The Use of Controlled Vocabularies and Structured Expressions in the Assurance of CPS

Katrina Attwood[1], Philippa Conmy[2] and Tim Kelly[1]

[1] Department of Computer Science, University of York, Deramore Lane, YORK YO10 5GH, United Kingdom; Tel +44 1904 325460; email: {katrina.attwood/tim.kelly}@york.ac.uk

[2] Rapita Systems Ltd, Atlas House, Osbaldwick Link Road, YORK YO10 3JB, United Kingdom; Tel +44 1904 413945; email: pconmy@rapitasystems.com

Abstract

To date, work on the development of assurance cases has largely been concerned with the broad structure and content of arguments to contextualise the data. However, at a more detailed level, use of natural language in an argument can lead to conflicting terminology, to difficulties in understanding the nature of the claims being made or to logical inferences which are obscure to the readers of the argument. This problem has become increasingly complex as more and more suppliers are involved in the development chain, making it more difficult to evaluate the strengths and weaknesses of assurance data or to re-use it. This paper explores the development of controlled vocabulary and structured expressions for CPS in the automotive domain, using the Semantics of Business Vocabulary and Business Rules (SBVR) to improve communication and to provide presents some formal consistency checking of content. We highlight the challenges this work has exposed.

Keywords: safety, assurance, controlled language, SBVR, automotive.

1 Introduction

The presentation of assurance cases is now standard practice in a number of safety-critical domains and is mandatory in several. Assurance cases typically comprise both reasoned arguments justifying claims relating to the safety, integrity and/or dependability of CPS and a variety of supporting evidence – analysis and test data, design information and process documentation. Although a considerable body of literature regarding safety-case praxis has been produced, the primary focus to date has been to provide guidance on the structure and content of the arguments, with relatively little attention paid to the language used to convey them. Graphical notations developed for the safety assurance domains (for example, the Goal Structuring Notation (GSN) [1] and the Claims-Argument-Evidence method [2]) inevitably foreground – and simplify – issues of logical flow and the overall readability of the argument, but provide limited guidance on how assertions and supporting statements should be phrased to ensure that the argument is correctly conveyed to a reader or assessor. In the GSN Community Standard [1], for example, less than 10% of the document is devoted to language issues as opposed to the definition, graphical representation, construction and review of argument structures. In practice, many assurance cases are not documented using graphical notations, but use either natural language alone or a combination of natural language and graphical notation for summary purposes.

Imprecise phrasing in assurance cases can lead to a number of problems, including:

- ***Inconsistency*** – terms may be used with different meanings at different points across an argument. This may lead to uncertainties in interpretation, particularly in the subjects of claims and assertions and the scope within which they are valid.
- ***Vagueness*** – without a precise definition of terminology, the author's intended meaning may not be properly conveyed to the audience, whether because there is no shared understanding of the terms used or because there is a failure to 'pin things down' adequately.
- ***Lack of focus in claims*** – in freeform text, it can be difficult to 'unravel' sentence structure so as to establish the scope of terms, i.e. how they influence other terms beyond the single phrasal structure in which they occur [3]. It can therefore be difficult to identify the claims the argument is making, since the relationships between the elements under discussion may not be made clear.

CPS are increasingly assembled by integrator organisations, using multiple components from a diffuse, multinational supply-chain [4]. Compositional approaches to certification mean that assurance data relating to discrete components need to be collected and matched to form an integrated system argument. There is a clear need for consistent usage of domain- and system-specific terminology throughout the supply-chain, and for a shared understanding of the nature and limitations of the claims and evidence being presented in the argument, and of the assumptions made about the operational context in which component behaviour is guaranteed.

We believe there is scope to use controlled language to provide more rigorous rhetorical structure in assurance cases for CPS. We propose a dual approach to address the problems of inconsistency and imprecision outlined above. First, we address semantic aspects by developing a domain dictionary, which provides unambiguous definitions of relevant concepts in the domain over which the argument ranges. Secondly, syntactic aspects are addressed by these definitions to specify claim types in the form of structured expressions to clarify the argument logic. The OMG's Semantics of Business Vocabulary and Business Rules (SBVR) specification [5] offers one means to implement this approach. SBVR provides for the formalized definition of domain concepts, together with the rules and assumptions which define the relationships between them. It contains an explicit model of formal logic, and thus provides a means for the capture of natural language expressions in a formal structure, suitable for machine-processing.

Two of the elements defined in SBVR are of particular significance for our approach: 'concepts' and 'fact types'. These form the basis for the development of the controlled lexicon and claim typology described in the two following sections.

2 Argument semantics: development of a controlled lexicon for safety assurance

In SBVR, a 'concept' is defined as "a unit of knowledge created by a unique combination of characteristics" [5]. Generally, this equates to a noun, or a noun-phrase (also referred to as a 'term'). In SBVR, concepts can be defined formally or informally. In a formal definition, each of the concepts referred to must be defined elsewhere in the vocabulary, thus making for a closed lexicon. Reserved terms to represent logical relationships between concepts are defined in [5]. The "General Concept" and "Concept Type" attributes can be used to specify hierarchical type-relationships between concepts. This is especially useful in the disambiguation of terminological mismatches in cross-domain "translation" scenarios, such as the comparison of concepts across different safety standards.

Our work in the OPENCOSS project [6] defined a preliminary SBVR vocabulary of concepts for assurance arguments. As in the SBVR specification [5], a graphical summary of concept relationships is provided for ease of reference (for human readers). The vocabulary provides a controlled language definition of concepts, artefacts and processes used in the domains of interest of OPENCOSS (railway, avionics and automotive), and thus provides a basis for comparison of usage between the domains. We do not seek to develop a unified, universal lexicon for assurance to be used across the target domains. Such an enterprise is fraught with difficulty, since the certification approaches differ fundamentally. As an illustration, consider the difficulties for a manufacturer seeking to reuse software developed according to IEC 61508 [7] in an avionics context, where certification to DO-178B is required [8]. An assurance argument in the original context – here expressed using SBVR, for clarity – might assert that "software module_Y *is developed to* safety integrity level SIL 4". In the avionics context, the manufacturer may wish to make a similar claim: "software component Y *is developed to* design assurance level DAL A". Since both the safety integrity level and the design assurance level are instantiations of the generic SBVR concept "Criticality Level" defined by OPENCOSS, it might be assumed that a direct 'translation' between the claims is possible. Examination of the diagrams summarizing the concept relationships for system and software architectures extracted from the SBVR vocabulary we have developed for the relevant standards, however, reveals that the situation is more complicated.

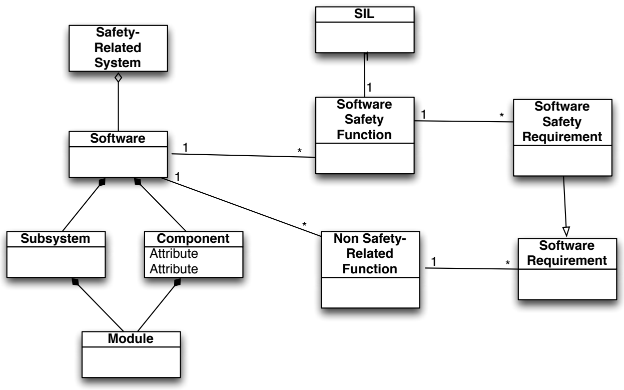


Figure 1 IEC 61508 software concept relationships

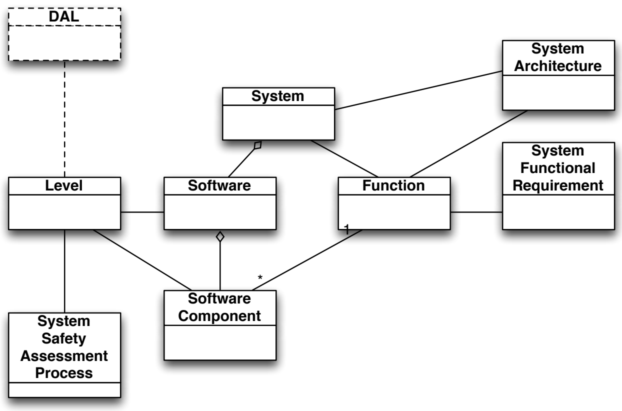


Figure 2 DO-178B software concept relationships

In IEC 61508, a SIL is directly attached to a (software) safety function which is modelled at system level. In DO-178B, however, the DAL is associated with a software system or component, and does not address the "function" concept at all. This implies that direct 'translation' of the claim cannot be made – it is not possible to convert a SIL directly into a DAL without considering the extra process-related concepts that arise because of the focus in DO-178B on the design of the system, rather than merely its functionality. Although a clear understanding of the terminology can be helpful in addressing this difficulty, what is required is not a definition of individual concepts in isolation, but an appreciation of the interrelationships between the concepts, since these provide constraints on reuse of the claim – and associated assurance data – here. Use of a closed SBVR vocabulary will ensure that these interrelationships are correctly identified. We should therefore consider there to be either a “partial map” or a “no map” relation between the concepts, and a full explanation of the discrepancies between the conceptual structure of the standards is required in order for an engineer to make informed decisions about the feasibility of or limitations on reuse, and on what extra assurance data may need to be provided in the DO-178B context.

A primary concern for the OPENCOSS project is to support reasoning about whether certification artefacts, such as analysis results, can be reused across domains and from one development project to another. In order to support this, the OPENCOSS vocabulary defines terminology at three levels of abstraction: we define vocabulary models to capture the generic vocabulary of safety standards relevant to the domains, organisation-specific terminology and project-specific terminology. Mapping relationships between concepts are used to capture traceability relationships between generic and system-specific concepts (e.g. the fact that a project-specific test plan is an instance of the test plan defined in the organisational model) and also to indicate the degree of “mapping” between concepts at the various levels (e.g. the degree to which the organisational definition of a test plan matches the characteristics of the generic artefact defined in the standard model and relating to a requirement of the standard).

The demonstration of assurance is a much wider and more complex concern than simply establishing conformance to a standard; and an argument is much more than a compliance checklist of processes and artefacts. Having clear definitions of terminology in which concepts are related both vertically by type and sub-type relations and horizontally by being defined in terms of one another in a closed lexicon can help in ensuring consistency of reference across assurance case modules. In particular, the terminology can be used to characterise the interfaces and interdependencies between argument modules, and to ensure that the terms of reference here are consistently understood. The layered vocabulary defined for the OPENCOSS project allows us to clarify the relationships between standards, industrial praxis and development projects, using the “mapping” relationships between concepts at the various levels of abstraction to make any gaps between standards’ requirements and projects’ actualities clear.

3 Argument semantics: structured claim types

One important means of maintaining consistency in the natural language used to convey the reasoning in an assurance argument is to specify types of claims. A taxonomy of claims can be superimposed on the general concerns of an argument structure identified in the literature (e.g. [9]) and can then be used to refine the logical structures provided in the argument fragment templates captured in GSN patterns such as those presented in [10]. The claim types characterise the types of concepts which are discussed in a particular part of the argument, and the features which are asserted in claims. We have identified several generic claim types for assurance arguments, as summarised in Table 1:

Claim Type	**Definition**
Activity-Artefact Claim	Claim relating to the production of particular artefacts as a result of particular safety analysis or development activities.
Artefact Compliance Claim	Claim relating to the presentation of a particular artefact necessary for compliance.
Artefact Adequacy Claim	Claim relating to the adequacy and appropriateness of a particular artefact, i.e. moving beyond compliance to a justification of the evidence artefacts provided. E.g., the adequacy of a fault tree
Activity Compliance Claim	Claim relating to the presence and features of features of a safety analysis or development activity necessary for compliance
Activity Adequcy Claim	Claim relating to the adequacy and appropriateness of a particular safety analysis or development activity
Component Development Claim	Claim relating to the adequacy and acceptability of the process by which a component has been developed
Fault Accommodation Claim	Claim relating to the accommodation or elimination of a fault
Hazard Mitigation Claim	Claim relating to the adequacy of hazard mitigation achieved by safety measures in the design

Table 1: Generic claim types for assurance

We can exploit the layered structure of the OPENCOSS vocabulary – where concepts are defined and “mapped” at the level of the standard, the industry model and the project – by defining domain-specific versions of these claim types in parameterised phrases used to populate the GSN argument patterns. These phrases can then be instantiated in component- or system-specific arguments using vocabulary relevant to that component derived from the project vocabulary model. The “Concept Type” mechanism in SBVR allows for the presentation of a series of potential instantiations of a given parameter from which the user can choose. In some cases, the “fact Type” mechanism in SBVR allows to generate the domain-specific claim type directly from the standard or industry vocabulary model.

The “Fact Type” in SBVR [5] is used to capture relationships between concepts defined in the vocabulary. A fact type is defined in [5] as “the meaning of a verb phrase that involves one or more nouns, whose instances are all actualities”. A fact type thus equates to a proposition ranging over the concepts represented by the nouns or noun-phrases, a statement of some relationship which can be evaluated logically as having a truth value. As with concepts, fact types can be defined formally – by means of a closed expression in which every term is defined elsewhere in the SBVR model – or informally, using terminology which is not controlled.

In some cases, the “fact type” mechanism in SBVR allows us to generate the domain-specific claim type, and the mapping between the standard (or industry) vocabulary and the project vocabulary provides possible terms with which the template phrase can be instantiated. For claims of the Activity-Artefact type, for example, the SBVR vocabulary derived from the terminology used in the safety standard should identify the types of concept over which the claim might range, by identifying relationships between particular activities and the artefacts they generate. A generic fact type of the sort artefact *is generated by* activity, for example, can be instantiated by traversing the SBVR "Concept Type" and "General Concept" fields in the standard-level vocabulary to identify a series of individual concepts of type "artefact" and type "activity"/ The list of possible concepts might be further reduced by pre- and post-conditions relating to the individual "artefact" and "activity" concepts identified in the project-level model, to present the argument developer with a list of candidate terms with which to instantiate the fact types reflecting the practice of the project. More complex fact types might be devised – around the basic claim structures – to reflect complex dependencies between activities.

4 Example

In this section, we present a simple example to illustrate the ways in which structured expressions using controlled vocabulary can be exploited to instantiate claims in an assurance argument. The example is based on a simplified, fictitious automotive anti-lock braking system (ABS), which is developed to ISO 26262 [11]. Correct operation of the ABS allows the wheels to maintain contact with the road surface during hard braking, preventing the wheels from locking and avoiding an uncontrolled skid. The system comprises a software controller, four wheel sensors (one for each wheel) and two hydraulic valves (one for each axel). The system has two basic operational scenarios. The software constantly monitors the speed at which the wheels rotate, measures via the wheel sensors. If it detects that one wheel is rotating at a slower speed than the others, the controller actuates the hydraulic valves to reduce hydraulic pressure to the brake, thus reducing braking force on that wheel and allowing it to turn faster. Alternatively, if the software detects that one wheel is turning significantly faster than the others, the valves are operated to increase hydraulic pressure to that wheel, thus increasing braking force to that wheel and slowing down its rotation. The software controller contains a critical function to calculate the hydraulic pressure demand value from the wheel speed sensor inputs. Failure of this function results in the incorrect braking force being applied to the wheel, which could result in a skid.

The assurance argument for the ABS software controller clearly needs to address the issue of potential faults in the hydraulic pressure demand calculation function. In this example, that issue will be addressed as part of a top-down argument concerning the mitigation of the "uncontrolled skid" hazard by the software. An argument of this type can be structured using the approach suggested in the high-level software safety argument pattern in [10], which is presented in Figure 3, using the GSN [1]:

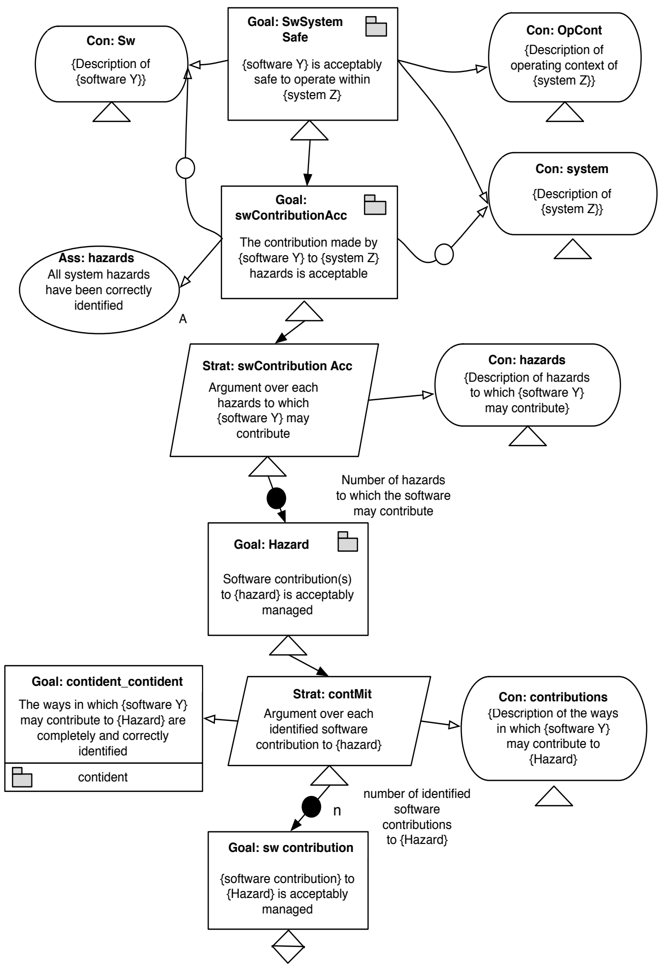

Con: Sw
{Description of {software Y}}
Goal: SwSystem Safe
{software Y} is acceptably safe to operate within {system Z}
Con: OpCont
{Description of operating context of {system Z}}
Con: system
{Description of {system Z}}
Goal: swContributionAcc
The contribution made by {software Y} to {system Z} hazards is acceptable
Ass: hazards
All system hazards have been correctly identified
A
Strat: swContribution Acc
Argument over each hazards to which {software Y} may contribute
Con: hazards
{Description of hazards to which {software Y} may contribute}
Number of hazards to which the software may contribute
Goal: Hazard
Software contribution(s) to {hazard} is acceptably managed
Goal: contident_contident
The ways in which {software Y} may contribute to {Hazard} are completely and correctly identified
contident
Strat: contMit
Argument over each identified software contribution to {hazard}
Con: contributions
{Description of the ways in which {software Y} may contribute to {Hazard}
number of identified software contributions to {Hazard}
n
Goal: sw contribution
{software contribution} to {Hazard} is acceptably managed

Figure 3 High-Level software safety argument pattern (from [10])

In the diagram, the rectangular boxes represent claims made about the software (these are called "Goals" in GSN). The top-level goal (Goal: SWSystemSafe) contains an overall claim that the software is acceptably safe to operate within the system in which it is located ({system Z}). The rounded rectangles attached by hollow arrows to this goal contain contextual statements required to further explain and validate the goal. Here, they refer to supporting documentation which provides descriptions of relevant aspects of the software design and the design and operational environment of {system Z}. The triangles underneath items indicate that textual information within curly braces requires instantiation in an argument relating to a real system. Goal:SWSystemSafe is refines into a lower-level claim (captured in Goal: swContributionAcc), which indicates that the argument will be made by considering the possible contributions that {software Y} could make to system-level hazards. The oval (Ass:hazards) represents an assumption on which this argument relies: in this case, that all of the system hazards have been identified correctly. The parallelogram (Strat:swContributionAcc) represents the strategy used to break down this general claim into more detailed ones. Here, the argument is structured by taking each of the system-level hazards to which the software may contribute in turn, and arguing that the software contribution to each has been managed. This strategy is realised in the statement of Goal:Hazard, which makes the claim that the software's contribution to a particular hazard ({Hazard}) is acceptably mitigated. An enumeration of the relevant hazards is provided as context to this argument, and is referred to in the GSN Context (Con:hazards). The solid circle on the decomposition arrow between Strat:swContributionAcc and Goal:Hazard indicates that Goal:Hazard and the subsequent argument is iterated for each of the hazards to which the software might contribute. Where a safety requirement exists which relates to the software's role in {Hazard}, this is explicitly stated, and referred to in the context Con:safetyRqt. Since software might contribute to the occurrence and effects of hazards in a number of different ways (depending on the nature of the hazard, the software and the system context), a further strategy (Strat:contMit) is applied, by which the claim concerning the safe management of these software contributions (captured in Goal:swContribution) is made and argued through for each potential contribution. This line of argument is made in the context of an enumeration of the potential contributions the software could make to the hazard (referred to in Con:contributions). Further confidence in the adequacy of the argument at this point is provided in a backing argument, which supports a claim that the list of potential software contributions to the hazard is complete and correct. This argument is made in a separate GSN module (contident), the structure of which is not outlined in full here. Goal:contident_contident provides a reference to the topmost claim in that backing argument, and serves to direct the reader's attention to the argument and evidence provided in the contident module.

Our discussion of the use of the SBVR vocabulary and claim types to develop and instantiate an argument draws on the lower part of the pattern in Figure 3, the claim in Goal:Hazard that the software's contribution to a particular Hazard is adequately managed and the subsequent argument addressing each potential way in which the software could contribute to the hazard.

The example requires two distinct SBVR vocabularies. Firstly, the ABS system is represented in a vocabulary, terms in which are drawn from the organisational vocabulary for the system as a whole. Concepts in this vocabulary serve to define concepts in the deployment context of the ABS software. The ABS software is also represented by a dedicated, project-level, vocabulary.

Figure 4 contains a restatement of the argument structure, which represents a partial instantiation of the template pattern presented in Figure 3, as an assurance argument for the ABS software. Here, Goal G1 represents an instantiation of Goal:Hazard in Fig. 3. Contexts C1 and C2 and G:backing_top are also instantiations of the parallel elements in the GSN pattern. The underlined terms here ("ABS software", "uncontrolled skid hazard", "safety requirement 123", "fault tree analysis") are instances of the more generic concept types used in Fig. 3, and are taken from the SBVR vocabulary for the ABS system (populated from project documents at the system level, such as system descriptions, requirements documents, system safety analysis).

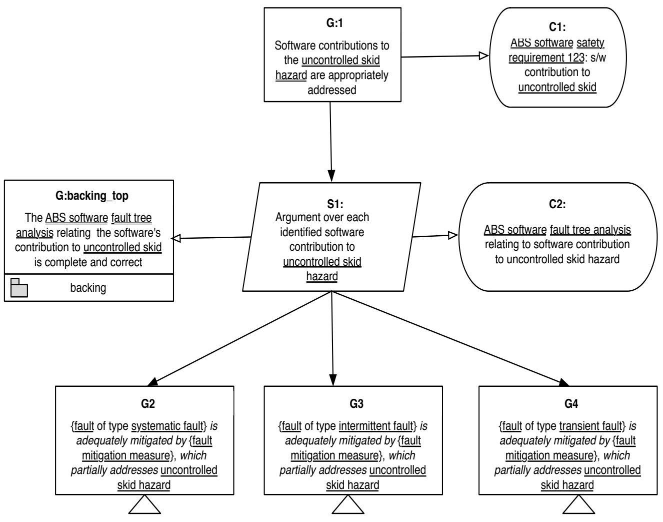
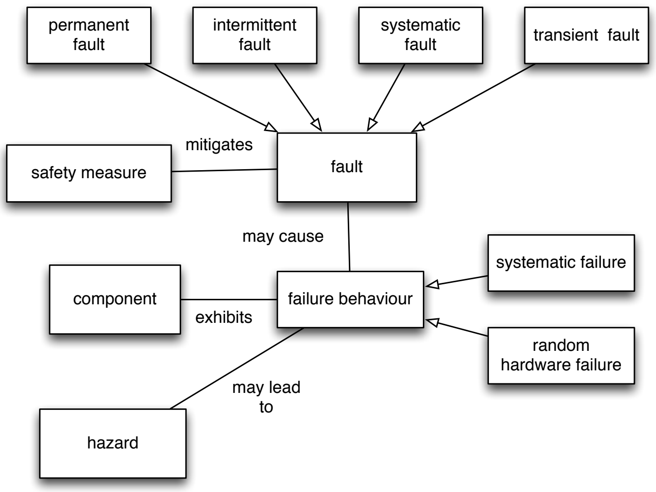

G:1
Software contributions to the uncontrolled skid hazard are appropriately addressed
C1:
ABS software safety requirement 123: s/w contribution to uncontrolled skid
G:backing_top
The ABS software fault tree analysis relating the software's contribution to uncontrolled skid is complete and correct
backing
S1:
Argument over each identified software contribution to uncontrolled skid hazard
C2:
ABS software fault tree analysis relating to software contribution to uncontrolled skid hazard
G2
{fault of type systematic fault} is adequately mitigated by {fault mitigation measure}, which partially addresses uncontrolled skid hazard
G3
{fault of type intermittent fault} is adequately mitigated by {fault mitigation measure}, which partially addresses uncontrolled skid hazard
G4
{fault of type transient fault} is adequately mitigated by {fault mitigation measure}, which partially addresses uncontrolled skid hazard

Figure 4 Restatement of lower portion of software safety argument pattern, indicating claim types

The claims captured in the statements in Goals G2, G3 and G4 represent standard-level representations of the generic claim types "Fault Accommodation Claim" and "Hazard Mitigation Claim" identified in Table 1 above. Here, they are parameterized with generic noun types drawn from the SBVR vocabulary for ISO 26262. These claims have an underlying conceptual model, which derives from ISO 26262, and relates a typology of faults to fault mitigation measures and characterises the relationship between faults and hazards[1]. This model, and the SBVR definitions for the concepts it identifies, are presented in Figure 5:



SBVR Concept Definitions

fault
Definition: abnormal condition that can cause failure of an element or an item
Dictionary Basis: ISO 26262 Part 1,§1.42 (adapted)
Possibility: fault *causes* at least one failure

[1] Note that ISO 26262 [11] identifies an additional subtype of fault, the concept "permanent fault". This concept requires a claim of a different type from those used to handle the other fault types, and it will be more difficult to make those claims generic. In order to simplify the discussion here and focus on the use of SBVR to populate generic claims, we have excluded "permanent fault" from the illustrative example here.

permanent fault
Definition: fault which occurs and then stays until removed or repaired
Dictionary Basis: ISO 26262 Part 1,§1.88
General Concept: fault

intermittent fault
Definition: a fault which occurs repeatedly and then disappears
Source: ISO 26262 Part 1,§1.42
Dictionary Basis: ISO 26262 Part 1,§1.42 note 2
General Concept: fault

systematic fault
Definition: fault which causes a failure which is manifested in a deterministic way and which can only be prevented by applying process or design measures
Source : ISO 26262 Part 1,§1.42 (adapted)
Dictionary Basis: : ISO 26262 Part 1,§1.131 (adapted)
General Concept: fault

safety measure
Definition: activity or technical solution put in place to avoid or control systematic failures and to detect or control random hardware failures or to mitigate effects of such failures which may lead to harm
Dictionary Basis: ISO 26262 Part 1 §1.110
Necessity: safety measure *includes* safety mechanism
safety measure *is specified in* functional safety requirement
Example: definition of software without the use of global variables
Synonym: means; control

failure behaviour
Definition: termination of an element's ability to perform a function as required or intended
Dictionary Basis: ISO 26262 Part 1,§1.39 (adapted)

systematic failure
Definition: failure which can be attributed deterministically to a certain cause, and which can be eliminated only by a change to the design or manufacturing process, to operational procedures, to documentation or to organisational factors
Dictionary Basis: ISO 26262 Part 1,§1.130 (adapted)
General Concept: failure
Necessity: systematic failure *is caused by* systematic fault

random hardware failure
Definition: failure that may occur unpredictably during the lifetime of a hardware element, according to some probability distribution
Dictionary Basis: ISO 26262 Part 1,§1.92
General Concept: failure
Necessity: random hardware failure *has* probability

component
Definition: element defined at an abstraction level below that of "the system", that is logically and technically separable and is comprised of more than one hardware part or of one or more software units
Source: ISO 26262 Part 3, §1
Dictionary Basis: ISO 26262 Part 1,§1.15
General Concept: element
Necessity: a component must *contain* at least one hardware part or a component must *contain* at least one software unit

hazard
Definition: potential source of harm caused by malfunctioning behaviour
of an item
Dictionary Basis: ISO 26262 Part 1,§1.56

Fact Types
fault *causes* at least one failure behaviour

failure behaviour *may lead to* hazard

systematic fault *may cause* systematic failure

safety measure *mitigates* fault

systematic failure *is caused by* systematic fault

random hardware failure *has* probability

component *exhibits* failure behaviour

hazard *has* cause

hazard *may be caused by* failure behaviour *which is exhibited by* component

hazard *has* effect

Figure 5: Conceptual model and SBVR definitions underlying the claim types defined in Figure 4

It will be clear that the first part of the claims in Goals G2, G3 and G4 have been derived straightforwardly from the conceptual model – they assert the relationship which is modelled between the "fault" and "safety measure" concepts, captured in the fact type safety measure *mitigates* fault. Note, however, that the claim generation is not automatic – understanding of the concepts of assurance and argumentation are required to lead to the concept of adequacy in association with fault mitigation, and thus to make the claim subjective (as the argument requires). The second part of the claim is not generated directly from a fact type or relationship, since there is no direct link in the conceptual model between the concepts of fault mitigation and the hazard. Instead, the relationship is obtained by traversing the contextual relationships between "fault", "failure behaviour" and "hazard". In order to produce an argument for a specific ABS system, the claim types captured in goals G2, G3 and G4 are instantiated by populating the parameterized noun phrases with concepts of appropriate types from the SBVR vocabulary defined for the specific ABS system – the project-level model. Figure 6 presents a partial instantiation of Goal G2:

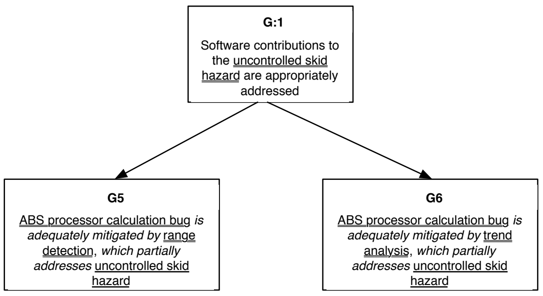

G:1
Software contributions to the uncontrolled skid hazard are appropriately addressed
G5
ABS processor calculation bug *is adequately mitigated by* range detection, *which partially addresses* uncontrolled skid hazard
G6
ABS processor calculation bug *is adequately mitigated by* trend analysis, *which partially addresses* uncontrolled skid hazard

Figure 6 Partial instantiation of claim type, using project-specific vocabulary

Here, Goal G2 from Figure 4 has been instantiated twice, populated using instances of the "systematic fault" and "fault mitigation measures' (a synonym for "safety measure") from the SBVR vocabulary for the actual ABS system (the project-level model). Note that the intention here is to show the population of the generic claim type using concrete instances from the vocabulary, rather than to present a complete argument. As it stands, the GSN fragment presented in Figure 6 suggests that the two goals G5 and G6, taken together, provide a sufficient argument that G1 holds in the context. Given the richness of the argument structure provided in Figure 4, this is clearly untrue: further instantiation of Goals G2, G3 and G4 are required to ensure adequate coverage of Goal G1. For simplicity, these additional goals (which can be instantiated from the SBVR vocabulary for the ABS system as G2 has been here) are not shown.

5 Related Work

There is only a limited amount of research which directly addresses the integration of controlled language approaches in the field of assurance argumentation. A methodology for argument development is presented in [12], which exploits the structural patterns presented in [10]. Generic patterns to help form software assurance arguments are also provided in, for example, [13], [14], [15] and [16]. Such patterns focus on the structure of the arguments and the issues they should address, rather than their phraseology or rhetoric and since they are by definition generic, it can be difficult to achieve consistency and completeness in the resulting argument instantiations. In none of these cases is explicit attention paid to the possible application of controlled natural language to enforce the patterns and assist the argument developer in making the reasoning clearer. The standard industry guidance on the development of GSN arguments [1] contains some general advice about sentence structure and a discussion of common language-based errors. These errors are identified at the level of the whole claim, rather than individual terms or phrases.

The OMG's *Structured Assurance Case Metamodel* [17] provides a metamodel of argumentation, including language aspects, and a discussion of the use of SBVR to realise assurance arguments. The technique described is, however, overly simplistic and is not fully realised in [17]: the present paper should be seen as part of an ongoing debate as to the utility of SBVR in the assurance argumentation field.

The authors of [18] define a restricted language to describe rely-guarantee conditions between software applications and computer hardware. Although this language can be used in the automated generation of a limited set of arguments concerning compositional behaviour of software elements, including failure behaviour, it is very limited in scope, and does not capture additional required information such as data concerning evidence supporting rely-guarantee claims or the degree of confidence which can be placed in them.

The OPENCOSS project [19] aims to develop technologies to support the cost-effective reuse of assurance information within and between safety-critical domains. Assurance arguments are used as the basis for communication of this information, and to support certification. This approach relies on the ability to communicate and compare relevant concepts across and within organisations and domains. However, there is no consistent conceptualisation and terminology to describe and manage assurance, let alone a "common certification approach" recognised by system integrators, the supply chain and assessors. OPENCOSS seeks to provide a basis for communication by developing a pragmatic approach, which identifies commonality and differences between the ways in which safety, assurance and certification are conceived, and provides means to compare them. The project has developed models of assurance assets, information, processes and concepts in safety standards, organisational practices and individual projects, using a generic metamodel of relevant concepts for safety assurance [6]. These models are supported by domain- and company-specific vocabularies which provide clear, controlled definitions of concepts which need to be addressed in safety arguments. A mapping technique is used to define relationships between concepts in both the models and the vocabulary at varying degrees of exactness, and tool support is provided to support engineers in making explicit the significant differences which need to be discussed in a justification of reuse.

Structured approaches to language are widely used in the requirements engineering domain. For example, the Attempto Controlled English (ACE) defines a structured natural language to support engineers in writing precise specifications which can be translated into semi-formal representations suitable for machine-checking [20]. Similarly, Denger et al [21] have identified natural language patterns to specify functional requirements for embedded systems. The CIRCE project [22] adopted model-based techniques to support the validation of natural language requirements, based on a lightweight formal model. In the safety-critical domain, the CLEAR methodology developed by the Dependability Research group at the University of Virginia uses insights from linguistics and cognitive psychology concerning the nature of linguistic error and presents a pattern-based technique to minimise miscommunication in requirements [23]. None of these methods explicitly address the issues relating to structured argumentation for assurance – for example, inherent subjectivity in claims -, although the relationship between requirements and argument claims appears to provide an interesting avenue for future research.

6 Conclusion

This paper has demonstrated the potential use of SBVR concept definitions and fact types to add rigour to the language used to convey assurance arguments for safety-critical CPS. We have described the use of a layered vocabulary and "mapping" to capture traceability relationships between concepts defined in safety standards, in organisation-specific practices and conventions and in individual projects, and have indicated how the mapping notion can be used to provide informed guidance on the transferability of concepts and the reusability of assurance assets between projects and across domains. Furthermore, we have provided an initial taxonomy of structured claim types, partially derivable from SBVR fact types, and have demonstrated how they can be used to constrain the language and focus of assurance arguments. Work to develop this method and to provide tooling is currently at an early stage. Theoretical work remains to be done to expand the taxonomy of claim types and refine their phrasing. There is also a need to explore the relationship between declarative fact types, requirements and argument claims more fully, in particular to find ways to address the subjective elements of claims in a formal or semi-formal lexicon for argumentation.

Acknowledgement

The work presented here was carried out as part of the OPENCOSS Project, No: 289011, funded by the European Commission under the FP7-ICR Framework. For further details, see the Project website: http://www.opencoss-project.eu

References

[1] *Goal Structuring Notation Community Standard*, Issue 1 (November 2011), Available for download from http://www.goalstructuringnotation.info

[2] http://www.adelard.com/asce/choosing-asce/cae.html

[3] E. Lapore (2009), *Meaning and Argument: an introduction to logic through language*, Second Edition (First Edition 2000), John Wiley and Sons.

[4] K. Attwood and P. Conmy (2013), *Nuanced term-matching to assist in compositional safety assurance, First International Workshop on Assurance Cases for Software-Intensive Systems (ASSURE 2013)*

[5] Object Modelling Group (2008), *Semantics of Business Vocabulary and Business Rules*, Version 1. Available for download at http://www.omg.org/spec/SBVR/1.0/

[6] OPENCOSS Consortium (2013), *Common Certification Language: Conceptual Model* (Version 1), Project deliverable D4.4. Available for download at http://www.opencoss-project.eu

[7] IEC (2009), *IEC 61508: International Standard – Functional safety of electrical/ electronic/ programmable electronic safety-related systems*

[8] RTCA (1992), *RTCA/DO-178B: Software considerations in airborne systems and equipment certification*

[9] R. Hawkins, T. Kelly, J. Knight and P. Graydon (2011), *A new approach for creating clear safety arguments,* in C. Dale and T. Anderson (eds) *Advances in Systems Safety: Safety-Critical Systems Symposium (SSS 11)*, Springer-Verlag, pp 3-24

[10] R. Hawkins and T. Kelly (2013), *A Software Safety Argument Pattern Catalogue*, University of York Department of Computer Science Report YCS-2013-482. Available for download at ftp://ftp.cs.york.ac.uk/reports/2013/YCS/482/YCS-2013-482.pdf

[11] ISO/FDIS (2011), *ISO/FDIS 26262 International Standard – Road Vehicles, Functional Safety*

[12] R. Hawkins and T. Kelly (2010), *A systematic approach to developing software safety cases,* Journal of System Safety, vol 46 no. 4, pp 25-33

[13] T. P. Kelly (1998), *Arguing safety – a systematic approach to managing safety cases,* D.Phil Thesis, University of York

[14] R. A. Weaver (2003), *The safety of software – constructing and assuring arguments,* PhD Thesis, University of York

[15] W. Wu (2007), *Architectural reasoning for safety-critical software applications*, PhD Thesis, University of York

[16] Industrial Avionics Working Group (2012), *Modular Software Safety Case Process Description*. Available for download at https://www.amsderisc.com/p-content/uploads/2013/01/MSSC_201_Issue_01_PD_2012_11_17.pdf

[17] Object Modelling Group (2013), *Structured Assurance Case metamodel (SACM)*, Version 1. Available for download at http://www.ormg.org/spec/SACM/

[18] B. Zimmer, S. Bürklen, M. Knoop, J. Höfflinger and M. Trapp (2001), *Vertical safety interfaces – improving the efficiency of modular certification*, in U. Voges (ed), *Computer Safety, Reliability and Security SAFECOMP 2001,* LNCS 2187, Springer-Verlag, pp 29-42

[19] http://www.opencoss-project.eu

[20] N. E. Fuchs, U. Schwertel and R. Schwitter (1999), *Attempto Controlled English – not just another logic specification language* in P. Flener (ed) (1999), *8th International Workshop on Logic-Based Program Synthesis and Transformation 1999*, LNCS 1559, Springer-Verlag, pp 1-20

[21] C. Denger, D. Berry and E. Kamsties (2003), *Higher-quality requirements specifications through natural language patterns*, IEEE Conference on Software: Science, Technology and Engineering, pp 80-90

[22] V. Abriola and V. Gervasi (2006), *On the systematic analysis of natural language requirements with CIRCE*, Automated Software Engineering, vol 13 no 1, pp 107-167

[23] K. S. Hanks, J. C. Knight, E. A. Strunk and S. R. Travis (2003), *Tools supporting the clear communication of critical application domain knowledge in high-consequence systems development,* in S. Anderson, M. Felici, B. Littlewood (eds), *Computer Safety, Reliability and Security SAFECOMP 2003*, LNCS 2788, Springer-Verlag, pp 317-330