Compositional Certification Conceptual Framework

**D5.3**

<table>
<thead>
<tr>
<th>Work Package:</th>
<th>WP5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dissemination level:</td>
<td>PU</td>
</tr>
<tr>
<td>Status:</td>
<td>Final</td>
</tr>
<tr>
<td>Date:</td>
<td>16 December 2013</td>
</tr>
<tr>
<td>Responsible partner:</td>
<td>Philippa Conmy (University of York)</td>
</tr>
<tr>
<td>Contact information:</td>
<td><a href="mailto:philippa.conmy@york.ac.uk">philippa.conmy@york.ac.uk</a></td>
</tr>
</tbody>
</table>

**PROPRIETARY RIGHTS STATEMENT**
This document contains information, which is proprietary to the OPENCOSS Consortium. Neither this document nor the information contained herein shall be used, duplicated or communicated by any means to any third party, in whole or in parts, except with prior written consent of the OPENCOSS consortium.
Contributors

<table>
<thead>
<tr>
<th>Names</th>
<th>Organisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alejandra Ruiz, Huáscar Espinoza</td>
<td>TECNALIA Research &amp; Innovation</td>
</tr>
<tr>
<td>Sunil Nair</td>
<td>Simula Research Laboratory</td>
</tr>
<tr>
<td>Katrina Attwood, Philippa Conmy</td>
<td>University of York</td>
</tr>
</tbody>
</table>

Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0.1</td>
<td>2013-09-18</td>
<td>ToC</td>
</tr>
<tr>
<td>V0.2</td>
<td>2013-11-12</td>
<td>First draft version</td>
</tr>
<tr>
<td>V0.3</td>
<td>2013-11-14</td>
<td>Ready for PB review</td>
</tr>
<tr>
<td>V1.0</td>
<td>2013-11-30</td>
<td>Reviewed first release</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

1 Executive Summary ........................................................................................................................................... 7

2 Introduction .................................................................................................................................................................... 8

3 Background Information ............................................................................................................................................. 12
   3.1 Related work and state of the art ................................................................................................................................. 12
   3.2 Expressing Arguments .................................................................................................................................................. 13
   3.3 Contracts for capturing dependencies .......................................................................................................................... 15
   3.4 Composition of Evidence ............................................................................................................................................ 17
   3.5 Summary ........................................................................................................................................................................ 18

4 Description of SACM/CCL model .............................................................................................................................. 19
   4.1 High level class diagrams for OPENCOSS argumentation meta-model ............................................................. 19
   4.2 Concepts ........................................................................................................................................................................ 22
      4.2.1 AssuranceCase class .............................................................................................................................................. 22
      4.2.2 Agreement class ..................................................................................................................................................... 22
      4.2.3 ArgumentationElement class (abstract) ................................................................................................................... 22
      4.2.4 Argumentation Class ............................................................................................................................................... 23
      4.2.5 ArgumentElement Class (Abstract) ........................................................................................................................ 23
      4.2.6 ReasoningElement Class (Abstract) ....................................................................................................................... 24
      4.2.7 Assertion Class (Abstract) ...................................................................................................................................... 24
      4.2.8 InformationElementCitation Class ......................................................................................................................... 24
      4.2.9 ArgumentElementCitation Class ............................................................................................................................. 25
      4.2.10 Claim Class ............................................................................................................................................................. 26
      4.2.11 EvidenceUseAssertion Class ................................................................................................................................ 27
      4.2.12 ArgumentReasoning Class .................................................................................................................................. 27
      4.2.13 AssertedRelationship Class (Abstract) .................................................................................................................. 28
      4.2.14 AssertedInference Class ....................................................................................................................................... 28
      4.2.15 Choice Class ........................................................................................................................................................... 29
      4.2.16 AssertedEvidence Class ......................................................................................................................................... 30
      4.2.17 AssertedContext Class ............................................................................................................................................ 31
      4.2.18 AssertedChallenge Class ......................................................................................................................................... 31
      4.2.19 AssertedCounterEvidence Class ........................................................................................................................... 32

5 Argument Templates and Vocabulary ....................................................................................................................... 34
   5.1 Introduction ................................................................................................................................................................. 34
   5.2 Argumentation strategy ................................................................................................................................................ 34
      5.2.1 Argument contents and architecture ........................................................................................................................ 36
   5.3 Overall Strategies for Modular Arguments - Architecture and Example Templates ....................................... 37
      5.3.1 SysSafe ................................................................................................................................................................. 38
      5.3.2 SysSafeReq ............................................................................................................................................................ 40
      5.3.3 CompNlnSys .......................................................................................................................................................... 42
      5.3.4 CompNDev ............................................................................................................................................................. 44
      5.3.5 SysAssump .............................................................................................................................................................. 45
      5.3.6 CompNAssump ...................................................................................................................................................... 46
      5.3.7 CompNSpec ............................................................................................................................................................ 48
      5.3.8 SysCompliance ...................................................................................................................................................... 50
      5.3.9 CompNCompliance ............................................................................................................................................... 51
      5.3.10 Variations .............................................................................................................................................................. 53
   5.4 Controlled language and formalisation of claim ................................................................................................. 53
5.4.1 Method 1: Strict Syntax and Grammar ................................................................. 53
5.4.2 Method 2: Content/Vocab/Claim Type ................................................................. 59
5.5 Summary .................................................................................................................. 74

6 Evidence Sets .............................................................................................................. 75
6.1 Introduction .................................................................................................................. 75
6.2 Assessing Evidence Sets ............................................................................................. 76
  6.2.1 Study design, Data collection and Analysis .......................................................... 76
6.3 Factors Considered for Assessing an Evidence Sets ................................................. 77
  6.3.1 Experience ............................................................................................................ 77
  6.3.2 Expertise ................................................................................................................ 78
  6.3.3 Historical Knowledge ............................................................................................ 78
  6.3.4 Provenance of person .......................................................................................... 78
  6.3.5 Provenance of tools .............................................................................................. 78
  6.3.6 Independence ......................................................................................................... 78
  6.3.7 Peer review ............................................................................................................ 78
  6.3.8 Best practices ........................................................................................................ 79
  6.3.9 Human emotions .................................................................................................. 79
  6.3.10 Personal relationship with people involved ......................................................... 79
  6.3.11 Claim .................................................................................................................... 79
  6.3.12 Argument ............................................................................................................. 79
  6.3.13 Mental Checklist ................................................................................................ 79
  6.3.14 Standard’s Checklist ......................................................................................... 79
  6.3.15 Safety culture ........................................................................................................ 80
  6.3.16 Heuristics .......................................................................................................... 80
6.4 Interview Questionnaire ............................................................................................. 80

7 Conclusions ................................................................................................................... 84

8 Abbreviations ................................................................................................................. 85

9 References ..................................................................................................................... 86
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Integration of assurance data in the same domain</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Relationship between design integration details and assurance integration claims</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>Relationship between components and assurance</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>Reusing assurance data between different domains</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>GSN Symbology</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>Example GSN Argument</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>High Level Modular Argument Structure and Architecture</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>Assurance Case class diagram</td>
<td>19</td>
</tr>
<tr>
<td>9</td>
<td>Argumentation Class Diagram</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>Relationships view diagram</td>
<td>21</td>
</tr>
<tr>
<td>12</td>
<td>Argumentation Class Example</td>
<td>23</td>
</tr>
<tr>
<td>13</td>
<td>InformationElementCitation Class Example</td>
<td>25</td>
</tr>
<tr>
<td>14</td>
<td>Claim Class Example</td>
<td>27</td>
</tr>
<tr>
<td>15</td>
<td>ArgumentReasoning Class Example</td>
<td>28</td>
</tr>
<tr>
<td>16</td>
<td>AssertedInference Class Example</td>
<td>29</td>
</tr>
<tr>
<td>17</td>
<td>Choice Class Example</td>
<td>30</td>
</tr>
<tr>
<td>18</td>
<td>AssertedEvidence Class Example</td>
<td>30</td>
</tr>
<tr>
<td>19</td>
<td>AssertedContext Class Example</td>
<td>31</td>
</tr>
<tr>
<td>20</td>
<td>AssertedChallenge Class Example</td>
<td>32</td>
</tr>
<tr>
<td>21</td>
<td>AssertedCounterEvidence Class Example</td>
<td>33</td>
</tr>
<tr>
<td>22</td>
<td>Relationships between patterns, instantiated patterns, and re-usable items in the argumentation approach</td>
<td>35</td>
</tr>
<tr>
<td>23</td>
<td>Top down and bottom up approaches to system certification</td>
<td>36</td>
</tr>
<tr>
<td>24</td>
<td>Combining top down and bottom up approaches for compositional certification</td>
<td>36</td>
</tr>
<tr>
<td>25</td>
<td>Overall Argument Structure</td>
<td>38</td>
</tr>
<tr>
<td>26</td>
<td>SysSafe Module Template Argument</td>
<td>39</td>
</tr>
<tr>
<td>27</td>
<td>SysSafeReq Module Template Argument</td>
<td>41</td>
</tr>
<tr>
<td>28</td>
<td>ComNlnSys Module Template Argument</td>
<td>42</td>
</tr>
<tr>
<td>29</td>
<td>CompNDDev Module Template Argument</td>
<td>44</td>
</tr>
<tr>
<td>30</td>
<td>CompNAssump Module Template Argument</td>
<td>47</td>
</tr>
<tr>
<td>31</td>
<td>CompNSpec Module Template Argument</td>
<td>49</td>
</tr>
<tr>
<td>32</td>
<td>SysCompliance Module Template Argument</td>
<td>50</td>
</tr>
<tr>
<td>33</td>
<td>CompNCompliance Module Template Argument</td>
<td>52</td>
</tr>
<tr>
<td>34</td>
<td>Contracts development phases</td>
<td>56</td>
</tr>
<tr>
<td>35</td>
<td>Model of Argument Agreement</td>
<td>56</td>
</tr>
<tr>
<td>36</td>
<td>Common Certification Language Simplified Overview (from [1])</td>
<td>59</td>
</tr>
<tr>
<td>37</td>
<td>Levels of Model in OPENCOSS (from [1])</td>
<td>60</td>
</tr>
<tr>
<td>38</td>
<td>IEC-61508 Software Concepts</td>
<td>62</td>
</tr>
<tr>
<td>39</td>
<td>DO-1788 Software Concepts</td>
<td>62</td>
</tr>
<tr>
<td>40</td>
<td>Different aspects which need to be addressed by the Vocabulary</td>
<td>64</td>
</tr>
<tr>
<td>41</td>
<td>Overall Structure of the Vocabulary</td>
<td>65</td>
</tr>
<tr>
<td>42</td>
<td>Partial SBVR example for Automotive Domain safety terminology</td>
<td>68</td>
</tr>
<tr>
<td>43</td>
<td>Generic claim type</td>
<td>72</td>
</tr>
<tr>
<td>44</td>
<td>Conceptual Model underlying claim type in Figure 43 (NB: fictitious)</td>
<td>73</td>
</tr>
<tr>
<td>45</td>
<td>Claim Type Instantiation in Argument Development</td>
<td>74</td>
</tr>
<tr>
<td>46</td>
<td>Combining different Evidence for Safety assessment</td>
<td>75</td>
</tr>
</tbody>
</table>
List of Tables

Table 1 SysSafe text to be instantiated .......................................................................................................... 39
Table 2 SysSafe Participant Goals .................................................................................................................. 40
Table 3 SysSafeReq text to be instantiated .................................................................................................... 41
Table 4 SysSafeReq Participant Goals ........................................................................................................... 41
Table 5 CompNInSys text to be instantiated .................................................................................................. 42
Table 6 CompNInSys Participant Goals ........................................................................................................... 43
Table 7 CompNDev text to be instantiated .................................................................................................... 45
Table 8 CompNDev Participant Goals ............................................................................................................. 45
Table 9 SysAssump text to be instantiated ..................................................................................................... 46
Table 10 SysAssump Participant Goals .......................................................................................................... 46
Table 11 CompNAssump text to be instantiated ........................................................................................... 47
Table 12 CompNAssump Participant Goals ..................................................................................................... 48
Table 13 CompNSpec text to be instantiated ................................................................................................. 49
Table 14 CompNSpec Participant Goals .......................................................................................................... 49
Table 15 SysCompliance text to be instantiated ............................................................................................ 50
Table 16 SysCompliance Participant Goals .................................................................................................... 50
Table 17 CompNCompliance text to be instantiated ........................................................................................ 52
Table 18 CompNCompliance Participant Goals ............................................................................................... 52
Table 19 Different contract technical approaches .......................................................................................... 54
Table 20 SBVR Attribute definitions (adapted from Annex C of [35]) ............................................................. 66
Table 21 SBVR Font Types (from Annex C of [35]) .......................................................................................... 67
Table 22 Initial Typology of Claims ................................................................................................................ 70
Table 23 Factors considered while assessing a set of evidence and the level of automation that can be achieved to implement them ................................................................................................................. 82
1 Executive Summary

This deliverable presents a conceptual framework for compositional certification, establishing principles and ideas. Our aim is to help support re-use of component level assurance data, thus improving the time taken to produce a system, and reduce the costs associated with development.

Three fundamental aspects of compositional certification are presented:

1. Composition of assurance arguments via:
   a. Argumentation patterns - in which we separate concerns between systems and components, and further sub-divide these into functionality, backing arguments and compliance.
   b. Language and expression - looking at both syntax and semantics of assurance claims, and the compliance, confidence and safety vocabulary we required to express and compare them.

2. Composition of evidence - which presents the results of a survey of industrial practitioners on assessing sets of evidence, and how their overall consistency and quality can be assessed.

3. A meta-model for argumentation - merging the Goal Structuring Notation (GSN) and Structured Assurance Case Metamodel (SACM).

The patterns research in 1.a above, was influenced and developed in conjunction with the Case study work in WP1. The vocabulary work (1.b) was developed in conjunction with WP4, and the CCL meta-model for vocabulary [1], as well as the case study work.

Both 1 and 2, above, link to the work in WP6 on evidence characterisation and capturing meta-data about evidence (see D6.1 [2] and D6.2 [3]).

Part 3 was originally part of the CCL work in WP4 (specifically D4.4 [1]).

All the work helps meet the research requirements identified in D5.2 [4]. The background information and related work in D5.1 [5] is also relevant. Close contact with Task 5.3 has provided input from this theoretical framework to the tooling being developed for D5.4 (software deliverable).
2 Introduction

We start this work from the assumption that a safety critical system is constructed from a number of different components. These components would be provided from a number of different suppliers to the main system developer. Component can refer to any sub-element of the system (assuming it is constructed from at least two different parts but, more likely, a large number of components). Compositional design techniques mean that these components can be brought together, using robust models of the components (describing their assumed use and offered functionality), thus assisting in design predictions of how they should work together prior to actual integration. We are concentrating on software intensive safety critical systems within this report due to expertise in the consortium, but the principles may be adapted to other areas in future (e.g. hardware parts).

Ideally, we wish to also have compositional certification of assurance data associated with each component. In this report we use the term certification to loosely mean the approval of a system, in terms of whether it is acceptably safe and also in terms of its compliance to a particular set of development and lifecycle criteria (e.g. a safety standard or company process regulations). Examples of certification bodies may be legal and strict e.g. the Civil Aviation Authority (CAA) [6], which regulates which aircraft can and can't fly in the U.K., or self-regulation such as Motor Industry Software Reliability Association (MISRA) [7], which is a combination of component manufacturers and industrialists coming together to combine best practice. Self-regulation may even be about approval in internal company policy. Assurance, then, covers a wide range of practice, from formal to informal methods of assuring that a system or component meets the specified properties.

Traditionally, certification has been a relatively top down process, assessing a design and performing different techniques to demonstrate system safety requirements are allocated to components, then implemented, and a certain quality level is assured. This data is often presented for assessment (either by an external body, or by the system developer) via a safety case, consisting of two parts [8]:

- **Evidence** about the item, e.g. test data, design to requirements mappings, formal specifications
- **Arguments** constructed from a series of claims about the system, typically starting with "system is acceptably safe", which is then deconstructed to a series of low level, easily verifiable claims, e.g. that a piece of software meets specified real-time constraints to respond to user input, on defined hardware.

![Figure 1 Integration of assurance data in the same domain](image-url)
The problem with the top down approach is that it is difficult to re-use assurance data, even if a component is able to be re-used in a different system. This data is typically extremely costly and time consuming to produce, but the component may not change when used in a different system. Our aim is to help support re-use of component level assurance data, thus improving the time taken to produce a system, and reduce the costs associated with development. The main approach taken within OPENCOSS is to re-use as much assurance data per component as possible, where appropriate, and integrate it with system assurance data. This is summarised in Figure 1.

There is a subtle point to note here. We are integrating assurance data, and hence this is not just about design integration. We don't only need to assess whether a component meets a specification, and why that specification is correct, but whether we are sufficiently confident that the data demonstrates a certain component property. Figure 2 shows a simple example: on the left hand side we show how a component is used to meet part of a car specification, on the right hand side we show, firstly the reasoning behind the specification i.e. mitigation of a collision, and secondly, the reasons why we believe using the component will sufficiently mitigate the hazard. Data represented in this diagram that should always be re-usable for assurance are the component property, component specification, and our confidence in the evidence of that property (with provisos that certain operating assumptions are met in the system, e.g. the WCET is based on a certain processor type, or the environmental conditions are within a certain temperature range). It's possible that other aspects, e.g. the reasoning, would be re-usable across multiple different car systems, however this is more complex and relies on a number of assumptions about similar operating conditions and performance of a vehicle.

![Figure 2 Relationship between design integration details and assurance integration claims](image)

Our aim is to be able to maximise the re-use of "system neutral" assurance data, such as static properties analyses and confidence in their assurance. Thus, the compositional certification approach relies on the ability to:

- Compose re-usable evidence/assurance data - linking evidence where required (see section 6)
• Create re-usable component level arguments - which provide information on our confidence that a component meets its specification, and the quality of the evidence (see section 5)
• Generate a specific system assurance case as necessary, ensuring certification and system requirements are satisfied (section 5) - whether the sum of the components, and their assurance data, meets both design and compliance requirements.

An alternative way of looking at this approach is shown in Figure 3. Represented on the left-hand-side is a generic component model, which meets a number of properties and other guarantees (dependent on whether certain rely conditions are met). E.g. component A will guarantee to provide accurate fin adjustments, if the data provided is within a specified range and in the expected scientific units. Or, component A will guarantee NOT to make an adjustment if it receives data outside the expected range. On the right-hand-side we illustrate that we need evidence and arguments to demonstrate that those guarantees/properties are actually as described, that they meet the component requirements, and also that they should be matched to actual system and certification requirements.

A further aim, in the future, is to facilitate the re-use of the data in multiple domains. The assurance data will usually have been acquired by following the recommended practice in a given domain standard, e.g. ISO26262 for automotive [9] and DO178B for avionics [10]. These standards/guidance may have difference practices from one another, therefore understanding is required of how well data from one domain meets the needs of another. OPENCOSS is developing the means to map assurance concepts between different domains, thus supporting a process such as that shown in Figure 4.

![Figure 3 Relationship between components and assurance](image-url)
Figure 4 Reusing assurance data between different domains
3 Background Information

This section of the document describes both an introduction to the area of research, and related research in this area. First we describe the general state of the art in compositional assurance, then we look at compositional arguments. Next we consider contracts as a means for expressing rely/guarantee conditions. Finally, considerations on compositional evidence are presented.

3.1 Related work and state of the art

There is limited academic research that directly addresses the issue of compositional assurance. Therefore, in this section we mainly discuss two things: some of the design paradigms which have been developed to assist compositional design and methods for expressing compositional arguments.

One design paradigm is Integrated Modular Avionics (IMA). This is a heterogeneous computing network used within the avionics domain. It uses three-layer computer architecture, separating applications (system, e.g. aircraft-specific but not hardware specific), from an operating system (hardware and platform neutral - shielding the applications from any hardware changes), and a hardware interface layer (hardware specific but not system specific). One purpose of this architecture is to streamline the upgrade and maintenance of both the applications and the underlying computing hardware. Any update to software or hardware must be accompanied by an update to assurance data. Using a suitable modular certification process, and relying on fixed software interfaces between aircraft specific and generic components, assists this, limiting the amount of re-work and re-analysis as much as possible.

Research and guidance relating to how to achieve modular analysis and modular certification practically can be found in [11][12][13]. In [11] the authors looked at modular arguments, and noted that it was important to identify change scenarios in advance of certification, and also the need to look at dependency-guarantee relationships (DGRs) between assurance arguments to look at interoperability. These are similar to the concept of design by contract, well established in the software domain [14]. The authors did not develop a method to assist in the definition and expression of the DGRs. In [12] failure mode analyses of an IMA operating system were performed, and a number of challenges found, including the need match informal analysis results of this technique with other informal analyses of software components, and to determine whether assumptions made about how the OS was being used were correct. We propose to improve this by using controlled language and expressions, where possible, in the assurance argument.

The avionics development guidance document DO-297 [13] defines a high-level process for certifying IMA, again requiring all assumptions to be stated when components are analysed independently, and a staged process to integration to help ensure these assumptions are as correct and complete as needed.

The automotive standard ISO26262 [9] has the concept of a Safety Element out of Context (SEooC). This allows a developer to produce a component, and analyse it, outside of a specific system development. The SEooC is developed to a set of assumed requirements, and makes assumptions about where and how it will be used in an actual system. The assumptions must be documented, and can then be validated during an actual vehicle development. The automotive domain also has a similar design paradigm to IMA called AUTOSAR [15], though there is much work to do in defining adequate interfaces [16].
3.2 Expressing Arguments

The Goal Structuring Notation (GSN) [17] has explicit extensions which allow arguments to be produced in modules per component - providing justifications and guarantees (backed with evidence) on an individual basis. The notion of modular safety cases has been introduced to facilitate effective development and maintenance of a safety case. Inspired by the notions of modularity in software and system engineering disciplines, modular safety cases are constructed from independently developed modules, with well-defined interfaces. The principal elements of the notation are shown in Figure 5 in the form of example instances of each concept from a safety argument:

When these elements of GSN are connected together, they are said to form a goal structure. A very simple example is shown in Figure 6. Sections of an argument can be grouped together in what is known as a module. The modules should be constructed of claims that can be reused within different system safety arguments, along with bespoke arguments about that system as necessary. For example, a component may have a specific Worst Case Execution Time, which is demonstrated via statistical testing. Arguments about this WCET may include information about the platform used for testing, confidence in the results (how many tests were run) and information on the testing technique used itself (how reliable it has been in the past). This information is reusable in multiple systems. In a specific system context, several WCETs may need to be reconciled as part of an overall system schedule, in order to meet a specific safety requirement. The latter bespoke argument may be via what is known as a Safety Case Contract which connects various argument modules together (Figure 7).
An example of a broad high level structure to a modular argument is shown in Figure 7. This argument has reused modules about an operating system (OS) and an application running on that OS, with claims reconciled via a safety argument contract. In addition, these are linked to a bespoke system level argument which also needs to link to claims in the modules.

If safety case modules are developed fully independently, with no coordination at the onset, it is highly unlikely that an adequate safety case for an integrated platform can be compiled in a ‘bottom-up’ fashion. Modules will define “public goals” which offer reasoning about certain aspects of the component or its behaviour. It may also have “away goals” which refer to gaps in reasoning. Ideally we should be able to match public to away goals, however in practice this is difficult. This is due to a number of factors as follows:
• Claims are expressed in a natural language, and as such may contain ambiguities in the way they are written.
• Claims about a component may be expressed at a different level of abstraction, due to the style of the argument developer. For example, software claims may be made at a low-level of detail (e.g. about freeness of boundary violations for an individual function) or at a higher level of abstraction (e.g. about general use of coding sub-sets and practices which mean that none of the software would have such violations, but it isn't explicitly stated).
• All argument claims are made based on a number of assumptions and within a given context. This is captured in the argument (via the special symbols shown above) but these need to be compared for each module to ensure that claims, which may superficially appear to match, do in reality. A more effective process model would require a generic safety case architecture to be established first, in order to identify key responsibilities of individual modules. In addition, establishing broad reusable principles for style and claim expression would mean that it is easier to link claims together. However, since components may be reused across multiple projects, or in different types of systems, this approach is not feasible in all cases. This is the approach taken in section 5.3.

The Structured Assurance Case Metamodel (SACM) [18] is a standard being developed by the OMG to model all kinds of argumentation concepts. It has a richer set of concepts than is made explicit in the GSN, for example the notion of how particular claims are used in the argument - e.g. as supporting or indirect, and umbrella types of element in the argument. In addition, it has some extra concepts such as counter-evidence and claims. However, it also lacks some of the GSNs features, most specifically modularity and some forms of patterning to provide argumentation templates. SACM doesn't have a graphical notation for ease of use.

Our adopted Argumentation MetaModel for OPENCOSS is shown in Section 4. This is a hybrid notation which combines features of SACM with those of GSN, thus giving us the best of both forms.

### 3.3 Contracts for capturing dependencies

As noted previously, the use of contracts in component-based development is a well-established research community. However there exists a wide range of views as to the nature of the contracts that are necessary to support safety-critical systems development, assurance and certification. There are challenges when using this approach when dealing with safety, and safety assurance, properties. Safety is a system property and because of that, it can be hard to define the contribution of components that have an impact on safety. Contract based approaches addressing safety have been proposed in the past regarding modular safety case development.

We have identified three phases in the use of contracts to support the certification of components. The first step is the use of design contracts to support the technical integration of different components within a system. Design contracts focus on the necessary conditions for correct component operation. In an integrated component configuration if component contracts are satisfied the set of components can be assumed to function correctly together.

An example of rich design contracts for assurance can be found in [19], where the authors present a restricted language for describing rely/guarantee conditions between software applications and computing hardware. This has the advantage that it can be used to generate automatically a limited set of arguments about the composed behaviour of the software, including for failure behaviour. However, it does not
capture additional information, such as relating the evidence supporting rely/guarantee claims or the confidence in them. In addition, it is limited to one specific set of software relationships related to resource management. Other notations, such as found in the ASSERT, CHESS and SPEEDS projects also offer methods for expressing design contracts. These are discussed in depth in OPENCOSS deliverable 5.1 [5]. This project does not attempt to extend these notations for the design aspects of compositionality. Instead we are concerned with supporting arguments about the validity of the rely/guarantee conditions given in the design contracts.

The context in which the component is going to be integrated is important and as indicated in [20] from the SEooC (Safety Element out of Context) perspective, the assumptions of the item can be understood as context characterization. In addition, to support safety assessment, failure behaviours of components, and their behaviour in the presence of failures, must be defined. Ruiz shows some needs of the industry in relation with the application of the SEooC concept and proposed the use of safety contracts as a possible strategy. A primary challenge is identifying all of the assumptions made and secondly envisaging all of the different contexts in which the element might be used.

The last step mentioned is that of assurance contracts. Assurance contracts define the set of claims that can be made about a component in terms of its assurance data. For example, the quality of some evidence and how well we believe that evidence supports claims made about the component, e.g. does the testing actually sufficiently stress test the software conditions we claim?

There is some guidance in current standards, for the compliance aspects of assurance, which address this problem in different ways. In ISO 26262 [9] Development Interface Agreements (DIA) are described as a way to specify both procedures and responsibilities allocated to distributed developments for items and elements. The DIA includes information beyond technical safety by addressing procedural and confidence related issues. The use of DIAs is intended to help address risks such as: a supplier with inadequate capability, improper understanding or definition of the boundary of component and its interactions with its environment, or failing to fulfil requirements.

In the avionics domain we can find similar requirements while talking about modules and application reuse on an IMA (Integrated Modular Avionics) platform. In DO-297[13] for reuse of component acceptance it is required that component limitations, assumptions, etc. are documented and a usage domain analysis is performed to ensure that it is being reused in the same way as it was originally intended. As in the automotive domain, in the avionics domain the adequacy of suppliers is a concern. Big companies such as Airbus are starting to put into practice a methodology to ensure the quality and capability of their suppliers specially for the critical functions. Jany presented [21] the plans for Airbus on the idea of extended airworthiness. The main issues being addressed were: delegation of authority, the cascade on certification requirement and the surveillance of suppliers.

In the avionics domain we can find similar requirements with respect to module and application reuse within an IMA (Integrated Modular Avionics) platform. In DO-297 [13] (amongst other requirements) it is required that limitations, assumptions, etc. are documented and a usage domain analysis performance to ensure that any component is being reused in a way that is compatible with the original design intent.

Other aerospace avionics guidelines such as AC 20-148 [22] concerning reusable software components indicate that in order to reuse components, stakeholders must identify any installation, safety, operational, functional and performance possible concerns. Developers need to state clearly the DO-178B objectives that are fully and partially addressed, and how compliance has been achieved. They need to state clearly the failure conditions, safety features, protection mechanism, architecture limitations, software levels, interface specification and the process for certification. AC 20-170 [23] defines incremental acceptance as, “A process for obtaining credit toward approval and certification by accepting or finding that an IMA module, and/or off-aircraft IMA system complies with specific requirements. This incremental acceptance is divided into tasks. Credit granted for individual tasks contributes to the overall certification goal.”
definition implies that the process in which the system assurance is performed is also important. At every stage some form of recognition is submitted in relation which a compliance data package. The process is divided into 6 tasks:

1. Module acceptance;
2. Application acceptance;
3. IMA system acceptance,
4. Aircraft integration of IMA system,
5. Change and reuse of modules or applications.
6. Reuse can be done at Task 1 and 2 level.

3.4 Composition of Evidence

Large systems such as those used in avionics, railway and automotive are often made of many components. For a relatively large system, there might be the need for collecting and assessing 1000s of pieces of evidence (resulting from 1000s of components) that would contribute towards building confidence in the safe operation of the system. When handling a large set of evidence for demonstrating safety, their composition has to be handled well. Composition of evidence can be described as follows.

- Evidence (particularly safety evidence) is produced by applying different analysis and testing techniques to a component, to back up what has been described in the contract (e.g. to prove that the execution time is the one specified). See also the evidence taxonomy in D6.1 [2]. Some types of evidence are easier to compose than others - for example some types of testing can be performed per software module, and fault trees are often composable. However, these may be linked to specific system requirements so care is needed to ensure that the properties being demonstrated are still valid in a new context. Other types of evidence may superficially seem easy to compose, but are harder in practice. For example, Worst Case Execution Time (WCET) values can be added together very easily, but values may have been derived using very different techniques (static analysis or probabilistic testing). Other types of evidence are unlikely to be composable (e.g. top down functional failure analysis).

More consideration has to be taken when the system is evolving and as a result the chain of evidence is evolving. A chain of certification evidence is a series of pieces of evidence that are related (e.g., the agent that has created a requirements specification, the test derived from the requirements, the agent that executed the tests, the report where the test results are documented, etc.). By evolution, we mean that a chain of evidence can suffer changes (e.g., a requirement is changed), and thus evolve. As a result of the change, the chain of evidence might not any longer be adequate for safety certification (e.g., the related test cases might have to be updated). When evidence changes, it must be possible to determine whether the set of safety certification evidence for a system is still adequate or if new evidence and, therefore, re-execution of certification-related activities are necessary.

Scenarios in which safety evidence evolves have been discussed in the past [24]. One particular scenario that corresponds to the compositional certification is when there is an incomplete set of evidence. This is probably the most basic situation in which a chain of evidence might not be adequate. It corresponds to the development scenario in which evidence is gathered and structured for a new system. Therefore, evidence is collected, or at least structured, progressively. Until all the pieces of evidence that are part of a chain of evidence have not been gathered and structured, such a chain is inadequate.

The challenge here is to not only assess the totality of certification (qualification) achievable for each component or module, but also its certifiability once it is in an ‘integrated’ state. For example, if the safety argument relies, in part, on reasoning on the properties of the components, then the system build process should leave evidence that the system has been built out of the specific versions of each component for which there is evidence that the component has the said properties. Each step in developing the software
needs to preserve the chain of evidence on which the argument that the resulting system is dependable will be based. This is the basic principle of a compositional certification approach.

3.5 Summary

The common theme in this section is of the complexity in composing and integrating assurance data produced from different sources. This motivates our research into how to identify mismatches, gaps and matches when composing assurance data. One method of reducing this problem is to use restricted language, which as described in section 5.4. Another is to use fixed patterns for arguing assurance as shown in Section 5.3. Finally, we explore the needs of composition of evidence in Section 6.
4 Description of SACM/CCL model

This section presents an adaptation of the OMGs SACM meta-model, allowing us to present arguments in a graphical form similar to GSN, but using the richer concepts provided by SACM. This is an extension/development of the argumentation metamodel originally presented as part of the CCL [1].

4.1 High level class diagrams for OPENCROSS argumentation meta-model

The first point of note is the modifications made on the SACM argumentation metamodel in order to include concepts for the modular argumentation and for patterns. The modifications proposed here try on one hand to minimise the impact the actual SACM meta-model and at the same time include the concepts of modular GSN. Some modifications have been made also to facilitate the task of implementing the metamodel.

The changes made in order to fulfil needs for modular argumentation and patterns are highlighted in green while, the changes made in order to make it connect with other parts of the CCL metamodels are highlighted in blue.

Figure 8 Assurance Case class diagram
Figure 9 Argumentation Class Diagram
Figure 10 Relationships view diagram
4.2 Concepts

This section describes the classes from Figure 8 to Figure 10.

4.2.1 AssuranceCase class

An AssuranceCase element.

**Superclass**
ModelElement

**Attributes**
- name: String
  A globally unique identified to the current assurance case.

**Associations**
- hasEvidences:Evidences[0..*]
  The evidence component of the assurance case.
- hasArgument:Argumentation[0..*]
  The argument component of an assurance case.

**Semantics**
The AssuranceCase element represents a justified measure of confidence that a system will function as intended in its environment of use. Assurance cases can be parts of bigger assurance case. This is the case of modular approaches where an assurance case module is included on a higher-level assurance case. When a component is integrated in a system, so does its assurance case and a component related the assurance case can be included in the system assurance case.

**GraphicalNotation**
None

4.2.2 Agreement class

It is an specialization of an AssuranceCase element.

**Superclass**
AssuranceCase

**Attributes**
none

**Associations**
- between: Argumentation[2..*]
  The argument components which conform the parts of the agreement.

**Semantics**
The Agreement element represents agreements between parts (Argumentation). Agreements are done between two or more Argumentation parts. It includes the premises and promises validated when both Argumentation are integrated.

**GraphicalNotation**

4.2.3 ArgumentationElement class (abstract)

An ArgumentationElement is the top level element of the hierarchy for argumentation elements.

**Superclass**
ModelElement

**Attributes**
- description: String
A description of the Argumentation entity.

- content: String
  Supporting content of the Argumentation entity.

**Semantics**
The ArgumentationElement is a common class for all elements within a structured argument.

**Graphical Notation**
None

### 4.2.4 Argumentation Class

The Argumentation Class is the container class for a structured argument. It can be understood either as the whole argumentation of an assurance case or by an argumentation module. These modules can contain another modules.

**Superclass**
ModelElement

**Attributes**

- location: String
  It identifies where a module of an argumentation is stored in order to be reused.

**Associations**

- consistOf:ArgumentElement[0..*]
  The ArgumentElements contained in a given instance of an Argumentation.

- contains:Argumentation[0..*]
  The nested Argumentation contained in a given instance of an Argumentation.

**Semantics**
Structured arguments represented using the Argumentation Metamodel are composed of ArgumentElements. Argumentation elements can be nested, in this case we can talk about argumentation modules that contain elements of argumentation.

For example, arguments can be established through the composition of Claims (propositions) and the AssertedInferences between those Claims.

Another example can be seen as an argumentation module which contains a composition of Claims as before but in this case, this argumentation module is composed by a set of Claims, InformationElementCitations and/or ArgumentElementCitation.

**Graphical Notation**

![Figure 11 Argumentation Class Example](image)

### 4.2.5 ArgumentElement Class (Abstract)

The ArgumentElement Class is the abstract class for the elements of any structured argument represented using the Argumentation Metamodel.

**Superclass**
ArgumentationElement

Semantics
ArgumentElements represent the constituent building blocks of any structured Argument. For example, ArgumentElements can represent the Claims and their structure made within a structured Argument.

GraphicalNotation
None

4.2.6 ReasoningElement Class (Abstract)

The ArgumentElement Class is the abstract class for the elements of any structured argument represented using the Argumentation Metamodel.

Superclass
ArgumentationElement

Semantics
ArgumentElements represent the constituent building blocks of any structured Argument. For example, ArgumentElements can represent the Claims and their structure made within a structured Argument.

GraphicalNotation
None

4.2.7 Assertion Class (Abstract)

The ArgumentElement Class is the abstract class for the elements of any structured argument represented using the Argumentation Metamodel.

Superclass
ArgumentationElement

Semantics
ArgumentElements represent the constituent building blocks of any structured Argument. For example, ArgumentElements can represent the Claims and their structure made within a structured Argument.

GraphicalNotation
None

4.2.8 InformationElementCitation Class

The InformationElementCitation Class enables the citation of a source that relates to the structured argument. The citation is made by the InformationElementCitation class. The declaration of relationship is made by the AssertedRelationship class (an AssertedContext or an AssertedEvidence relationship).

Superclass
ArgumentElement

Attributes
- url: String
  An attribute recording a URL to external evidence.
- toBeInstantiated: Boolean
  It indicates whether the element needs to be instantiated specifically for the actual argumentation as it is part of a pattern or it just specifies the pattern.
- type: InformationElementType
  It indicates the typology of the information used.

Associations
- artefact:Artefact[0..*]
  The artefacts referenced by the current InformationElementCitation object. Artefact is a concept described in the Evidence model.
It is necessary to be able to cite sources of information that support, provide context for, or provide additional description for the core reasoning of the recorded argument. InformationElementCitations allow the citation of this information within the structured argument, thereby allowing the relationship between this information and the argument to also be explicitly declared.

The url attribute is to be used only when the argumentation aspects of the SACM are complied with. If compliance is claimed against both the argumentation and evidence packages, then the association to Evidence::Artefact shall be used to reference evidence by means of a URL.

**Graphical Notation**

```
<table>
<thead>
<tr>
<th>type=&quot;context&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>type=&quot;solution&quot;</td>
</tr>
</tbody>
</table>
```

**Example**

```
ProcCon
Measurements on Processor A
```

```
SWAccReport
Evidence artefact report
```

**Figure 12 InformationElementCitation Class Example**

### 4.2.9 ArgumentElementCitation Class

The ArgumentElementCitation Class cites an Argumentation, or an ArgumentElement within another Argumentation, for use within the current Argumentation.

**Superclass**

ArgumentElement

**Attributes**

- **type**: CitationElementType
  - It indicates the typology of the information used.

**Associations**

- **citesElement**: ArgumentElement[0..*]
  - References an ArgumentElement within another Argument.

**Semantics**

Within the actual Argumentation (package) it can be useful to be able to cite elements of another Argumentation (i.e., ArgumentElements) to act as explicit proxies for those elements acting within the argumentation structure. For example, in supporting a Claim it may be useful to cite a Claim or InformationElementCitation declared within another Argumentation. It can also be useful to be able to cite entire Argumentations. For example, in supporting a Claim it may be useful to cite an existing (structured) Argumentation.

This concept is key to understand the modular argumentation. There are times when it becomes necessary to be able to make a reference from the argument of one case module to some defined context that exists within the boundary of another, or to a Claim that is supported within another argumentation structure.

**Graphical Notation**

```
<table>
<thead>
<tr>
<th>type=&quot;context&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>type=&quot;solution&quot;</td>
</tr>
<tr>
<td>type=&quot;claim&quot;</td>
</tr>
</tbody>
</table>
```
4.2.10 Claim Class

Claims are used to record the propositions of any structured Argumentation. Propositions are instances of statements that could be true or false, but cannot be true and false simultaneously.

**Superclass**
Assertion

**Attributes**
- **assumed**: Boolean
  An attribute recording whether the claim being made is declared as being assumed to be true rather than being supported by further reasoning
- **toBeSupported**: Boolean
  An attribute recording whether further reasoning has yet to be provided to support the Claim (e.g., further evidence to be cited).
- **public**: Boolean
  An attribute recording whether the preposition described in the claim is publicly visible to other arguments and this way is able to be references in other structures of argumentation.
- **toBeInstantiated**: Boolean
  An attribute recording whether the claim needs to be instantiated for the actual argumentation of is just the specification of a pattern

**Associations**
- **choice**: Choice[0..1]
  References a ChoiceElement. A claim can be decomposed in a choice of options

**Semantics**
The core of any argument is a series of claims (premises) that are asserted to provide sufficient reasoning to support a (higher-level) claim (a conclusion).

A Claim that is intentionally declared without any supporting evidence or argumentation can be declared as being assumed to be true. It is an assumption. However, it should be noted that a Claim that is not ‘assumed’ (i.e., assumed = false) is not being declared as false.

A Claim that is intentionally declared as requiring further evidence or argumentation can be denoted by setting toBeSupported to be true.

Claims are related with InformationElementCitation through the AssertedEvidence relationship. Claims are also related with other claims if decomposition is needed by the AssertedInference relationship.

Also if a claim is reference by a CitedElement, then this is done by the AssertedInference relationship. This is the case in modular argumentation when in an argumentation module a claim described in another argumentation module is cited. In this case the claim reference should have the public attribute equal true.

**Invariants**
Self.assumed and self.toBeSupported cannot both be true simultaneously

**Graphical Notation**

```
<table>
<thead>
<tr>
<th>assumed</th>
<th>toBeSupported</th>
<th>toBeInstantiated</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>assumed</td>
<td>true</td>
<td>toBeSupported</td>
</tr>
<tr>
<td>assumed</td>
<td>false</td>
<td>toBeInstantiated</td>
</tr>
</tbody>
</table>

assumed=false       assumed=false       assumed=false
```
4.2.11 EvidenceUseAssertion Class

A sub-type of Claim used to record propositions (assertions) made regarding an InformationElementCitation being used as supporting evidence to the Argument. This is intended to be used as an interface element to external evidence. An evidence use assertion is a minimal assertion (proposition) about an item of evidence, and there is no supporting argumentation being offered within the current structured argument.

Superclass
Claim

Semantics
Well supported arguments are those where evidence can be cited that is said to support the most fundamental claims of the argument. It is good practice that these fundamental claims of the argument state clearly the property that is said to exist in, be derived from, or be exhibited by the cited evidence. Where such claims are made these are said to be basic EvidenceUseAssertions.

4.2.12 ArgumentReasoning Class

ArgumentReasoning can be used to provide additional description or explanation of the asserted inference or challenge that connects one or more Claims (premises) to another Claim (conclusion). ArgumentReasoning elements are therefore related to AssertedInferences and AssertedChallenges. It is also possible that ArgumentReasoning elements can refer to other structured Arguments as a means of documenting the detail of the argument that establishes the asserted inferences.

Superclass
ReasoningElement

Attributes
- toBeSupported: Boolean
  An attribute recording whether further reasoning has yet to be provided to support the reasoning (e.g., further evidence to be cited).
- toBeInstantiated: Boolean
  An attribute recording whether the reasoning needs to be instantiated for the actual argumentation of is just the specification of a pattern

Associations
- hasStructure:Argument[0..1]
  Optional reference to another structured Argument to provide the detailed structure of the Argument being described by the ArgumentReasoning.

Semantics
The argument step that relates one or more Claims (premises) to another Claim (conclusion) may not always be obvious. In such cases ArgumentReasoning can be used to provide further description of the reasoning steps involved. An ArgumentReasoning can be relate with an InformationElementCitation through the AssertedContext relationship.

**Graphical Notation**

![Graphical Notation](image)

**Example**

```
S1
Argument by addressing all identified operating hazards
```

**Figure 14 ArgumentReasoning Class Example**

### 4.2.13 AssertedRelationship Class (Abstract)

The AssertedRelationship Class is the abstract association class that enables the ArgumentElements of any structured argument to be linked together. The linking together of ArgumentElements allows a user to declare the relationship that they assert to hold between these elements.

**Superclass**
Assertion

**Attributes**

- multiplicity: AssertedMultiplicityExtension

**Semantics**
In the SACM, the structure of an argument is declared through the linking together of primitive ArgumentElements. For example, a sufficient inference can be asserted to exist between two claims (“Claim A implies Claim B”) or sufficient evidence can be asserted to exist to support a claim (“Claim A is evidenced by Evidence B”). An inference asserted between two claims (A – the source – and B – the target) denotes that the truth of Claim A is said to infer the truth of Claim B.

### 4.2.14 AssertedInference Class

The AssertedInference association class records the inference that a user declares to exist between one or more Assertion (premises) and another Assertion (conclusion) or between Argument modules in order to define the argumentation architecture or structure. It is important to note that such a declaration is itself an assertion on behalf of the user.

**Superclass**
AssertedRelationship

**Attributes**

- multiplicity: AssertedMultiplicityExtension

An attribute used while specifying patterns to indicate whether the inference is multiple, optional or one to one.
• cardinality: String

An attribute used while specifying patterns to record the number of times the inference should be instantiated afterwards.

**Semantics**
The core structure of an argument is declared through the inferences that are asserted to exist between Assertions (e.g., Claims). For example, an AssertedInference can be said to exist between two claims ("Claim A implies Claim B"). An AssertedInference between two claims (A – the source – and B – the target) denotes that the truth of Claim A is said to infer the truth of Claim B.

An AssertedInference can relation a claim with another claim for example for decomposition needs. An AssertedInference can relation a claim with an ArgumentElementCitation when that cited element reference a Claim in another argumentation structure or module.

**Invariants**
context AssertedInference
inv SourceMustBeClaimOrArgumentElementCitation : self.source->forAll(s|s.oclIsTypeOf(Claim)) or t.ocllsTypeOf(InformationElementCitation))
inv TargetMustBeClaimOrAssertedRelationshipOrArgumentElementCitation : self.target -> forAll(t|t.ocllsTypeOf(Claim) or t.ocllsTypeOf(AssertedRelationship) or t.ocllsTypeOf(AssertedInferenceCitation))

**Graphical Notation**
multiplicity=normal  multiplicity=optional  multiplicity=multi

**Example**

Control system is safe

All identified hazards eliminated /sufficiently

Software developed to I.L. appropriate to hazards involved

**Figure 15 AssertedInference Class Example**

### 4.2.15 Choice Class

This class is a subtype of the AssertedInference Class. It is used to denote possible alternatives in satisfying an inference.

**Superclass**
AssertedInference

**Attributes**
• sourceMultiextension: AssertedMultiplicityExtension

An attribute used while specifying patterns to indicate whether the source of the inference is multiple, optional or one to one.

**Semantics**
It is used to denote possible alternatives in satisfying an inference. It can represent 1-of-n and m-of-n selection, an annotation indicating the nature of the choice to be made.
4.2.16 AssertedEvidence Class

The AssertedEvidence association class records the declaration that one or more items of Evidence (cited by InformationItems) provides information that helps establish the truth of a Claim. It is important to note that such a declaration is itself an assertion on behalf of the user. The information (cited by an InformationItem) may provide evidence for more than one Claim.

**Superclass**
AssertedRelationship

**Attributes**
- confidence:ConfidenceType
  It records the confidence given to the evidence, whether is can refute the claim related, or confirm or support or challenge it
- multiplicity: AssertedMultiplicityExtension
  An attribute used while specifying patterns to indicate whether the inference is multiple, optional or one to one.
- cardinality: String
  An attribute used while specifying patterns to record the number of times the inference should be instantiated afterwards.

**Semantics**
Where evidence (cited by InformationItems) exists that helps to establish the truth of a Claim in the argument, this relationship between the Claim and the evidence can be asserted by an AssertedEvidence association. An AssertedEvidence association between some information cited by an InformationElementCitation and a Claim (A – the source evidence cited – and B – the target claim) denotes that the evidence cited by A is said to help establish the truth of Claim B.

An AssertedEvidence can relate an InformationElementCitation with a Claim

**Invariants**
context AssertedEvidence
inv SourceMustBe InformationElementCitation : self.source->forAll(s|s.oclIsTypeOf(InformationElementCitation))
inv TargetMustBeClaimOrAssertedRelationship : self.target->forAll(t|t.oclIsTypeOf(Claim) or t.oclIsTypeOf(AssertedRelationship))

**Graphical Notation**
multiplicity=normal  multiplicity=optional  multiplicity=multi

**Example**

![Figure 17 AssertedEvidence Class Example](image)

System X is safe

Interactions between system functions are non-hazardous

All system functions are independent (no interactions)
4.2.17 AssertedContext Class

The AssertedContext association class declares that the information cited by an InformationElementCitation provides a context for the interpretation and definition of a Claim or ArgumentReasoning element.

**Superclass**
AssertedRelationship

**Attributes**
- multiplicity: AssertedMultiplicityExtension
  
  An attribute used while specifying patterns to indicate whether the context reference is multiple, optional or one to one.
- cardinality: String
  
  An attribute used while specifying patterns to record the number of times the context should be instantiated afterwards.

**Semantics**

Claim and ArgumentReasoning often need contextual information to be cited in order for the scope and definition of the reasoning to be easily interpreted. For example, a Claim can be said to be valid only in a defined context (“Claim A is asserted to be true only in a context as defined by the information cited by InformationItem B” or conversely “InformationItem B is the valid context for Claim A”). A declaration (AssertedContext) of context (InformationItem) for a ReasoningElement (A – the contextual InformationItem – and B – the ReasoningElement) denotes that A is asserted to be valid contextual information for B (i.e., A defines context where the reasoning presented by B holds true).

An AssertedContext can relation an InformationElementCitation with a ReasoningElement
An AssertedContext can relation an InformationElementCitation with a ArgumentElementCitation when that cited element reference a ReasoningElement in another argumentation structure or module.

**Invariants**
context AssertedContext
inv SourceMustBeInformationElementCitation : self.source -> forAll(s | s.oclIsTypeOf(InformationElementCitation))
inv TargetMustBeReasoningElementOrArgumentElementCitation : self.target -> forAll(t | t.oclIsTypeOf(ReasoningElement) or t.oclIsTypeOf(ArgumentElementCitation))

**Graphical Notation**

multiplicity=normal  multiplicity=optional  multiplicity=multi

Example

![Figure 18 AssertedContext Class Example](image)

4.2.18 AssertedChallenge Class

The AssertedChallenge association class records the challenge (i.e., counter-argument) that a user declares to exist between one or more Claims and another Claim. It is important to note that such a declaration is itself an assertion on behalf of the user.

**Superclass**
AssertedRelationship

**Semantics**
An AssertedChallenge by Claim A (source) to Claim B (target) denotes that the truth of Claim A challenges the truth of Claim B (i.e., Claim A leads towards the conclusion that Claim B is false). This concept is used on a review process in order to indicate the weakness of an assertion associated with a claim.

**Invariants**
context AssertedChallenge
inv SourceMustBeClaim : self.source->forAll(s | s.oclIsTypeOf(Claim))
inv TargetMustBeClaimOrAssertedRelationship : self.target->forAll(t | t.oclIsTypeOf(Claim) or t.oclIsTypeOf(AssertedRelationship))

**Graphical Notation**

![Diagram of AssertedChallenge](image)

**Figure 19 AssertedChallenge Class Example**

### 4.2.19 AssertedCounterEvidence Class

AssertedCounterEvidence can be used to associate evidence (cited by InformationElements) to a Claim, where this evidence is being asserted to infer that the Claim is false. It is important to note that such a declaration is itself an assertion on behalf of the user.

**Superclass**
AssertedRelationship

**Semantics**
An AssertedCounterEvidence association between some evidence cited by an InformationNode and a Claim (A – the source evidence cited – and B – the target claim) denotes that the evidence cited by A is counter-evidence to the truth of Claim B (i.e., Evidence A suggests the conclusion that Claim B is false).

**Invariants**
context AssertedCounterEvidence
inv SourceMustBeInformationElement : self.source->forAll(s | s.oclIsTypeOf(InformationElement))
inv TargetMustBeClaimOr AssertedRelationship : self.target->forAll(t | t.oclIsTypeOf(Claim) or t.oclIsTypeOf(AssertedRelationship))

**Graphical Notation**

![Diagram of AssertedCounterEvidence](image)
Example

SW is safe

SW is from a reputable supplier

Supplier reputation is not guarantee of safety

The records from the last 20 years indicate a correct safety methodology implantation

Figure 20 AssertedCounterEvidence Class Example
5 Argument Templates and Vocabulary

5.1 Introduction

This section of the document describes the compositional and modular argument approach being used within OPENCOSS.

First we introduce the overall strategy we are taking, and the assumptions on which it is based. Then we discuss the overall argument architecture, in terms of structure and argument modules. We then look at the likely content of these modules, and how they can be interlinked (via the use of controlled language and/or strict contract expressions). We show a few examples to demonstrate the principles.

5.2 Argumentation strategy

The assumptions and principles underpinning the compositional argument structure are now discussed. First we note that it is difficult to write *re-usable* arguments about specific evidence that must be composed for a specific system in order to be useful or to demonstrate certain system properties. However, we can write *re-usable patterns* for how such arguments may be structured. For example, fault trees will need to be combined for multiple components, in order to demonstrate a system level failure probability. We can write a re-usable pattern which highlights typical strengths and weaknesses of a fault tree. This can then be instantiated for a specific fault tree and potentially re-used for multiple arguments. We can also write patterns with suggested strategies for arguing about how fault trees can be successfully composed together. However, the specifics for a given project cannot be re-used. Argument patterns contain text to be instantiated (e.g. names of components) as well as choices of strategies to follow when decomposing claims. Figure 21 shows which items may or may not be re-used during the compositional argument process.
An argumentation class (also known as an argument module) is a collection of argument elements. We can encapsulate arguments associated with one component in a module, or in a set of modules, (note that the definition of a component is at the discretion of the manufacturer, and may be relatively small or consist of several smaller components). Each component must have at least one associated argument module in the final safety argument, however, an argument module may discuss multiple components (e.g. for composed evidence).

The re-usable arguments are not only created in order to capture the end point of the development process, but also during component development, to help drive the development process. In a traditional, top-down, approach we would drive system development from system requirements down to component design and then construction. In a bottom-up approach (with a previously developed component) we would reconstruct the design process and "retro-fit", matching system level requirements. These contrasting methods are shown in Figure 22:
5.2.1 Argument contents and architecture:

This section discusses the content, or subject matter, which should be expressed in the safety argument. There is a substantial body of work using top down, monolithic safety cases, however, very little has been published on the creation of modular, and re-usable arguments. Therefore, the important aim of this work is that we should still express the following content, but ensure we can modularise it, to support re-use and composability wherever possible.

Each argument contains a number of different strands of information. There are multiple ways to decompose a component argument to include/separate/link this information (each pattern contains multiple information), but we wish to balance the argument to separate:

1. Common concerns to all arguments namely
   a. Compliance (e.g. to a company process or standard).
   b. Confidence (in both how compelling the argument is, as well as the quality/provenance of the supporting evidence)
   c. Risk (showing that hazards have been adequately met)
d. Assumptions/Context - background information within which this argument was constructed.

2. Areas/strands of the development process
   a. High-level predictive analyses (for validation purposes and to drive the design, e.g. early FMEAs to derive safety requirements). Evidence and output from these is typically linked to further analyses and artefacts.
   b. Lower-level and confirmatory analyses or design artefacts (for verification purposes, e.g. static code analysis results, testing, code, functional properties). These may be more likely to be standalone, and not required as input to other items.

5.3 Overall Strategies for Modular Arguments - Architecture and Example Templates

The overall architectural structural concept for assurance is shown in Figure 24. The argument has been modularised to separate concerns for organisational purposes (dependent on the supply chain), and also to encapsulate similar concepts in a module. However, we still maintain the ability to trace safety requirements to verified properties of the individual, and integrated components. In Figure 24, the architecture has been presented in a grid format to show the different stakeholders and concepts being explored for each module. We split horizontally into modules from suppliers for each component (in the bottom half) and those from the system integrator (top half). The vertical splits are an extension of the ideas in [1], in which the authors separated the reasoning purpose of each of the parts of an argument. The items in the far left column are directly relevant to the safety function of the system. The middle and right hand columns contain the columns relating to backing arguments - both generic backing and also for compliance to standards.

Each module is now discussed in turn, with the following standard format (adapted from the established method by Kelly in [8] by including descriptions of references to other modules):

**Module Name**: The name of the module

**Module Intent**: Short textual summary of the module.

**Relies Upon**: The names of modules which are expected to support away goals.

**Provides to**: The names of modules which it is expected to support (via public goals)

**Example Content**: GSN examples and typical claims that will be provided. They are provided as patterns and include the following sections: Structure, Participant description, applicability, consequences, implementation and possible pitfalls.

**Please note**: each template argument is provide to suggest possible content and starting points to complete the various modules in the overall structure. In each case, filling in the argument structure on its own will not be sufficient to produce a complete system argument. Future work in Task 5.4 will present further guidance and methodological guide in this matter.
5.3.1 SysSafe

**Module Description:** This is the highest level module - describing the safety of the overall system. It describes the derivation of system safety requirements, from system hazards. It discusses decomposition and allocation of SSRs to various system parts. SSRs may include functionality, failure behaviour, configuration of the component, confidence in the component, reliability of the component etc.

**Relies Upon:** CompNinSys, SysSafeReq, SysAssump, SysCompliance

**Provides to:** None
Participant description:
The following table describes the text to be instantiated within various claims in the pattern. Then each of the goals is described in turn (the numbers refer to the goals within the diagrams).

### Table 1 SysSafe text to be instantiated.

<table>
<thead>
<tr>
<th>Text to be instantiated</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{acceptably safe}</td>
<td>Every safety critical system has residual risks (e.g. very unlikely events that are too costly to prevent or that have not been detected or mitigated, or the task itself is inherently risky, such as parachuting). This text should be instantiated to describe what the acceptable level of safety is, perhaps partly by design quality assurance, probability of certain events, or qualitative discussion.</td>
</tr>
<tr>
<td>{system}</td>
<td>The name or ID of the system itself.</td>
</tr>
<tr>
<td>{operating conditions}</td>
<td>The operating or deployment environment for the system, defined as a series of conditions.</td>
</tr>
<tr>
<td>{design assumptions}</td>
<td>The design assumptions - i.e. what was assumed during development, defined as a series of conditions. These should be compared with the actual operating conditions, and become important guarantees for safety. This may include further conditions than operational ones, e.g. the tools used for development, as this may impact on the ability to integrate certain components.</td>
</tr>
<tr>
<td>{safety requirements}</td>
<td>Descriptions of required behaviour of the system, and its constituent components, which mitigate, or prevent hazards within the system.</td>
</tr>
<tr>
<td>{component N}</td>
<td>An ID should be provided for each component within the system</td>
</tr>
<tr>
<td>{hazard(s)}</td>
<td>The name, or name of the list of hazards being addressed by a particular goal.</td>
</tr>
</tbody>
</table>
Table 2 SysSafe Participant Goals

<table>
<thead>
<tr>
<th>Element ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C100</td>
<td>The contextual description of what is &quot;acceptably safe&quot; for the whole argument.</td>
</tr>
<tr>
<td>G100</td>
<td>Top claim for the entire argument - that the system is acceptably safe to operate in the defined operating conditions.</td>
</tr>
<tr>
<td>C101</td>
<td>A full description or model of the system.</td>
</tr>
<tr>
<td>G101</td>
<td>The first claim to be developed in the system level safety argument, is to demonstrate that the hazards identified are valid. When developing this goal the author may appeal to many things - previous knowledge of similar systems, competency of staff, computer simulations, etc.</td>
</tr>
<tr>
<td>G102</td>
<td>If we have a valid list of hazards, then it follows that these should be examined in order to identify the safety requirements to manage them appropriately.</td>
</tr>
<tr>
<td>G111</td>
<td>This claim addresses completeness - that all hazards have been examined in turn, and in combinations considered if necessary.</td>
</tr>
<tr>
<td>G112</td>
<td>This claim is about the need to show the analysis performed was sufficient. Strictly speaking this may be a backing argument, but is listed here to be directly accessible.</td>
</tr>
<tr>
<td>G113</td>
<td>The initial analysis of hazards, and subsequent consideration of how the chosen components are used, may lead to emergent hazards. This claim shows that they should be addressed. It is optional - and only needed if there are emergent hazards. N.B. this may be related to G523 in CompNInSys.</td>
</tr>
<tr>
<td>G103</td>
<td>To meet this claim it must be shown that all the allocated safety requirements are met (on the assumption that all of them are essential for an acceptable level of system safety).</td>
</tr>
<tr>
<td>G501</td>
<td>This is an away goal link to a claim in CompNInSys about an individual component meeting its requirements. It is repeated as necessary.</td>
</tr>
<tr>
<td>G201</td>
<td>An away goal link to a backing argument about the validity and quality of the safety requirements allocations in SysSafeReq.</td>
</tr>
<tr>
<td>G301</td>
<td>An away goal link to SysAssump, which is a contract module ensuring that the assumed operating and design conditions for the components sufficiently meet those of the system. It also links to their definition.</td>
</tr>
<tr>
<td>G401</td>
<td>The final strand of the argument covers compliance or conformance to a standard. This is also a contract module, which ensure that compliance of components is either sufficient (or deficits are covered) to meet system assumptions.</td>
</tr>
</tbody>
</table>

**Applicability:** This type of argument should be generated for every system. The template may be reusable, although an instantiated version is not.

**Consequences:** A number of goals in this structure will need further development - G101, G112, and G113.

**Implementation and Possible pitfalls:** Further understanding of how best to integrate details on emergent hazards may be required. This pattern relies on a number of different away goals, and data that is defined elsewhere. Some understanding of the adequacy of these linked argument modules is required. Compliance to a standard is not a necessary claim to fulfil the structure unless the definitions of "acceptably safe" include the naming of standards (C100).

### 5.3.2 SysSafeReq

This module is a backing argument discussing the quality of the system safety requirements allocation.

**Relies Upon:** None

**Provides to:** SysSafe

**Example Structure:**
Participant description:

Table 3 SysSafeReq text to be instantiated.

<table>
<thead>
<tr>
<th>Text to be instantiated</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{safety requirements}</td>
<td>Descriptions of required behaviour of the system, and its constituent components, which mitigate, or prevent hazards within the system.</td>
</tr>
<tr>
<td>{technique}</td>
<td>The type of one or more processes or methods used for allocating and analysing the safety requirements.</td>
</tr>
</tbody>
</table>

Table 4 SysSafeReq Participant Goals

<table>
<thead>
<tr>
<th>Element ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G201</td>
<td>The top level goal, provided as a guarantee to other modules as required.</td>
</tr>
<tr>
<td>G211</td>
<td>A completeness claim - that each safety requirement is allocated to at least one component (it may be further broken down or met by multiple components if required). For example, a fuel tank measurement system may be provided by two independently functioning components, one measuring the depth of liquid and one measuring the weight.</td>
</tr>
<tr>
<td>G212</td>
<td>A claim that the techniques used have been applied properly. This also links to the work in WP6 of OPENCOSS on Evidence characterisation.</td>
</tr>
</tbody>
</table>

Applicability: This argument should be use whenever additional claims about the veracity of the safety requirements allocation is needed.

Consequences: This argument structure is meant to stimulate suggestions for how to argue about the quality of safety requirements allocation, rather than be a complete method. Further work may be needed by the argument integrator to complete a satisfactory argument.

Implementation and Possible pitfalls: One possible issue with applying this pattern is to ensure the most up to date version of the safety requirements allocation is being examined. There may also be a question of
what constitutes complete - whilst each safety requirement may be allocated once, it might be more
effective in a different way or in combinations. Therefore, some understanding and assessment of the
content of the allocations is needed. This also links to the work in WP6 of OPENCOSS on Evidence
characterisation.

5.3.3 CompNInSys

Module Description: This module should show how {component N} meets the allocated system safety
requirements. Ideally this will be matching of public goals and away goals at the component level to away
goals from SysSafe. In practice more complex alignment may be required.

Relies Upon: CompNDev

Provides to: SysSafe

Example Structure:

![Diagram showing the structure of CompNInSys module]

Participant description:

Table 5 CompNInSys text to be instantiated.

<table>
<thead>
<tr>
<th>Text to be instantiated</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{system}</td>
<td>The name or ID of the system itself.</td>
</tr>
<tr>
<td>{safety requirements}</td>
<td>Descriptions of required behaviour of the system, and its constituent components, which mitigate, or prevent hazards within the system.</td>
</tr>
<tr>
<td>{component N}</td>
<td>An ID should be provided for each component within the system</td>
</tr>
<tr>
<td>{hazard(s)}</td>
<td>The name, or name of the list of hazards being addressed by a particular goal.</td>
</tr>
<tr>
<td>{property}</td>
<td>A feature, function, or attribute of the component in question. These are wide</td>
</tr>
</tbody>
</table>
ranging and may be grouped. E.g. failure rate, response to fault,

Table 6 CompNinSys Participant Goals

<table>
<thead>
<tr>
<th>Element ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G501</td>
<td>The top level goal is to ensure that the component in question sufficiently meets the safety requirement. The definition of sufficient is very wide and varied, and is not discussed here. It will relate to the definition of adequacy from the other suppliers though.</td>
</tr>
<tr>
<td>Str501</td>
<td>The argument strategy chosen in this example template is to iterate over each safety requirement, or group them if possible (for example requirements relating to a certain type of failure).</td>
</tr>
<tr>
<td>G511, G512</td>
<td>There is a choice between using either G511 or G512 as sub-claims, depending on whether the safety requirement is completely covered by one component, or by the combined properties of multiple components. G511 reflects the simpler case, 512 the more complex version.</td>
</tr>
<tr>
<td>G601</td>
<td>This is an away goal link to a claim in another module about a component's property of interest. This structure doesn't show any additional reasoning that may be required here, for example, a claim about why a certain property is felt to meet the safety requirement.</td>
</tr>
<tr>
<td>G521</td>
<td>If the evidence about a particular property is lacking, then additional evidence may be required (this covers both properties of a single component, or of multiple components). It is presented as a public goal here, for reference in other modules e.g. SysCompliance, but it may also lead to the creation of a new module if necessary.</td>
</tr>
<tr>
<td>G513</td>
<td>When re-using a component (particularly a generic, cross domain component) it is likely that it provides more properties than those actually needed to support the system safety requirements. This optional argument strand covers this situation.</td>
</tr>
<tr>
<td>G522, G523</td>
<td>One of these goals should be developed further to support the parent goal - either showing that the additional behaviour doesn't impact on the required behaviour, or that any emergent hazards or failure modes are managed. The latter claim (G523) is public, as it may be required to fulfil the SysSafe module (G113).</td>
</tr>
</tbody>
</table>

**Applicability:** This is an extremely important module for the argument, and forms the contract "glue" between system claims (what is needed) and component claims (what is offered). It will be needed for every safety requirement and every component in the system.

**Consequences:** As a consequence of instantiating this pattern, G512, G522 or G523 may need further development. In addition, further requirements for evidence may have been generated.

**Implementation and possible pitfalls:** The developer of this argument should consider how well each component meets the requirements (and possible rely/guarantee relations between it and another component).

For example, component N may offer some behaviour to manage a failure which is of significance to the system, but rely on component M passing on information about that behaviour. A more specific example might be that component N provides a warning due to an overrunning process, but component M must detect that overrun. The evidence about component M may be weak. How well does the argument stand up in this situation?

There is an issue with this argument that loops of logic may appear (particularly when two or more components meet a single safety requirement, and are mentioned in several strands of the argument). An additional pitfall is that the information and features are examined in such fine detail that an enormous argument is produced which is very difficult to check for errors and contradictions. The trick here is to
identify properties at the correct level of abstraction, and rely on evidence of groups of properties when appropriate.

5.3.4 CompNDev

Module Description: This module exposes the component properties, and how they were either derived or verified. It consists of public goals only, but is linked closely to backing arguments about compliance and specification validity.

Relies Upon: None
Provides to: CompNInSys, CompNAssump, CompNCompliance
Example Structure:

![Diagram of the CompNDev Module Template Argument]

Participant description:
Table 7 CompNDev text to be instantiated.

<table>
<thead>
<tr>
<th>Text to be instantiated</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(component N)</td>
<td>An ID should be provided for each component within the system</td>
</tr>
<tr>
<td>(development tier)</td>
<td>A specified phase of development (e.g. design, implementation, build, test). See Hawkins existing patterns in [26].</td>
</tr>
<tr>
<td>(technique)</td>
<td>A defined method for demonstrating a feature (static analysis), for producing part of a component (manufacturing standard, compiler).</td>
</tr>
<tr>
<td>(property)</td>
<td>A feature, function, or attribute of the component in question. These are wide ranging and may be grouped. E.g. failure rate, response to fault,</td>
</tr>
<tr>
<td>(artefact)</td>
<td>A link to a piece of evidence, such as the results of running a technique (e.g. source code, object code, fault tree etc.).</td>
</tr>
<tr>
<td>(sufficiently)</td>
<td>A measure of how well a component has been demonstrated to meet its specification. There are many ways of doing this, e.g. by appealing to standards for some degree of quality, using a formal proof of code to specification, test data for specified failure rates of a physical component etc.</td>
</tr>
<tr>
<td>(specification)</td>
<td>A clear, consistent, description of the component and all its properties.</td>
</tr>
</tbody>
</table>

Table 8 CompNDev Participant Goals

<table>
<thead>
<tr>
<th>Element ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G601</td>
<td>Captures the top claim of this argument - that a component has met its specification.</td>
</tr>
<tr>
<td>C601</td>
<td>Publicly available specification of the component.</td>
</tr>
<tr>
<td>C602</td>
<td>Definition of sufficiently.</td>
</tr>
<tr>
<td>Str601</td>
<td>The argument is to be broken down over a series of properties. There may need to be additional argumentation to show that these properties cover all of the specification. These properties need to be preserved throughout the development process hence each tier is considered.</td>
</tr>
<tr>
<td>G621, G622</td>
<td>There is a choice between arguing that a property has been demonstrated at this tier, or saying that the property is not relevant at this tier.</td>
</tr>
<tr>
<td>Sol621</td>
<td>Publicly available link to an evidence artefact, supporting the claims in G621.</td>
</tr>
</tbody>
</table>

**Applicability:** Every component must have some demonstration that it meets a specification. The rigour required to show this may depend on its expected integrity (e.g. a component for low integrity systems may not as exacting evidence as a high integrity component).

**Consequences:** After instantiating this template there will be a large number of instantiated versions of G601, G621 and G622. The total of G621 and G622 should equal that for G601. G601 should appear as many times as there are properties.

**Implementation and Possible pitfalls:** Although G621 is shown here as directly linking to Sol621, it should ideally be expanded to justify why the artefact demonstrates the property presented.

### 5.3.5 SysAssump

**Module Description:** This module should reconcile assumptions from the different suppliers. Assumptions is a wide ranging term here, and includes all contextual information, such as deployment environment (e.g. for road use, within a certain temperature range, given processor), development environment (e.g. factory conditions, development tools).

**Relies Upon:** CompNSuppAssump

**Provides to:** SysSafe

**Example Structure:**

**Participant description:**
Table 9 SysAssump text to be instantiated.

<table>
<thead>
<tr>
<th>Text to be instantiated</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{component N}</td>
<td>An ID should be provided for each component within the system</td>
</tr>
<tr>
<td>{type(s) of assumptions}</td>
<td>Dependent on the claim being instantiated, this should either be a reference to the whole set of assumption types, or a reference to a single type. E.g. a reference to a documented list, or a link to a row in a table within that document.</td>
</tr>
<tr>
<td>{system}</td>
<td>The name or ID of the system itself.</td>
</tr>
</tbody>
</table>

Table 10 SysAssump Participant Goals

<table>
<thead>
<tr>
<th>Element ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G301</td>
<td>The top goal in this argument module reflects its purpose - to ensure that the assumptions made during component development are sufficiently similar to those for the system they are used in.</td>
</tr>
<tr>
<td>Str301</td>
<td>The strategy chosen for this example is to consider types of assumptions in turn, grouped as appropriate. For example, all assumptions relating to software timing properties, or corrosion rates of equipment.</td>
</tr>
<tr>
<td>G311</td>
<td>This is a claim about the specific, verified, list of assumptions for the system which needs to be checked for each component.</td>
</tr>
<tr>
<td>Sol311</td>
<td>A link to the list.</td>
</tr>
<tr>
<td>G701</td>
<td>An away goal link to a public goal in CompNSuppAssump which should demonstrate that a component was developed with a particular assumption (not whether that assumption was correct for this system - see G313). Assumptions become contextual &quot;rely&quot; information to a component, if the assumptions are not then integrating the argument modules (and the components themselves) will be invalid.</td>
</tr>
<tr>
<td>G312</td>
<td>An alternative claim to G701, is that a particular assumption type is not relevant for a given component. This would need some kind of expansion as necessary, but for comp eteness all components need to be considered (obviously, some physical characteristics of the system may have no bearing on a purely logical component).</td>
</tr>
<tr>
<td>G313</td>
<td>This part of the pattern will need the most detailed development. It is unlikely that the lists of assumptions, and types of assumptions for the components and systems will be easy to directly compare. They may be formulated at different levels of abstraction, or (even when easily comparable) have some mismatches. Part of the work described in the next sections of this document (5.4) is an attempt to make this type of matching process simpler.</td>
</tr>
</tbody>
</table>

**Applicability:** This argument is required whenever there is a potential issue with mismatched contextual information and assumptions. I.e. whenever components have been developed independently.

**Consequences:** After implementing this argument, there will be a need to develop G313 and several iterations of G312.

**Implementation and Possible pitfalls:** The difficult area in developing this argument is the need for a consistent set of assumptions, across both this and the CompNAssump module pattern.

### 5.3.6 CompNAssump

**Module Description:** Each supplier must provide a set of assumptions about their component. Following earlier discussions in this document we note two things:
1) The assumptions are likely to become "rely" conditions on {component N}
2) The term "assumptions" captures a large and varied amount of information. There are design assumptions, from which the component specification was developed. In addition there are usage and deployment assumptions, which need to be adhered to when the component is used (e.g. environmental
conditions, input output values, failure behaviour of other components, expected hazards of the system...). Further research into suitable starting lists for these is recommended. Some exist for certain domains e.g. in section 4.2.10 of DO-297 [13], which provides these for modular avionics systems.

This pattern is intended to be instantiated several times, for each type of assumption the supplier believes is relevant. As such, CompNAssump as a module will consist of several re-usable fragments. It is anticipated that each of these should be considered for the SysAssump module in turn - and may lead to missing assumptions being identified for SysAssump.

**Relies Upon:** None  
**Provides to:** SysAssump  
**Example Structure:**

![Diagram](image)

**Participant description:**

**Table 11 CompNAssump text to be instantiated.**

<table>
<thead>
<tr>
<th>Text to be instantiated</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{component N}</td>
<td>An ID should be provided for each component within the system</td>
</tr>
<tr>
<td>{type(s) of assumptions}</td>
<td>Dependent on the claim being instantiated a reference to a single type of assumption. E.g. a reference to a documented list, or a link to a row in a table of assumptions.</td>
</tr>
<tr>
<td>{development tier}</td>
<td>A specified phase of development (e.g. design, implementation, build, test). See Hawkins existing patterns in [26].</td>
</tr>
<tr>
<td>{assumptions}</td>
<td>The full set of assumptions.</td>
</tr>
</tbody>
</table>
Table 12 CompNAssump Participant Goals

<table>
<thead>
<tr>
<th>Element ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G701</td>
<td>This is a public goal, designed to be linked to the SysAssump argument as necessary, and for every time it is instantiated.</td>
</tr>
<tr>
<td>Strat701</td>
<td>The strategy for this pattern is to look at whether the assumption is held to be valid for each specified tier of development.</td>
</tr>
<tr>
<td>G721, Sol721</td>
<td>There is choice of choosing this claim, which requires evidence that the assumption has been held to be true during this tier of development (and a link to that evidence) or G722</td>
</tr>
<tr>
<td>G722, Sol722</td>
<td>The alternative to G721, is to provide a justification as to why the particular assumption is not relevant for this tier.</td>
</tr>
<tr>
<td>C701</td>
<td>A link to the list of assumptions.</td>
</tr>
</tbody>
</table>

**Applicability:** Each component should have this argument pattern instantiated, as it is vital to understand the context and assumptions that were made when developing the component if we are to successfully integrate them.

**Consequences:** After instantiating this pattern, there should be several versions of G701 for each identified type of assumption in C701.

**Implementation and Possible pitfalls:** As for SysAssump, the main pitfall in instantiating this pattern is completeness of the assumptions. It is extremely difficult to fully capture the range of assumptions about deployment, particularly for a very generic component such as an operating system.

### 5.3.7 CompNSpec

**Module Description:** This module describes the specification of the component, with respect to the assumptions, and looks at the completeness and appropriateness of the specification that was produced.

**Relies Upon:** CompNAssump

**Provides to:** Potentially CompNDev

**Example Structure:**
Participant description:

Table 13 CompNSpec text to be instantiated.

<table>
<thead>
<tr>
<th>Text to be instantiated</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{component N}</td>
<td>An ID should be provided for each component within the system</td>
</tr>
<tr>
<td>{assumptions}</td>
<td>The full set of assumptions.</td>
</tr>
<tr>
<td>{specification}</td>
<td>A clear, consistent, description of the component and all its properties.</td>
</tr>
<tr>
<td>{property}</td>
<td>A feature, function, or attribute of the component in question. These are</td>
</tr>
<tr>
<td></td>
<td>wide ranging and may be grouped. E.g. failure rate, response to fault,</td>
</tr>
</tbody>
</table>

Table 14 CompNSpec Participant Goals

<table>
<thead>
<tr>
<th>Element ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G801</td>
<td>The top claim that the specification is complete, consistent and correct given the assumptions made.</td>
</tr>
<tr>
<td>C601</td>
<td>A link to the specification</td>
</tr>
<tr>
<td>C701</td>
<td>A link to the set of assumptions</td>
</tr>
<tr>
<td>Str801</td>
<td>The proposed strategy in this example is to rigorously consider each assumption and each property in turn, thus providing some confidence in completeness.</td>
</tr>
<tr>
<td>G811, G812</td>
<td>A choice of goals is presented, either an argument to be developed that the property in question reflects the assumptions, or to demonstrate that it is not relevant for the property.</td>
</tr>
</tbody>
</table>
**Applicability**: This argument is optional, but provides further of confidence that the assumptions listed by the developer are useful. By instantiating this argument the developer has an extra opportunity to review and consider both the specification and the assumptions.

**Consequences**: After instantiating this pattern a large number of G811 and G812 goals will need to be developed further.

**Implementation and Possible pitfalls**: It may be useful to group certain properties which are not relevant for a given assumption in order to make the argument simpler in presentation and development (e.g. no operating temperature properties are relevant to an assumption about input data correctness).

### 5.3.8 SysCompliance

**Module Description**: This module integrates the various pieces of compliance data and attempts to show that they are acceptable for the system compliance needs.

**Relies Upon**: CompNCompliance, CompNInSys

**Provides to**: SysSafe

**Example Structure**:

![Figure 31 SysCompliance Module Template Argument](image)

**Participant description**:

**Table 15 SysCompliance text to be instantiated.**

<table>
<thead>
<tr>
<th>Text to be instantiated</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{component N}</td>
<td>An ID should be provided for each component within the system</td>
</tr>
<tr>
<td>{stds/quality}</td>
<td>A specified development standard or quality document.</td>
</tr>
</tbody>
</table>

**Table 16 SysCompliance Participant Goals**

<table>
<thead>
<tr>
<th>Element ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G401</td>
<td>This claim summarises the aim of the module - to demonstrate that the processes used during development meet the expectations of the system</td>
</tr>
</tbody>
</table>
The strategy for this pattern is again divide and conquer - looking at each component in turn and considering the quality of the evidence, linking to additional evidence if it has been required.

This is an away goal link to CompNCompliance, for the top level claim about the compliance of a component to a given std.

There is a choice of claims for further development here, which cover the different evaluations of G901. Only one is required. G411 expresses the case where there is a compliance deficit (e.g. only compliance to SIL3 is shown when SIL4 may be ideal), however this can be shown to be acceptable (e.g. that many SIL3 components are used to meet an overall SIL4 requirement). G412 covers the case where the deficit has led to more evidence being generated. G413 covers the (ideal) case where the component is at least as compliant as needed for the overall system.

This is an optional away goal reference to CompNInSys, which is required if further evidence of the components combined behaviour with another component was generated (for example, end to end timing behaviour).

**Applicability:** Whenever there is a need for compliance to standards to be shown, an instantiation of this type of argument pattern is required. For example, in aviation compliance to DO178B guidance is typical practice to ensure certification of the airframe or an engine.

**Consequences:** After instantiating this argument one of G411, G412, and G413 will require further development. If G521 exists then it should also be explored/developed in order to determine the level of compliance of the related evidence.

**Implementation and Possible pitfalls:** Judging whether an instantiation of this pattern is satisfactory may be extremely subjective, and it may be hard to compare or assess compliance information. For example, if a fault tree has been generated then this may comply with the standard, but it still needs to be of sufficient quality to demonstrate the related properties.

### 5.3.9 CompNCompliance

**Module Description:** This module attacks the issue of how well the design artefacts produced during development (and the way they were conducted) comply with a specified standard.

**Relies Upon:** None

**Provides to:** SysCompliance

**Example Structure:**
Participant description:

Table 17 CompNCompliance text to be instantiated.

<table>
<thead>
<tr>
<th>Text to be instantiated</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(component N)</td>
<td>An ID should be provided for each component within the system</td>
</tr>
<tr>
<td>(stds/quality)</td>
<td>A specified development standard or quality document.</td>
</tr>
<tr>
<td>(development tier)</td>
<td>A specified phase of development (e.g. design, implementation, build, test). See Hawkins existing patterns in [26].</td>
</tr>
<tr>
<td>(technique)</td>
<td>A defined method for demonstrating a feature (static analysis), for producing part of a component (manufacturing standard, compiler).</td>
</tr>
<tr>
<td>(artefact)</td>
<td>A link to a piece of evidence, such as the results of running a technique (e.g. source code, object code, fault tree etc.).</td>
</tr>
<tr>
<td>(specified degree)</td>
<td>This needs to be instantiated on a case by case basis, dependent on how well the overall set of artefacts for a component complies with the std.</td>
</tr>
<tr>
<td>(std requirement)</td>
<td>From a list of stds requirements, as created from either a CCL standard model instantiation, or from the standard itself or company version of that standard.</td>
</tr>
</tbody>
</table>

Table 18 CompNCompliance Participant Goals

<table>
<thead>
<tr>
<th>Element ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G901</td>
<td>The aim of this argument is to establish that the component meets a std to a defined level.</td>
</tr>
<tr>
<td>Str901</td>
<td>The strategy in this example template is to examine the application of all</td>
</tr>
</tbody>
</table>
techniques at each tier of development (the latter allows the argument to be built up during development, and instantiated one at a time). There are alternatives, such as considering each standard requirement without development tiers, or a justification for why following one standard is in some way equivalent to another.

<table>
<thead>
<tr>
<th>G911, G912, G913</th>
<th>At this level of decomposition there is a choice between a) demonstrating that application of a technique is adequately compliant to the relevant std. b) showing that there is a deficit (note that this argument module doesn’t address whether that is acceptable or not - that’s only determined in SysCompliance) is public goal, or c) discussing why a particular (std requiremnt) was not met. One should be chosen for each {std requirement}, with associated {technique}, at each {tier}, and then developed to justify the choice. This will link to the work in WP6 on evidence characterisation for G911 and (12 C911) A contextual link to an artefact which is used as evidence/solution, as found in CompNDev. It is required information for G911 and G912.</th>
</tr>
</thead>
</table>

**Applicability:** It is anticipated that any developer producing a component to conform or comply with a specific standard would need to provide information/arguments about the degree of compliance.  
**Consequences:** After instantiating this pattern, a large number of versions of G911, G912 and G913 should be created for further development. This number should equal the number of defined std requirements.  
**Implementation and Possible pitfalls:** Describing how well a standard requirement has been fulfilled may need to appeal to qualitative arguments, which are hard to justify.

### 5.3.10 Variations

Future versions of these templates need to consider systems and sub-systems (i.e. have more levels of decomposition and heirarchy).

### 5.4 Controlled language and formalisation of claim

One area of concern which has been discussed in the previous sections, is to use a controlled language to express goals, and to make it easier to match claims in an argument, such as those found in the compliance arguments. The structure that has been put together allows us to argue about mismatches, however without the use of a strict vocabulary, it can be difficult to compare key concepts in argument modules (such as hazard, fault, failure, function...). This section of the document presents two alternative methods of addressing this. In the next section, 5.4.1, a strict sentence syntax and grammar is described, for describing properties in an argument. In section 5.4.2, a wider ranging vocabulary and claim taxonomy is presented which may be used for multiple different syntaxes. Both angles of this issue are being investigated by OPENCOSS.

#### 5.4.1 Method 1: Strict Syntax and Grammar

##### 5.4.1.1 Background

There are some identified reasons behind formalizing text, and multiple ways of expressing rely/guarantee conditions. [27] provides the following examples:

- Avoid human errors
- Support for validation or checking
- Interoperability between different suppliers
- Facilitate the integration of the components within the system
Previous projects have tried different approaches in order to try to reduce the ambiguity on contracts and promote the best practices on the domain. On safety assurance, the GSN community released a standard which include as an extension concepts of modular safety cases [15]. The UK IAWG (Industrial Avionics Working Group) have proposed a generic pattern for safety case contract modules in [28], they proposed that GSN might be used in order to capture the rationale behind the safety goal relationships, where an "away goal" requiring support in one module, cannot be directly mapped to a "public goal" elsewhere. This way strategies, justifications, and context are also included on the contract and the rationale is made explicit.

John Rushby [29] proposed formalised safety cases: “formalization of some elements may allow the context for human reviewers (e.g., assumptions) to be more precisely articulated and checked.” On this direction but from the design point of view the project SPEEDs [30] developed and implemented a formal meta-modeling language and the syntax of component contracts. These contracts define the premises and promises of the component in order to behave in a specific way and an attribute designating its viewpoint.

Another project proceed with the idea, CESAR [32] defined the CESAR Meta Model (CMM) that includes the concept of ‘rich’ components, which can be connected and integrated in hierarchies. There can be different kinds of rich components such as operational actors, functions, logical components or technical components depending on the perspective. The CMM is based on an integration of component-based design with contracts based on input from the SPEEDs project, EAST-ADL2 (traceability, verification and validation) from ATESS project and the own CESAR Requirements Management Meta-Model (RMM). CHESS project [32] also defined a component model but focusing on safety, reliability, performance and robustness characteristics.

Each of the approaches typically focuses on solving one objective. The following table shows some of the approaches already explored for improving the definition of contracts:

<table>
<thead>
<tr>
<th>Approach</th>
<th>Description</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Formal language</strong></td>
<td>Specification of a formal meta-modeling language for design contracts. It provides information about the component behaviour, variables and interfaces but not about the implementation</td>
<td>[30]</td>
</tr>
<tr>
<td></td>
<td>Specification of a formalization of safety cases. Safety argumentation can be logical deduction, probabilistic, expert judgement or historical experience. Formalizing some elements supports precision and checking methods</td>
<td>[29]</td>
</tr>
<tr>
<td><strong>Metamodell</strong></td>
<td>The ‘Rich Component’ Metamodel focuses on the integration of component-based design by the use of contracts from different perspectives: such as operational actors, functions, logical components or technical components.</td>
<td>[31]</td>
</tr>
<tr>
<td><strong>Reference architecture</strong></td>
<td>In different domains there have been initiatives to define a reference architecture with an open API e.g. AUTOSAR. These reference architectures can be decomposed into different components. The integration of these components is implementation independent and is aided by well-defined interfaces</td>
<td>[15][33]</td>
</tr>
<tr>
<td><strong>Properties modelling</strong></td>
<td>Formal and structured property modelling.</td>
<td>[34]</td>
</tr>
<tr>
<td><strong>Pattern</strong></td>
<td>Definition of a generic pattern for safety case contracts. They propose the GSN notation as a way to structure agreements between safety case modules.</td>
<td>[28]</td>
</tr>
</tbody>
</table>

Table 19 Different contract technical approaches
These approaches try to solve parts of the whole problem from different perspectives. Some approaches, such as those that concentrate on defining reference architectures, focus on design standardization and component integration rather than certification. (Although the argument can be made that they reduce the costs of development and certification through establishing standardized interfaces.)

Guidelines from the standards offer the best practices and interpretations of the standards in order to comply with certain requirements. Those best practices can be modelled within the different technical approaches and impact on the methodology for the system development. Different technical measures can be put into place in order to assure the correct and complete following of the guidance and practices.

OPENCOSS proposes to use formalized contracts through a well-defined and structured contract ‘grammar’ to support how users may systematically assure safety of their system while integrating components. In order to do so we propose the definition of a BNF (Backus Normal Form or Backus–Naur Form) grammar. In this structure we will take into account the different views of contracts. AC 20-148 [22] states that, "identify any installation, safety, operational, functional, or performance concern". We organise our contract grammar around these aspects to help identify such concerns. Fenn [28] proposes to use argumentation not only on safety cases but also on safety contracts, so our grammar should support argumentation. Rushby [29] has previously identified different types of argumentation. These types can be used to help provide extra structure to the argumentation aspects of the contract grammar. OPENCOSS has continued the work on that direction and enhance this work. The next section 5.4.1.2 has more information regarding this issue.

One of the benefits of formalizing safety contracts will be the possibly of tool support for checking or generating contracts. BNF (Backus Normal Form or Backus–Naur Form) notation can support the creation of a formal grammar that supports the technical approaches and instantiate the guidance from the standards. Moreover, with the provision of a defined grammar for safety contracts we will be able to support validation of contracts (e.g. helping identify incomplete contracts).

### 5.4.1.2 Strict Syntax Format and concepts

The use of contracts in component based development is a well-known approach in the development of complex systems. It is based on the idea of “divide and conquer” where a complex development is done by the sum of various smaller and more manageable blocks or components that together, conform to the whole. In this environment agreements are required.

Agreements, also known as contracts, are associated to two or more these blocks. An agreement consists of set argumentation assertions which defines constraints over the behaviours and compliance means. Regarding the components or modules associated with the agreement we can see as two different types:

- A deployable component that can consist of a set of software and/or hardware
- An argumentation module with not specific map of a deployable component but just part of the argumentation related to a specific subject.

OPENCOSS proposes to follow the following methodology when developing the system:

1. Develop a component or select an existing component for reuse
2. Specify the information included in a future contract regarding the component.
3. (Optional) If we are creating a contract between 2 or more components we will inherit the data specified by each of the components by itself
4. Analyse the inherited data in order to find correspondences and inconstancies. If any constraint is violated then the agreement will be invalid. For some premises a deeper analysis might be needed in order to confirm the compatibility.
Contracts are agreements between parts where some properties are guarantee based on the validity of some conditions. Contracts record agreements in terms of promises and premises. Contracts identify the different characteristics or which specifies behaviour for the components related where premises are valid and the promises or guarantees are ensured to be true.

Premises and promises are the core of the contracts. Premises need to be validated before the contract promises can be fulfilled. Those premises are typically identified at the component level. Promises can be made at component level but also new promises can appear as the integration of components enables new promises (regarding the composition of components) to be made. The documentation of assumptions and intended context of use are seen as premises here. They indicate the boundaries and operation conditions that ensure the correct and safe use of the component.

Promises and premises are closely interconnected. Guarantees identified at component level but promises that are not ensured and validated by contracts could make the contracts not valid. It is also important to consider behaviour, not only nominal behaviour but also failure and degraded behaviour are important to consider for both the safety contracts and assurance contracts.

- Premises
- Promises
- Reasoning
- Viewpoints

As it is shown the agreement is decomposed into 4 main parts:

```
'Agreement' name=ID
'modules:' (module+=ID)+
'premises:' '{'{assertionDefinition}*'}'
'promises:' '{'{assertionDefinition}*'}'
```

BNF (Backus Normal Form or Backus–Naur Form) notation can support the creation of a formal grammar that supports the technical approaches and instantiate the guidance from the standards. The below grammar is presented in BNF notation. For this reason we have used the conventions {def}* to denote 0 or more repetitions {def}+ or def to indicate at least 1 or more repetitions.
'reasoning:' '{strategyDefinition}*''

**Viewpoints**

We propose that safety contracts could also benefit from a multi-viewpoint approach. Flood and Habli [35] have proposed multi-view safety cases in order to facilitate the understanding of the safety argumentation abstracting those elements that are of interest for particular stakeholders.

When we look deeper on the standards requirements for concepts such as SEooC, a good example for modular certification, we see different types of information are required:

- On the scope
- On functional safety requirements
- On technical safety requirements
- On System level
- On the design
- On the development
- On hardware level
- On software level
- Work products for the ISO 26262 compliance

Other aerospace avionics guidelines such as AC 20-148 [22] concerning reusable software components indicate that in order to reuse components, stakeholders must identify any installation, safety, operational, functional and performance possible concerns. We can extract from that some viewpoints defined there but could increase in a future:

- Installation
- Safety
- Functionality
- Performance
- Timing

The degree of detail and concepts managed by each view is different however all the information is needed for the contract validation. This can be easily assimilated as the degree in complexity between the safety requirements and technical safety requirements. This is done at system level but also at component level by using the assumptions and guarantees concepts as a way to represent the differences in the granularity of the assumptions.

But all of these viewpoints are not complete isolated, premises and promises are inter-related. Even more, they linked to evidences and claims which argument safety of the system as a whole.

Using viewpoints will let us handle the different aspects in a unified framework, this way different types of information and with differences in detail in a common and systematic way structuring the information and this way helping to assure completeness.

**Premises & Promises**

Premises and Promises and be seen as the same concept from the grammar’s point of view although not from the semantic point of view. Premises describes the assumptions made by each of the modules where Promises describes the guarantees made by each of the modules. Both of them are defined in form of assertion in relation with one or more viewpoints.

```
assertionDefinition:
    assertion+= viewpointId* 'for' p+=assertionpattern;
```
Assertions are defined as an instantiation of a pattern already defined. These patterns are the connection to claim typology, and vocabulary that will be explained on the next section.

Assertion patterns are defined as:
assertionpattern : 
    assertionpattern= Noun v+=Verb f+=Fact
;

Assertion patterns can be focus on different topics, about

- Non-functional properties of a component or the whole system
- the behaviour of the component
- compliance about specific requirements from standards or guidelines that the component needs to assure.

This pattern are specified as sentences with nouns or phrasal nouns, verbs and facts. These nouns verbs and facts are inherited from the claim types that have been identified.

Premises examples:
- Failure condition FCa is assumed to lead to safety requirement R1 for the function f1
- The user should ensure that a PARTITION_ID value attributed for a partition is unique among all partitions of a module and for a given position

Promises examples:
- The WCET for component C1 is X units Un. Determination is defined in Doc1, method for determination is Method1 and validation can be checked on Ref2
- Component C1 has functionality F1 complies with requirement R1 and R3, method for validation is Method M2 and results for validation can be checked at Report1
- Component C1 has a level of compliance: partly compliance with standard DO178B objective O13, the specific mean of compliance is Analysis Ana1 to a level DAL C, the level of data is Available and the status is Approved

**Reasoning**

As discussed, the grammar should support argumentation. Premises and promises are deeply interconnected and so if a premise is not valid, a promise might be affected and not be guaranteed. In order to identify these dependencies and offering the possibility to indicate the rationale under some guarantee, we are able to use the reasoning concept. It follows two objectives

- Traceability of the impact of the premises
- Details of the implications of some guarantees

strategyDefinition:
    strategy+=STRING

The validation of assertions will be done by comparing each of the premises and promises. Some of the validation can be easily done by using a Boolean assertion of comparison (e.g. match, less than, greater than, etc.), however the matching and consistency checks for the general case are non-trivial due to the complexity and diversity of the kinds of assumptions.

If the assumptions are shown to be invalid, the impact analysis and the related configuration management and change management workflows will support the further modifications of the various safety work products for the envisioned aims.
5.4.2 Method 2: Content/Vocab/Claim Type

Argument modules and contracts used in the compositional argument approach established in this document will need to be populated using natural language vocabulary. For ease of communication between engineers on a given project, across a supply chain and in a regulatory context – we propose that the natural language used be constrained: i.e. that the meanings assigned to particular words and phrases should be clearly defined and rigorously adhered to in the OPENCOSS approach. Being clear about the precise meaning of terminology within a particular domain, organisation or project will enable OPENCOSS to have a more informed approach to the reuse of assurance artefacts.

Within OPENCOSS, the mechanism used to define the controlled language forms part of the Common Certification Language (the CCL). In this section, we describe the role and presentation of terminology within the CCL and explore how controlled vocabulary approaches can be used within the compositional argument approach.

5.4.2.1 Vocabulary in the CCL

As outlined in D4.4 [1], and summarized in the simplified overview presented in Figure 35 below, the CCL is a hybrid “language”, in which a hierarchy of models is instantiated from domain-specific vocabularies designed for each of the OPENCOSS target domains – avionics, railway and automotive.

The inter-related models that form the CCL metamodel [1] are presented at the conceptual level (Figure 35). The intention of this model is to define the key concepts required for the three core interrelated OPENCOSS concerns: safety argumentation, compliance management and evidence characterization. The conceptual model currently provides minimal information required to characterize these concepts, and indicates the key relationships between them, necessary for mapping standards, company standards and projects at the lower levels of the approach. These concepts are captured in a series of models in the current version of D4.4 [1], which also provides additional metamodels capturing concepts relating to mappings and vocabulary management.

The OPENCOSS approach also defines other models at lower levels of abstraction, as shown in Figure 36. These models represent the relationships between assurance artefacts at different levels of abstraction:
the safety standard, company best practice and guidance documents, and an individual project (which is usually developed to a company standard). As outlined in [1], these models are related to one another, and to the CCL Conceptual Model by “mapping” relationships, which enable us to capture subtle details of mismatches – partial mappings – between concepts, thus providing the basis for informed decisions about the logical relationships asserted between argument fragments, the limitations and scope of claims they make with respect to evidence and the degree to which rely-guarantee relationships required by argument contracts hold.

Figure 36 Levels of Model in OPENCOSS (from [1])

Mappings between concepts identified in models at these different levels will need to be recorded. The Internal Industry Model or Process (Level 1B – sometimes called the “Company Standard”) represents the company-specific process documents, best-practice documents, guidance and generic requirements to which the project is developed, and will typically include concepts drawn from a number of different standards, historical projects, internal documentation and so on.\(^1\) In general, the relationship between the Internal Industry Model and the Standard Model – which presents a model of an individual safety standard at Level 1) can be treated as an extension or a development – many of the concepts will be drawn from a particular standard, but there will be some additional material (perhaps drawn from other safety standards, for example). It is unlikely, but possible, that the Internal Industry Model will map directly to the Standard Model – i.e. it will be a clone of the standard, without additional concepts. In such cases, the “mapping” between Levels 1 and 1B can be taken as a “full mapping” (see sections 3.3 and 5.9 in [1]). OPENCOSS will

\(^1\) Indeed, it should be noted that the “Company Standard” modelled at Level 1B may not actually exist as a specific document, but may need to be derived from internal process guidance, background “general knowledge” etc.
also need to take account of the relationship between the Project Model (Level 2) and the Internal Industry Model (Level 1B). Ideally, this will be a simple instantiation (i.e. a “full mapping”). There may be cases where there is a direct mapping between the standard and the project (bypassing the Internal industry Model). These will require detailed analysis according to the “mapping” types established in D4.4, as no universal concept mapping can necessarily be asserted. In the cross-domain case, OPENCOSS will need to consider the relationships between concepts in the Internal Industry Model and the Standard Model for the reuse domain. For example, when a component developed according to DO-178B/C is being considered for reuse within the railway domain, the mapping between the Internal Industry Model for the component developers and CENELEC will need to be considered. If the project model is a simple instantiation of the Internal Model, the relationship between the project model and CENELEC can be inferred from this (if not, the relationship between the Project Model and the Internal Industry Model will also have to be considered in detail). The Internal Industry Model → Reuse Standard Model will always need to be discussed in terms of conceptual “mappings” of varying degrees of inexactitude.

The model-based aspects of the CCL can be seen as providing syntax for safety argumentation, compliance management and evidence characterization. For the concerns of assurance and asset reuse to be properly addressed, however there is also a need for semantics, in the sense of a richer understanding of the meanings of relevant concepts in the domain and of how they relate to one another. Figure 35 indicates that these meanings are addressed in OPENCOSS by the provision of a Vocabulary. The Vocabulary is orthogonal to the CCL Conceptual Model: as explained below, it addresses concepts additional to those presented there. It is, however, used in the reification of the various models presented in OPENCOSS, at levels 1, 1B and 2 in Figure 36. The vocabulary thus provides a controlled language definition of concepts used in the various domains of interest of OPENCOSS (railway, automotive and aerospace), and thus provides a basis for comparison (via a matching technology) and informed discussion (via controlled assurance arguments) of the artefacts and processes modelled in the project. Thus, words which have a clear definition in the vocabulary – for usage within a given domain – should be considered reserved words, only to be used with that meaning within the project.

Terminology to be included in the Vocabulary can be derived from the domain standards, from company standards and best practice documents, as well as from the individual projects. Providing the definitions for the terms is not an easy task: although some definitions can be taken from the glossaries of the standards or organisations, many relevant concepts are not defined, and considerable domain expertise and linguistic skill is required to ensure that the definitions provided are correct and adequate. It is also interesting to note that, in many cases, terms and concepts are used “in practice” (i.e. in the body text of the standards, or in company-specific information) in ways which differ – sometimes subtly – from the definitions published in the standards’ glossaries. For consistency and also to support machine-based reasoning, there is a need for terms and concepts to be defined with respect to one another. This issue is discussed in more detail in section 5.4.2.2 below.

Most of the Vocabulary will be structured per domain: i.e. there will be a separate vocabulary for the railway domain, one for avionics and one for automotive. The vocabulary will also be layered, so that it corresponds to the levels of abstraction indicated in Figure 36. That way, it will be possible for the differences between concepts in the different domains to be seen, and for the limitations on reuse between domains (and between projects) to be made clear. OPENCOSS will not, in general, try to create a unified language, but to provide “mappings” between the usages across domains and projects, where reuse is required. The theoretical basis for “mappings” is discussed in more detail in section 3.5 of D4.4, where three types of “mapping” are identified (full map, partial map and no map). A short example will be useful here to illustrate some of the difficulties.

Suppose that a manufacturer wishes to reuse some software developed according to IEC 61508 [38] in an avionics context, where certification to DO-178B [10] is required. A claim is made in the original 61508-based assurance case that “Software Module Y has been developed for a SIL 4 system”. The manufacturer
wishes to make a claim in the reuse context – i.e. in the DO-178B context – that “Software Component Y has been developed for a DAL A system”. Examination of the conceptual models for system and software architecture in both IEC 610508 and DO-178B (Figure 37 and Figure 38 respectively) reveal some important differences:

Figure 37 IEC-61508 Software Concepts

Figure 38 DO-178B Software Concepts
The most significant differences here relate to the nature of the Criticality Level (as the generic concept is conceived in the CCL conceptual model) and of the architecture of the system. In IEC 61508, a SIL is directly attached to a safety function (in this case, a software safety function) which is modelled at system (in this case, software system) level. In DO-178B, however, the Development Assurance Level (DAL) is associated directly with a software system or a 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 Safety Integrity Level (SIL) directly into a DAL without considering the extra process-related requirements that arise because of the focus in DO-178B on the structure of the system rather than merely its functionality. Although a clear understanding of the terminology can be helpful in addressing this difficulty, clear contextual understanding is required to understand the constraints on reuse here. In OPENCOSS terms, we should 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 a user to make informed decisions about the feasibility or limitations on reuse, and on what extra information is required.

In IEC 61508, the high-level concepts for software architecture include “software module”, a container class. The IEC 61508 glossary provides the following natural language definition of “software module”: “a construct that consists of procedures and/or data declarations and that can also interact with other such constructs”. Although 61508 clearly has a concept of “component”, this is not defined at all in the glossary, and quite likely refers to hardware. In DO-178B, it is clear that “software” is an aggregation of “software components”, to which a DAL can be applied directly. The DO-178B glossary defines “component” as follows: “a self-contained part, combination of parts, subassemblies or units, which performs a distinct function of a system”. Clearly, there is some cross-over with the concept of “software module”, which may be sufficient to consider a “full map” relation, depending on the detail at which the argument needs to consider the concept. The easiest way for this to be resolved in WP5, however, would be with the development of a generic Component Model for OPENCOSS, to which the terminology of the standards and domains can be mapped. This way, it can be ensured that the component-level argument modules to be used in the compositional argument structure are expressed at a similar level of detail, with scope over the same type of system element.

In practice, it may be that an element of unification of the vocabulary is possible, since some concepts will be used in exactly the same way across the three domains. It is likely, for example, that this may be the case with some of the language used for the discussion of safety concepts, or for the characterization of evidence, which may be generic. However, it cannot simply be assumed that terminology is used with exactly the same connotations or contextual understanding across the three domains, and any unification can only be carried out after the vocabulary has been properly modelled for each of the three domains.

There is also a generic Vocabulary which runs alongside each of these domain-specific vocabularies. This is the vocabulary of OPENCOSS-specific concepts, which will contain clear definitions of concepts and terminology required within the OPENCOSS Project itself. This terminology will include clear definitions of the generic concepts in the CCL conceptual model, the mapping types, evidence characterization criteria (to support justification of the suitability of evidence to support particular argument claims) and a standardised component model (to be used in compositional arguments).

The concepts identified in the CCL conceptual model cannot be considered as a complete set of all of the relevant concepts (or superconcepts) that are required for argumentation and reuse. The conceptual model represents vocabulary only in the plane of Assurance, in the sense of what the standard requires,

---

2 Note that the correct word to use here – in philosophical and linguistic terminology – is “domain”. However, it should be clear that the term “domain” already has a reserved meaning in OPENCOSS – to refer
artefacts etc. Standards-based assurance is, however, only one of the concerns which will need to be discussed in a compositional assurance argument, or in a reasoned justification of reuse. The vocabulary will therefore need to include concepts and terminology from a number of different planes, as indicated in Figure 39:

**Figure 39 Different aspects which need to be addressed by the Vocabulary**

- **Plane of Assurance**
  - Terminology relating to artefacts, activities and requirements
  - e.g. ASIL, Test Plan

- **Plane of System and Environment**
  - Terminology relating to the structure and nature of the system and its operating environment
  - e.g. Item, Element, State

- **Plane of Safety**
  - Terminology relating to the conceptualisation and demonstration of safety
  - e.g. Hazard, Fault, Failure, Controllability, Risk

The overall structure of the OPENCOSS Vocabulary will reflect division across the domains of interest to the project (railway, automotive, avionics) and the planes of interest to the safety argumentation and reuse (assurance, system/environment, safety). An additional vocabulary module containing definitions of the OPENCOSS-specific terminology will also be included. The vocabulary structure is presented in Figure 40.

to the application domains for the Platform (automotive, avionics and railway). We have therefore adopted ‘plane’ instead.
One final observation is required here. The preceding discussion has given the impression that it is straightforward to divide terminology into the three ‘planes of interest’ introduced here – i.e. to be absolute as to whether a concept belongs in the assurance plane or the safety plane. In reality, however, this is far from being the case. A simple example will illustrate the problem. In Section 3 of ISO 26262 [9] the nature of the harm or potential harm caused by a hazard is determined by calculations involving three qualities: severity, probability of exposure and controllability. The hazard is assigned a severity class (S0, S1, S2 or S3), a probability class (E0, E1, E2, E3 or E4) and a controllability class (C0, C1, C2 or C3), which is used to specify the ASIL relating to the function of an item or element. Each of the terms underlined here is a reserved term, with a meaning defined in the ISO 26262 vocabulary. Clearly, they are heavily intertwined, and definitions are mutually dependent. Yet terms from each of the three planes can be identified here:

Plane of Assurance: severity class (S0-S3), probability class (E0-E4), controllability class (C0-C3), ASIL
Plane of System/Environment: function, item, element
Plane of Safety: harm, potential harm, hazard, severity, probability of exposure, controllability

Strictly speaking, this means the definitions given above are ISO 26262 specific only and a direct mapping is not possible at all (which is a pre-condition for transparent certification). The only possibility is that the use of existing ISO-26262 compliant artefacts create a basis that will reduce the corresponding effort in another domain.

5.4.2.2 Proposed Technique for Representing Vocabulary

We propose the use of parts of the Semantics of Business Rules Vocabulary (SBVR) technique for the representation of the OPENCOSS Vocabulary. SBVR is a publicly available OMG Specification [36], which is integrated with the OMG Model-Driven Architecture approach. SBVR aims to provide a basis for a declarative, descriptive domain model for a business or similar organisation, by means of formalized definitions of concepts, together with rules and facts which define the relationships between them. SBVR is underpinned by an explicit model of formal logic, and thus provides a means for the capture of natural language expressions in a formal structure, which is suitable for machine processing. The OMG Standard for SVBR provides an XMI schema to allow for the interchange of rules and vocabularies between organisations and tools.

Two of the elements defined in SBVR are of particular significance to the OPENCOSS Vocabulary: “concepts” and “fact types”. In SBVR, a “concept” is defined as “a unit of knowledge created by a unique
combination of characteristics”. Generally, this equates to a noun, or a term (e.g. a phrase which has a distinct meaning in a given context). SBVR [Annex C of [36]] specifies a number of possible attributes of the concept which are required in a rich definition. The following have been adopted as suitable attributes for definition of a concept in the OPENCOSS Vocabulary:

Table 20 SBVR Attribute definitions (adapted from Annex C of [36])

<table>
<thead>
<tr>
<th>SBVR Attribute Name</th>
<th>Definition (from Annex C of [36])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary representation</td>
<td>The main form of the entry relating to the concept in the vocabulary. Primary representations can be terms, names or fact types (see below for a discussion of fact types).</td>
</tr>
<tr>
<td>Definition</td>
<td>An expression that can be logically substituted for the primary representation. In SBVR, a definition can be given in formal or informal notation. The aim in OPENCOSS is to use the formal, controlled English notation described below.</td>
</tr>
<tr>
<td>Source</td>
<td>An indication of the document or vocabulary in which the concept appears</td>
</tr>
<tr>
<td>Dictionary Basis</td>
<td>If the concept is defined in a commonly used natural language dictionary which supports the use of the primary representation according to the definition given in the SBVR specification, reference may be made to that here. Note that a dictionary basis is not necessarily the same as the definition given in the SBVR definition field, but merely provides external support for the usage imposed by that definition.</td>
</tr>
<tr>
<td>General Concept</td>
<td>A concept, elsewhere in the SBVR definition, which generalizes the concept defined in the entry.</td>
</tr>
<tr>
<td>Concept Type:</td>
<td>This caption is used to specify the type of the entry concept.</td>
</tr>
<tr>
<td>Example</td>
<td>Examples of the usage of the concept according to the stated definition</td>
</tr>
<tr>
<td>Synonym</td>
<td>Another designation that can be substituted for the primary representation, for terms and names</td>
</tr>
<tr>
<td>Synonymous Form</td>
<td>Another designation that can be substituted for the primary representation, for fact types</td>
</tr>
</tbody>
</table>

The General Concept and Concept Type attributes are particularly useful in OPENCOSS, since they allow us to specify hierarchical type-relationships between concepts. This typing will be particularly significant in the disambiguation of terminological mismatches in cross-domain “translation” scenarios. For OPENCOSS, we also add a third ‘type’ relationship to the SBVR definition. Where a concept is defined in – or specialises – the CCL Conceptual Model, an SVBR attribute “CCL Concept Type” is added to record this fact, allowing for traceability between the CCL Conceptual Model and relevant vocabulary definitions.

Relationships between elements are captured by “fact types”. A fact type is an extension of a concept, defined in [36] as follows: “a concept that is 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, 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.

Fact types form the basis of SBVR “rules”, which are defined as “elements of guidance which introduce an obligation or a necessity”. Two classes of obligation are introduced in the SBVR attribute definitions in Annex C of [36]. The ‘Necessity’ caption is used to state something that is necessarily true, while the
‘Possibility’ caption captures the fact that something is not prevented by the definition (i.e. it is possible). [3] defines keywords for the phrasing of these rules in accordance with formal logic, e.g. to capture concepts such as quantification, modal operands and logical operations (see Annex C of [36], particularly section C1.1.1).

SBVR defines a form of Structured English to represent concepts, rules and fact types, to ensure logical completeness (when the definitions used are fully formal) and consistency for machine-processing. In addition to the reserved words defined in the SBVR standard, four font types are used in the SBVR model. Again, the significance and use of the font types is discussed in full in Annex C of [36] – the explanation here is not intended to be comprehensive but to give an overview of the notation for OPENCOSS partners. Table 21 demonstrates the font types:

Table 21 SBVR Font Types (from Annex C of [36])

<table>
<thead>
<tr>
<th>SBVR Font Type</th>
<th>Explanation (from Annex C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>term</td>
<td>Term font is used to designate a (general) noun concept</td>
</tr>
<tr>
<td>Name</td>
<td>Name font is used to designate an individual concept – a name.</td>
</tr>
<tr>
<td>verb</td>
<td>The verb font is used to designate fact types – usually a verb, a preposition or a combination thereof. In vocabulary entries, fact type forms usually use singular active forms of verbs</td>
</tr>
<tr>
<td>keyword</td>
<td>The keyword font is used to represented reserved terms used to construct statements (e.g. ‘each’ and ‘it is obligatory that’. These keywords are defined in the formal logic model in [36].</td>
</tr>
</tbody>
</table>

SBVR has been proposed as the means to represent the vocabulary parts of the CCL for three reasons. Firstly, it provides a means for the unambiguous definition of concepts of relevance to OPENCOSS and its target domains, by means of establishing reserved terms for use in the project. In principle, SBVR provides a mechanism for a fully formal definition of domain semantics – by the use of fully controlled vocabulary. In practice, however, a fully closed model is unlikely to be required by OPENCOSS, though more work is required to establish the lassitude which can be permitted in the vocabulary model, especially in terms of the support for cross-domain reuse. Secondly, the type relationships afforded by the General Concept, Concept Type and CCL Concept Type categories in the SBVR concept definitions allow for a clear account of typing relationships between concepts at different levels of abstraction. This will be an important aspect of the “mapping” on which the OPENCOSS approach relies. Finally, explicit ‘fact-type’ relationships between concepts (and their formalization in terms of rule structures) allows for the explicit capture of relationships between concepts (and also for explicit statements as to where relationships are not permitted). As discussed briefly in the next section, these are likely to be the basis for the specification of some of the permitted claim types and expressions in the argument templates developed for the compositional approach.

Work on the definition of domain-specific SBVR vocabularies for OPENCOSS is currently in its early stages. An SBVR model for Part 3 of ISO 26262 (The “Concept Phase”) is currently in preparation. As in the OMG SBVR standard [3], visual summaries are used along with the textual SBVR definitions to indicate the context in which the controlled definitions are offered, and to prompt users as to which related concept definitions they need to be aware of in order to understand a particular concept in its 26262 context. Figure 41 presents SBVR definitions of a group of concepts in the “Safety Plane” and the “Assurance Plane” from ISO 26262. Note that this work is not currently complete and requires validation with domain specialists.

3 Note that, although the SBVR specification [3] does mandate the use of colours as part of the font style definition, this aspect is generally treated as non-essential in literature concerning the technique.
5.4.2.2.1 SBVR example for Automotive domain: text

**harm**
Definition: physical injury or damage to the health of traffic participants or material damage which arises from a hazard
Source: ISO 26262 Part 1, §1.57
Dictionary Basis: ISO 26262 Part 1, §1.56
Necessity: harm has severity

**harm has severity**
Definition: relation by which the effect of harm or potential harm is measurable

**Potential harm**
Definition: harm which is not actual, but which may arise from a hazard
Source: ISO 26262 Part 3, §7.4.3.2
Dictionary Basis:
General Concept: harm

**actual harm**
Definition: observable harm which arises from a hazard
Source: ISO 26262 Part 3, §7.4.3.2
General Concept: harm
injury
Definition: physical hurt caused to a traffic participant
Source: ISO 26262 Part 3, §7.4.3.3
General Concept: harm

material damage
Definition: detriment caused to vehicles, equipment or the environment
Source: ISO 26262 Part 3, §7.4.3.3
General Concept: harm

impact factor
Definition: quality used to assess the likely seriousness of a potential hazard in a given operational scenario
Source: ISO 26262 Part 3, §7.2

severity
Definition: qualitative estimate of the extent of potential harm or actual harm that may arise or has arisen from a hazard
Source: ISO 26262 Part 3, §7.4.3.2
Dictionary Basis: ISO 26262 Part 1, §1.120 (adapted)
General Concept: impact factor
Concept Type:
Necessity: severity of potential harm or actual harm arising from a given hazard is represented by at least one severity class

Probability of exposure
Definition: qualitative estimate of the likelihood that an item will be in an operational situation that could result in a given hazard if coincident with a given failure mode
Source: ISO 26262 Part 3, §7.2
Dictionary Basis: ISO 26262 Part 1, §1.37 [exposure] (adapted)
General Concept: impact factor
Concept Type:
Necessity: hazard is assigned to probability class
probability class is assigned on the basis of a combination of hazard and environmental conditions
Possibility:
Reference Scheme:
Note: actual number of vehicles equipped with the item under analysis not considered when estimating probability of exposure; exposure demonstration is based on a representative sample of operational situations for the target markets

controllability
Definition: qualitative estimate of the probability that the driver or other traffic participants would be able to take actions to avoid harm in the event of a hazard, possibly with support from external measures
Source: ISO 26262 Part 3, §7.2
Dictionary Basis: ISO 26262 Part 1, §1.19
General Concept: impact factor
Necessity: hazard is assigned to a controllability class
Controllability assumes driver characteristics

severity class
Definition: one of four levels reflecting the relative severity of harm or potential harm
Source: ISO 26262 Part 3, §7.4.3.2
Necessity: hazard is assigned to severity class
Possibility: severity class is assigned on the basis of a combination of injuries
severity class is assigned on the basis of a combination of instances of material damage
severity class is assigned on the basis of a combination of injuries and material damage

**Probability class**
Definition: one of five levels reflecting the relative likelihood that an item will be exposed to a given hazard in a particular operational scenario
Source: ISO 26262 Part 3, §7.4.3.4
General Concept:
Necessity: hazard is assigned to probability class

**Controllability class**
Definition: one of four levels reflecting the relative likelihood that the driver or other traffic participants would be able to take actions to avoid harm in the event of a hazard
Source: ISO 26262 Part 3, §7.4.3.7
Necessity: hazard is assigned to a controllability class

### 5.4.2.3 3. Claim Typology and Vocabulary Use in Compositional Argumentation

The compositional argumentation patterns presented in Section 5.3 above will be populated and instantiated using natural language statements. The natural language used will need to reflect the logical structure of the argument which is suggested by the GSN notation: unless the phrasing “inside the symbols” is coherent and adequately captures the argument, the logic of the discourse will not stand. Coherent use of terminology is particularly important in a compositional argument structure, where there is a need to “match up” argument fragments from a number of different sources (e.g. from a number of different component suppliers contributing aspects of an integrated system) to form an overall argument. The SBVR vocabulary model will provide consistent, specific definitions, which should help to ensure consistent terms of reference and comparison throughout the argument. The SBVR model should also provide an explicit definition of the context relevant to particular concepts within the specified domain.

**Table 22 Initial Typology of Claims**

<table>
<thead>
<tr>
<th>Claim Type</th>
<th>Definition</th>
<th>E.g. from patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity-Artefact Claim</td>
<td>claims relating to the production of particular artefacts as a result of particular activities</td>
<td>G212: &quot;{technique} used for (safety requirements) allocation has been applied methodically&quot;</td>
</tr>
<tr>
<td>Artefact Compliance Claims</td>
<td>claims relating to the necessity of particular artefacts for compliance.</td>
<td>G913: &quot;{component N} has not implemented {std requirement} at {tier} for {defined reasons}&quot; (note that {std requirement} may be for an activity or an artefact.)</td>
</tr>
<tr>
<td>Artefact Adequacy Claims</td>
<td>claims relating to the adequacy and appropriateness of particular artefacts – i.e. moving beyond compliance into a justification of the evidence artefacts provided. E.g. the adequacy of a fault tree</td>
<td>G211: &quot;Each {safety requirement} has been allocated to one or more components&quot;</td>
</tr>
<tr>
<td>Activity Compliance Claims</td>
<td>Claims relating to the necessity of particular activities for compliance</td>
<td>G913: &quot;{component N} has not implemented {std requirement} at {tier} for {defined reasons}&quot;</td>
</tr>
<tr>
<td><strong>Activity Adequacy Claims</strong></td>
<td>Claims relating to the adequacy and appropriateness of particular activities – e.g. the suitability of a particular analysis technique.</td>
<td>See WP6 deliverable [2] and section 6.3 for some discussions of adequacy.</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| **Component Development Claims** | Claims relating to the adequacy of the process by which a component has been developed | G712: "{type of assumption} is specified, valid, and verified at {development tier}"  
G621: "{property} is enforced in {tier} by application of {technique} as shown in {artefact}" |
| **Fault Accommodation Claims** | Claims relating to the accommodation or elimination of a fault | G522: "Additional functionality of {component N} doesn't contradict required functionality" or example in Figure 44. |
| **Hazard Mitigation Claims** | Claims relating to the adequacy of hazard mitigation achieved by safety measures in the design | G113: "Emergent {hazards} have been identified and addressed" |

One important means of constraining the vocabulary which can be used in an assurance argument is to specify the types of claim structures which can be used in an argument, possibly constraining their use at particular points in the argument. A typology of claim types can be superimposed on the general concerns of an argument structure outlined in the ‘layered’ patterns, presented in Table 22, to characterise the types of concepts which are discussed at a particular point and the features which are asserted as claims in the argument. Although this aspect of the language work in OPENCOSS is at an early stage, several types of claims – characterised by the kinds of concepts and relationships they make assertions about – have already been identified, as follows:

Further work is required to validate and extend this typology of claims, and to identify language structures to express them adequately. It is believed that the verb phrases indicating certain types of claims may be able to be isolated and generalized from existing GSN argument patterns [26] and from a study of publicly available assurance arguments from the OPENCOSS target domains. The Safety Case Repository assembled by the Dependability Research Group at the University of Virginia [37] provides several such arguments, which use a variety of graphical and natural language representations. Considerable expertise will be required to extract claims from natural language arguments and then to generalize them into reusable claim types. Although the identification of claims in GSN is substantially easier (due to the notation’s forcing of goal and context statement definitions), it should be noted that expertise will be needed to assess the suitability of the examples obtained from the survey: inclusion of an argument in a research repository indicates only that the argument is publicly available for study – not that it represents best, or even desirable, practice!

To some extent, verb phrases may be extractable from the SBVR fact types in the vocabulary models for the target domains. For claims of the Activity-Artefact type, for example, SBVR fact types should identify the
types of concept over which the claim might range. SBVR fact types are not adequate, however, as sources of claims without modification, for several reasons. Firstly, the SBVR fact types are descriptive – they provide a means of traversing the domain model in terms of the relationships between concepts. Argument claims are not descriptive – they are assertions regarding particular domain concepts and – in some cases – the relationships between them. Argument claims are subjective – they rely not on the truth or otherwise of descriptive statements, but on assertions relating to subjective concepts such as adequacy, confidence and appropriateness. Although the SBVR fact types may serve as the basis for the relationships between concepts, considerable expertise is required to adapt the resulting language structures to capture a subjective claim.

Once the verb phrases have been identified, a library of generic claim types can be identified. Rules can be specified to indicate which types of noun are required to instantiate the claim types implied by particular patterns. SBVR allows us to type nouns using the General Concept aspect of the concept definition. This means that there is some support for consistency checking of the relevance of language used in claims across the argument.

Claim types can be represented as GSN goals, parameterized with generic noun types drawn from the SBVR Concept Type hierarchy. Figure 42 represents a generic claim type referring to fault mitigation:

\[
\text{\{fault of type systematic fault\} is adequately mitigated by \{fault mitigation measure\}, which partially addresses \{hazard\}}
\]

Figure 42 Generic claim type

Obviously, this claim has an underlying conceptual model, which derives from the vocabulary of a standard, and relates a typology of faults to fault mitigation measures and faults to hazards. This model is presented in Figure 43 below. It should be noted that, although the model shares some characteristics with that derived from ISO 26262 for OPENCROSS, it is fictitious and has been devised for this illustrative example. Definitions of the concepts have not been provided in the example.
The first part of the claim in Figure 42 has been derived straightforwardly from the conceptual model – it asserts the relationship which is modelled between the “fault” and “fault mitigation measure” concepts. In a full SBVR definition, this relationship would be presented as a fact type, and the claim could be generated from the fact type. 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”.

The claim type is instantiated by populating the parametized noun phrases with concepts of appropriate types from the vocabulary models provided for the project, as described in Figure 36 above. One such instantiation is represented in Figure 44, demonstrating how the claim types can be used to construct argument fragments. The claim type is instantiated to form two claims relating to the satisfaction of a higher-level claim relating to hazard mitigation.
5.5 Summary

This section has introduced the conceptual compositional argumentation framework. This is being addressed in two ways:

1. The development of an argumentation strategy and architecture which supports the integration of various component level arguments into one system level argument. Several suggested templates for each section of the architecture are presented. These were developed, in part, in response to the needs of the case studies being explored in WP1.

2. The development of syntax and vocabulary to assist in the expression of claims within the argument. An example of how to apply this work was presented.

Further work instantiating and developing these concepts will be provided in the methodological guide of Task 5.4.
6 Evidence Sets

6.1 Introduction

Safety evidence plays an important role in providing confidence in the safe operation of the system during the safety certification and assurance process. Safety evidence can be broadly defined as “information or artefacts that contribute to developing confidence in the safe operation of a system” Evidence consists of a collection of documents that provide evidentiary support to a set of claims in an argument. In other words, evidence is information, based on established fact or expert judgment, which is presented to show that the claim to which it relates is valid (i.e., true) in the context of the argument. Anything that supports the claim can be presented as evidence. Often, this information is a record of some sort, demonstrating that a certain event or process took place. Evidence can be diverse as various things or artefacts may be produced as evidence, such as documents, expert testimony, test results, measurement results, records related to process, product, and people, etc.

A typically large system such as those used in the avionics, railway and automotive domains are often made of many components or sub-systems. For each subsystem or component there might be the need for collecting 1000s of these evidence artefacts from different components to demonstrate the safety of the system in a specific environment. Since safety is a system property, the judgement about the safety of the system as a whole relies heavily on the aggregate of confidence in different evidence from different sources (sub-systems or components) as shows in Figure 45.

![Figure 45 Combining different Evidence for Safety assessment](image-url)


6.2 Assessing Evidence Sets

Despite the progress in processes and methods targeted at facilitating and improving the safety assurance and certification of critical systems, there exist aspects of these activities that are inherently related to human judgment and cannot be fully automated. As shown in Figure 45, evidence artefacts from different components are composed to assess the combined safety of the system as a whole. When composing evidence information from various sources it is necessary to assess the evidence sets for certain properties:

- Completeness of Evidence set
- Sufficiency of Evidence set
- Confidence of Evidence set

These properties can be assessed in many ways, such as checklists, quantitative assessment, or its ability to support a structured safety argument (See D6.1 [2] and D6.2 [3]). Although most of these techniques depend on human decisions, research on confidence in system safety is at a very early stage, especially when compared to the research and results provided on human judgment in other fields.

A semi-structured interview questionnaire was developed to better understand the decision making process of assessing evidence sets. The in-depth interview study involved experts in various domains, who either provide or assess evidence information for safety certification or assessment process. The results of the interview study was then discussed with a focus group consisting of a panel of 15 international experts from three different domains namely avionics, railway and automotive. The aim of the study was to identify the various factors that are considered by the experts during safety certification and assurance to assess the acceptance of a safety evidence set or item. As a result we wanted to explore the possible automation support that can be provided for these factors.

6.2.1 Study design, Data collection and Analysis

The study was designed with a high-degree of discussion between the interviewer and interviewee that was later complimented with the focus group discussion. The experts were chosen with convenience sampling. Experts included system suppliers and safety assessors from within the project. They were contacted by e-mail and were introduced to the scope of the study. We chose seven experts from three domains namely avionics, railway and automotive. These were the experts who were contacted and were willing to be interviewed. The interview instrument was designed with questions that focused in understanding how the experts make the decision of accepting an evidence artefact. We tested the interview instrument with a pilot interview. Some questioned were clarified but none changed. The structure of the interview was tested and improved before the main interviews. A summary of the interview questions is available in 6.4.

The interviews were conducted on either skype or on phone. One interview was conducted at a time. The same interviewer conducted all interviews. The interviews were recorded for analyses later and notes were taken while the interview was being conducted. All interviews were in a time frame of 90 to 120 mins. The complete interview was not transcribed but rather important quotes were transcribed.
The interview data was analysed later. The content analysis involved marking and coding interesting sections of the recording. The coding strategy used was a mind map that involves a series of nodes and bubbles to represent the interview data.

### 6.3 Factors Considered for Assessing an Evidence Sets

The following section details some of the results obtained from the interview study with regards to evidence sets. The interview questionnaire included other questions related to specific evidence items that are not discussed in this document. This is to keep the discussion in the scope of compositional certification and with results with regards to argumentation.

Seven experts were interviewed in total for the study. Following are the list of experts and their domains.

- 2 System/ component suppliers from the Railway domain
- 2 Safety assessors from the Railway domain
- 1 Safety expert from the automotive domain
- 1 safety assessor from the Avionics domain
- 1 System/component supplier from the automotive domain

All the interviewees had extensive experience ranging from 5-15 years of experience in the field of safety engineering. We had chosen experts from three different domains so as to get a wide global picture of the state of the practice in evidence assessment.

One main goal of the study was to identify the various factors that experts take into account when assessing evidence sets. In relation to OPENCOSS, this was more of a requirements gathering process for the tool development and implementation of both WP5 dealing with arguments and WP6 dealing with evidence item per se. As a result of identifying the various factors that are taken into account during assessment of evidence sets and items, we explored on the possible level of automation support that can be provided to these factors in a tooling environment.

Several factors were identified from the coding schema used for analysing the interview data. An initial mind map coding of the interview data was made and used for discussion with a 15-member focus group. The focus group was made of experts in safety engineering from seven different countries and 3 different domains. The discussion in the focus group was a more informal discussion in which the preliminary results were presented the focus group. The discussion did not raise any new factors used for evidence assessment. The group agreed to the initial results presented and did not find fault with the initial results. Discussion was mostly about how explicit these factors were mentioned in the interview. In the sense that the group agreed that almost all the factors mentioned by the interviewees were commonly used for evidence assessment and mentioned that these are factors that are not normally mentioned out so explicitly. Many of the factors could be grouped into categories and hence categorisation of the factors was performed. 15 categories were identified from all the answers (commonly). The following sections define and discuss each of the factors that are commonly considered when assessing evidence sets.

#### 6.3.1 Experience

All the seven interviewees mentioned that there is a great deal of experience that plays a major role in assessing the evidence sets. We define Experience as “Knowledge or practical wisdom gained over a period of time from what one has observed, encountered or undergone” [39]. In terms of the level of automation support that can be provided for this factor, the associated data of experience can be collected and recorded but the factor cannot be fully automated into a tool.
6.3.2 Expertise

The interviewees answered a variation of the experience as mentioned above. They mentioned that working in a domain for several years alone won’t help in assessing evidence sets but the know how of the domain is very important. We categorised this as domain knowledge or more generally the expertise and it can be defined as “Valid knowledge, skills and the know-how in an area/domain or system” [39]. In terms of automation support, associated data can be recorded and a knowledge base can be developed and automated. However a number of challenges might have to be addressed like for example how to collect the knowledge and the time to train the system to self learn.

6.3.3 Historical Knowledge

Experts also mentioned the use of previous data about the evidence that gives them confidence. The previous use could be a positive or a negative one. They also mentioned that historical knowledge not just about the use of the evidence set gives them confidence but also the fact that it has been used else where previously before gives them the confidence that it’s a good enough evidence set. We define this as historical knowledge “Knowledge based on or reconstructed from an event in the past” [39]. For automation support this can be fully automated by storing and accessing in a data base the data and information pertaining to the past.

6.3.4 Provenance of person

All the experts mentioned that one of the major factors that they consider while assessing evidence set (apart from the evidence set itself) is the person responsible for creating and maintain the evidence set. We define this as “Characteristics of the evidence creator that correspond to the information related to his/her competence, skills, knowledge, capacity and qualification for the task” [39]. For automation support, associated data can be recorded and accessed for decision-making. Statistical inference could be made based on reviews/feedbacks or recommendations of the person. For example the tool could implement a sort of rating schema for each person involved in the project based on his experience, expertise, past performance and so on rated by the manager.

6.3.5 Provenance of tools

Similar to the provenance of the person, experts also mentioned that they gain confidence in the evidence sets based on the tools that were used to produce or generate these evidence sets. We define this as “Characteristics of the tool used to create the evidence that correspond to the quality, ability or accomplishment that makes the tool suitable for the task” [39]. The automation support could be very similar to the one provided for person.

6.3.6 Independence

All the experts agreed that there should be independence between the different teams, and members per se working in the project. We define independence as “Separation of responsibilities where the objectivity of the verification and validation processes is ensured by virtue of their "independence" from the development team”. In short, the person verifying the item (such as a requirement or source code) may not be the person who authored the item and this separation must be clearly documented. This factor can be implemented with full automation support in a tool with constraints stating that the same person cannot generate and evaluate the evidence sets.

6.3.7 Peer review

Six experts out of seven agreed that the evidence assessment or the evidence set itself should be peer reviewed. We define peer review as “Evaluation of a person’s work by one or more people of similar competence to the producer of the work.” [39]. This factor is hard to automate since this is a manual task. However automation support can be provided to handle the peer reviewing process for example allocating reviewers and enabling the review process.
6.3.8 Best practices

Experts mentioned that they assess safety evidence sets based on the best practices that are used to in generating them. We define best practice as “Methods or techniques that has consistently shown results superior to those achieved with other means, and that is used as a benchmark” [39]. Best practices cannot be fully automated with tool support however this can be facilitated.

6.3.9 Human emotions

Four out of seven experts agreed human emotions are involved when assessing some evidence sets. This is normally based on the mood and setting of the environment. We define this as “A mental state that arises spontaneously rather than through conscious effort and is often accompanied by physiological changes (normally “a feeling”)” [39]. This is a challenging factor to automate. The main reason being human emotions can be very inconsistent and unpredictable. Moreover emotions are very hard to capture in a machine or tool.

6.3.10 Personal relationship with people involved

Some experts mentioned that they gain confidence in the sets of evidence from the personal relationship that they have with the people involved in generating or assessing the evidence. We define this as “A connection, association or involvement among the parties involved in the assessment process.” This might be a connection between the supplier and the assessor established in the past on working together in similar settings. This factor cannot be automated fully since human connections entail emotions.

6.3.11 Claim

One factor that was mentioned by all the interviewees was the claim that was associated to the evidence. We define claim as “Propositions being asserted in relation to system safety (or other safety-related system properties)” [18]. Checking the evidence association to the claim can be automated. However checking the rigor of the evidence to the claim and its sufficiency is hard. It still requires human or manual check. Hence this cannot be fully automated.

6.3.12 Argument

Another factor that was mentioned by all the experts for assessing evidence sets was argument of the use of the evidence for a particular claim. We define argument as “A body of information (or reasons) presented with the intention to establish one or more claims about system safety through the presentation of related supporting claims, pieces of evidence, and contextual information” [18]. In essence, an argument aims to justify the validity of a piece of evidence for a claim. Similar to claim, this cannot be fully automated, as the rigor of the argument is hard to verify, however support for the same can be provided.

6.3.13 Mental Checklist

When asked about how the experts assess the set of evidence, all replied with checklist as a common way of doing it. They mentioned that the checklists are normally not written down but more of a mental checklist obtained from experience and expertise. We define mental checklist as “A list of items, as names or tasks that are formulated from experience, expertise and know-hows of a domain/system, for comparison, verification, or other checking purposes”. This cannot be automated unless we manage to elicit the expert’s mental checklist into a formal checklist against which assessment of evidence can be performed.

6.3.14 Standard’s Checklist

Similar to a mental checklist, 3 experts (specially the safety assessors) mentioned that sometimes they use a checklist that is extracted from the safety standard that they show compliance to. These are normally safety requirements checklist coming from the safety standard. We define this as “A list of items, as names or tasks that are formulated from the requirements of a safety standard, for comparison, verification, or
other checking purposes”. This again cannot be automated unless all the requirements from the particular safety standard is extracted and converted into a tool-based checklist.

6.3.15 Safety culture

Some experts conveyed their opinion mentioning that sometimes they assess safety evidence sets based on the safety culture of the organisation. The experts mention that different organisations have different levels of safety culture developed over a period of time from the experience and knowledge of working in a particular domain. We define safety culture as “the ways in which safety is managed in the workplace, and often reflects "the attitudes, beliefs, perceptions and values that employees share in relation to safety” [40]. This cannot be automated into a tool since this is a factor that is bred over time in the working environment and mind set of the organisation.

6.3.16 Heuristics

The last category that we identified from the experts answers was heuristics. We identified that experts sometimes use trial and error method to assess evidence sets and this is purely based on heuristics. This is an ad-hoc process and does not follow any systematic schema. We define this as “Experimental or trial-and-error methods and techniques used for learning, discovery, or problem solving” [39]. The essence that this is an ad-hoc process makes it difficult to provide automation support for this factor.

Table 23 summarises the above discussed factors, their definition and the level of automation support that can be provided for the same.

6.4 Interview Questionnaire

Please note that only questions associated to sets of evidence are presented here. The interview discussion extended in the area of specific evidence type properties and how they are assessed. To keep the document in relevance to compositional certification and argumentation related work these questions are not presented here.

Completeness Set

• How do you determine whether the collection of evidence being offered up in an assessment (project) is complete?
• What is your definition of complete?
• What criteria and process do you use to judge completeness?
• Are the criteria and process you apply applied loosely or strictly? (For example, can you think of an occasion when it has been necessary to make an exception to your own criteria and process?)
• How do you ensure that the criteria and process are applied consistently?

Set vs. Item Differences

• How does the criteria and process applied for examining the evidence set (as a whole) vary, if it does, from the criteria and process applied for a specific item of evidence?

Completeness vs. Sufficiency Set

• Is there, in your opinion, a difference between judging the completeness of a set of evidence and the sufficiency of a set of evidence? What are the differences?

If (any hint of) YES:
Sufficiency Set

• How do you determine whether the collection of evidence being offered up in an assessment (project) is sufficient?
• What is your definition of `sufficient’?
• What criteria and process do you use to judge sufficiency?
• Are the criteria and process you apply applied loosely or strictly? (For example, can you think of an occasion when it has been necessary to make an exception to your own criteria and process?)
• How do you ensure that the criteria and process are applied consistently?

Sufficiency Set vs. Item

• How does the criteria and process applied for examining the sufficiency of the evidence set (as a whole) vary, if it does, from the criteria and process applied for a specific item of evidence?

Confidence Set

• How do you establish (your own) confidence in the collection of evidence being offered up in an assessment (project)?
• What factors influence this confidence (positively or negatively) in the evidence set?
• Where have they come from?
• Are these factors investigated systematically for each assessment?
• How do you ensure that these factors are examined consistently?
• How is confidence in the evidence set documented / recorded?
• Is it assessed qualitatively or quantitatively?

Confidence Set vs. Item

• How does the approach used for judging confidence in the evidence set (as a whole) vary from, if it does, approach used for judging confidence in a specific item of evidence?

Set – Item Confidence Relationship

• How do your judgment of confidence in the evidence set (as a whole) relate to your judgments of confidence in a specific item of evidence?

Improvements

• In your experience and opinion how could judgments of completeness, sufficiency and/or confidence in safety assessments be improved?
Table 23 Factors considered while assessing a set of evidence and the level of automation that can be achieved to implement them

<table>
<thead>
<tr>
<th>No</th>
<th>Factors influencing acceptance decisions</th>
<th>Descriptions of factors</th>
<th>Level of automation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Experience</td>
<td>Knowledge or practical wisdom gained over a period of time from what one has observed, encountered or undergone [39]</td>
<td>Semi-automated</td>
</tr>
<tr>
<td>2.</td>
<td>Expertise</td>
<td>Valid knowledge, skills and the know-how in an area/domain or system [39]</td>
<td>Fully-automated</td>
</tr>
<tr>
<td>3.</td>
<td>Historical Knowledge</td>
<td>Knowledge based on or reconstructed from an event in the past [39]</td>
<td>Fully-automated</td>
</tr>
<tr>
<td>4.</td>
<td>Provenance of person</td>
<td>Characteristics of the evidence creator that correspond to the information related to his/her competence, skills, knowledge, capacity and qualification for the task [39]</td>
<td>Fully-automated</td>
</tr>
<tr>
<td>5.</td>
<td>Provenance of tools</td>
<td>Characteristics of the tool used to create the evidence that correspond to the quality, ability or accomplishment that makes the tool suitable for the task [39]</td>
<td>Fully-automated</td>
</tr>
<tr>
<td>6.</td>
<td>Independence</td>
<td>Separation of responsibilities where the objectivity of the verification and validation processes is ensured by virtue of their &quot;independence&quot; from the development team. In short, the person verifying the item (such as a requirement or source code) may not be the person who authored the item and this separation must be clearly documented.</td>
<td>Fully-automated</td>
</tr>
<tr>
<td>7.</td>
<td>Peer review</td>
<td>Evaluation of a person’s work by one or more people of similar competence to the producer of the work. [39]</td>
<td>Cannot be automated</td>
</tr>
<tr>
<td>8.</td>
<td>Best practices</td>
<td>Methods or techniques that has consistently shown results superior to those achieved with other means, and that is used as a benchmark. [39]</td>
<td>Semi-automated</td>
</tr>
<tr>
<td>9.</td>
<td>Human emotions</td>
<td>A mental state that arises spontaneously rather than through conscious effort and is often accompanied by physiological changes (normally “a feeling”) [39]</td>
<td>Cannot be automated</td>
</tr>
<tr>
<td>10.</td>
<td>Personal relationship with people involved</td>
<td>A connection, association or involvement among the parties involved in the assessment process. This might be a connection between the supplier and the assessor established in the past on working together in similar settings.</td>
<td>Cannot be automated</td>
</tr>
<tr>
<td>11.</td>
<td>Claim</td>
<td>Propositions being asserted in relation to system safety [18](or other safety-related system properties)</td>
<td>Semi-automated</td>
</tr>
<tr>
<td>12.</td>
<td>Argument</td>
<td>A body of information (or reasons) presented with the intention to establish one or more claims about system safety through the presentation of related supporting claims, pieces of evidence, and contextual information [18]. In essence, an argument aims to justify the validity of a piece of evidence for a claim.</td>
<td>Semi-automated</td>
</tr>
<tr>
<td>13.</td>
<td>Mental Checklist</td>
<td>A list of items, as names or tasks that are formulated from experience, expertise and know-hows of a domain/system, for comparison, verification, or other checking purposes.</td>
<td>Cannot be automated</td>
</tr>
<tr>
<td>14.</td>
<td>Standard’s Checklist</td>
<td>A list of items, as names or tasks that are formulated from the requirements of a safety standard, for comparison, verification, or other checking purposes.</td>
<td>Fully-automated</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>14.</strong></td>
<td><strong>Safety culture</strong></td>
<td>Safety culture is the ways in which safety is managed in the workplace, and often reflects “the attitudes, beliefs, perceptions and values that employees share in relation to safety” [40]</td>
<td>Cannot be automated</td>
</tr>
<tr>
<td><strong>15.</strong></td>
<td><strong>Heuristics</strong></td>
<td>Experimental or trial-and-error methods and techniques used for learning, discovery, or problem solving [39]</td>
<td>Semi-automated</td>
</tr>
</tbody>
</table>
7 Conclusions

This report has presented the OPENCOSS conceptual framework for compositional assurance. This has consisted of a series of argument patterns for assurance, which can be combined and instantiated, to form a coherent system level argument. The argument covers safety function, backing arguments about confidence in processes and evidence, and compliance/conformance to standards. The syntax and semantics of claims within the argument has been defined, using example vocabularies to indicate how to ensure common understanding when different individuals have developed assurance data and artefacts.

The report also presented findings on a survey of industrial practitioners on assessment of sets of evidence, where the evidence has been gathered from different sources, but an overall understanding of its quality and confidence together is required. This identified several key factors of importance and assessed which types of assessment were automatable.

The following areas have been identified for future work:

1. Advice on a methodological guide for development of patterns and claims, to be provided in Task 5.4.
2. Further examples of instantiating patterns and claim types
3. Extension of the vocabulary and vocabulary mappings to support cross domain understanding
4. Methodological guidance on assessment of evidence sets, to be provided in Task 5.4.
8 Abreviations

AUTOSAR - AUTomotive Open System ARchitecture
BNF - Backus Naur Form
CAA - Civil Aviation Authority
CCL - Common Certification Language
DGR - Dependency Guarantee Relationship
FAA - Federal Aviation Authority
GSN - Goal Structuring Notation
ISO - International Standards Organisation
IMA - Integrated Modular Avionics
IWAG - International Avionics Working Group
MISRA - Motor Industry Software Reliability Association
OMG - Object Management Group
OPENCOSS - Open Platform for EvolutioNary Certification OfSafety-critical Systems
SACM - Structured Assurance Case Metamodel
SEooC - Safety Element out of Context
WCET - Worst Case Execution Time
WP - Work Package
9 References

[6] Civil Aviation Authority - http://www.caa.co.uk
[18] OMG: Structured Assurance Case Metamodel (SACM), 2013
Compositional certification conceptual framework


[32] D2.1 – CHESS Modelling Language and Editor CHESS Project; Deliverable. http://api.ning.com/files/IV00Zi2n8N6um45W000QxMxogYOoBb7Vh2lI4O0JR1W1AW6v5L5zTVx rz*x2t94lvKd5S8hEtQx9Lih*etowoQWgaqzVC/D2.1CHESSModellingLanguageandeditor.pdf;PDF-Document; Last visit: 2013-06-18

[33] ARINC 653 Avionics Application Software Standard Interface

[34] ATTEST2 Project, URL: http://www.atesst.org Last visit: 25/06/2013


