, e.g., maintainability, compliance with safety principles, reducing the cost of re-certification, modularity, and composition of arguments, and (ii) *automation*, e.g., in the assembly of heterogenous data in the overall

[1] Note that the figure only shows some key steps and data relevant for this paper, and is not a comprehensive representation. Additionally, the figure shows neither the iterative and phased nature of the involved activities nor the feedback between the different processes.

[2] Uncertainty analysis [5] is out of the scope of this paper.

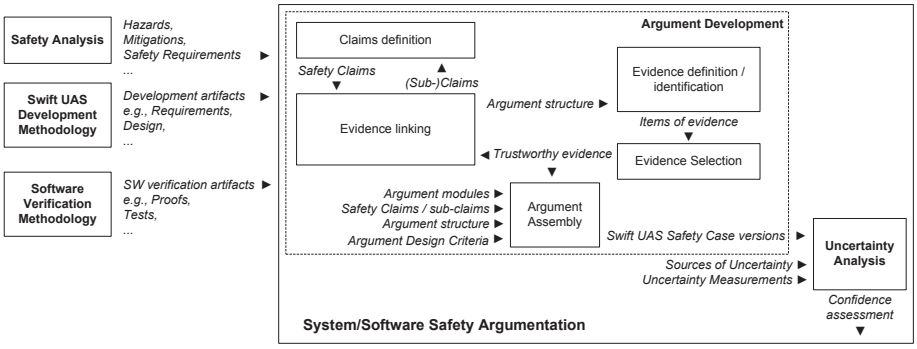

Fig. 1. Safety assurance methodology showing the data flow between the processes for safety analysis, system development, software verification, and safety argumentation.

safety case, including argument fragments and argument modules created using manual, automatic, and semi-automatic means [6].

Safety argumentation, which is phased with system development, is applied starting at the level of the system and then repeated at the software level. Consequently, the safety case produced itself evolves with system development. Thus, similar to [11], we may define a *preliminary*, *interim*, and *operational* safety case reflecting the inclusion of specific artifacts at different points in the system lifecycle. Alternatively, we can also define finer grained versions, e.g., at the different milestones defined in the plan for system certification[3].

4 Towards a Lightweight Methodology

The goal of a *lightweight* version of our methodology (Fig. 1), is to give systems engineers a capability to (i) continue to maintain the existing set of artifacts, as per current practice, (ii) automatically generate (fragments of) a safety case, to the extent possible, rather than creating and maintaining an additional artifact from scratch, and (iii) provide different views on the relations between the requirements and the safety case.

Towards this goal, we characterize the processes involved and their relationship to safety cases. In this paper, we specifically consider a subset of the artifacts, i.e., tables of (safety) requirements and hazards, as an idealization[4] of the safety analysis and development processes. Then, we transform the tables into (fragments of) a *preliminary* safety case for the Swift UAS, documented in the Goal Structuring Notation (GSN) [8]. Subsequently, we can modify the safety case and map the changes back to (extensions of) the artifacts considered, thereby maintaining both in parallel.

[3] Airworthiness certification in the case of the Swift UAS.

[4] We consider idealizations of the processes, i.e., the data produced, rather than a formal process description since we are mainly interested in the relations between the data so as to define and automate the transformations between them.

Hazards Table

ID	Hazard	Cause / Mode	Mitigation	Safety Requirement
HR.1.3	Propulsion system hazards			
HR.1.3.1	Motor overheating	Insufficient airflow	Monitoring	RF.1.1.4.1.2
		Failure during operation		
HR.1.3.7	Incorrect programming of KD motor controller	Improper procedures to check programming before flight	Checklist	RF.1.1.4.1.9

System Requirements Table

ID	Requirement	Source	Allocation	Verification Method	Verification Allocation
RS.1.4.3	Critical systems must be redundant	AFSRB	RF.1.1.1.1.3		
RS.1.4.3.1	The system shall provide independent and redundant channels to the pilot	AFSRB			

Functional Requirements Table

ID	Requirement	Source	Allocation	Verification Method	Verification Allocation
RF.1.1.1.1.3	FCS must be dually redundant	RS.1.4.3	FCS	Visual Inspection	FCS-CDR-20110701, TR20110826
RF.1.1.4.1.2	CPU/autopilot system must be able to monitor engine and motor controller temperature.	HR.1.3.1	Engine systems	Checklist	Pre-flight checklist
RF.1.1.4.1.9	Engine software will be checked during pre-deployment checkout	HR.1.3.7	Pre-deployment checklist	Checklist	Pre-deployment checklist

Fig. 2. Tables of hazards, system and functional requirements for the Swift UAS (excerpts).

4.1 Process Idealizations

We consider three inter-related tables as idealizations of the safety analysis and development processes for the Swift UAS; namely: the hazards table (HT), the system requirements table (SRT), and the functional requirements table (FRT)[5].

Fig. 2 shows excerpts of the three tables produced in the (ongoing) development of the Swift UAS. As shown, the HT contains entries of identified hazards, potential causes, mitigation mechanisms and the corresponding safety requirements. The requirements tables contain specified requirements, their sources, methods with which they may be verified, and verification allocations, i.e., links to artifacts containing the results of verification. Requirements can be allocated either to lower-level (functional) requirements or to elements of the physical architecture.

Fig. 2 mainly shows those parts of the tables that are relevant for defining transformations to an argument structure. Additionally, we are concerned only with a subset of the set of requirements, i.e., those which have a bearing on safety. Since we are looking at snapshots of development, the tables are allowed to be incomplete, as shown in Fig. 2. We further assume that the tables have undergone the necessary quality checks performed on requirements, e.g., for consistency.

Entries in any of the tables can be hierarchically arranged. Identified safety requirements in the HT need not have a corresponding entry in the SRT or FRT. Additionally,

[5] Strictly speaking, this table contains lower-level requirements and not only functional requirements; however, we use the terminology used by the engineers of the Swift UAS.

requirements identified as safety-relevant in either of the requirements tables need not have a hazard, from the hazards table, as a source (although to ensure full traceability, both of these would be necessary). The HT, as shown, are a simplified view of hazard analysis as it occurs at a system level. In practice, hazard analysis would be conducted at different hierarchical levels, i.e., at a subsystem and component level.

For now, we consider no internal structure to the table contents, and simply assume that there are disjoint, base sets of hazards (H), system requirements (R_s), functional requirements (R_f), verification methods (V), and external artifacts (A_r). The set of external artifacts contains items such as constraints from stakeholders, artifacts produced from development, e.g., elements of the physical architecture, concepts of operation, results of tests, etc. We also consider a set of causes (C) and mitigation mechanisms (M). Without loss of generality, we assume that hazards and requirements have unique identifiers. Additionally, we assume the sets V, A_r, C, and M each have a unique "blank" element, shown in the tables as a blank entry.

The HT consists of rows of type

$$\texttt{hazard} \times \texttt{cause}^* \times \texttt{mitigation}^* \times \texttt{safety\_requirement}^* \tag{1}$$

Definition 1. *A* hazards table*, HT, is set of hazard entries ordered by a tree relation $\rightarrow_h$, where a* hazard entry *is a tuple $\langle h, c, m, sr\rangle$, in which $h \in H$, $c \subseteq C$, $m \subseteq M$, and $sr \subseteq (R_s \cup R_f)$.*

The SRT and FRT each have rows of type

$$\texttt{requirement} \times \texttt{source}^* \times \texttt{allocation}^* \times \texttt{verif\_method}^* \times \texttt{verif\_alloc}^* \tag{2}$$

Definition 2. *A* system requirements table*, RT_s, is a set of system requirements entries ordered by a tree relation $\rightarrow_s$, where a* system requirements entry *is a tuple, $\langle r, so, al, vm, va\rangle$, in which $r \in R_s$, $so \subseteq (H \cup A_r)$, $al \subseteq (R_f \cup A_r)$, $vm \subseteq V$, and $va \subseteq A_r$.*

Definition 3. *A* functional requirements table*, RT_f, is a set of functional requirement entries ordered by a tree relation $\rightarrow_f$, where a* functional requirement entry *is a tuple $\langle r, so, al, vm, va\rangle$ in which $r \in R_f$, $so \subseteq (H \cup A_r \cup R_s)$, $al \subseteq A_r$, $vm \subseteq V$, and $va \subseteq A_r$.*

Thus, in an SRT (i) a source is one or more hazard or external artifact, (ii) an allocation is a set of functional requirements or a set of artifacts, and (iii) a verification allocation is a set of artifacts. Whereas in a FRT (i) a source is a hazard, external artifact or system requirement, (ii) an allocation is a set of artifacts, and (iii) a verification allocation links to a specific artifact that describes the result of applying a particular verification method.

Given the base sets and the definitions 1 – 3, we can now define:

Definition 4. *A* requirements specification*, $\mathfrak{R}$, is a tuple $\langle HT, RT_s, RT_f\rangle$.*

We consider a safety case as the result of an idealized safety argumentation process, and document its structure using GSN. We are concerned here with development

snapshots, however, so want to define a notion of partial safety case. Here, we ignore semantic concerns and use a purely structural definition. Assuming finite, disjoint sets of goals (G), strategies (S), evidence (E), assumptions (A), contexts (K) and justifications (J), we give the following graph-theoretic definition:

Definition 5. *A* partial safety case*, $\mathfrak{S}$, is a tuple $\langle G, S, E, A, K, J, \mathtt{sg}, \mathtt{gs}, \mathtt{gc}, \mathtt{sa}, \mathtt{sc}, \mathtt{sj}\rangle$ with the functions*

– $\mathtt{sg} : S \to \mathcal{P}(G)$*, the subgoals to a strategy*
– $\mathtt{gs} : G \to \mathcal{P}(S) \cup \mathcal{P}(E)$*, the strategies of a goal or the evidence to a goal*
– $\mathtt{gc} : G \to \mathcal{P}(K)$*, the contexts of a goal*
– $\mathtt{sa} : S \to \mathcal{P}(A)$*, the assumptions of a strategy*
– $\mathtt{sc} : S \to \mathcal{P}(K)$*, the contexts of a strategy*
– $\mathtt{sj} : S \to \mathcal{P}(J)$*, the justifications of a strategy*

We say that g' is a subgoal of g whenever there exists an $s \in \mathtt{gs}(g)$ such that $g' \in \mathtt{sg}(s)$. Then, define the descendant goal relation, $g \rightsquigarrow g'$ iff g' is a subgoal of g or there is a goal g'' such that $g \rightsquigarrow g''$ and g' is a subgoal of g''. We require that the $\rightsquigarrow$ relation is a directed acyclic graph (DAG) with roots R.[6]

4.2 Mapping Requirements Specifications to Safety Cases

We now show how a requirements specification (as defined above) can be embedded in a safety case (or, alternatively, provide a safety case skeleton). Conversely, a safety case can be mapped to an extension of a requirements specification. It is an extension because there can be additional sub-requirements for intermediate claims, as well as entries/columns accounting for additional context, assumptions and justifications. Moreover, a safety case captures an argument design that need not be recorded in the requirements.

In fact, the mapping embodies the design decisions encapsulated by a specific argument design, e.g., argument over an architectural breakdown, and then over hazards. A given requirements specification can be embedded in a safety case (in many different ways), and we define this as a relation. Based on definitions 1 – 5, intuitively, we map:

– hazard, requirement, causes $\mapsto$ goal, sub-goal
– allocated requirements $\mapsto$ sub-goals
– mitigation, verification method $\mapsto$ strategy
– verification allocation $\mapsto$ evidence
– requirement source, allocated artifact $\mapsto$ goal context

We want to characterize the minimal relation which should exist between a requirements specification and a corresponding partial safety case. There are various ways of doing this. Here, we simply require a correspondence between node types, and that "structure" be preserved.

We define $x \leq x'$ whenever (i) $x \to_s x$, or (ii) $x \to_f x$, or (iii) $x \to_h x$, or (iv) $x = r, x' = al$, $\langle r, so, al, vm, va\rangle \in RT_s$ and $al \in RT_f$, or (v) $x = h, x' = sr$, $\langle h, c, m, sr\rangle \in HT$ and $sr \in (RT_s \cup RT_f)$.

[6] Note that we do not require there to be a unique root. A partial safety case is, therefore, a forest of fragments. A (full) *safety case* can be defined as a partial safety case with a single root, but we will not use that here. Informally, however, we refer to partial safety cases as safety cases.

Definition 6. *We say that a partial safety case,* $\mathfrak{S} = \langle G, S, E, A, K, J, \mathtt{sg}, \mathtt{gs}, \mathtt{gc}, \mathtt{sa}, \mathtt{sc}, \mathtt{sj} \rangle$*,* extends *a requirements specification,* $\mathfrak{R} = \langle HT, RT_s, RT_f \rangle$*, if there is an embedding (i.e., injective function),* ι*, on the base sets of* $\mathfrak{R}$ *in* $\mathfrak{S}$*, such that:*

- $\iota(H \cup C \cup R_s \cup R_f) \subseteq G$
- $\iota(V \cup M) \subset S$
- $\langle r, so, al, vm, va \rangle \in (RT_s \cup RT_f) \Rightarrow \begin{cases} \iota(so) \in \mathtt{gc}(\iota(r)), \\ \iota(vm) \in \mathtt{gs}(\iota(r)), \\ \iota(va) \subseteq \mathtt{sg}(\iota(vm)) \cap E \end{cases}$
- $x \leq x' \Rightarrow \iota(x) \rightsquigarrow \iota(x')$

Whereas goal contexts may be derived from the corresponding requirements sources, strategy contexts, assumptions and justifications are implicit and come from the mapping itself, e.g., as boilerplate GSN elements (See Fig. 3, for an example of a boilerplate *assumption* element). Note that we do not specify the exact relations between the individual elements, just that there is a relation.

4.3 Architecture of the Argument

The structure of the tables, and the mapping defined for each table, induces two patterns of argument structures. In particular, the pattern arising from the transformation of the HT can be considered as an extension of the *hazard-directed breakdown pattern* [12]. Thus, we argue over each hazard in the HT and, in turn, over the identified hazards in a hierarchy of hazards. Consequently, each defined goal is further developed by argument over the strategies implicit in the HT, i.e., over the causes and mitigations.

Similarly, the pattern induced by transforming the SRT and FRT connects the argument elements implicit in the tables, i.e., requirements (goals), and verification methods and verification allocations (strategies), respectively. Additionally, it includes strategies arising due to both the hierarchy of requirements in the tables, and the dependencies between the tables. Specifically, for each requirement, we also argue over its allocation, e.g., the allocation of a functional requirement to a system requirement, and its children, i.e., lower-level requirements. The links between the tables in the requirements specification define how the two patterns are themselves related and, in turn, how the resulting safety case fragments are assembled.

4.4 Transformation Rules

One choice in the transformation is to create goals and strategies that are not marked as *undeveloped* (or *uninstantiated*, or both, as appropriate), i.e., to assume that the completeness and sufficiency of all hazards, their respective mitigations, and all requirements and their respective verification methods, is determined prior to the transformation, e.g., as part of the usual quality checks on requirements specifications. An alternative is to highlight the uncertainty in the completeness and sufficiency of the hazards/requirements tables, and mark all goals and strategies as *undeveloped*. We pick the second option, i.e., in the transformation described next, all goals, strategies, and evidence that are created are undeveloped except where otherwise indicated.

We give the transformation in a relational style, where the individual tables are processed in a top-to-bottom order, and no such order is required among the tables.

Hazards Table: For each entry in the HT (Fig. 2),

(H1) For an entry {Hazard} in the *Hazard* column with no corresponding entries, {Cause} in the *Cause/Mode* column, {Mitigation} in the *Mitigation* column, or {Requirement} in the *Safety Requirement* column, respectively,
   (a) Create a top-level goal "{Hazard} is mitigated", with the hazard identifier as context. Here, we are assuming that this top-level entry is a "container" for a hierarchy of hazards, rather than an incomplete entry.
   (b) The default strategy used to develop this goal is "Argument over identified hazards", with the associated assumption "Hazards have been completely and correctly identified to the extent possible".

(H2) For each lower-level entry, {Hazard}, in the hierarchy,
   (a) Create a sub-goal, "{Hazard} is mitigated", of the parent goal.
   (b) The way we further develop this sub-goal depends on the entries {Cause}, {Mitigation} and {Requirement}; specifically,
      i. For one or more causes, the default strategy is "Argument over identified causes", with "Causes have been completely and correctly identified to the extent possible" as an assumption, and "{Cause} is managed" as the corresponding sub-goal for each identified cause. Then develop each of those sub-goals using "Argument by {Mitigation}" as a strategy.[7]
      ii. For no identified causes, but one or more mitigations specified, create an "Argument by {Mitigation}" strategy, for each mitigation.
      iii. When no cause/mitigation is given, but a safety requirement is specified, then create a strategy "Argument by satisfaction of safety requirement".
      iv. If neither a cause, mitigation nor a safety requirement is given, then assume that the entry starts a new hierarchy of hazards.
   (c) The entry in the *Safety Requirement* column forms the sub-goal "{Safety Requirement} holds", attached to the relevant strategy, with the requirement identifier forming a context element.

System/Functional Requirements Tables: For each entry in either of the SRT/FRT (Fig. 2),

(R1) The contents of the *Requirements* column forms a goal "{System Requirement} holds" if the SRT is processed, or "{Functional requirement} holds" if the FRT is processed. Additionally, if the entry is the start of a hierarchy, create a strategy "Argument over lower-level requirements" connected to this goal. Subsequently, for each lower-level entry in the hierarchy, create a goal "{Lower-level requirement} holds" from the content of the *Requirements* column.

[7] An alternative strategy could be "Argument by satisfaction of safety requirement", assuming that the entry in the *Safety Requirement* column of the hazards table is a safety requirement that was derived from the stated mitigation mechanism.

(R2) (a) the *Source* column forms the context for the created goal/sub-goal. Additionally, if the source is a hazard, i.e., (an ID of) an entry {Hazard} in the HT, then the created goal is the same as the sub-goal that was created from the *Safety Requirement* column of the HT, as in step (H2)(c).

(b) the *Allocation* column is either a strategy or a context element, depending on the content. Thus, if it is

i. an allocated requirement (or its ID), then create and attach a strategy "Argument over allocated requirement"; the sub-goal of this strategy is the allocated requirement[8].

ii. an element of the physical architecture, then create an additional context element for the goal.

(c) the *Verification method* column, if given, creates an additional strategy "Argument by {Verification Method}", an uninstantiated sub-goal connected to this strategy[9], and an item of evidence whose content is the entry in the column *Verification allocation*.

We now state (without proof), that the result of this transformation is a well-formed partial safety case that extends the requirements specification.

5 Illustrative Example

Fig. 3 shows a fragment of the Swift UAS safety case, in the GSN, obtained by applying the transformation rules (Section 4.4) to the HT and FRT (Fig. 2), and assembling the argument structures. Note that a similar safety case fragment (not shown here) is obtained when the transformation is applied to the SRT and FRT.

We observe that (i) the argument chain starting from the top-level goal G0, to the sub-goals G1.3 and G2.1 can be considered as an instantiation of the hazard-directed breakdown pattern, which has then been extended by an argument over the causes and the respective mitigations in the HT (ii) the argument chains starting from these sub-goals to the evidence E1 and E2 reflects the transformation from the FRT, and that, again, it is an instantiation of a specific pattern of argument structures, and (iii) when each table is transformed, individual fragments are obtained which are then joined based on the links between the tables (i.e., requirements common to either table). In general, the transformation can produce several unconnected fragments. Here, we have shown one of the two that are created.

The resulting partial safety case can be modified, e.g., by including additional context, justifications and/or assumptions, to the goals, sub-goals, and strategies. In fact, a set of allowable modifications can be defined, based on both a set of well-formedness rules, and the activities of argument development (Fig. 1). Subsequently, the modifications can be mapped back to (extensions of) the requirements specification.

Fig. 4 shows an example of how the *Claims definition* and *Evidence linking* activities (Fig. 1) modify the argument fragment in Fig. 3. Specifically, goal G2 has been

[8] This will also be an entry in the *Requirements* column of the FT.

[9] A constraint, as per [8], is that each item of evidence is preceded by a goal, to be well-formed.

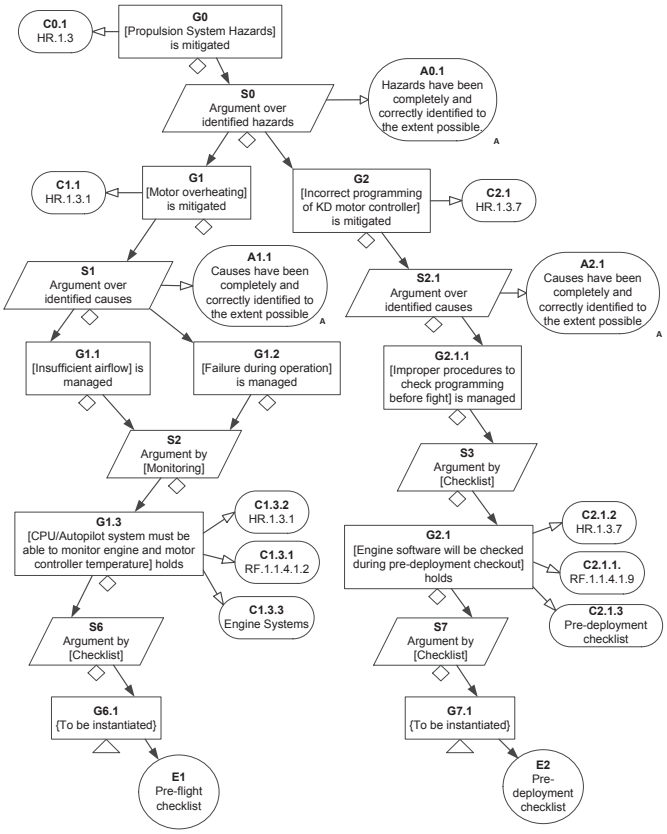

Fig. 3. Fragment of the Swift UAS safety case (in GSN) obtained by transformation of the hazards table and the functional requirements table.

further developed using two additional strategies, StrStatCheck and StrRunVerf, resulting in the addition of the sub-goals GStatCheck and GRunVerf respectively. Fig. 5 shows the corresponding updates (as highlighted rows and *italicized* text) in the HT and SRT respectively, when the changes are mapped back to the requirements specification. Particularly, the strategies form entries in the *Mitigation* column of the HT, whereas the sub-goals form entries in the *Safety Requirement* and *Requirement* columns of the HT and the SRT respectively. Some updates will require a modification (extension) of the tables, e.g., addition of a *Rationale* column reflecting the addition of justifications to strategies. Due to space constraints, we do not elaborate further on the mapping from safety cases to requirements specifications.

6 Conclusion

There are several points of variability for the transformations described in this paper, e.g., variations in the forms of tabular specifications, and in the mapping between these

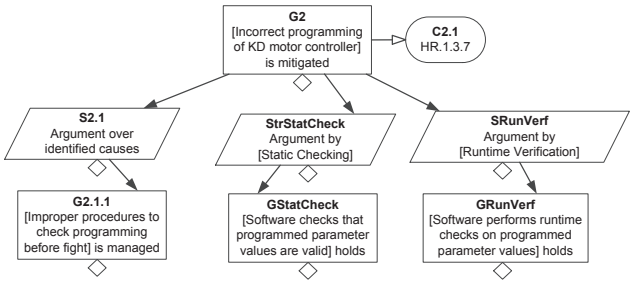

Fig. 4. Addition of strategies and goals to the safety case fragment for the Swift UAS.

Hazards Table

ID	Hazard	Cause / Mode	Mitigation	Safety Requirement
HR.1.3	Propulsion system hazards			
HR.1.3.1	Motor overheating	Insufficient airflow	Monitoring	RF.1.1.4.1.2
		Failure during operation		
HR.1.3.7	Incorrect programming of KD motor controller	Improper procedures to check programming before flight	Checklist	RF.1.1.4.1.9
		-	*Static checking*	*GStatCheck*
		-	*Runtime Verification*	*GRunVerf*

System Requirements Table

ID	Requirement	Source	Allocation	Verification Method	Verification Allocation
RS.1.4.3	Critical systems must be redundant	AFSRB	RF.1.1.1.1.3		
RS.1.4.3.1	The system shall provide independent and redundant channels to the pilot	AFSRB			
GStatCheck	*Software checks that programmed parameter values are valid*	*HR.1.3.7*			
GRunVerf	*Software performs runtime checks on programmed parameter values*	*HR.1.3.7*			

Fig. 5. Updating the requirements specification tables to reflect the modifications shown in Fig. 4.

forms to safety case fragments. We emphasize that the transformation described in this paper is *one* out of many possible choices to map artifacts such as hazard reports [9] and requirements specifications to safety cases. Our main purpose is to place the approach on a rigorous foundation and to show the feasibility of automation.

We are currently implementing the transformations described in a prototype tool[10]; although the transformation is currently fixed and encapsulates specific decisions about the form of the argument, we plan on making this customizable. We will also implement abstraction mechanisms to provide control over the level of detail displayed (e.g., perhaps allowing some fragments derived from the HT to be collapsed).

We will extend the transformations beyond the simplified tabular forms studied here, and hypothesize that such an approach can be extended, in principle, to the rest of the data flow in our general methodology so as to enable automated assembly/generation of safety cases from heterogeneous data. In particular, we will build on our earlier work on generating safety case fragments from formal derivations [1]. We also intend to clarify how data from concept/requirements analysis, functional/architectural design,

[10] **A**dvo**CATE**: **A**ssurance **C**ase **A**utomation **T**ools**e**t.

preliminary/detailed design, the different stages of safety analysis, implementation, and evidence from verification and operations can be transformed, to the extent possible, into argument structures conducive for assembly into a comprehensive safety case.

We have shown that a lightweight transformation and assembly of a (preliminary) safety case from existing artifacts, such as tabular requirements specifications, is feasible in a way that can be automated. Given the context of existing, relatively mature engineering processes that appear to be effective for a variety of reasons [14], our view is that such a capability will ameliorate the adoption of, and transition to, evidence-based safety arguments in practice.

Acknowledgements. We thank Corey Ippolito for access to the Swift UAS data. This work has been funded by the AFCS element of the SSAT project in the Aviation Safety Program of the NASA Aeronautics Mission Directorate.

References

[1] Basir, N., Denney, E., Fischer, B.: Deriving safety cases for hierarchical structure in model-based development. In: 29th Intl. Conf. Comp. Safety, Reliability and Security. (2010)
[2] Bishop, P., Bloomfield, R.: A methodology for safety case development. In: Proc. 6th Safety-critical Sys. Symp. (Feb 1998)
[3] Davis, K.D.: Unmanned Aircraft Systems Operations in the U.S. National Airspace System. FAA Interim Operational Approval Guidance 08-01. (Mar 2008)
[4] Denney, E., Habli, I., Pai, G.: Perspectives on Software Safety Case Development for Unmanned Aircraft. In: Proc. 42nd Annual IEEE/IFIP Intl. Conf. on Dependable Sys. and Networks. (Jun 2012)
[5] Denney, E., Pai, G., Habli, I.: Towards measurement of confidence in safety cases. In: Proc. 5th Intl. Symp. on Empirical Soft. Eng. and Measurement. pp. 380–383 (Sep 2011)
[6] Denney, E., Pai, G., Pohl, J.: Heterogeneous aviation safety cases: Integrating the formal and the non-formal. In: Proc. 17th IEEE Intl. Conf. Engineering of Complex Computer Systems. (Jul 2012)
[7] Dodd, I., Habli, I.: Safety certification of airborne software: An empirical study. Reliability Eng. and Sys. Safety. 98(1), pp. 7–23 (2012)
[8] Goal Structuring Notation Working Group: GSN Community Standard Version 1 (Nov 2011). http://www.goalstructuringnotation.info/
[9] Goodenough, J.B., Barry, M.R.: Evaluating Hazard Mitigations with Dependability Cases. White Paper (Apr 2009). http://www.sei.cmu.edu/library/abstracts/whitepapers/dependabilitycase_hazardmitigation.cfm/
[10] International Organization for Standardization (ISO): Road Vehicles-Functional Safety. ISO Standard 26262 (2011)
[11] Kelly, T.: A systematic approach to safety case management. In: Proc. Society of Automotive Engineers (SAE) World Congress (Mar 2004)
[12] Kelly, T., McDermid, J.: Safety case patterns – reusing successful arguments. In: Proc. IEE Colloq. on Understanding Patterns and Their Application to Sys. Eng. (1998)
[13] NASA Aircraft Management Division: NPR 7900.3C, Aircraft Operations Management Manual. NASA (Jul 2011)
[14] Rushby, J.: New challenges in certification for aircraft software. In: Proc. 11th Intl. Conf. on Embedded Soft. pp. 211–218 (Oct 2011)
[15] Scolese, C.J.: NASA Systems Engineering Processes and Requirements. NASA Procedural Requirements NPR 7123.1A. (Mar 2007)