Towards a Formal Basis for Modular Safety Cases

Ewen Denney and Ganesh Pai

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

Abstract. Safety assurance using argument-based safety cases is an accepted best-practice in many safety-critical sectors. Goal Structuring Notation (GSN), which is widely used for presenting safety arguments graphically, provides a notion of modular arguments to support the goal of incremental certification. Despite the efforts at standardization, GSN remains an informal notation whereas the GSN standard contains appreciable ambiguity especially concerning modular extensions. This, in turn, presents challenges when developing tools and methods to intelligently manipulate modular GSN arguments. This paper develops the elements of a theory of modular safety cases, leveraging our previous work on formalizing GSN arguments. Using example argument structures we highlight some ambiguities arising through the existing guidance, present the intuition underlying the theory, clarify syntax, and address *modular arguments*, *contracts*, *well-formedness* and *well-scopedness* of modules. Based on this theory, we have a preliminary implementation of modular arguments in our toolset, AdvoCATE.

Keywords: Safety Cases, Argument Structures, Modularity

1 Introduction

Modular safety arguments are desirable for a number of reasons: first, they can be useful to *manage safety case size*, and also to *improve comprehensibility*, by providing an abstract, architectural view of the argument that clarifies the relevant relationships between various argument components. Second, modularity is useful to *minimize and contain the impact of required changes* to a safety case, and, consequently, maintain the assurance provided [1]. Thirdly, they can support *distributed and concurrent development* of the different argument modules [2]. The vision is that a modular organization will facilitate module replacement during an *incremental certification* process [3], so that an argument module can be exchanged for another that meets the same safety properties whilst also protecting intellectual property, e.g., by exposing only the *public details* of arguments as appropriate. To support this vision, the Goal Structuring Notation (GSN) [4], a widely-used graphical notation for presenting safety arguments, provides a notion of modular arguments. Still, there has been limited experience with using modular safety cases (to our knowledge). We believe that this is due to, in part, insufficient tools and techniques supporting both a practical and correct use of the modular safety case concept.

Indeed, in a cost-benefit study of modular safety case development [5], the lack of adequate tool support was one of the concerns identified amongst the risks of using modular arguments. Additionally, although there have been various efforts at both formalizing the GSN, e.g., [6], [7], and standardization [4], [8], the GSN Standard [4] leaves many questions open (see Section 3 for details). This, in turn, presents challenges for tool development (e.g., to build well-formed modular arguments), with existing tools that implement modular GSN varying in how they handle modules. This paper is a first attempt at closing that gap. Our goal is to provide a formal basis for an implementation that will provide, to the extent possible, automation in creating and manipulating modular arguments. Another aim of this paper is to clarify the GSN Standard, i.e., to make explicit numerous constraints that were hitherto implicit rather than a critique, or radical replacement, of the existing notation. Specifically, our paper makes the following contributions: *i*) a formal definition of *modular arguments* and *contracts*, clarifying their permissible interconnections; *ii*) extending modules with a notion of hierarchy, giving a rigorous definition for *containment* and *scope*, and using these to formulate a notion of *well-formedness*; and *iii*) foundations for implementing modular organization in tools supporting GSN, e.g., our toolset, AdvoCATE [9], or others such as ASCE[1].

2 Modular Extensions to Goal Structuring Notation

We use intra-module GSN (Fig. 1) in an argument (within a specific module) to refer to other modules, and/or invoke specific elements in other modules (using, respectively, so-called *module reference*, and *away* nodes). An away goal (context, or solution) in one module essentially repeats a *public* goal (context, or evidence item) present in another module, indicated by a '🗀' annotation placed at the top right of the node (e.g., Fig. 1, goal node G1). Thus, other modules can access (i.e., reference) a public node of a given type using the corresponding type of away node. Each away node additionally contains a reference to the module containing the public node (e.g., Fig. 1, away goal node AG1). As shown, we can link to away goals using both the *Is Supported By* (→) and *In Context Of* (⇾) relations. The former implies claim refinement (i.e., by a public claim in the referenced module). The latter is a substitution for a justification node, but where *more* justification is required than can be provided by a justification node alone, and where the additional justification is provided in a different module.

GSN also provides a concept of *contract module*, containing a definition and/or justification of the relationship between two or more modules, in particular how a claim in one (or more) module(s) support(s) the argument in the other(s) [4]. As originally conceived, (argument) contracts were meant to represent relations between goals, context and evidence of modules participating in a composition, and were documented using *contract tables*. Currently, the GSN standard provides little guidance to specify the contents of a contract. However, to integrate and view a contract within a common framework, the use of GSN itself has been proposed to specify contracts within contract modules [10], [11], which is the convention we will follow here. Thus, we use intra-module GSN to specify a contract, although away goals have a slightly different interpretation: as the link to the modules accessing the contract (see Fig. 3 for an example), in addition to a contextual use, i.e., to provide justification in the contract as appropriate.

[1] Assurance and Safety Case Environment, available at: http://www.adelard.com/asce/

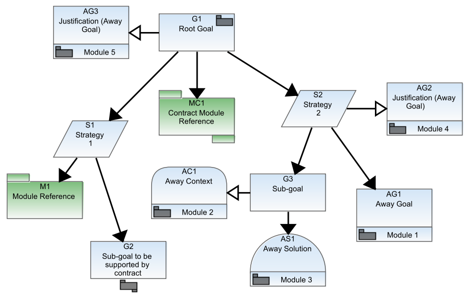


Fig. 1. Intra-module GSN

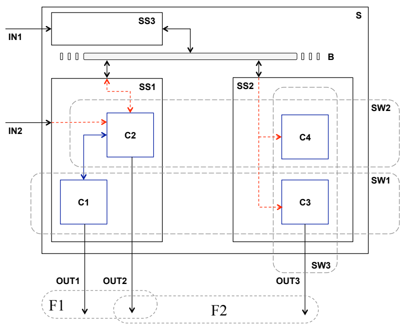


Fig. 2. Motivating example system.

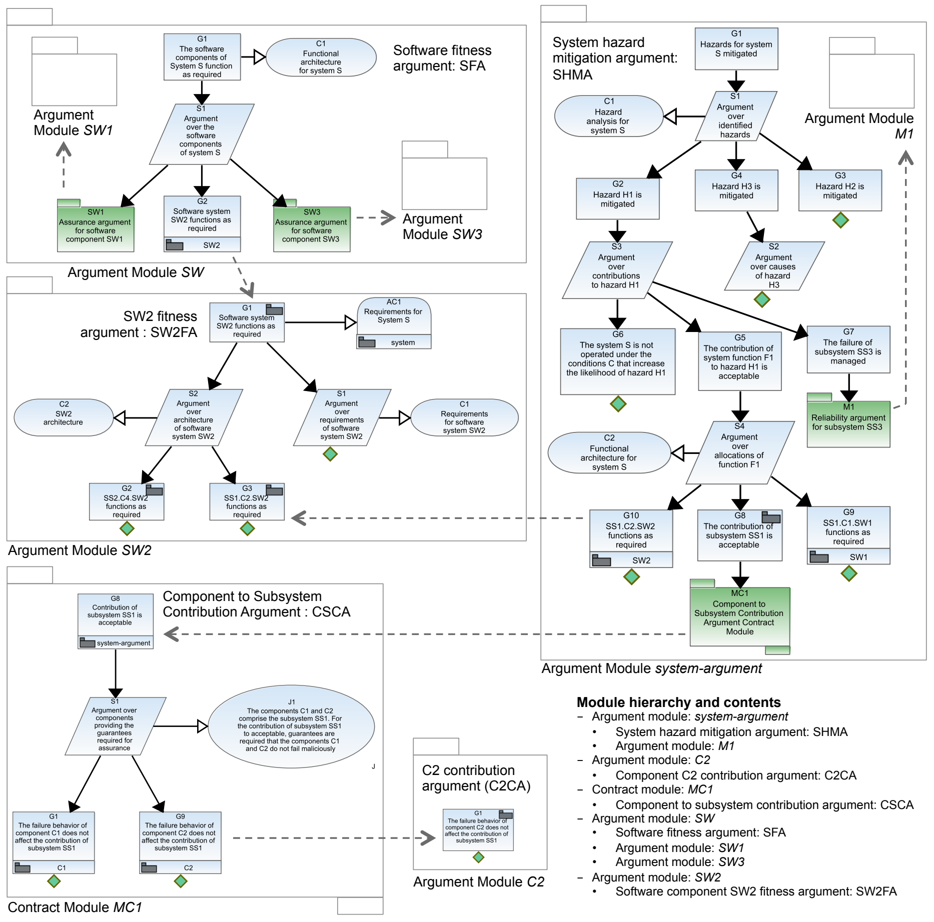
When an argument is supported in an, as yet, unspecified module but the contract of support is available in a contract module, we show the reference to that contract using a *contract module reference* node (e.g., Fig. 1, node MC1). When the contract itself is unspecified, we use the '*to be supported by contract*' (⊟) annotation (e.g., Fig. 1, goal node G2). This annotation is analogous to, and mutually exclusive with, the '*to be developed*' (*tbd*) annotation (◇) in non-modular GSN.

Effectively, we use inter-module GSN to specify a *module view* that shows modules and their relationships. We can link modules to *a*) other modules, using the ➝ and/or ⇾ links, and *b*) contract modules, using the ➝ link. The contract module explicates the support relationship between the modules to which it is linked. In this paper, we primarily focus on intra-module GSN, deferring inter-module GSN to future work.

3 Motivating Example

We present an example to illustrate some of the ambiguities that arise when using the currently available guidance on modular structuring of GSN arguments [4]. Consider a system S comprising subsystems SS1, SS2, and SS3 communicating over a bus B, and providing the functions F1 and F2 by processing inputs IN1 and IN2 (Fig. 2). Subsystems SS1 and SS2 contain the components (C1, C2), and (C3, C4) respectively. The software operating on system S uses software components SW1, SW2, and SW3, deployed such they require the subsystem components (C1, C3), (C2, C4), and (C3, C4) respectively. The hazard analysis for system S indicates that there are three hazards H1, H2, and H3, to be managed for assuring system safety. In particular, it is determined that the function F1 and subsystem SS3 contribute to hazard H1. The system S can represent, for example, an idealized and abstract *integrated modular avionics* (IMA) system, where different software applications of varying safety-criticality operate on a number of COTS computing modules connected by a real-time computer network.

Fig. 3 shows a possible modular organization of argument structure fragments addressing different assurance concerns for system S, together with the module hierarchy, and module contents. As shown, the module *system-argument* contains (a fragment of) a *hazard mitigation argument* (labeled SHMA) asserting the mitigation of the identified hazards. Here, we develop the claim of mitigating hazard H1 (goal node G2) into a


Software fitness argument: SFA
G1 The software components of System S function as required
C1 Functional architecture for system S
S1 Argument over the software components of system S
Argument Module *SW1*
SW1 Assurance argument for software component SW1
G2 Software system SW2 functions as required
SW2
SW3 Assurance argument for software component SW3
Argument Module *SW3*
Argument Module *SW*
SW2 fitness argument : SW2FA
G1 Software system SW2 functions as required
AC1 Requirements for System S
system
C2 SW2 architecture
S2 Argument over architecture of software system SW2
S1 Argument over requirements of software system SW2
C1 Requirements for software system SW2
G2 SS2.C4.SW2 functions as required
G3 SS1.C2.SW2 functions as required
Argument Module *SW2*
System hazard mitigation argument: SHMA
G1 Hazards for system S mitigated
C1 Hazard analysis for system S
S1 Argument over identified hazards
Argument Module *M1*
G2 Hazard H1 is mitigated
G4 Hazard H3 is mitigated
G3 Hazard H2 is mitigated
S3 Argument over contributions to hazard H1
S2 Argument over causes of hazard H3
G6 The system S is not operated under the conditions C that increase the likelihood of hazard H1
G5 The contribution of system function F1 to hazard H1 is acceptable
G7 The failure of subsystem SS3 is managed
M1 Reliability argument for subsystem SS3
C2 Functional architecture for system S
S4 Argument over allocations of function F1
G10 SS1.C2.SW2 functions as required
SW2
G8 The contribution of subsystem SS1 is acceptable
G9 SS1.C1.SW1 functions as required
SW1
MC1 Component to Subsystem Contribution Argument Contract Module
Argument Module *system-argument*
G8 Contribution of subsystem SS1 is acceptable
system-argument
Component to Subsystem Contribution Argument : CSCA
S1 Argument over components providing the guarantees required for assurance
J1 The components C1 and C2 comprise the subsystem SS1. For the contribution of subsystem SS1 to acceptable, guarantees are required that the components C1 and C2 do not fail maliciously
J
G1 The failure behavior of component C1 does not affect the contribution of subsystem SS1
C1
G9 The failure behavior of component C2 does not affect the contribution of subsystem SS1
C2
Contract Module *MC1*
C2 contribution argument (C2CA)
G1 The failure behavior of component C2 does not affect the contribution of subsystem SS1
Argument Module *C2*

Module hierarchy and contents
- Argument module: *system-argument*
  - System hazard mitigation argument: SHMA
  - Argument module: *M1*
- Argument module: *C2*
  - Component C2 contribution argument: C2CA
- Contract module: *MC1*
  - Component to subsystem contribution argument: CSCA
- Argument module: *SW*
  - Software fitness argument: SFA
  - Argument module: *SW1*
  - Argument module: *SW3*
- Argument module: *SW2*
  - Software component SW2 fitness argument: SW2FA

Fig. 3. Possible modularization of (fragments of) argument structures addressing different assurance concerns for the system S of Fig. 2. Note that module boundaries, cross-module linking, and containment, are only illustrative and not part of GSN.

claim that the contribution of the system function F1 to hazard H1 is acceptable (goal node G5). To support that claim, we assert that *i*) the software that provides function F1, i.e., those portions of the software components SW1 and SW2 operating on SS1, is acceptable and operates as required, and *ii*) the contribution of subsystem SS1 (to the function F1) is acceptable. In Fig. 3, we have shown those sub-claims as away goals (nodes G9 and G10), and as a public goal (node G8) that refer, respectively, to the supporting argument modules *SW1*, *SW2* and the contract module *MC1*. The latter, in particular, is the interface to the argument modules providing assurance that the failure behavior of the components of SS1 is acceptable, e.g., module *C2*. The argument module *SW* contains a *software fitness argument* (labeled SFA), which asserts that the software components of the system behave as required.

This argument is itself modular, with references to other modules that contain arguments assuring the fitness of the individual software components: namely, *SW1* and *SW3* (shown as *sub-modules* contained within *SW*), and *SW2* (shown as a *sibling* of *SW*). Assuring software fitness may be required independently of assuring hazard mitigation, although, as shown in Fig. 3, the latter depends on elements of the former. In particular, in argument SHMA, the away goal node G10 in the module *system-argument* invokes the public goal (i.e., node G3) in module *SW2*.

Whilst creating such modular arguments, we found the GSN standard and the literature on practical usage of modular arguments, e.g., [10], [11] to be unclear on the characteristics of well-formed modular arguments, and a number of specific questions arose: (1) Is it permissible to include the module *SW2* within the module *SW*, similar to modules *SW1* and *SW3*? In general, what is the *scope* of public argument elements? (2) Can elements of the module *SW*, e.g., SFA, refer to elements of the module *system-argument*, e.g., SHMA, thus resulting in cyclic links between the two modules, and potentially in the overall argument? In general, what constraints apply across module boundaries? (3) If properties of the components of subsystem SS1 are known, can the hazard mitigation argument SHMA in the module *system-argument* additionally develop the claim in goal node G8? In general, can a claim be supported by multiple contracts, or can a claim be supported by both a contract and a local argument? (4) What are the valid elements of a contract argument represented using GSN, e.g., as proposed in [10]? In general, is a contract argument subject to the same constraints as any other (modular/non-modular and non-contract) argument structure? (5) What are the valid properties of the nodes specified with modular GSN, e.g., can away goals be undeveloped (i.e., annotated with $\diamond$), as shown in Fig. 3? (6) Supposing module *SW* contained module *SW2*, but the latter offered no public goals (like module *SW1*), is it meaningful to construct and invoke a single contract (referencing the module *SW*), from multiple goal nodes of the hazard mitigation argument (containing different claims about the software)? In general, is it permissible for multiple claims in one (or more) module(s) to invoke a common contract? Towards addressing these issues, next (Section 4) we give rigorous definitions for modular GSN extensions, in particular intra-module GSN.

4 Formalization

First, we recall the (non-modular) definition of argument structure [6], [7] (Definition 1), which we will extend to formalize the notion of a single modular argument structure, i.e., an individual diagram, to account for intra-module GSN (Definition 2). Then, we define *contracts* (Definition 3). Subsequently, we will extend the formalization to interconnected collections of modules, clarifying containment (Definition 4), and scope (Definition 5). Thereafter, we will define the characteristics of a well-formed module hierarchy (Definition 6). Note that Definitions 2, 5 and 6 work together: Definition 2 gives structure on individual modules; Definition 5 gives the permissible and required connections between arguments, and Definition 6 gives additional constraints that must hold between linked arguments.

Definition 1 (Argument Structure). *An* argument structure *S is a tuple $\langle N, l, \rightarrow\rangle$ comprising: a set of nodes N; a family of labeling functions l_X, where $X \in \{t, d, m, s\}$,* *giving the node fields* type*,* description*,* metadata *(i.e., attributes), and* status*; and* $\rightarrow$ *is the connector relation between nodes. Let* $\{g, s, e, a, j, c\}$ *be the node types* goal, strategy, evidence, assumption, justification*, and* context *respectively. Then,* $l_t : N \rightarrow \{g, s, e, a, j, c\}$ *gives node types,* $l_d : N \rightarrow$ *string gives node descriptions,* $l_m : N \rightarrow \mathcal{P}(A)$ *(where* A *is an attribute set) gives node instance attributes, and* $l_s : N \rightarrow \mathcal{P}(\{tbd\})$ *gives node development status. We define the transitive closure,* $\rightarrow^*: \langle N, N\rangle$*, in the usual way, and require the connector relation to form a* finite forest *with the operation* $\texttt{root}(\rightarrow, r)$ *checking if the node* r *is a root of the forest. Furthermore, the following structural conditions must be met:*

1. $\texttt{root}(\rightarrow, r) \Rightarrow l_t(r) = g$*, i.e., each root of the argument structure is a goal;*
2. $n \rightarrow m \Rightarrow l_t(n) \in \{s, g\}$*, i.e., connectors only leave strategies or goals;*
3. $(n \rightarrow m) \wedge [l_t(n) = g] \Rightarrow l_t(m) \in \{s, e, a, j, c\}$*, i.e., goals do not connect to other goals;*
4. $(n \rightarrow m) \wedge [l_t(n) = s] \Rightarrow l_t(m) \in \{g, a, j, c\}$*, i.e., strategies do not connect to other strategies or evidence;*
5. $tbd \in l_s(n) \Rightarrow l_t(n) \in \{g, s\}$*, i.e., only goals and strategies can be undeveloped.*

Definition 1 gives a *strict* notion of argument—i.e., a tree, rather than a directed acyclic graph (DAG)—where separate goals cannot share evidence, and goals require intermediate strategies. Both these conditions, which are often violated in practice, can be captured by a more relaxed definition (not given here). Now, let mr, and cr, be two additional node types, denoting *module reference* and *contract module reference* respectively, in addition to the node types given in Definition 1.

Definition 2 (Modular Argument Structure). *A* modular argument structure*, (or module, for short),* M*, is a tuple* $\langle N, l, t, d, \rightarrow\rangle$*, where* N *and* $\rightarrow$ *are as in Definition 1;* d *is a module description string. Again, as in Definition 1,* l *is the same family of functions where* $l_t : N \rightarrow \{s, g, e, a, j, c, mr, cr\}$ *gives node types;* $l_d : N \rightarrow$ *string gives node descriptions; and* $l_m : N \rightarrow \mathcal{P}(A)$ *gives node instance attributes.* $l_s : N \rightarrow \mathcal{P}(\{tbd, tbsbc, public, away, contextual\})$ *gives node status, i.e., whether a node is, respectively,* 'to be developed'*,* 'to be supported by contract'*, declared* public*, referencing an* away node*, or* 'used in context'*.* t *is a family of functions that gives the target of the nodes referencing other modules: for module reference,* x*,* $t_r(x)$ *gives the target module, and for an away node,* $t_a(x)$ *gives the pair of module and public node. Let* $\mathcal{I}_m$ *and* $\mathcal{I}_n$ *be sets of identifiers (IDs) distinct from* N*, representing modules (and contracts; see Definition 3) and nodes external to* M*. Then, we have the maps* $t_a : \{n \in N \mid away \in l_s(n)\} \rightarrow \mathcal{I}_m \times \mathcal{I}_n$*, and* $t_r : \{n \in N \mid l_t(n) \in \{mr, cr\}\} \rightarrow \mathcal{I}_m$*. We require individual modular argument structures to form forests. Additionally, the following structural conditions*[2] *must be met:*

1. *The conditions in items 1, 2, 4, and 5 of Definition 1 hold;*
2. *Only goal, evidence, and context nodes can be marked as public or away nodes:* $public, away \in l_s(n) \Rightarrow l_t(n) \in \{g, e, c\}$*;*
3. *Only goals are marked as* to be supported by contract*:* $tbsbc \in l_s(n) \Rightarrow l_t(n) = g$*;*
4. *Nodes cannot be both public and away:* $\nexists n \in N \,.\, \{public, away\} \subseteq l_s(n)$*;*

[2] To save space, we consider free variables to be implicitly universally quantified.

5. *Nodes with status* tbd *and* tbsbc *are mutually exclusive:* $\nexists n \in N \,.\, \{\mathit{tbsbc}, \mathit{tbd}\} \subseteq l_s(n)$*;*
6. *Goals with status* away *or* tbsbc*, and (contract) module references have no outgoing links:* $\mathit{away} \in l_s(n)$ *or* $\mathit{tbsbc} \in l_s(n)$ *or* $l_t(n) \in \{mr, cr\} \Rightarrow \nexists m \in N \,.\, n \rightarrow m$*;*
7. $\mathit{contextual} \in l_s(n) \Rightarrow ([\mathit{away} \in l_s(n) \wedge l_t(n) = g] \vee l_t(n) = mr) \wedge \exists m \in N . m \rightarrow n$*, i.e., contextual nodes are away goals or module references, and are link targets;*
8. $n \rightarrow m \wedge l_t(n) = l_t(m) = g \Rightarrow \mathit{contextual} \in l_s(m)$*, i.e., goal-to-goal links are contextual;*
9. *Goals supported by contract module references must be public, with no other out-links:* $n \rightarrow m \wedge l_t(n) = g \wedge l_t(m) = cr \Rightarrow (\mathit{public} \in l_s(n) \wedge n \rightarrow m' \Rightarrow m = m')$.

The standard permits away goals and module references to be linked to by both ➝ and ⇾ relations, whereas contract module references can only be linked to using the ➝ relation (e.g., see Fig. 3). Since our definition contains a single connector relation $\rightarrow$ where we derive the link type from the types of source/target nodes, we introduce an additional status '*contextual*' to represent the situation when goals or module references are used contextually. As in Definition 1, we require individual modular argument structures to form forests. However, whereas a non-modular argument is expected to be a tree eventually (with a single top-level claim), a completed modular argument can naturally consist of several trees (see Fig. 3). Also, note that all nodes have unique IDs, including away goals and the public goals that they reference. The forest condition rules out cycles in the non-modular equivalent of modular arguments (see Fig. 5) and multiple parents (i.e., DAGs). This, and the condition of item 8, reflect the strict notion of argument in Definition 1. Later (Definition 5), we will give a condition to constrain inter-module cycles. The standard implies that module references cannot have status tbd, since an undeveloped goal and a goal supported by a module reference are alternative ways of stating that "support (for the claim) is (yet) to be provided". However, from an implementation standpoint, it seems reasonable to have a user preference that a tbd status on module references be derived from the corresponding module body; we therefore do not prevent tbd status on nodes of type mr.

As mentioned earlier, we consider contracts to be represented also using GSN. The simplest form of a contract specified using GSN contains *i*) an away goal referencing a public goal in the *consumer* (or source) module—i.e., the module that invokes the contract—which is *ii*) developed using an appropriate strategy into *iii*) one or more module references to, or away goals that reference public goals in, the *provider* (or target) module(s)—i.e., the module(s) that are the target of the contract (e.g., see contract module *MC1*, in Fig. 3, for an example). In a contract, we term the away goals referring to consumer modules as the *in*–nodes, whereas the *out*–nodes of the contract are the away goals, contexts, solutions and module references that refer to provider modules.

Formally, if $l_t(n) = g, \mathit{away} \in l_s(n), \neg\texttt{leaf}(\rightarrow, n)$, and $\forall n' \,.\, [\texttt{leaf}(\rightarrow, n') \Rightarrow n \rightarrow^* n']$, then the node n is an *in*–node, $\mathit{in}(n)$. Similarly, it is an *out*–node, $\mathit{out}(n)$, if $\texttt{leaf}(\rightarrow, n)$ and $(l_t(n) = mr \vee \mathit{away} \in l_s(n))$. We give the target module of a node x as $\mathit{tmod}(x) = M$, if $l_t(x) \in \{g, c, e\}, \mathit{away} \in l_s(x)$, and $t_a(x) = \langle M, \_ \rangle$, or $l_t(x) = mr$ and $t_r(x) = M$. Here, note that the notions of *in*– and *out*–node are defined on a single module diagram, and do not depend on the connections to other modules (which we characterize later in Definition 5, when we define module scope).

Next, we formalize contracts.

Definition 3 (Contractual Argument Structure). *A* contractual argument structure *(*contract*, for short) C, is a tuple $\langle N, l, d, t, \rightarrow\rangle$ that satisfies the same conditions as Definition 2, with the exception of condition (6) for away goals. That is, away goals are allowed to have outgoing links. The following additional conditions hold:*

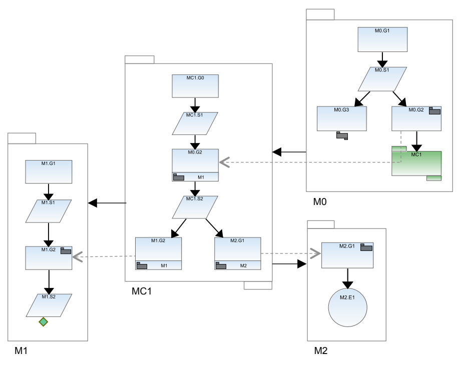
1. *There exists at least one* in*–node (i.e., a non-leaf away node above all leaf nodes): $\exists n \in N \,.\, in(n)$;*
2. $\texttt{leaf}(n) \Rightarrow l_t(n) \notin \{s, cr\} \wedge [l_t(n) = g \Rightarrow away \in l_s(n)]$*, i.e., each leaf node is either an away goal, a module reference, an evidence node, or a contextual element (context, assumption, or justification node). Hence,* out*–nodes will either be away nodes (goals, contexts, evidence), or module references;*
3. *There exists at least one* out*–node: $\exists n \in N \,.\, out(n)$;*
4. *Nodes cannot be public: $public \notin l_s(n)$;*
5. *All away nodes are either* in*– or* out*–nodes: $away \in l_s(n) \Rightarrow in(n) \vee out(n)$.*

From Definition 3, we observe the following: first, since contracts do not satisfy item 6 of Definition 2, formally they are not modules. Informally, though, we can think of them as a special kind of module, and will write '*(non-)contract module*' when the difference needs emphasis. Next, item 2 permits *local*, i.e., non-away, evidence[3], and not all branches need have *out*–node leaves; hence we also require the condition of item 3. It is a matter of interpretation that non-leaf away nodes can be only *in*–nodes. This, combined with item 2 implies that *in*–nodes must be above *out*–nodes and, in particular, that a node cannot be both an *in*–node and an *out*–node. Additionally, since the intended role of a contract is to provide an interface between the assertions of provider modules and the guarantee(s) required by consumer modules, it seems reasonable to constrain the way in which the contract is accessed. Thus, by prohibiting public nodes, i.e., item 4, we preclude access to those elements of the contract, that are not *in*–nodes. For the same reason, *in*–nodes are necessarily above all *out*–nodes. So also, the latter must be necessarily leaves since the premises of a contract are developed externally by provider modules, rather than internally to a contract. However, note that item 5 does not restrict an *out*–node from being used contextually. Fig. 4 illustrates some of the variety in contractual argument structures, arising from the conditions of Definition 3.

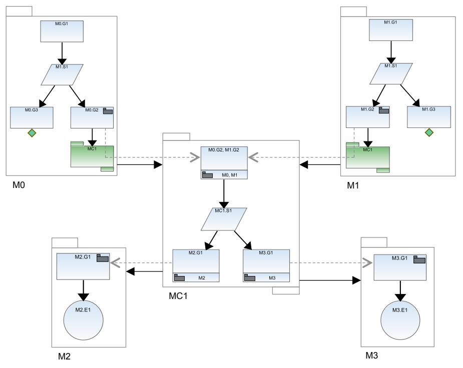
Our definition does not excessively constrain the type of GSN structures that may be used to specify a contract. Thus, we allow additional internal nodes, e.g., we do not require the root to be an away goal, although children of *out*–nodes are prohibited (Fig. 4a). We also permit contracts to contain multiple *in*–nodes, e.g., a chain of *in*–nodes (Fig. 4c), so that the away goals are still linked to consumer modules (also see Definition 5). Fig. 4 additionally shows some of the permissible links to/from module contracts, which we will clarify when we formalize module *scope* (Definition 5).

Given a set of modular/contractual argument structures, intuitively, we can logically collect and organize them according to (domain-)specific concerns, which gives rise to *containers* for the collections (e.g., SW and SW2 in Fig. 3) and a hierarchy (e.g., see Fig. 3, bottom right).

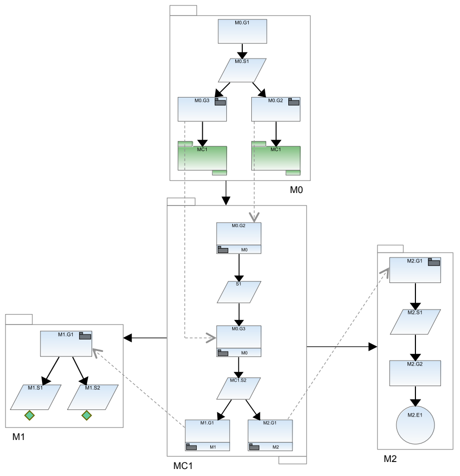
[3] The rationale is to directly resolve *auxiliary* subgoals internal to the contract, without needing to create an additional, external module.



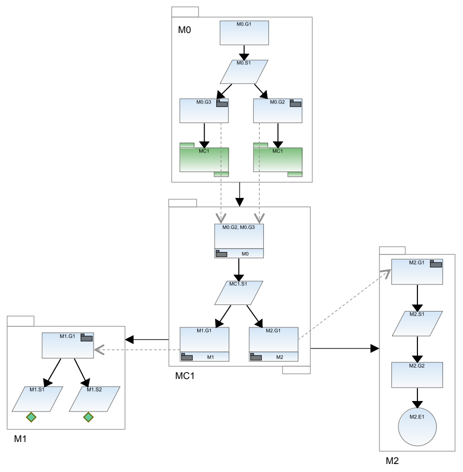
(a) Consumer module M0 invoking a non-root *in*–node in contract module MC1, linking to a non-root public goal in provider module M1.



(b) Common (root *in*–node in) contract module MC1 invoked from multiple consumer modules M0 and M1.



(c) Different argument legs in consumer module M0 (different claims) invoking different *in*–nodes of the same contract module MC1.

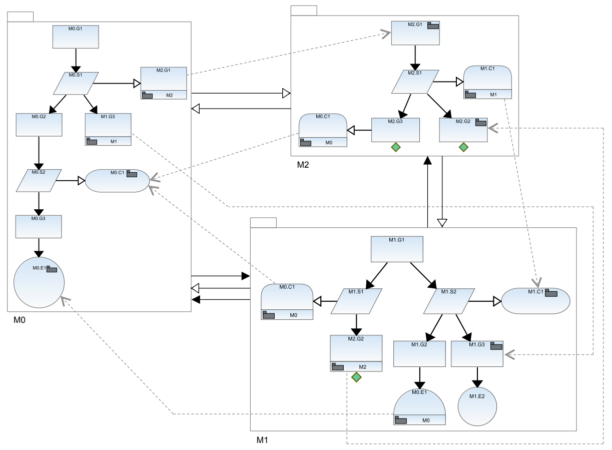


(d) Different argument legs in consumer module M0 (different claims) invoking common root *in*–node in same contract module MC1.

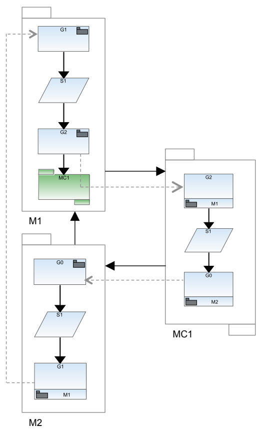
Fig. 4. Examples of constraints on contract modules showing internal/external links.

Definition 4 (Module Hierarchy). *A* module hierarchy, $\mathcal{H}$, *is a tuple* $\langle \mathcal{I}_m, \mathcal{I}_a, \mathcal{I}_c, \mathcal{A}, \mathcal{C}, \mathcal{M}_a, \mathcal{M}_c, < \rangle$*, comprising distinct sets of module container IDs,* $\mathcal{I}_m$*; modular argument IDs,* $\mathcal{I}_a$*; contractual argument IDs,* $\mathcal{I}_c$*; modular arguments,* $\mathcal{A}$*; contractual arguments,* $\mathcal{C}$*; and mappings* $\mathcal{M}_a : \mathcal{I}_a \to \mathcal{A}$*, and* $\mathcal{M}_c : \mathcal{I}_c \to \mathcal{C}$*, along with a forest* $\langle \mathcal{I}, < \rangle$*, where* $\mathcal{I} = \mathcal{I}_a \cup \mathcal{I}_c \cup \mathcal{I}_m$*, such that* $i \in \mathcal{I}_a \cup \mathcal{I}_c \Rightarrow \texttt{leaf}(<, i)$*, and* $\texttt{root}(<, i) \Rightarrow i \in \mathcal{I}_m$.

The forest represents the *containment relation* between modules, for which there need be no single top-level module. For convenience, we will abuse notation and write $M = \langle N, \ldots \rangle$ when we mean $\mathcal{M}_a(M) = \langle N, \ldots \rangle$, and $n \in M$ rather than $n \in N$.



(a) Permitted cyclic links not inducing a loop



(b) Illegal modularization inducing a loop

Fig. 5. Examples of constraints on cyclic links between modules.

A module hierarchy represents a snapshot of a possibly incomplete collection of safety arguments under development and, thus, although arguments must be leaves (and within a container), we do not require all leaves to be arguments. That is, during development, we allow a module leaf (with no argument within). We also allow a tree with a single node, i.e., an empty module. Since Definition 4 allows forests, we allow multiple arguments in a single module, and multiple argument fragments (with distinct roots) in a single argument. We will use *module* to refer to both modular arguments and contracts, as well as their containers, when the distinction is not significant. Next, we formalize what it means for the links given by away nodes and module/contract module references to exist in the appropriate location in a module hierarchy. First, write $x \dashrightarrow y$ when x and y are in separate modules and there is a reference link from x to y, i.e., x is an away node pointing to public node y. We extend $\dashrightarrow$ from nodes to modules so that $M_1 \dashrightarrow M_2$ when there exist nodes $m_1 \in M_1$ and $m_2 \in M_2$ such that $m_1 \dashrightarrow m_2$ or $tmod(m_1) = M_2$ (which we can write as $m_1 \dashrightarrow M_2$). We write $x \twoheadrightarrow y$ when there exists a z such that $x \dashrightarrow z \rightarrow^* y$.

Definition 5 (Well-scoped Module Hierarchy). *A module hierarchy, $\mathcal{H} = \langle \mathcal{I}_m, \mathcal{I}_a, \mathcal{I}_c, \mathcal{A}, \mathcal{C}, \mathcal{M}_a, \mathcal{M}_c, < \rangle$, is* well-scoped *if the following conditions hold:*

1. *For every away node in every non-contract module there is a corresponding public node in a separate non-contract module: $\forall n_1 \in M_1 \, . \, away \in l_s(n_1) \Rightarrow \exists M_2 \in \mathcal{I}_a, m_2 \in M_2 \, . \, public \in l_s(m_2) \wedge t_r(n_1) = \langle M_2, m_2 \rangle$;*
2. *For every module (contract module) reference, the corresponding appropriate type of module (contract module) exists, and is distinct from any of its container modules: $n \in M, l_t(n) = mr \wedge t_r(n) = M' \Rightarrow M' \in \mathcal{I}_m \wedge M \nleq M'$; and $n \in M, l_t(n) = cr \wedge t_r(n) = M' \Rightarrow M' \in \mathcal{I}_c \wedge M \nleq M'$;*
3. *If a goal node is supported by a contract module reference to a contract C, then the goal is a public node, and it corresponds to a non-leaf away node (i.e., an* in*–node)*

in C: $l_t(n) = g \wedge l_t(c) = cr \wedge n \rightarrow c, t_r(c) = C \Rightarrow public \in l_s(n) \wedge \exists k \in C \,.\, away \in l_s(k) \wedge t_a(n) = \langle C, k \rangle$*;*

4. Anti-cycle condition*: If* $n_1 \rightarrow_1^* m_1$ *in argument module* M_1 *and* $n_2 \twoheadrightarrow^* n_1$ *and* $m_1 \twoheadrightarrow^* m_2$*, then* $m_2 \not\rightarrow_2^* n_2$ *in argument module* M_2*;*
5. Scope: *Inter-module links respect the module hierarchy, i.e., if* $M_1 \dashrightarrow M_2$ *then either* M_1 *and* M_2 *are siblings or* M_2 *is a child of* M_1*:* $\exists M_3 \in \mathcal{I}_a \,.\, M_1, M_2 < M_3$ *or* $M_2 < M_1$*;*

Additionally, for every contract, C*, the following conditions hold:*

6. Out*–nodes link to separate argument trees (which can be in the same module*[4]*), i.e.,* $out(n) \wedge out(n') \wedge t_a(n) = \langle M, g \rangle \wedge t_a(n') = \langle M', g' \rangle \Rightarrow \nexists m \,.\, m \rightarrow^* g, g'$*; and* $t_a(n) = \langle M, \_ \rangle \wedge t_r(n') = M' \wedge t_r(n'') = M'' \Rightarrow (M \neq M' \wedge M' \neq M'')$*;*
7. In*– and* out*–nodes link to different modules:* $in(x) \wedge out(y) \Rightarrow tmod(x) \neq tmod(y)$*;*
8. *At least one* in*–node is linked to a source:* $\exists n \in \mathcal{C} \,.\, in(n) \wedge \exists m \in \mathcal{M}_a \,.\, public \in l_s(m) \wedge n \dashrightarrow m$*;*
9. *All* out*–nodes are linked to targets. Thus, if a leaf is an away node*[5] *then there exists a linked public node in the corresponding module*[6]*:* $\forall n \in C \,.\, away \in l_s(n) \Rightarrow \exists m \in \mathcal{M}_a \,.\, public \in l_s(m) \wedge n \dashrightarrow m$*.*

From Definition 5, our notion of scope (item 5) is that modules should only access those nodes of their siblings declared to be public, but not their siblings' children modules nor their own grandchildren[7] based on the principle that to access a public node of a child module, the parent must itself use and then redeclare that node. Well-scoping forces the external sets of IDs in Definition 2 (i.e., $\mathcal{I}_m$ and $\mathcal{I}_n$) to be drawn appropriately from within the module hierarchy. Although we permit cycles between modules (Fig. 5a), we need to prevent cycles in the underlying 'unfolded' argument; thus, it is not sufficient to state the *anti-cycle condition* (item 4) for pairs of modules, i.e., we must consider arbitrary length chains. (Fig. 5b illustrates the underlying rationale).

Note that items 6–9 apply to contracts; thus, although item 1 seems to cover item 9, the former applies to *non-contract modules*. The intuition underlying the latter is that a contract may provide support to multiple consumer modules, e.g., via reuse. Thus, we permit an *in*–node to be invoked by multiple consumer modules (Fig. 4b), as well as by multiple claims in the same consumer module (Fig. 4d). This is useful in a scenario where a common contract provides support for multiple arguments, or for a claim that appears in related argument legs. Consequently, we require all *out*–nodes to be linked to provider modules, but not all *in*–nodes need be linked. Also, item 1 precludes referring to public nodes in a contract, by definition (Definition 3, item 4).

Items 8 and 9 prohibit nested contracts or references to other contracts, primarily to simplify formalization. In general, away nodes never point *into* a contract; rather, the *in*– and *out*–nodes (goals) of a contract refer *out* to the corresponding public goals

[4] Recall that (modular) argument structures are forests.

[5] Module references are handled by item 2 of Definition 5.

[6] Note that the public goal node need not be the root of module M (Fig. 4a).

[7] A possible relaxation of this condition would be that modules *can* access child modules of siblings, i.e., in $\mathcal{H}$, the module containing the target of an away node is $\leq$ the module containing the source node. Another possible relaxation is to allow a module access to its grandchildren. However, these alternatives limit the benefits of encapsulation.

of the provider and consumer modules respectively. However, note that for contracts (Fig. 4) the $\rightarrow$ link between an away *in*–node of a contract and its referenced public leaf goal in the provider module points in the *opposite* direction.

Finally, we can define when a hierarchy is well-formed.

Definition 6 (Well-formed Hierarchy). *A well-scoped module hierarchy, $\mathcal{H}$, is a* well-formed hierarchy *if:*

1. *The properties of away–public node pairs are related, i.e.,* type *and* description *are equal,* status *is equal for tbd and $tbsbc$ (which can only apply to goals), while* metadata *of the away node are a superset of those of the public nodes: for each module/contract module $M = \langle N, l, t, d, \rightarrow \rangle \in \mathcal{M} \cup \mathcal{C} \in \mathcal{H}$, $M' = \langle N', l', t', d', \rightarrow' \rangle$, $n \in N$, and $t_a(n) = \langle M', n' \rangle$—i.e., n is an away node, n' is a public node—(and M, M' do not violate Definition 5), we have $l_t(n) = l'_t(n')$, $l_d(n) = l'_d(n')$, $l_m(n) \subseteq l'_m(n')$, $l_s(n) \cap \{tbd, tbsbc\} = l'_s(n') \cap \{tbd, tbsbc\}$, and $n' \in M', M \neq M'$;*
2. *Module/contract module reference descriptions equal those of the provider module/contract module (due to well-scoping, Definition 5, the diagram types are correct): if $t_r(n) = M = \langle N, l, t, d, \rightarrow \rangle$ then $l_d(n) = d$, and $l_t(n) = mr \Rightarrow M \in \mathcal{I}_a$ and $l_t(n) = cr \Rightarrow M \in \mathcal{I}_c$.*

Formally, away nodes also have a *description*, *metadata*, and tbd status, but not independently of the public goal node. Specifically, the *description*, and $tbd/tbsbc$ status are the same as that of the public goal node. Moreover, the away node inherits the metadata of the public node, although we allow it to have additional attributes. Similarly, the description of a module/contract module reference is the same as that of the target module/contract module. The rationale for allowing additional metadata for away, module reference and contract module reference nodes, is that they are *private* to a module. A module user may augment such nodes with additional semantic information, local to the containing module, and specific to a user perspective and/or the context of usage, beyond what has been added to the target node (e.g., the intended use of the module, or other intellectual property). Alternatively, since the target of an away node is public, a reasonable assumption is that its metadata is also public and, therefore, inherited.

5 Discussion

Based on our formalization for modular arguments we now have a rigorous basis to distinguish modularity from hierarchy [12] in safety arguments. Informally, however, hierarchy can be considered as a *vertical* abstraction of structure, whereas modularity is useful when a *horizontal* abstraction is required (although it can be applied in both dimensions). We can now respond to the questions posed in Section 3 (in the same order): (1) In general, Definition 5, item 5, clarifies the scope of public nodes in a modular argument. In particular, in Fig. 3, we disallow module *SW* from containing module *SW2*, as it violates scope. Accordingly, module *SW1* also ought not to be contained in module *SW*. If such containment is required, the away goals of the argument SHMA, in module *system-argument*, should be replaced with equivalent public goals that either reference a contract offered by module *SW*, or are marked with status $tbsbc$; (2) Definition 5 also provides the constraints on links across module boundaries. In particular, the *anti-cycle* condition (item 4) allows cycles in modules but prohibits them in the underlying arguments. Thus, elements of the argument SFA in module *SW* can reference elements of the argument SHMA in module *system-argument*, subject to the constraints imposed by our formal definitions; (3) Definition 2, item 9 prohibits claim support though multiple contracts, since the presence of a contract module reference from a public goal precludes any other out links. However, we note that this condition indicates a conflict in the GSN standard guidance, which does not prohibit goals (in non-modular arguments) to be both supported directly by evidence, and also by other intermediate goals; (4) In general, contracts are subject to most of the same constraints as modular arguments (Definition 2), but must meet the specific constraints given by Definition 3; (5) Definition 6 clarifies the valid properties of modular GSN nodes. In particular, item 1 of Definition 6 permits away goals to have a tbd ($\Diamond$) status; (6) Finally, our formalization permits multiple claims in one module (or multiple modules) to invoke a common contract (Definition 3, Fig. 4), although the invocation of multiple contracts is prohibited (Definition 2).

6 Concluding Remarks

We have given the elements of a theory for formalizing modular safety arguments that provides a rigorous basis for tool implementation, and our focus has been mainly intra-module GSN. The current implementation of modules in our toolset, AdvoCATE [9], is preliminary and not all checks have yet been implemented. By formally defining modular arguments, contracts, scope, and well-formedness, we have clarified and made explicit many structural assumptions that were not previously described in the literature. We have also developed a theory of modular patterns (omitted here due to space limitations), which presents numerous design choices and subtleties. Though a lack of space has precluded our giving a correctness theorem, intuitively, all references are well-defined in a well-scoped and well-formed module hierarchy. To formalize this, we can interpret modules as non-modular arguments via a notion of *unfolding*, capturing the intuition behind inter-module links as denoting an underlying monolithic argument.

Many of the conditions that required formalization are quite intricate and the proscriptions of the standard often seem ad hoc. A more abstract approach to module contracts, extended with a formal assume/guarantee language would be useful [13]. Various restrictions made through our formalization could be relaxed, e.g., nesting contracts and references to other contracts seems to be reasonable. Our formalization is partly based on the guidance for modularity in the GSN standard [4] which, itself, is based upon the work in [2] and [14]. However, in general, the standard only provides limited guidance on the interconnections between modular arguments, contracts, the related constraints, and issues of scope. Moreover, it does not address hierarchy in (modular/non-modular) arguments. Earlier research on modular software safety arguments [11], [15] has addressed scope and containment, albeit only informally, and is compatible with our formalizations (Section 4). Neither contemporary work on formalizing GSN modules [16], nor notation agnostic meta-models of safety argumentation (to our knowledge) [8], have considered issues of well-formedness and scope. As future work, we will extend the theory to account for *module views*, i.e., inter-module GSN, and their relationship to the underlying modules. We also intend to look at additional aspects, e.g., how context shared by collaborating modules will be managed, and the relationship to modularization concepts in other modeling languages such as Unified Modeling Language (UML), etc. Elsewhere, we developed a notion of argument *query* [7] for individual argument structures. Extending this to modules would require a notion of multi-diagram view, which is another avenue for future work.

Acknowledgement. We thank Jane Fenn (BAE Systems), Ibrahim Habli, and Richard Hawkins (University of York) for earlier discussions on modular GSN. We also thank the anonymous reviewers for their helpful comments. This work was supported by the SASO project of the NASA ARMD Airspace Operations and Safety Program.

References

1. Despotou, G., Kelly, T.: Investigating the Use of Argument Modularity to Optimise Through-Life System Safety Assurance. In: 3rd IET Intl. Conf. System Safety. (Oct. 2008) pp. 1–6
2. Kelly, T.: Managing Complex Safety Cases. In: Current Issues in Safety-Critical Systems. Springer London (2003) pp. 99–115
3. Fenn, J., Hawkins, R., Williams, P., Kelly, T., Banner, M., Oakshott, Y.: The Who, Where, How, Why And When of Modular and Incremental Certification. In: 2nd IET Intl. Conf. System Safety. (Oct. 2007) pp. 135–140
4. Goal Structuring Notation Working Group: GSN Community Standard v1 (Nov. 2011)
5. Kelly, T., Bates, S.: The Costs, Benefits, and Risks Associated With Pattern-Based and Modular Safety Case Development. In: Proc. UK MoD Equipment Safety Assurance Symp. (Oct. 2005)
6. Denney, E., Pai, G.: A Formal Basis for Safety Case Patterns. In: Computer Safety, Reliability and Security (SAFECOMP 2013). LNCS 8153. (2013) pp. 21–32
7. Denney, E., Naylor, D., Pai, G.: Querying Safety Cases. In: Computer Safety, Reliability and Security (SAFECOMP 2014), LNCS 8666. (Sep. 2014) pp. 294–309
8. Object Management Group: Structured Assurance Case Metamodel (SACM) version 1.0. Formal/2013-02-01 (Feb. 2013)
9. Denney, E., Pai, G., Pohl, J.: AdvoCATE: An Assurance Case Automation Toolset. In: SAFECOMP 2012 Workshops. LNCS 7613. (Sep. 2012) pp. 8–21
10. Fenn, J., Hawkins, R., Williams, P., Kelly, T.: Safety Case Composition Using Contracts - Refinements based on Feedback from an Industrial Case Study. In: Proc. 15th Safety Critical Systems Symp. (SSS' 07). (Feb. 2007)
11. Industrial Avionics Working Group: Modular Software Safety Case Process GSN – MSSC 203 Issue 1 (Nov. 2012)
12. Denney, E., Pai, G., Whiteside, I.: Formal Foundations for Hierarchical Safety Cases. In: Proc. 16th Intl. Symp. High Assurance Sys. Eng. (HASE 2015). (Jan. 2015)
13. Warg, F., Vedder, B., Skoglund, M., Söderberg, A.: SafetyADD: A tool for safety-contract based design. In: Proc. 25th Intl. Symp. Soft. Rel. Eng. Workshops (ISSREW 2014). (2014)
14. Kelly, T.: Concepts and Principles of Compositional Safety Case Construction. Technical Report COMSA/2001/1/1, University of York (2001)
15. Industrial Avionics Working Group: Modular Software Safety Case Process Description – MSSC 201 Issue 1 (Nov. 2012)
16. Matsuno, Y.: A Design and Implementation of an Assurance Case Language. In: 44th Intl. Conf. Dep. Sys. Networks (DSN 2014). (June 2014) 630–641