ExplicitCase: Tool-support for Creating and Maintaining Assurance Arguments Integrated with System Models

Carmen Cârlan, Vivek Nigam, Sebastian Voss and Alexandros Tsalidis

fortiss — An-Institut Technische Universität München, Germany

{carlan, nigam, voss, tsalidis}@fortiss.org

I. Abstract

***Abstract*—Assurance cases are collections of standard-mandated documents that entail the specification of system's objectives and a collection of processes, development or verification evidence regarding the satisfaction of the respective objectives. A considerable amount of work has been done in the direction of modelling assurance cases, to support communication and reasoning regarding the system's safety. In this work, we present a set of features of ExplicitCase - a tool for modeling assurance cases. While there is a plethora of tools for creating and managing model-based assurance cases, the uniqueness of our tool is that it integrates assurance case models with system models created in AutoFOCUS3 (AF3) - an open-source model-based development tool for embedded software systems. While trying to keep up with state-of-the-art assurance case editors, the newly implemented features support assurance case creation using typed patterns, change impact analysis for assurance cases, assessment of the confidence in the created assurance arguments, export of the argumentation diagrams generated in ExplicitCase and integration of assurance case models with system models created in AutoFOCUS3. In particular, based on the integration with AF3 system models, we propose automatic support for detecting the impact of a change within system models on the assurance case model, thus enabling the integrated development of system and assurance case models.**

***Index Terms*—assurance cases, model-based development, change impact analysis**

II. Introduction

Standards in various safety-critical domains mandate the existence of safety assurance cases. A safety assurance case is a structured argumentation on system safety, based on evidence proving the satisfaction of the system's safety goals, in a certain context, under certain assumptions [1]. Assurance cases comprise the collection of all documents generated during the execution of standard-mandated activities. In the last years, extensive research has been done in the direction of model-based assurance cases. First, model-based assurance cases provide an overview of the argumentation underlying in the documents. Second, they support automatic manipulation of assurance cases. There already exists a plethora of tools for editing and managing model-based safety cases. In previous work we also propose such a tool, based on a metamodel built in compliance with the standard Goal Structuring Notation (GSN) [2]. While other assurance case editors integrated with system modeling tools exist, our tool's uniqueness lies on the fact that we integrate assurance case models with system models and verification artefacts generated with AutoFOCUS3 (AF3) [3]. AF3 is an open-source model based development tool for distributed, reactive, embedded software systems, based on the FOCUS semantics. ExplicitCase offers computer-aid for writing assurance argumentation regarding the correctness of the software automatically generated from AF3 models. Such assurance argumentation is necessary for software certification.

Maksimov et. al. [4] have done a survey regarding existing assurance case editors. They define the following six distinct tool functionalities specific to such editors: *creation*, *maintenance*, *assessment*, *integration*, *reporting* and *collaboration*. They then define four levels of the implementation of these functionalities (from *D* - weakest support to *A* - strongest support), while enumerating a set of features corresponding to each level. Given the tool features we implemented in previous work [5], we have achieved level *B* for the *creation*, *maintenance*, *assessment* and *integration with other artefacts* of assurance cases.

a) Our long-term goal: For software certification, users of AF3 modelling embedded software systems need to provide assurance argumentation. System models or verification results in AF3 may act as evidence supporting the assurance argumentation. Our long-term goal is to provide our users with a tool equipped with all state-of-the-art functionalities of an assurance case editor, while enabling the integrated development of assurance case and AF3 system models. Such integrated development allows the assurance case model to evolve along the system models created in AF3.

To achieve our long-term goal and to keep up with the state-of-the-art assurance case editors, in this work, we implement or extend five functionalities of the tool:

- **Support for Assurance Case Creation.** In previous work we present how assurance cases can be modeled in ExplicitCase [6]. In this work, we offer support for the specification and instantiation of argumentation patterns that can be reused in assurance cases of other projects, aiding users to develop convincing assurance cases.

- **Support for Assurance Case Maintenance.** During system development, the system artefacts modeled in AF3 may undergo changes. Therefore, support for maintaining the system's assurance case model given a change in referenced system models is necessary. First, we provide automatic annotation of challenged assurance case elements given changes in referenced AF3 system model elements. Second, we implement in ExplicitCase a state-of-the-art change impact analysis algorithm [7].
- **Support for Assessment of Assurance Cases.** Previously, we presented on-the-fly checks of the syntax of assurance arguments in ExplicitCase. One other possibility for assessing assurance cases is to assess the confidence in the validity of the argumentation. Based on the work of Duan et. al. [8], we implement in ExplicitCase a new feature allowing users to annotate GSN-based nodes with information regarding their confidence in the validity of argumentation claims.
- **Support for Assurance Case Report Generation.** The GSN-based diagrams created in our tool shall be used in different documents. Consequently, we implement a feature supporting the export of diagrams generated in ExplicitCase into *pdf*, *png* or *jpeg* files that can be then integrated into other documents.
- **Support for Assurance Case Integration with other Artefacts.** To support automatic creation and maintenance of assurance arguments, there is a need for the integration of assurance case models with system artefacts. ExplicitCase enables integrated development of assurance case and AF3 native system models, as presented in previous work [5]. To support the various needs of users, in this work we extend this feature, by allowing users to reference any type of AF3 native system model elements to any core assurance case element (i.e., goal, strategy, solution, context, justification, assumption). Furthermore, while previously one assurance case element could only have one reference to a single AF3 native system model element, now it may reference a set of elements.

b) Paper structure: First, we provide some background information regarding model-based assurance cases (see Section III). Second, in Section IV, we present how we support the creation and instantiation of GSN-based assurance case patterns. Third, in Section V, we present how we enable multiple, constraint-free hyperlinking. We then continue with Section VI, where we present how users of ExplicitCase can perform change impact analysis on assurance case models. In the next two sections, we present two new features of ExplicitCase that enable users to specify quantitative confidence assessment in the assurance case models and to export the GSN-based diagrams to external files. Towards the end, we discuss on the contribution of this work, by comparing ExplicitCase with state-of-the-art assurance case editors (see Section IX). We conclude with a summary and a discussion about possible directions of future work (see Section X).

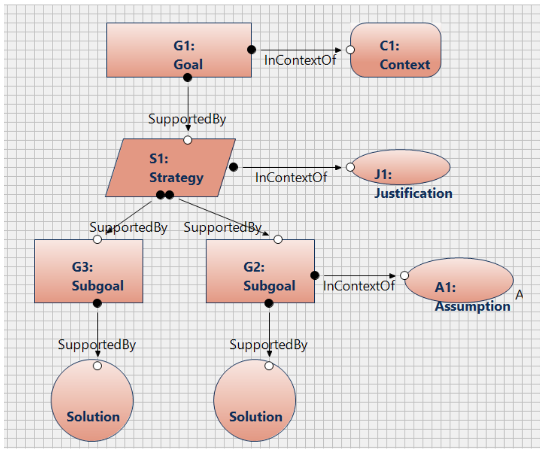


Figure 1: Illustration of the GSN elements.

III. Preliminaries

a) Goal Structuring Notation (GSN): One manner to model assurance cases is based on the standardized and commonly used Goal Structuring Notation (GSN) [2]. GSN supports the graphical representation of the logical flow between safety claims, depicted as *goals*, *strategies* for decomposition of the goals in subgoals and evidence for the truth of the claims, depicted as *solution* elements, with the help of the *SupportedBy* relationship (see Figure 1). With GSN one can also depict contextual information regarding the argumentation, via *context* and *assumption* nodes and the rationale behind the arguments with the help of *justification* nodes.

b) GSN patterns: GSN has also an extension dedicated for assurance case patterns. Similarly to design patterns, assurance case patterns are argumentation fragments that may be reused for arguing about the satisfaction of similar goals, in similar contexts [9]. The claims within patterns may contain words in curly brackets that shall be replaced with project specific information, when the pattern is applied in an assurance case. Matsuno et. al. [10] go a step forward and introduce *typed patterns*, where these words in curly brackets are actually variables of certain type. Typed patterns enforce the instantiation of claims within patterns with the correct type of information. Elements in patterns may be annotated as *uninstantiated* when their claims require project-specific information. Also, goals and strategies may be annotated as *undeveloped* if they require to be further developed by subgoals or solutions. On the one hand, optional relationships and entities can be used for describing the parts of the argumentation that are not mandatory to be used in the instantiated argumentation. On the other hand, relationships having their multiplicity attribute set to true enable the usage of multiple instances of a certain part of the pattern.

c) Change impact analysis for safety case: Kelly et. al. [7] propose a change management process for GSN-based assurance cases. This process has two phases: the damage and the recovery phase. Whereas in the damage phase, an impact analysis for a challenge to the validity of a GSN node is done on the entire GSN argument, in the recovery phase the assurance case is adjusted to the change. Before the impact analysis, one needs to identify the cause of a challenge in an assurance case. Challenges in assurance cases may have different causes, such as adding a new requirement, a change in system design, or a change in the operational context. Then, the GSN elements that may be *challenged* due to this cause shall be identified. The potential area of argumentation impacted by a challenge in one GSN node may be identified automatically, using an algorithm. The outcome of the change impact analysis is a set of GSN elements annotated as *potentially impacted*. The selection of the actually impacted nodes is to be done by the safety engineer.

d) ExplicitCase: The current work extends the functionality of ExplicitCase [5], a model-based assurance case editor. In previous work, we presented out proposed GSN-based metamodel [6]. In *ExplicitCase*, all classes modeling GSN nodes (e.g., *goals* or *solutions*) extend the *ArgumentElement*, which is an abstract class for depicting any type of GSN node. Assurance cases in *ExplicitCase* have a modular structure, being formed of *ArgumentModules*. An argument module is a group of highly-coupled GSN nodes, which can communicate with other modules via interfaces. Usually, a change in a module does not affect other modules. To support users in building correct GSN argumentation structures, the tool implements modeling constraints and on-the-fly checks, which enforce the best practices for creating assurance cases. Also, the tool allows referencing external files in GSN elements.

System modeling in AutoFOCUS3 AutoFOCUS3 (AF3) [3] is a model-based development tool. With AF3 one can model different levels of abstractions (or views) for a system. At the *requirements* level, the user can model plain-text or formalized requirements. The *logical architecture* (*LA*) describes the logical structure of a system in form of *components* and*communication links*. The components have *interfaces* and their behavior may be modeled via *state* or *mode automaton*. The *technical architecture* (*TA*) is a model of the hardware: processors, buses, actuators and sensors. In AF3, *deployments* - allocations of components from the logical architecture on components from the platform architecture and *schedules* - order of tasks executions can also be modeled. At the *implementation* level, the *source code* is automatically generated from the aforementioned models.

IV. Creation of Convincing Assurance Arguments

To support ExplicitCase's functionality of creating assurance cases, we implement a feature enabling users to create and instantiate GSN-based patterns. Patterns are argumentation structures already successfully used in previous projects that can be re-used in similar projects, thus supporting the creation of convincing arguments.

A. Metamodel Extension for Patterns

To enable the creation and manipulation of patterns in ExplicitCase, we extended our metamodel, as presented in Figure 2. First, we add the extensions recommended by the GSN standard [2]. Consequently, we add to the metamodel a class representing the option entity. Then, we enhance the *ArgumentElement* class with attributes to specify *undeveloped* and *uninstantiated* elements and we add to the *AssuranceCaseConnection* class attributes to depict *multiplicity* and *optionality*. Second, we enable typed claims by adding to the *ArgumentElement* class an attribute of type *Claim*. A claim is a construct specifying the *text-based claim*, the *pattern claim*, based on which the text-based claim is to be created, and a set of *variables* (i.e., the words in curly brackets). A variable has an artefact type and a value. Variable types may be any type of AF3 native system model elements, strings or documents.

B. Pattern Creation

An argument module within ExplicitCase may be saved as a pattern in the AF3 assurance case pattern library if it has no references to any artefacts (see Figure 2). When creating a pattern, the user may want to annotate an argument element as being uninstantiated or undeveloped. As a visual-aid, uninstantiated and undeveloped entities are depicted in different colors. Furthermore, a pattern may contain option entities and optional and generalized relationships.

a) Typed claims: The claims in argument elements within patterns may contain words in curly brackets that shall be then replaced with information specific to the system the pattern is instantiated for. While such claims may be written as strings, in ExplicitCase we also enable the specification of typed claims. After writing the text of the claim, the user may create for each world in curly brackets a variable, declaring the variable's type.

C. Pattern Instantiation

To use a pattern in a current assurance case, the user shall add the respective pattern from the *Assurance Case Pattern Library* in the current assurance case by drag-and-drop (see Figure 2). An instance of a pattern is actually an argument module. Then, the user shall go through each element within that argument module and instantiate it.

a) Undeveloped and uninstantiated entities: If a goal is annotated as *undeveloped* in the pattern, then it shall either be further-developed by subgoals or a solution or it shall be replaced by an away goal referencing a goal in another module, further developing the argumentation. Similarly, an *undeveloped* strategy shall be added subgoals. To annotate a GSN entity as *instantiated*, the entity's claim needs to have all its claim variables instantiated.

b) Option entities: Option entities are only in patterns. When instantiating the pattern, the engineer shall select the subgoals that will be used. After the selection, the unused goals, together with the option entity are automatically deleted.

c) Optional relationships: When instantiating a pattern, the user may choose to delete certain optional relationship. If the target argument element of the respective relationship is not connected to any other part of the argumentation, then the respective element is deleted, together with the respective relationship and all its children elements (i.e., subgoals, strategies, contextual elements).

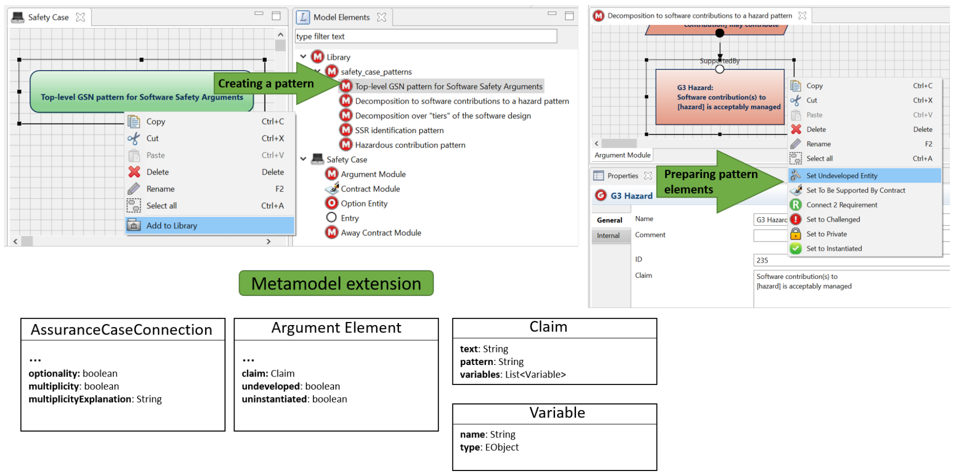


Figure 2: Illustration of the patterns feature.

d) Generalized relationships: If the multiplicity annotation of a relationship is set to true, the user may choose to multiply the argumentation targeted by this relationship.

e) Typed instantiation of claims: If the claim of a node in the instantiated pattern is typed (i.e., it contains variables), the user can instantiate the claim's variables by selecting from the AF3 elements of the type of the respective variables. Some claims have, however, words in curly brackets without any dedicated variables. These words are to be instantiated manually, in the dedicated text field.

V. Integration with AF3 System Models

A safety assurance case is a collection of all the existing evidence and supporting material that the system satisfies its safety objectives. From safety analysis results to code file, any system development artefact may be referenced in the system's assurance case if it can somehow support the argumentation about the system's safety. In related work, we notice how hazards, hazard analysis, test results or design components are referenced in safety case fragments and patterns. As such, one feature of ExplicitCase is the support for integrating the assurance case metamodel with other AF3 metamodels.

a) Multiple hyperlinking: In several published assurance case fragments or patterns a GSN node references multiple artefacts of the same type (e.g., hazards, requirements etc.). For example, one context claim within the *SSR Identification Pattern* described by McDermid [11] references to multiple requirements: *SSRs relevant to design decisions are SSRs*. As such, we enable that a single assurance case element can reference a list of AF3 native model elements of the same type (e.g., a list of *requirements*).

b) Integration with different AF3 "native" metamodels: The user can annotate any assurance case model element with references to any AF3 native model elements of different types. For example, a goal can reference to both a *requirement* and a *component* within the *logical architecture*. Furthermore, we now enable referencing *modes* and *mode automata*, *states* and *components* of logical and technical architectures.

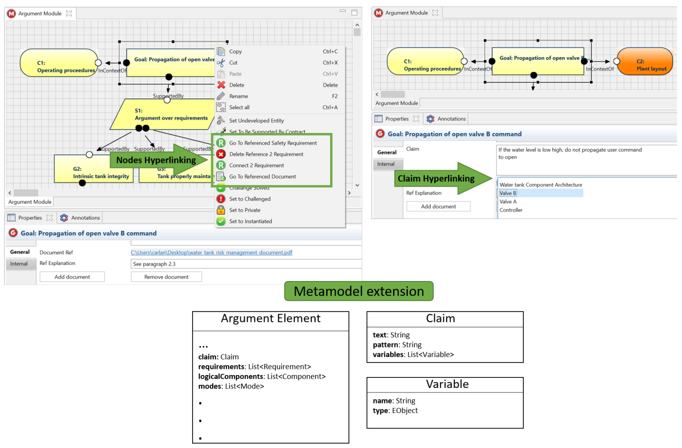
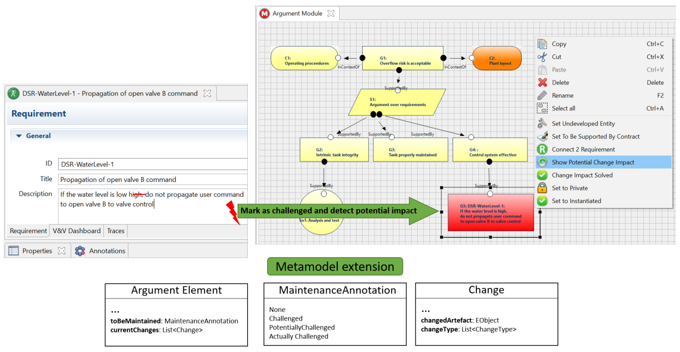


Figure 3: Illustration of the integration feature.

c) Metamodel extension for hyperlinking: Any assurance case element of type assumption, context, goal, justification, solution and strategy can reference to a set of types of AF3 model element: requirements, logical architecture/logical component, platform architecture/platform architecture component, mode automaton/mode, state automaton/state, deployment. As such, we extend the *ArgumentElement* class with corresponding attributes. To enable references to artefacts in the claim within a GSN node, each *ArgumentElement* has a *Claim*, as defined in Section IV.

VI. Change Impact Analysis for Assurance Cases

Artefacts can evolve during a system's lifecycle (i.e., system development phase and operational phase), undergoing different types of changes. For example, a claim within the assurance case model may be invalidated because of changing requirements, additional safety evidence or a change in the system architecture. As such, impact analysis is necessary to guarantee that system safety and the corresponding argumentation are not jeopardised. In ExplicitCase, given certain types


DSR-WaterLevel-1 - Propagation of open valve B command

Requirement

General

ID: DSR-WaterLevel-1

Title: Propagation of open valve B command

Description: If the water level is low ~~high,~~ do not propagate user command to open valve B to valve control

Requirement
V&V Dashboard
Traces
Properties
Annotations
Argument Module
C1: Operating proceedures
G1: Overflow risk is acceptable
C2: Plant layout
InContextOf
SupportedBy
S1: Argument over requirements
G2: Intrinsic tank integrity
G3: Tank properly maintained
G4: : Control system effective
Sn1: Analysis and test
G5: DSR-WaterLevel-1: If the water level is high, do not propagate user command to open valve B to valve control
Mark as challenged and detect potential impact
Copy Ctrl+C
Cut Ctrl+X
Paste Ctrl+V
Delete Delete
Rename F2
Select all Ctrl+A
Set Undeveloped Entity
Set To Be Supported By Contract
Connect 2 Requirement
Show Potential Change Impact
Change Impact Solved
Set to Private
Set to Instantiated
Metamodel extension

Argument Element	MaintenanceAnnotation	Change
... **toBeMaintained**: MaintenanceAnnotation **currentChanges**: List<Change>	None Challenged PotentiallyChallenged Actually Challenged	... **changedArtefact**: EObject **changeType**: List<ChangeType>

Figure 4: Illustration of the maintenance feature.

of changes in AF3 native model elements, the argumentation nodes referencing these model elements are automatically annotated as *challenged* (see Figure 8). Furthermore, we implement the change impact analysis recommended by Kelly and McDermid [7].

A. Metamodel Extension for Maintenance

To reason about the impact a change in the system artefacts may have on assurance case elements, there is a need for annotating assurance case elements with maintenance metadata. Therefore, in this work, we extend the assurance case metamodel implemented in ExplicitCase such that maintenance information can be specified for each of the assurance case elements. The extension is based on the terminology proposed in the work of Kelly and McDermid [7] (see Figure 4). As such, we add to the *ArgumentElement* class a new attribute named *toBeMaintained*, which depicts the type of affection of a assurance case elements caused by a change - *challenged*, *potentially impacted* or *actually impacted*. Furthermore, for any assurance case element, we enable the specification of types of changes that may affect the respective element, and also the current list of changes that caused an element to be annotated as needing maintenance. In the *currentChanges* attribute we save the list of changes that determined that an assurance case element needs maintenance. We specify a type of change two-fold: the type of artefact that would undergo the change and the type of the respective change. Examples of types of changes may be: the addition, deletion or modification of a system artefact or a modified claim.

B. Manual Maintenance Annotations

The user may manually annotate any GSN node as *to be maintained* (i.e., *challenged*, *potentially impacted* or *actually impacted*). As a visual aid, *challenged* nodes are depicted in red, *potentially impacted* nodes in yellow and *actually impacted* nodes in orange. Also, the user may optionally input information regarding the change impacting the node, namely he or she can reference the changed artefact (either AF3 native model elements or external files) and select the change type from a list of predefined types of changes. When the desired change is not within the list, the user may also directly define a new type of change. In the model, the change impacting the node is added to the list of current changes affecting the respective node. After the user deals with the impact of certain changes on a node, he or she may select the changes whose impact has been solved and delete them. The node will remain *to be maintained* till the impact of all current changes have been dealt with.

C. Computer-aided Maintenance Annotations

ExplicitCase supports the semi-automation of assurance case maintenance. First, for a set of predefined types of system changes, the tool offers automatic detection of challenges within the assurance case. Second, the user may choose to run an automatic impact analysis for detecting the potentially impacted assurance case nodes, given a challenged node just by pressing a button.

1) Computer-aided Identification of Challenges: For certain types of changes we enable the automatic detection of challenged nodes. Whenever an artefact generated in AF3 undergoes a change, the assurance case elements referencing it are automatically annotated as *challenged*, while also saving the changed artefact and the type of change.

a) Changes in external files: When the path to an external file referenced by an assurance case element is changed - added, deleted or modified, the referencing assurance case element are annotated as *challenged*. For example, the path to the external file referencing test results may be modified, meaning that the document may have been changed.

b) Changes in assurance case model elements: When the claim within a assurance case element is modified, the elements having a direct connection to the respective element are annotated as *challenged*. For example, when the claim of a goal is modified, its sub-goals may not be sufficient anymore to satisfy it and therefore they shall be annotated as *challenged*. When a node is deleted, depending on the type of that node, the tool automatically does certain annotations. Some of these annotations are not maintenance-specific annotations. For example, when a solution is deleted, the goals supported by that solution are annotated as *undeveloped*. When a goal is deleted, both its parent goal and its sub-goals are annotated as *challenged*. The deletion of a strategy does not challenge any nodes, but the goal supported by the respective strategy is then directly connected with the strategy's sub-goals, or is marked as *undeveloped*, if the respective strategy was undeveloped. When context, assumption, or justification elements are deleted, the nodes directly connected to such elements are annotated as *challenged* since, an argument is only valid in a certain context. Therefore, when the context is modified, the validity of the argument shall be rechecked.

c) Changes in AF3 native model elements: In ExplicitCase, we support the automatic identification of the *challenged* elements within an assurance case, given certain types of changes within certain types of referenced AF3 native model elements. When a reference is deleted or added, the referencing assurance case element is annotated as challenged. We assume that a change in the name or the id of an element does not have an impact on the assurance case. Furthermore, we support the automatic identification of challenges in assurance cases for the following modifications:

- A modification in a requirement's claim;
- A modification in a logical component - a port is deleted or added or its state or mode automaton is modified;
- A modification in the logical architecture - a component is added or deleted;
- A modification in a state or mode automaton - a state/mode or a transition is added or deleted;
- A modification in the platform architecture - a node is added or deleted;
- A modification in the allocation of logical components to hardware nodes (e.g., a logical component is allocated to another node).

D. Computer-aided Change Impact Analysis

We implemented in ExplicitCase the change impact analysis proposed by Kelly and McDermid [7], which can detect all the nodes potentially impacted by a challenge of a certain node. The analysis will annotate the potentially impacted nodes (see Figure 4, where the potential impact analysis for the challenged goal *G5* is displayed). In our tool, the change impact analysis does not go beyond the boarders of the argument module containing the challenged element. While the potential impact may be automatically computed, the actual impact is to be determined only by a safety engineer. After the potentially impacted nodes are correspondingly annotated, the safety engineer can go through all of these claims and, depending on the case, annotate them as either *actually challenged* or as not impacted. For example, in Figure 4, context node *C2* is annotated as *actually impacted*.

VII. Quantitative Confidence Assessment

Recent works [12], [13], [8] have proposed mechanisms for associating GSN-arguments with quantitative values. These values are inspired by Dempster-Shafer Theories [14] containing three values for, respectively, the Belief, Disbelief, and Uncertainty, and are used to denote the confidence level on the safety of an item.

We implemented in AutoFOCUS the approach proposed by Duan *et al.* [8], which computes the belief, disbelief and uncertainty of a GSN-argument based on the *safety defeaters*. A safety defeater is anything that can reduce the confidence on the argument [12], such as, a software bug.

Consider the GSN-argument depicted in Figure 5. It contains a main hazard which is broken down into two hazard sub-goals. Each GSN goal is annotated with the number of defeaters outruled and the total number of defeaters. In the tool, this is shown by the pair of numbers on the top left corner of GSN goals. For example, the top goal in Figure 5 is annotated with 15/29 denoting that 15 out of 29 safety defeaters have been outruled. Users can only enter these numbers for the leaf goals by editing their property sections, as illustrated by Figure 6. Also, a weight denotes the importance of these goal. From the data on the leaf nodes, the values of

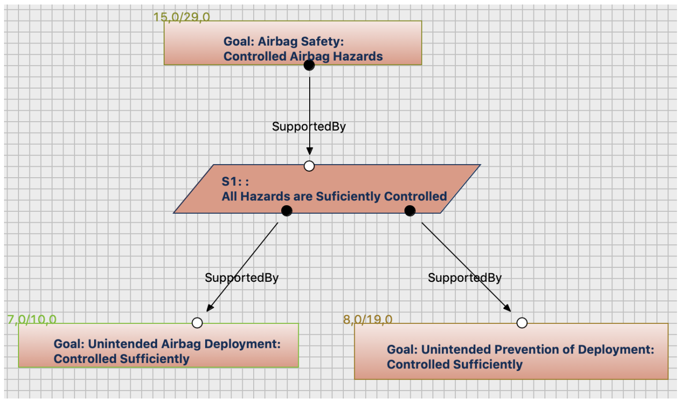
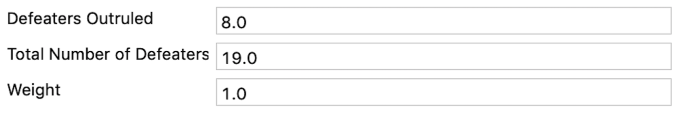

15,0/29,0
Goal: Airbag Safety:
Controlled Airbag Hazards
SupportedBy
S1: :
All Hazards are Suficiently Controlled
SupportedBy
SupportedBy
7,0/10,0
Goal: Unintended Airbag Deployment:
Controlled Sufficiently
8,0/19,0
Goal: Unintended Prevention of Deployment:
Controlled Sufficiently

Figure 5: Illustration of the quantitative evaluation of a GSN, where *b* is for belief, *d* for disbelief, *u* for uncertainty, and *w* for weight.



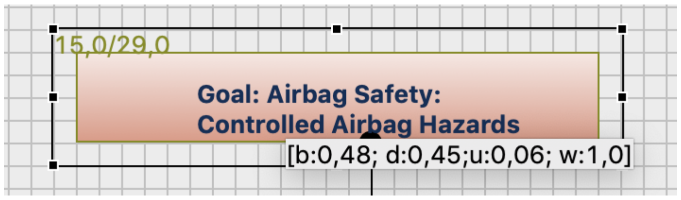
Defeaters Outruled	8.0
Total Number of Defeaters	19.0
Weight	1.0

Figure 6: Illustration of the Safety Defeaters Property Section.

outruled and total defeaters for the remaining GSN nodes are computed by a weight sum.

Intuitively, the greater the total number of defeaters, the lower the uncertainty is. Moreover, the greater the number of outruled defeaters the greater the belief on the GSN-argument and the lower the disbelief. The exact values for belief, disbelief and uncertainty can be computed from the values of outruled and total number of defeaters. We refer to the work [8], [15] on how exactly these values are computed. One feature of the approach described in [8], not present in the approach described in [13], is that a marginal increase on the number of identified defeaters does not greatly affect the values for belief, disbelief and uncertainty.

The belief, disbelief and uncertainty for the top most goal of Figure 5 is shown by simply hovering the mouse over the goal as illustrated by Figure 7. The color of the numbers shown in the goal reflect the level of confidence. Red colors indicating a higher disbelief, while a green color a higher belief.



[b:0,48; d:0,45;u:0,06; w:1,0]

Figure 7: Illustration of the Dempter-Schafer evaluation of a GSN goal.

VIII. Generation of Safety Case Reports

GSN diagrams do not model the entire assurance case and, as such, they do not replace all the documents within an assurance case. Instead, they represent an abstract overview of the argumentation and are included in other documents. Therefore, there is a need to export the GSN diagrams created in ExplicitCase into a format so that they can be easily integrated in text-based documents as figures. Consequently, we implement a new feature within our tool that allows the user to export an assurance case or an argumentation module diagram into SVG, pdf and png, and $jpeg$ files, only by pushing a button (see Figure 8).

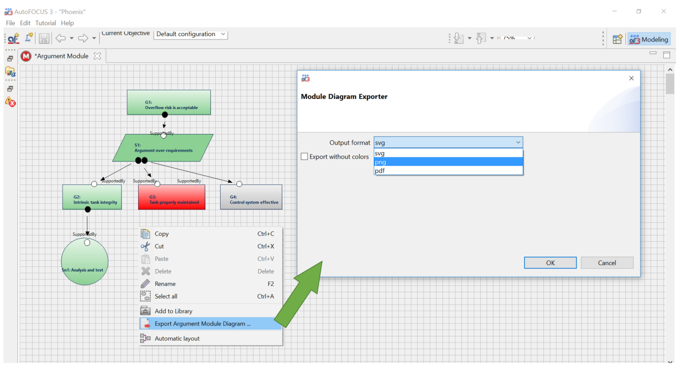


Figure 8: Illustration of the exporting feature of a GSN diagram.

IX. Related Work

There is a large pallet of assurance case editor tools [4], each supporting various functionalities. In this section, we discuss how our tool differentiates itself from other tools.

Assurance case patterns. Not many tools support the creation and instantiation of GSN-based assurance case patterns. While the Papyrus extension proposed by Huhn and Zechner [16] implements the pattern extensions proposed in the GSN standard [2], DCase [10] also implements typed patterns. On the one hand, CertWare [17] provides an argumentation pattern library. AdvoCATE [18], on the other hand, does not only support specification of patterns, but also automated pattern instantiation with data extracted from external sources. Similar to DCase [10], ExplicitCase supports the creation and instantiation of typed patterns.

Integration with system modeling tools. Tools like MMINT-A [19], Resolute [20], ACCESS [21], HIP-HOPS [22] Safety.Lab [23] and OpenCert [24] support integration of assurance case models with system models. In MMINT-A [19] the user can specify references to any type of system models, while our assurance case models can only reference AF3 native system models. Whereas Safety.Lab [23] enables direct references from GSN-based elements to requirement and hazard models, in ExplicitCase references to more types of system models can be specified. However, in Safety.Lab [23] the possible traces have semantics, meaning that the user is constrained and even obliged to define certain types of traces between elements. Other tools offer connections of assurance case models with system models specified in different languages - in Resolute [20] and GAGE [25] assurance cases may reference AADL models, whereas in HIP-HOPS [22] and ACCESS [21] system models are specified in their own defined specification language. To support the usage of system elements out of context in different systems, the assurance cases in OpenCert [24] can be integrated with system models from CHESS tool.

Assurance case maintenance. A variety of tools offer some kind of automatic support for assurance case maintenance. ETB [26] supports maintenance of claims and evidence from formal verification techniques. Tools like GAGE [25], Resolute [20], Access [21] and MMINT-A [19] are similar to our approach, since they support maintenance of assurance cases given changes in referenced system models. In GAGE [25], claim are formalized and their validity can be evaluated against the AADL system models. Similar to our computer-aided detection of challenged GSN nodes, in Resolute [20] the claims invalidated by changes in system models are automatically flagged, given that evidence in assurance cases directly reference system design artefacts. In ACCESS [21], when system models change, part of the assurance argumentation is regenerated based on assurance case patterns. In contrast, most similar to our approach, MMINT-A [19] proposes a change impact analysis given changes in referenced system models.

Quantitative confidence assessment. Similar to our tool, NOR-STA [27] uses Dempster-Shaffer theory to assess the confidence engineers have in the argumentation claims. They also offer different coloring for different degrees of confidence. In contrast to our work, EviCA [28] quantifies confidence in argumentation claims using Evidential Reasoning.

Generation of reports. NOR-STA [27], AdvoCATE [18] and SafeEd support report generation. While SafeEd supports the generation of natural language documentation and reports for the assurance case - including quantitative assessments of the assurance case (e.g., how many goals are undeveloped), ExplicitCase currently only supports the export of the created GSN-based argumentation diagrams.

Different tools offer different features. Only few tools, such as NOR-STA [27], DCase [10] and AdvoCATE [18] support almost all features our tool supports, whereas tools as GAGE [25], Resolute [20], Access [21] and MMINT-A [19] mainly focus on supporting integration with system models. While being comparable with the features of state-of-the art assurance case editors, in ExplicitCase features such as multi-users or versioning are not available.

X. Conclusions

In this paper, we presented a set of novel features of the ExplicitCase tool - a model-based assurance cases editor already presented in Cârlan et. al. [5]. First, we better support assurance case creation by enabling users to create and instantiate patterns for their assurance cases. Second, to increase the tools' support for maintenance, we implement an automatic change impact analysis. Third, in addition to the automatic syntactic checks of the assurance case, now the user can quantitatively assess the assurance argumentation's confidence. Furthermore, the tool has been extended so that the user can export the created GSN diagrams in different file formats. Finally, the user can now connect any type of safety case element to one or more AF3 native system model elements of any type.

For validation, as future work, we plan on using the tool in real-world projects to assess its scalability and applicability. Also, we intend to further work on extending the tool as follows: **1)** Modifying our current metamodel so that it is compliant with the standardized Safety Assurance Case Metamodel; **2)** Defining semantics for the references to AF3 native system model elements; **3)** Providing tool-support for identifying the actual impact of certain types of system changes on selected fragments of the safety case.

Acknowledgment

The research leading to these results is partially funded by Collaborative Embedded Systems (CrEST) project, a project funded by the German Federal Ministry of Education and Research.

References

[1] R. Bloomfield and P. Bishop, "Safety and assurance cases: Past, present and Possible Future – an Adelard perspective," in *Making Systems Safer - Proceedings of the Eighteenth Safety-Critical Systems Symposium*. London: Springer London, 2010, pp. 51–67.

[2] "GSN community standard version 1," Nov. 2011. [Online]. Available: http://www.goalstructuringnotation.info/documents/GSN_Standard.pdf

[3] V. Aravantinos, S. Voss, S. Teufl, F. Hölzl, and B. Schätz, "AutoFOCUS 3: Tooling concepts for seamless, model-based development of embedded systems," in *Proceedings of 8th International Workshop for Model Based Architecting and Construction of Embedded Systems*. IEEE Computer Society, 2015, pp. 19–26.

[4] M. Maksimov, N. L. S. Fung, S. Kokaly, and M. Chechik, "Two decades of assurance case tools: A survey," in *Proceedings of 37th International Conference on Computer Safety, Reliability, and Security - SAFECOMP Workshops*, ser. Lecture Notes in Computer Science, vol. 11094. Springer, 2018, pp. 49–59.

[5] C. Cârlan, S. Barner, A. Diewald, A. Tsalidis, and S. Voss, "Explicitcase: Integrated model-based development of system and safety cases," in *Proceedings of 36th International Conference on Computer Safety, Reliability, and Security - SAFECOMP Workshops*, ser. Lecture Notes in Computer Science, vol. 10489. Springer, 2017, pp. 52–63.

[6] S. Voss, B. Schätz, M. Khalil, and C. Cârlan, "Towards modular certification using integrated model-based safety cases," in *Proceedings of 25th International Conference on Computer Aided Verification - Workshop on Verification Assurance*, ser. Lecture Notes in Computer Science, vol. 8044. Springer, 2013.

[7] T. P. Kelly and J. A. McDermid, "A systematic approach to safety case maintenance," in *Proceedings of 18th International Conference on Computer Safety, Reliability, and Security - SAFECOMP*, ser. Lecture Notes in Computer Science, vol. 1698. Springer, 1999, pp. 13–26.

[8] L. Duan, S. Rayadurgam, M. Heimdahl, O. Sokolsky, and I. Lee, "Representation of confidence in assurance cases using the beta distribution," in *Proceedings of 17th Conference on High Assurance Systems Engineering Symposium - HASE*. IEEE Computer Society, 2016, pp. 170–171.

[9] T. P. Kelly and J. A. McDermid, "Safety case construction and reuse using patterns," in *Proceedings of 16th International Conference on Computer Safety, Reliability and Security - SAFECOMP*. Springer, 1997, pp. 55–69.

[10] Y. Matsuno, H. Takamura, and Y. Ishikawa, "A dependability case editor with pattern library," in *Proceedings of 12th Conference on High Assurance Systems Engineering Symposium - HASE*. IEEE Computer Society, 2010, pp. 170–171.

[11] J. A. McDermid, "Safety and dependability," in *Dependable Software Systems Engineering*, ser. NATO Science for Peace and Security Series, D: Information and Communication Security. IOS Press, 2015, vol. 40, pp. 128–169.

[12] L. Duan, S. Rayadurgam, M. P. E. Heimdahl, A. Ayoub, O. Sokolsky, and I. Lee, "Reasoning about confidence and uncertainty in assurance cases: A survey," in *Software Engineering in Health Care*, M. Huhn and L. Williams, Eds. Springer, 2017, pp. 64–80.

[13] R. Wang, J. Guiochet, and G. Motet, "Confidence assessment framework for safety arguments," in *Proceedings of 36th International Conference on Computer Safety, Reliability, and Security - SAFECOMP*, ser. Lecture Notes in Computer Science, vol. 10489. Springer, 2017, pp. 55–68.

[14] A. P. Dempster, "Upper and lower probabilities induced by a multivalued mapping," 1967.

[15] A. Jøsang, "A logic for uncertain probabilities," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 9, no. 3, pp. 279–212, 2001.

[16] M. Huhn and A. Zechner, "Analysing dependability case arguments using quality models," in *Proceedings of 28th International Conference on Computer Safety, Reliability, and Security - SAFECOMP*, ser. Lecture Notes in Computer Science, vol. 5775. Springer, 2009, pp. 118–131.

[17] M. R. Barry, "Certware: A workbench for safety case production and analysis," in *Proceedings of Aerospace Conference*. IEEE Computer Society, 2011, pp. 1–10.

[18] E. Denney and G. Pai, "Tool support for assurance case development," *Automated Software Engineering*, vol. 25, no. 3, pp. 435–499, 2018.

[19] N. L. S. Fung, S. Kokaly, A. D. Sandro, R. Salay, and M. Chechik, "MMINT-A: A tool for automated change impact assessment on assurance cases," in *Proceedings of 37th International Conference on Computer Safety, Reliability, and Security - SAFECOMP Workshops*, ser. Lecture Notes in Computer Science, vol. 11094. Springer, 2018, pp. 60–70.

[20] A. Gacek, J. Backes, D. D. Cofer, K. Slind, and M. Whalen, "Resolute: an assurance case language for architecture models," in *Proceedings of the 2014 ACM SIGAda annual conference on High integrity language technology - HILT*. ACM, 2014, pp. 19–28.

[21] X. Larrucea, A. Walker, and R. C. Palacios, "Supporting the management of reusable automotive software," *IEEE Software*, vol. 34, no. 3, pp. 40–47, 2017.

[22] A. Retouniotis, Y. Papadopoulos, I. Sorokos, D. Parker, N. Matragkas, and S. Sharvia, "Model-connected safety cases," in *Proceedings of 5th International Symposium for Model-Based Safety and Assessment - IMBSA*, ser. Lecture Notes in Computer Science, vol. 10437. Springer, 2017, pp. 50–63.

[23] D. Ratiu, M. Zeller, and L. Killian, "Safety.lab: Model-based domain specific tooling for safety argumentation," in *Proceedings of 34th International Conference on Computer Safety, Reliability, and Security - SAFECOMP Workshops*, ser. Lecture Notes in Computer Science, vol. 9338. Springer, 2015, pp. 72–82.

[24] I. Sljivo, B. Gallina, J. Carlson, H. Hansson, and S. Puri, "Tool-supported safety-relevant component reuse: From specification to argumentation," in *Proceedings of 23rd Ada-Europe International Conference on Reliable Software Technologies*, ser. Lecture Notes in Computer Science, vol. 10873. Springer, 2018, pp. 19–33.

[25] S. Björnander, R. Land, P. Graydon, K. Lundqvist, and P. Conmy, "A method to formally evaluate safety case arguments against a system architecture model," in *2nd edition of the IEEE Workshop on Software Certification - WoSoCER*. IEEE Computer Society, 2012.

[26] S. Cruanes, G. Hamon, S. Owre, and N. Shankar, "Tool integration with the evidential tool bus," in *Proceedings of 14th International Conference of Model Checking, and Abstract Interpretation - VMCAI*, ser. Lecture Notes in Computer Science, vol. 7737. Springer, 2013, pp. 275–294.

[27] J. Górski, A. Jarzebowicz, J. Miler, M. Witkowicz, J. Czyznikiewicz, and P. Jar, "Supporting assurance by evidence-based argument services," in *Proceedings of 31st International Conference on Computer Safety, Reliability, and Security - SAFECOMP Workshops*, ser. Lecture Notes in Computer Science, vol. 7613. Springer, 2012, pp. 417–426.

[28] S. Nair, N. Walkinshaw, T. Kelly, and J. L. de la Vara, "An evidential reasoning approach for assessing confidence in safety evidence," in *Proceedings of 26th IEEE International Symposium on Software Reliability Engineering - ISSRE*. IEEE Computer Society, 2015, pp. 541–552.