Developing an Argument for Def Stan 00-56 from Existing Qualification Evidence

Zoë Stephenson[1], Tim Kelly[1], Jean-Louis Camus[2]

1: High-Integrity Systems Engineering Group, Department of Computer Science, University of York, Heslington, York YO10 5DD, UK

2: Esterel Technologies S. A., Parc Euclide, 8 rue Blaise Pascal, 78996, Elancourt, France

Abstract: Commonly-used civil guidance and standards in the safety-critical software industry (IEC 61508, EN 50128, DO-178B) constrain development activity and generate process and product evidence. However, procurements for UK defence systems must be supported with a safety case assessed against Def Stan 00-56 Issue 4. This paper studies the use of evidence from civil guidance and standards in arguments towards DS 00-56. The approach is centred on a particular application, the KCG qualified code generator, and is based on a generic software contribution argumentation approach. The results show that issues arise in substantiating failure conditions, choosing a suitable level of detail in the argument and relating detailed explanations to the structure of the evidence. Explicit argumentation was found to be useful in addressing each of these issues.

Keywords: Certification, hazard analysis, safety case, argumentation

1. Introduction

Every industry is governed by a body of standards, and the safety-critical software industry is no different. Commonly-used standards include IEC 61508 [1], EN 50128 [2] and DO-178B [3]. These standards recommend particular activities that contribute towards the safety of a product under development. Examples of these activities include hazard analysis, failure modes and effects analysis, configuration control, traceability and test coverage analysis. Documentation of these activities and their results is submitted for independent assessment against the requirements or objectives of the standard.

When the product under development is not a complete system - such as a component, a service layer, an operating system or a development tool - it is common to assess the product against the needs of many different safety standards. The commercial benefit of this extra effort is the ability to offer the product to customers in different sectors, and the standards are similar enough that there is a great deal of overlap in the evidence produced.

For products that are used in safety-critical systems procured by the UK Ministry of Defence, assessment against Def Stan 00-56 [4] is required. This is a different style of standard to those described above; instead of recommending activities, it requires that particular goals are achieved and that the safety of the product is argued in a separate safety case.

This shift in approach means that it is not appropriate to simply reuse qualification evidence. However, matching up the structure of a compelling and defensible argument with the body of existing evidence is not straightforward. To help address this problem, we are customising a generic argument approach [5] to create guidance for users of the so-called “Esterel SCADE certification factory”, involving SCADE Suite tools, in particular the KCG qualified code generator.

Section 2 describes the operation of the KCG qualified code generator in more detail, to serve as background for the discussion. In section 3, we describe the structure of an assurance case to support the use of such tools. The experience from development of this assurance case is expanded in section 4, identifying the key issues and the strategies taken to address them, particularly in terms of the benefits of explicit argumentation. Related work is briefly studied in section 5 and conclusions are given in section 6.

2. KCG qualified code generator

The SCADE suite provides graphical and textual modelling capabilities for control algorithms, with an emphasis on well-defined, unambiguous behaviours based on formal semantics. The KCG tool converts a SCADE model into an equivalent program in the C language, ready for integration with user-supplied I/O and scheduling code. To support the assertion that the resulting object code has the same behaviour as the SCADE model, there are several Conditions of Use that the user must follow, for example:

- The model (and the underlying formal system) assumes that processing is much quicker than input/output interaction, and uses a model of instantaneous computation of outputs from inputs at each processing step. The C implementation only satisfies this assumption if the worst-case execution time of the code is less than the iteration rate of the model and the software.

- The generated C code is trusted, but the compiler must also be trusted to produce a correct output. Evidence to support this must be generated by the user for the particular tool-chain for their target platform. The code generator generates only a very limited range of C constructs, so a test suite to test those constructs is feasible and a ready-made test suite of this kind is available with the code generator.

3. Assurance case structure

The overall assurance case is structured into a number of parts, as shown in Figure 1:

- An argument that shows that the KCG tool itself was developed in an appropriate way. For example, this argument includes details of regular reviews, personnel competence and the hazard identification process.
- An argument that shows that the KCG tool correctly transforms input to output - that the resulting object code has the same behaviour as that specified in the model. This refers to the KCG development argument.
- An argument to show that the development of KCG adequately meets the objectives of DD 00-56.
- A set of custom argument patterns for users of the KCG tool. The patterns show how to link the above three arguments into a software contribution argument for the user's own software system, which in turn should be linked to the safety case for the system into which the software is ultimately integrated.

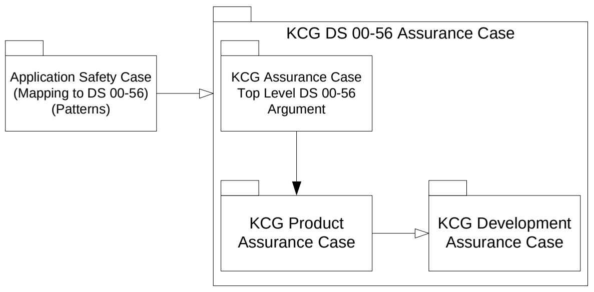
KCG DS 00-56 Assurance Case
Application Safety Case (Mapping to DS 00-56) (Patterns)
KCG Assurance Case Top Level DS 00-56 Argument
KCG Product Assurance Case
KCG Development Assurance Case

Figure 1: Argument relationships

Each part of the assurance case has its own purpose. This structure helps with development and review of the individual arguments, as well as managing the complexity of the argument.

3.1 Goal structuring

The argument was developed in a top-down fashion using the Goal Structuring Notation (GSN). GSN is a graphical notation for the representation of structured argument. The main graphical elements of the notation are shown in Figure 2. With a GSN structure, there is a top-level claim, a goal, the truth of which the writer wishes to convince the reader. The supporting elements describe the shape of the argument, which is carried in the text labels and in further discursive material. The argument eventually reaches assumptions, justifications and solutions that collectively form the evidence on which the argument is based.

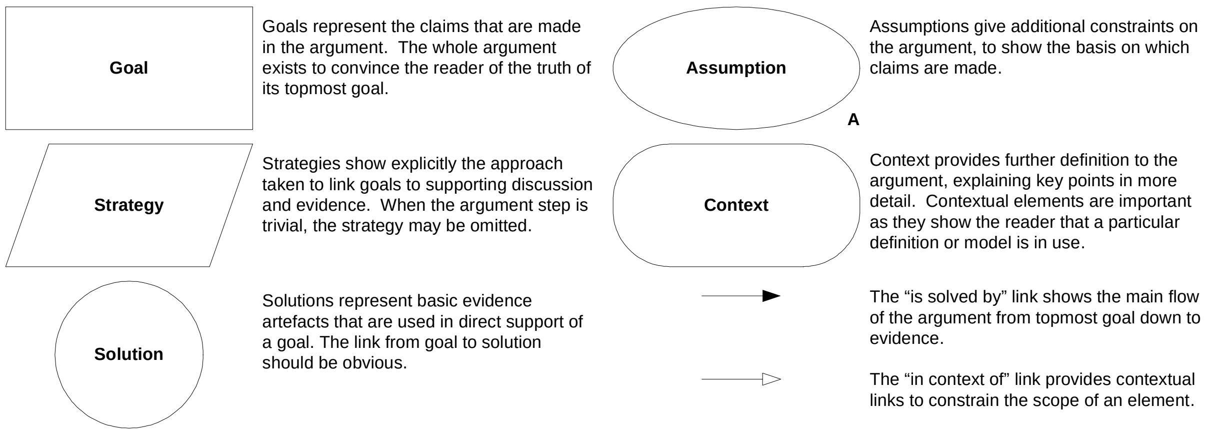
Goal
Goals represent the claims that are made in the argument. The whole argument exists to convince the reader of the truth of its topmost goal.
Assumption
A
Assumptions give additional constraints on the argument, to show the basis on which claims are made.
Strategy
Strategies show explicitly the approach taken to link goals to supporting discussion and evidence. When the argument step is trivial, the strategy may be omitted.
Context
Context provides further definition to the argument, explaining key points in more detail. Contextual elements are important as they show the reader that a particular definition or model is in use.
Solution
Solutions represent basic evidence artefacts that are used in direct support of a goal. The link from goal to solution should be obvious.
The “is solved by” link shows the main flow of the argument from topmost goal down to evidence.
The “in context of” link provides contextual links to constrain the scope of an element.

Figure 2: main GSN symbols

Table 1: Instantiation and discussion of potential top-level goals

Pattern element	Instantiation	Suitability
Goal: SwSystemSafe: {software Y} is acceptably safe to operate within {system Z}	KCG qualified code generator is acceptably safe to operate within development process for {system Z}	This goal references the surrounding development process operated by the KCG tool user. This claim requires additional evidence beyond that pertaining to KCG.
Goal: SwContributionAcc: The contribution made by {software Y} to {system Z} hazards is acceptable	The contribution made by the KCG qualified code generator to {system Z} hazards is acceptable	This goal references the hazards in the system for which KCG produces software. Not all of these hazards will relate to software behaviour, and very few will relate to the behaviour of KCG. The scope is still too large to be supported by evidence about KCG.
Goal: Hazard: Software contribution(s) to {Hazard} is acceptably managed	Goal: Hazard: Software contribution to deviation of C code behaviour from input SCADE model is acceptably managed	This goal identifies an argument that is directly concerned with the transformation from the input SCADE model to C code. This directly matches the scope of the available evidence.

4. Issues in argument construction

4.1 Identification of the goal

Issue: For an argument based on existing evidence, it can be problematic to decide on the overall claim that should be made. If the claim is too broad, there will be too little evidence to support it. If the claim is too narrow, then it will not provide the assurance needed for the product.

Approach: To address this issue, the existing evidence was studied to determine the scope to which it applies. The main body of evidence for the KCG qualified code generator is test evidence concerned with the transformation from the SCADE input source to the equivalent C code. Based on this description, the overall claim should relate to this transformation.

To set up an appropriate top-level claim, the high-level software safety argument pattern from the Interim Standard of Best Practice for Software in the Context of Def Stan 00-56 Issue 4 [4] was used. This pattern present a series of possible top-level goals, each within smaller scopes, which were used as a menu from which to select the most appropriate overall claim. The three public goals in the pattern are those that are intended for use as the top-level goal of the software contribution argument. Table 1 shows these public goals along with their instantiation for the KCG assurance case and corresponding comments on suitability.

For the evidence supporting KCG, the most appropriate choice was that with the most specific scope. The higher-level goals identified in the pattern are then the responsibility of the tool user. The explicit GSN structure brings an additional benefit in demonstrating to the tool user and to other stakeholders exactly the type of claim being made and how this claim relates to its surrounding claims.

4.2 Argument over sources of deviation

Issue: The pattern-based argument is centred on sources of deviation of the generated C code from the input SCADE model, but no advice is given with the pattern on the process of identifying deviations.

Approach: Several existing development documents contained expert analysis of potential failure conditions using architectural models and systematic analysis. To substantiate the claim that the failure conditions have been correctly identified, a separate analysis was conducted and the results were compared to the original findings to confirm that independent analyses end up with the same results.

To create this view of the possible sources of deviation, a three-step process was used:

- KCG architectural documentation was consulted to produce a data-flow architectural model of KCG. The model was also informed by discussions with KCG developers, to ensure that an appropriate level of detail was represented. The data-flow style is particularly appropriate for KCG because of its focus on transforming one representation of behaviour to another.

- HAZOP analysis was conducted on the architectural model. The analysis began with customisation of the general-purpose HAZOP guidewords to custom forms as shown in Table 2. Each column shows customised guide words for different types of flow found in the data-flow architectural model. The narrow scope of the analysis and the systematic behaviour of the tool made it easy to identify mitigating actions during this analysis step. This step produced a list of around 70 individual consequences, with around 20 of those able to introduce a deviation that would not necessarily be detected by some existing process.
- While the HAZOP covered individual data flows, there are specific implementation concerns relating to the platform differences between the SCADE semantics and the target execution platform:
  - finite storage
  - finite accuracy
  - non-zero execution time
  - use of pointers

  A dedicated gap analysis was conducted in these areas to identify the platform gaps and their overall effect. For example, the SCADE semantics assume instantaneous execution of the code compared to the iteration rate of the control loop, whereas the eventual object code has some non-zero execution time. It is therefore up to the KCG tool user to determine the worst-case execution time of the control loop on the target platform and ensure that this is compatible with the iteration rate.

The previous analysis of KCG failure conditions was integrated into these newer analysis results. This process was performed alongside the other steps to ensure that all of the different analyses would fit together. This integration effort prompted the platform gap analysis and also revealed some missing domain knowledge that resulted in an incomplete customisation of the HAZOP guide words.

Table 2: Customised HAZOP guidewords for KCG architecture analysis

Guide Word	Translation	Hand-Code	Control	Feedback	Service
No	The translation simply does not take place.	No user-supplied code is given.	▨	Nothing reported about the behaviour of the tool.	No service is available.
Less	▨	▨	▨	▨	▨
More	▨	▨	▨	▨	▨
Part Of	Some part of the SCADE model isn't translated, or some part of the C code isn't compiled.	Not all of the necessary user-supplied code is given.	▨	The feedback reports are missing feedback about some part of the process or the model.	Only some of the required services are available.
As Well As	Something extra is put into the C code or the object code that was not intended.	The user supplies additional code that is not necessary.	▨	The feedback reports give spurious additional information.	▨
Reverse	▨	▨	▨	Tool indicates opposite success.	▨
Other Than	Translation creates different behaviour to that of the model.	User supplies incorrect code.	The wrong options are given to the tool.	The tool indicates a different result to the actual result obtained.	The service operates incorrectly.

Table 3: Summary of mitigation areas by responsible agent

User actions	Installer actions	Developer actions
• Verification of user-supplied code, type specification, command line and customisation • Validation of correct tool invocation and outcome • Use of the tool on multiple development systems • Verification of post-generation code modifications • Analysis of object code properties	• Verification of correct tool and OS version • Verification of tool command line and file processing capabilities	• 100% requirements test coverage • Full MC/DC • Absence of unintended functions • Verification of command line and file processing • Analysis of response to resource constraints • Analysis of underlying OS and library services

4.3 Presentation of analysis results

Issue: The HAZOP and gap analyses produced a large number of possible failure conditions, but there were many similarities in the conditions and significant similarities across the suggested mitigations. In all, the 68 failure conditions gave rise to 37 individual mitigation actions. Presenting this explicitly according to the pattern structure would lead to a significant replication of information across a complex GSN structure, for which the benefit was unclear. While the underlying argument would be the same in each case, it was important to be able to present the information with a useful amount of detail.

Approach: This issue was tackled in six steps:

- The failure conditions were classified according to their ability to cause a deviation of the C code behaviour from the model behaviour. A brief analysis of the classification showed that mitigations for the critical failure conditions would also mitigate the non-critical failure conditions. This reduces the number of mitigations that need to be addressed in the argument.
- The available mitigations were collated and grouped into user actions, installer actions and developer actions. Table 3 gives a summary of the mitigation areas. This gives a better picture of the overall scope of mitigations.
- The options for presentation of failure conditions were enumerated:
    - o A single reference in the argument to the full set of failure conditions.
    - o A reference to each of the 23 critical failure conditions (those leading to undetected deviation) plus a combined reference to the non-critical failure conditions.
    - o An individual reference to each of the 68 failure conditions.
- The options for presentation of mitigating actions were enumerated:
    - o A single reference to the full set of mitigating actions.
    - o A reference to actions grouped according to the 3 responsible agents.
    - o A reference to actions grouped according to the 13 mitigation areas.
    - o An individual reference to each of the 37 mitigating actions.
- The options for argument structure in this area were sketched out as shown in Table 4. The table gives a crude account of the overall complexity of the graphical structure for different levels of detail in the argument presentation.
- An appropriate combined representation was chosen. In this case, the argument was structured with the failure conditions grouped together into a single entity and the mitigating actions divided up into the three stakeholder areas. The division into mitigation areas could also have been usable at this level of the argument, although the choice of mitigation areas came relatively late in the course of the project.

Table 4: Complexity indication for detail in failure conditions and mitigations

Failure Conditions				**Mitigations**
	Single reference to all mitigations: 1	Division across responsible agents: 3	Division into mitigation areas: 13	Individual mitigations: 37
Single reference to all failure conditions: 1	1	3	13	37
Separate critical failure conditions with additional "others": 24	24	72	312	888
Individual conditions: 68	68	204	884	2516

4.4 Link from mitigating actions to evidence

Issue: Having decided to group mitigating actions according to responsible agent, a suitable structure was needed to describe the link to the evidence in terms of goals, strategies and solutions.

Approach: Goals were formulated for the mitigating actions in terms of subsets of the failure conditions. The failure conditions that need mitigation by the user are labelled user-specific failure conditions, similarly for the installer and the developer.

These actions are carried out in two entirely different contexts. The developer actions are all carried out during the development of the KCG tool, while the installer and user actions are all carried out after the user receives the tool. This identification of three types of mitigating condition and two contexts in which mitigations occur leads to a GSN structure as shown in Figure 2.

Arguing effective mitigation through developer actions is achieved by showing that failure conditions relate to safety requirements, that safety requirements trace to implemented functionality, and that the implementation satisfies the requirements. This links directly into the test evidence produced by following the guidance of the civil standards.

Arguing effective mitigation through user actions is

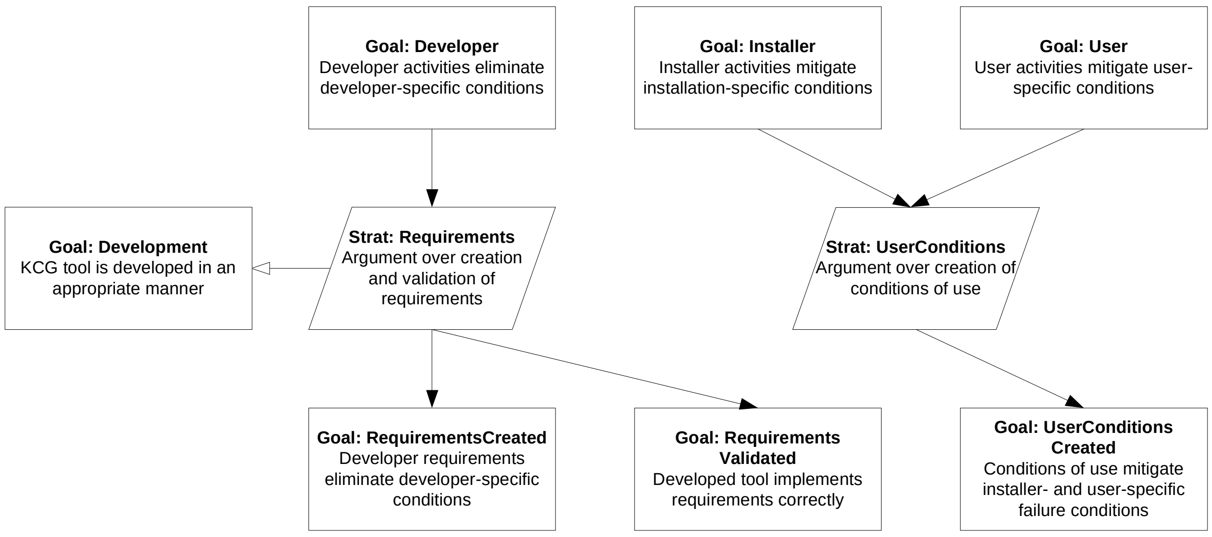
Goal: Developer
Developer activities eliminate developer-specific conditions
Goal: Installer
Installer activities mitigate installation-specific conditions
Goal: User
User activities mitigate user-specific conditions
Goal: Development
KCG tool is developed in an appropriate manner
Strat: Requirements
Argument over creation and validation of requirements
Strat: UserConditions
Argument over creation of conditions of use
Goal: RequirementsCreated
Developer requirements eliminate developer-specific conditions
Goal: Requirements Validated
Developed tool implements requirements correctly
Goal: UserConditions Created
Conditions of use mitigate installer- and user-specific failure conditions

Figure 2: GSN fragment showing mitigation argument structure

more problematic, as the tool developer has no control over the tool user. The strongest argument that may be made here is that the tool developer identifies conditions that the user must follow so that the tool output does not deviate from the behaviour specified by the input. When the tool user comes to make claims about the system built using that tool, that user must then demonstrate that these conditions of use have been met.

To assist the tool user, the customised software contribution patterns that are available to the KCG tool user identify exactly where this argument should be made.

4.5 Traceability supporting completeness

Issue: At several points in the argument, there is a need to demonstrate completeness. In some cases, we found that an appeal could reasonably be made to the traceability relationships in development artefacts. In other cases, while such evidence could be used to support an argument, the evidence had not already been generated as an existing development artefact. In this situation, the benefit of the additional evidence must be traded off against the effort involved in generating the evidence.

Approach: There were three points at which traceability was a strong candidate for support of completeness:

- The identification of failure conditions: for this area, we appeal to the traceability from the failure conditions to the data-flow architectural model of the KCG tool, and from the identified failures to the failures obtained from previous domain expert analysis. Issues identified from the traceability structure were fed back into the analysis process and prompted the use of explicit platform gap analysis.
- The combination of mitigations covering failure conditions: here the argument must show that developer, installer and user mitigation activities are complete with respect to the failure conditions that they address. The argument refers to the traceability from mitigation actions to the failure conditions that they address, but does not call for specific analysis of coverage, as such analysis is trivial with just three types of mitigation activity. If there were many more activity types (or perhaps responsible agents) then an explicit coverage analysis would be appropriate.
- The combination of guidance areas covering the intent of a standard: the top of the KCG development assurance argument is divided into system development guidance areas on the basis of a recognition of system development themes:
    - o Responsibility assignment: in this area, the organisation must identify appropriate roles and skills, identify the personnel in those roles, and demonstrate that those people have those skills.
    - o Safety management: here, the process must focus on the analysis of failure conditions and the management of risk.
    - o Configuration management: strong configuration management is essential as backing for a product argument.
    - o Continuous safety: safety must be addressed before development begins, during development, at the point of deployment and in the context of the use of the tool.
    - o Traceability: strong traceability is vital for support of a product argument.
    - o Coverage: this area embodies the idea that 100% of the requirements should be shown to be correctly implemented and no unintended functionality should be present.

  To show that these areas match with the content of appropriate existing guidance, the argument refers to three areas in which process guidance is already given (DO-178B, CAST-13 and IEC 61508). There is currently no explicit identification of any traceability mapping from the process areas to the contents of those guidance documents and standards. This step in the argument is instead taken to represent best practice and professional judgement.

4.6 Explicit product and process evidence

Issue: Some of the evidence produced by following industry guidance will relate to the correctness of the product with respect to its intended behaviour. Other evidence will show that the product was developed competently in accordance with industrial best practice. It is not common to find the distinction made explicit in industry guidance, nor in the evidence so produced, yet the distinction is a basic feature of an explicit argument.

Approach: The assurance argument is structured as a product argument that appeals to a process argument (an argument for development assurance). Beneath this structure, one group of evidence relates to the product argument, showing that KCG in itself has the required integrity, while other evidence relates to the process argument, showing the suitability of the development process for a critical development.

For example, software unit test reports and analysis of the adequacy of the conditions of use appear in the KCG product argument, while coverage reports and configuration management audits appear in the KCG development process argument.

The assurance gained from the process argument provides confidence that appropriate methods were employed in generating the product and the evidence that supports the claim that that product meets its requirements.

4.7 Good practice

Issue: In many parts of the argument, the claim is essentially that some evidence is adequately generated. There is no standard approach for linking such a claim to existing evidence.

Approach: To identify appropriate evidence, the argument was built with the following four aspects of good practice in mind:

- Definition of a process that is capable of generating the required evidence.
- Assessment of the process definition to show that it is appropriate for the type of evidence being used. For example, in KCG development there is independence between implementation and testing.
- Evidence that the process has been carried out effectively, such as logs or audits.
- Evidence produced by the process itself, such as functional test results or coverage data.

In many cases these aspects are distributed around the argument. For the example of functional test results, the four aspects arise as follows:

- The testing process definition is given in the test plans, and these are referenced from the KCG development argument, under goals concerned with continual safety consideration throughout the lifecycle.
- Adequacy of the testing process definition is not shown directly, but is established by association with the guidance given in established standards.
- Evidence that testing has been carried out effectively is found in the KCG development process argument, demonstrating 100% requirements coverage and MC/DC.
- The actual test evidence is used in the argument structure to substantiate the claim that the safety requirements for the KCG have been met.

Assessment of evidence according to these four aspects provided a useful rule of thumb when identifying the role of particular pieces of available evidence.

4.8 Support for complex guidance

Issue: When the end user of the KCG tool is producing software that is governed by DS 00-56, some additional guidance will be necessary. To ensure that the guidance is followed, the value of that guidance - or the potential cost of not following that guidance - must also be conveyed to the user.

Approach: The argument structure supports this in two ways:

- The KCG user is given conditions of use that constrain the installation and use of the tool to the context within which KCG assurance claims are valid. This involves areas such as the operating system used by the tool, command line options and the setting up of optimised operations in the generated code. As a stand-alone list of conditions, some effort could be involved to track down the rationale for each condition. In the argument structure, the conditions of use trace to installer and user activities that mitigate specific failure conditions, which in turn links to a higher goal involving more general identification, elimination and mitigation of failure conditions. A final step from this point shows that the conditions in question are the critical failure conditions from the hazard analysis and the platform gap analysis. The analysis report classifies the mitigating actions and links them to the conditions of use, completing the picture.
- The KCG user is given guidance on constructing a software safety case. This is based on customised software safety argument patterns, indicating the tiers of development for use of KCG and the links from KCG-specific tiers to the KCG product argument and its supporting material.

4.9 Direct objective mapping

Issue: The audience for a safety case is diverse. For some readers, the important information will be the safety argument and the degree to which it convinces the reader of its claim. For other readers, the important information will be the satisfaction of the particulars of DS 00-56. It is difficult to structure a safety case so that it will satisfy a range of different stakeholders.

Approach: For the KCG assurance case, a particular structure was used to link that assurance case into the tool user software safety case in a way that allows reference to adherence to a particular standard. The structure is shown in Figure 3.

The DS 00-56 conformance argument supports an overall KCG tool dependability claim, which in turn supports a claim that the output of KCG, when compiled and executed on the target platform, accurately implements the SCADE model. This claim provides backing for the assertion that validation in the SCADE environment is sufficient evidence for the correctness of the design tier in the decomposition.

5. Related work

Argumentation research for safety cases focuses on the abstract concepts and general principles of top-down argument construction. Examples include argumentation patterns [8], safety case modularity [9], assurance sufficiency [10] and argument justification [11]. Nevertheless, arguments are almost never constructed purely top-down; there is always some notion of the available evidence and the support that it may give to an argument. Working from the evidence to identify appropriate goals may not seem ideal, but it has the advantage of focusing on the power of the evidence rather than searching for ways to support an *a priori* belief, a danger highlighted in the Haddon-Cave report [12, p259, pt 5]. In practice, a combination of top-down principles, patterns and existing evidence will be applied in creating a safety case.

While there is little existing research directly related to the use of evidence from one standard in support of another, several parts of this paper are essentially practical examples of more general principles:

- The allocation of evidence between process and product arguments supports the targeted process argument approach of Habli and Kelly [6]. They advocate process arguments that are structured around concerns that are particular to the project, rather than around the general process structure imposed by a standard. For KCG, an intermediate approach was taken in which the process argument stands separately but makes specific claims about areas that attract concern.
- The issue of identifying and obtaining appropriate traceability information is also found in other work [7]. The identification of the need for traceability is similar here, but instead of relying solely on existing traceability information, Ridderhof et al propose tool support to allow for the construction of arbitrary traceability links. They identify automated traceability management as a key issue in the success of such an approach.

6. Summary and conclusions

Given appropriate evidence, construction of an explicit argument should be straightforward. However, when the argument is made against one standard and the evidence is from another, there is the potential for difficulty in deciding on the argument structure and the exact evidence to include. In producing an assurance argument for the KCG

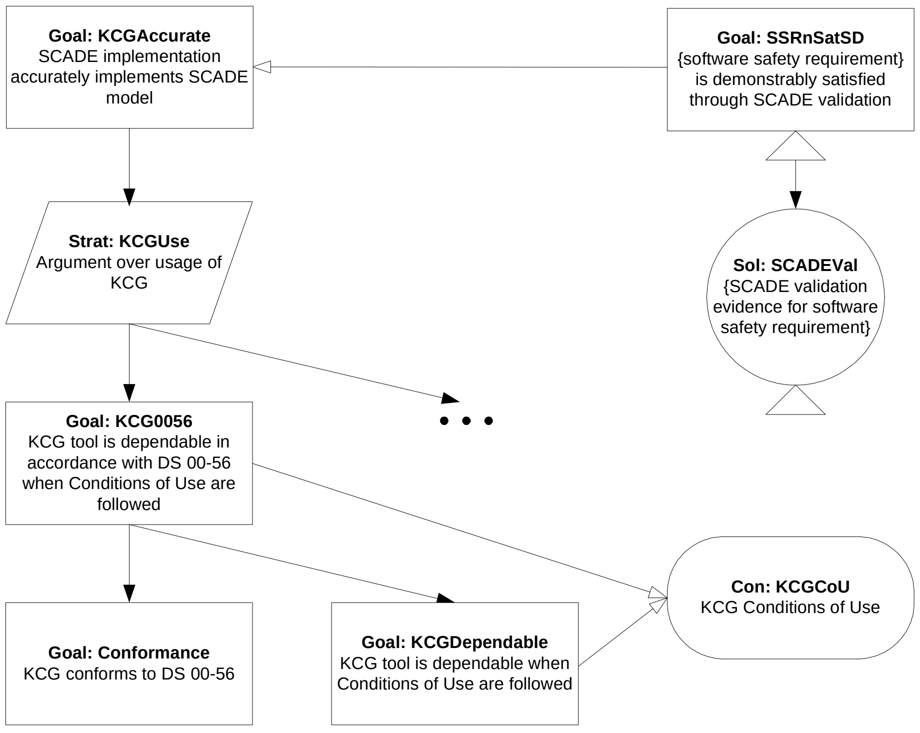
Goal: KCGAccurate
SCADE implementation accurately implements SCADE model
Goal: SSRnSatSD
{software safety requirement} is demonstrably satisfied through SCADE validation
Strat: KCGUse
Argument over usage of KCG
Sol: SCADEVal
{SCADE validation evidence for software safety requirement}
• • •
Goal: KCG0056
KCG tool is dependable in accordance with DS 00-56 when Conditions of Use are followed
Con: KCGCoU
KCG Conditions of Use
Goal: Conformance
KCG conforms to DS 00-56
Goal: KCGDependable
KCG tool is dependable when Conditions of Use are followed

Figure 3: GSN fragment showing argument integration structure

qualified code generator intended for DS 00-56, we found that we could use a wide range of evidence from the development and analysis of the tool under DO-178B, EN 50128 and IEC 61508.

In terms of evidence generation, our efforts were centred on additional support for the claim that sources of deviation were correctly identified. The remaining argument was supported with evidence generated through alternative standards.

The overall argument construction effort revolved around:

- Correctly identifying the relevant topmost goal, by instantiating a list of suggested topmost goals and comparing against the available scope of evidence;
- Presenting the results of hazard analysis in the argument in a manageable way;
- Providing an appropriate link from high-level mitigations to evidence;
- Linking to evidence that demonstrates completeness of analysis;
- Dividing existing evidence into process and product evidence;
- Where evidence is presented for a claim about the product, looking for backing evidence to show good practice in the generation of that evidence;
- Providing a route for the identification of rationale behind the guidance given to the tool user;
- Describing the relationship between the available evidence, the supporting arguments and the individual requirements of DS 00-56.

While many parts of this work represent instantiations of general principles, we believe there is novelty and value in our approach to finding an appropriate presentation mechanism for a large body of evidence and in the method of scope comparison to document the selection of an appropriate top-level goal.

7. References

[1] IEC: "Functional safety of electrical/ electronic/ programmable electronic safety-related systems", IEC 61508, 2000

[2] CENELEC: "*Railway applications. Communications, signalling and processing systems. Software for railway control and protection systems*", EN 50128, 2001

[3] RTCA/EUROCAE: "*Software Considerations in Airborne Systems and Equipment Certification*", DO-178B, 1992

[4] Ministry of Defence: "Safety Management Requirements for Defence Systems", Def Stan 00-56, Issue 4, 2007

[5] Menon, C; Hawkins, R. and McDermid, J.: "*Interim Standard of Best Practice on Software in the Context of DS 00-56 Issue 4*", SSEI report SSEI-BP-000001, 2009

[6] Habli, I. and Kelly, T: "*Achieving Integrated Process and Product Safety Arguments*", The Safety of Systems, Part 2, Proceedings of the Fifteenth Safety-critical Systems Symposium, Springer Verlag 2007, pages 55-68

[7] Ridderhof, W.; Gross, H.-G. and Doerr, H: "*Establishing Evidence for Safety Cases in Automotive Systems – A Case Study*", Proceedings of SAFECOMP 2007, LNCS 4680, 2007, pp 1-13

[8] Kelly, T. P. and McDermid, J. A.: "*Safety Case Construction and Reuse Using Patterns*", Proceedings of SAFECOMP 16, Springer Workshops in Computing, 1997, pp55-69

[9] Althammer, E.; Schoitsch, E.; Eriksson, H. and Vinter, J.: "*The DECOS Concept of Generic Safety Cases - A Step Towards Modular Certification*", EUROMICRO-SEAA, 2009, pp537-545

[10] Hawkins, R. D. and Kelly, T. P. "*Software Safety Assurance - What is Sufficient?*", IET System Safety Conference, 2009

[11] Bishop, P. G.; Bloomfield, R. and Guerra, S. "*The Future of Goal-Based Assurance Cases*", Proceedings of the Workshop on Assurance Cases, 2004

[12] Haddon-Cave QC, C.: "*An Independent Review into the Broader Issues Surrounding the Loss of the RAF Nimrod MR2 Aircraft XV230 in Afghanistan in 2006*", 2009

8. Glossary

GSN:	Goal Structuring Notation
HAZOP:	Hazard and Operability Study
KCG:	Esterel Qualified Code Generator
PDF:	Portable Document Format