Collaborative Large-scale Integrating Project

Open Platform for EvolutioNary Certification Of Safety-critical Systems

Prototype implementation of tools for Argumentation/Compositional Certification

D5.5

<table>
<thead>
<tr>
<th>Work Package:</th>
<th>WP5: Compositional Certification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dissemination level:</td>
<td>CO</td>
</tr>
<tr>
<td>Status:</td>
<td>Draft</td>
</tr>
<tr>
<td>Date:</td>
<td>12/08/2014</td>
</tr>
<tr>
<td>Responsible partner:</td>
<td>Jan Mauersberger (IKV), Thomas Richardson (UoY)</td>
</tr>
<tr>
<td>Contact information:</td>
<td><a href="mailto:mauersberger@ikv.de">mauersberger@ikv.de</a>, <a href="mailto:thomas.richardson@york.ac.uk">thomas.richardson@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>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan Mauersberger, Sascha Baumgart</td>
<td>IKV</td>
</tr>
<tr>
<td>Alejandra Ruiz and Mª Carmen Palacios</td>
<td>TEC</td>
</tr>
<tr>
<td>Yaping Luo, Martijn Klabbers</td>
<td>TUE</td>
</tr>
<tr>
<td>Sunil Nair</td>
<td>SIM</td>
</tr>
<tr>
<td>Eric Verhulst</td>
<td>ALT</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>2014-02-03</td>
<td>Initial version</td>
</tr>
<tr>
<td>V0.2</td>
<td>2014-02-20</td>
<td>First update of the document and the structure</td>
</tr>
<tr>
<td>V0.3</td>
<td>2014-02-27</td>
<td>Second update with contributions to chapter 2 and 4</td>
</tr>
<tr>
<td>V0.4</td>
<td>2014-02-28</td>
<td>Third update with contributions to chapter 3</td>
</tr>
<tr>
<td>V0.5</td>
<td>2014-12-08</td>
<td>Review</td>
</tr>
<tr>
<td>V0.6</td>
<td>2014-12-09</td>
<td>Incorporated review comments</td>
</tr>
<tr>
<td>V0.7</td>
<td>2014-12-11</td>
<td>Added chapter about EviCA</td>
</tr>
<tr>
<td>V1.0</td>
<td>2014-12-12</td>
<td>Ready for PB review</td>
</tr>
<tr>
<td>V1.1</td>
<td>2014-12-15</td>
<td>Reviewed first release</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

Abbreviations .............................................................................................................. 7

1 Executive Summary........................................................................................................ 8

2 Implementation of OPENCORES platform 2nd prototype tools........................................... 9
   2.1 Scope and Implemented Functionality ......................................................................... 9
   2.2 Installation Guides & User Manuals ............................................................................. 17
   2.3 Source Code .............................................................................................................. 18

3 Research and Investigation ............................................................................................. 22
   3.1 Evidence Assessment ............................................................................................... 22
   3.2 Vocabulary Support .................................................................................................. 23
       3.2.1 SBVR ............................................................................................................... 23
       3.2.2 Implementation with SBVR support .................................................................... 24
   3.3 Difference Analysis ................................................................................................. 25

4 Outlook .......................................................................................................................... 27

5 Appendix A ...................................................................................................................... 28
List of Figures

Figure 1 Functional decomposition of the OPENCOSS platform .............................................................. 9
Figure 2 Palette with supported Argumentation classes in the argumentation editor .............................. 11
Figure 3 Structure of the argumentation contract grammar ........................................................................ 11
Figure 4 Example of a claim with mark-ups ............................................................................................. 12
Figure 5 Example of content assistant using a vocabulary in the ISO26262 context ................................. 13
Figure 6 Example of a pattern at creation phase ....................................................................................... 14
Figure 7 Templates view let the user select the library of patterns stored ................................................. 14
Figure 8 Configuration wizard page from the export wizard ...................................................................... 17
Figure 9 Argumentation management plugins .......................................................................................... 20
Figure 11 Methodology of the approach .................................................................................................... 24
Figure 12 Tool support for using SBVR vocabulary for safety cases ......................................................... 25
Figure 13 Exemplary simplified difference between argumentation structures ......................................... 26
List of Tables

Table 1. Component level requirements for Reuse of existing argumentation ......................................... 28
Table 2. Component level requirements for Specification of Assumptions .................................................. 28
Table 3. Component level requirements for Validation ................................................................................ 29
Table 4. Component level requirements for Gap Analysis ........................................................................... 29
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application programming interface</td>
</tr>
<tr>
<td>CCL</td>
<td>Common Certification Language</td>
</tr>
<tr>
<td>DSL</td>
<td>Domain Specific Language</td>
</tr>
<tr>
<td>DX.Y</td>
<td>OPENCOSS deliverable X.Y</td>
</tr>
<tr>
<td>EMF</td>
<td>Eclipse Modelling Framework</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>GSN</td>
<td>Goal Structure Notation</td>
</tr>
<tr>
<td>SBVR</td>
<td>Business Vocabulary and Business Rules</td>
</tr>
<tr>
<td>TX.Y</td>
<td>OPENCOSS task X.Y</td>
</tr>
<tr>
<td>XText</td>
<td>Framework to build DSL with eclipse</td>
</tr>
</tbody>
</table>
# Executive Summary

This document is a summary umbrella document of implementation and investigation work done in T5.3 “Tool support for compositional certification” and the continuation of D5.4. It covers work that has been done during development of prototype 1 and prototype 2 phases.

In particular the following core items constitute D5.5 achievements, referenced by this document:

- Installable OPENCOSS Platform tools 2nd prototype
- User Manuals and installation Instruction
- Source code description

In parallel to the implementation work, further research and investigation on potential implementation techniques, frameworks and tooling strategies but also on theoretical groundwork have been performed by partners participating in the T5.3 task. The results are briefly outlined in this document and they will be used in the 3rd prototype phase if applicable.
2 Implementation of OPENCROSS platform 2nd prototype tools

The main goal of T5.3 in general is the provision of tools and OPENCROSS platform services to support argumentation and assurance pattern management. Following the general OPENCROSS strategy of an incremental approach for research and development, the first prototype phase did concentrate on the general validation and implementation of the compositional certification conceptual framework and on the quick provision of end user tools to support other activities in the project such as the analysis of the industrial case studies and the analysis of further tool requirements and usage scenarios. The integration into the OPENCROSS platform, the alignment with other parts of OPENCROSS and the stepwise extension towards further framework concepts have been subject of the second prototype phase or will be part of the third and last prototype phase.

To achieve these goals, the Eclipse RCP platform – including supplementary Eclipse packages such as EMF, GMF or Xtext – were chosen as the technology foundation for the implementation work. It allowed rapid prototyping but still supports extensibility in later phases due to high integration capabilities.

This deliverable is mainly concerned with the Argumentation Management. This part of the OPENCROSS platform manages argumentation information in a modular fashion. It includes mechanisms to support compositional safety assurance and assurance patterns management. The implemented functionality in the first prototype is described in the next sub-section.

While the first prototype phase (see D5.4) was concentrating on the provision of the functional aspects of the tool, the second prototype phase did concentrate on integration aspects with other parts of the OPENCROSS tool platform, for example with the project repository.

2.1 Scope and Implemented Functionality

![Functional decomposition of the OPENCROSS platform](image-url)
The scope for the prototype is the provision of modelling tools for modular argumentation structures and assurance patterns as well as for supplementary functions such as preliminary pattern instantiation, context based user guidance, vocabulary support and contract definition. The major scope is highlighted with a red circle on the Figure 1 showing the general functional overview of the OPENCOSS platform.

As stated above, the “Argument Management” part is in charge of the argumentation along the assurance project. The user will be able to benefit from reuse by using previous approved modules for argumentation or instantiating argumentation patterns that reflect a set of best practices. He should be supported by additional functions depending on the environment of the project and the prescriptive knowledge (i.e. the safety standard or other standards the project has to deal with). This section details both the satisfied requirements and the deployed components to show the implementation scope of the first prototype.

From the requirements point of view this phase focuses on a set of high level requirements as defined in D5.2. Each requirement together with the implementation done so far and implementing the requirement is shortly outlined in the following sections.

- Provide a consistent and constrained means for the expression of safety argument claims

Safety argumentation is developed using the argumentation model which has been developed within OPENCOSS. For the first prototype a graphical argumentation editor has been developed which implements the CCL argumentation model using the GSN graphical notation. The tool palette shown in Error! Reference source not found. (see Figure 2) indicates the different argumentation classes supported in the first prototype.
Arguments structures and GSN diagrams are stored in individual model files in the Eclipse workspace.

- Provide a consistent and constrained means for the expression of safety argument contracts

To approach this requirement a DSL and contract editor has been created. This editor will support users while creating a contract using a textual language. It implements the grammar for contracts that is being defined on the context of T5.2. More information about the grammar is available in deliverable D5.3.

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

Figure 3 Structure of the argumentation contract grammar

Contracts are stored as independent files and are referenced in the argumentation when composing two or more different modules of argumentation. Note that the idea of using a dedicated DSL and a contract editor was reviewed and finally dropped in the last (current) prototype phase. An alternative approach is presented in D5.6 – this document will be updated accordingly when tools have been developed.
• Provide a consistent and constrained means for the expression of **contextual information** used in safety arguments

While creating argumentation claims some pieces of information can be seen as properties. These properties need to be constrained and are referenced in a consistent and coherent way. In order to prevent users to include invalid expressions an initial support for using vocabularies in argumentation has been developed. The information on the claims and in future also the context will be validated against the vocabulary.

• Provide a consistent and constrained means for the **expression of assumptions** used in safety arguments

Based on the initial support for simple vocabularies in the CCL and on means to support a more structured argumentation, a set of additional supporting features have been implemented atop the graphical argumentation editor. The language used to express claims has been enriched by supporting dedicated markups. For example to express references to other elements in the argumentation structure by means of id: markup or to express typed pattern variables by using var: markups. References to external files or hyperlinks are supported as well. Markups can be used while editing an argument text and they are rendered (similar to hypertext markups and in line with the GSN standard) in case the argument element is shown on a GSN diagram. Using the provided set of markups a first step towards semi-formal or even formal argumentation is supported.

Beside the markups, the user is provided with a content assistant that – depending on the context, on the already entered text and on the vocabularies that are available in the current project – proposes a set of texts or markups that consistently fit the current context. For that purpose a simple vocabulary editor is available to create and organize vocabulary terms according to the current CCL version. Available vocabularies (vocabulary files in the project, potentially the SBVR) are dynamically loaded and the terms are used by the content assist. Content assistance is also available for all markups, e.g. for the id: markup the IDs of referable elements are provided.

![Figure 4 Example of a claim with mark-ups](image-url)
• Develop a library of **reusable pattern-based structural templates** for modular arguments

As at theory level, different patterns have been conceived, thus tool support to create, store and instantiate these patterns was prepared. In the tools, an argument library is stored, respectively saved, in a special directory that is defined in the preferences. Argument patterns can be checked while developing argumentation using the patterns view which let users select stored argument patterns and just by using the drag and drop function, patterns are instantiated in the target argumentation.
Figure 6 Example of a pattern at creation phase

Figure 7 Templates view let the user select the library of patterns stored.
• Provide a means for managing change within a **modularized argument**

Argumentation management defines a class called module which lets user store argumentation in a modular way. Argumentation ready for reuse can be instantiated as a module on the actual argumentation and by doing so all the previous argumentation will be reused on the new project.

A library of ready to use modules is stored in a separate location within the tool. The content of the library (all contained pre-define modules) can be browsed and accessed using the “Templates” view on the Module Explored section similar to a “Pattern” view. However, these modules do not need the phase of instantiating them because the included information is ready to be used and does not need adaptation nor modification.

• Integration with CCL

One of the objectives of the tooling is to support safety argumentation from the early stages. In this way, the Argumentation Editor has been enhanced with the automatic creation of the high level safety argument from the concepts and terms currently modeled in the CCL. In order to achieve this objective a model to model transformation is required since model transformations provide a mechanism for automatically creating or updating target models based on information contained in existing source models.

The first step in this topic has been to conduct a survey about Model Transformation languages. Since the introduction of the MDE/MDA/MDD ideas for software systems development several years ago, a number of different (Meta)-modelling and model transformation languages have been proposed. Moreover, new model transformation languages will continue to appear, following different paradigms and approaches. For instance, recently several approaches adopting the Model Transformation By-Example (MTBE) paradigm have been proposed. So, in order to assess the selection a set of ideal characteristics of model transformation languages has been established. Therefore, taking into account such relevant characteristics and the model to model transformation requirements of the OPENCOSS project, the Epsilon Transformation Language (ETL) has been selected. ETL is a hybrid model-to-model transformation language with the following main features:
- It can handle several source and several target models.
- It offers rule scheduling functionality: lazy rules are only executed when they are explicitly called, guarded rules are only executed if their guard evaluates to true, greedy rules are executed whenever possible.
- Rules can be reused and extended through rule inheritance.
- External code can be executed from within the transformation rule.
- Epsilon languages provide excellent Eclipse-based tools that are supported by stable execution engines.

The second step in this topic has been to implement the required model to model transformations. In our case, Baseline models are transformed into Argumentation models as follows:
- RefActivities and its subactivities are transformed into a hierarchy of claims.
- RefRequirements and its subrequirements are transformed into a hierarchy of subclaims marked as “underdeveloped” and “to be supported”. These subclaims are also related to their father claims (coming from RefActivities and its subactivities).
- RefArtefacts are transformed into Information Element Citations of Solution type.
- Relationships between all above instances are created.

**NB:** Only instances marked as “isSelected” are automatically transformed.
In addition, a trace of these transformations is available. It provides a log of the execution of the transformation. It maps elements of the source model that were matched by a transformation rule to the elements of the target model that were produced by the transformation rule. In OPENCOSS traces can be stored through two complementary mechanisms: in a separate trace model or in a log file. These traces are useful as a basis for synchronization or incremental execution of model transformations as well as a debugging aid for the transformation process itself.

- **Integrate the argumentation tooling with the common infrastructure of the OPENCOSS platform**

During the second prototype phase it was envisaged to support a common central data infrastructure for sharing data with other tools and platform services. The main task of the development was the investigation in the available persistence techniques and frameworks available for Eclipse and EMF based tools. In addition the possibilities to still support both, a central database storage as well as a local file based storage was analyzed. Main goal was to support both in parallel to let the OPENCOSS community choose the appropriate technique when developing additional tools atop the OPENCOSS platform and frameworks. For argumentation tooling the second approach was followed, i.e. the tools are still EMF/GMF and local file based but there is support to export argumentation model to a central data storage and also to import models from the central data storage.

On the server side, the technology of choice is a relational database as it is the most common way to store data. The relational paradigm is well-supported by tools and frameworks and several widespread open-source relational databases are freely available. We currently use a PostgreSQL 9.3 server, but this can be exchanged easily. The database schema is generated by Teneo from the meta-model definition of the first prototype. This allows us to stick with the rapid prototyping approach, where changes to the meta-model do not break the implementation. The last major task on the server side is to create a RESTful API to give clients yet another way to access the common storage. The EMFT Texo project is currently being evaluated as a framework to generate such API.

On the client side, the first prototype had to be integrated with the common storage. The standard framework for object-relational mapping in Java programs is Hibernate. It requires an XML configuration file that defines how Java objects are stored in the database. Teneo not only creates such a mapping file from the EMF meta-models, it also generates the database schema.

The first approach was to use the import and export extension points of the Eclipse platform. The model data, e.g. from safety cases or evidence models, is currently stored in separate files inside projects in the Eclipse workspace. We provided export and import wizards to put model files into the database and then used Hibernate and Teneo to handle the data transfer. This approach worked well with the advantage being that it is a controlled push and pull mechanism for the common storage. The main issues were that the references between model files could not resolved and the integration with the editors was not very good.

References between models can occur when a GMF diagram is stored in a different file than the data model or when a model element references another model element in a different file. Both cases happen quite frequently. When a file containing such a reference is exported to the common storage, those references cannot be resolved because the target model element is missing. This is not acceptable, so we modified the exporter to export whole projects instead of single model files. Now, before the project is exported, every model file is loaded into a project model that also contains enough information about paths and file names to restore the project structure during the import.
Parallel to the export and import functionality, we are working to integrate the editors of the first prototype directly with the common storage. The important frameworks here (in addition to Hibernate and Teneo) are CDO and Dawn. CDO handles, among other things, sessions and model locking, while Dawn provides some integration with EEF and GMF editors. The main obstacle is that the integration provided by Dawn is clearly not sufficient. Additional implementation is required to achieve a seamless user experience. The common storage integration already adds to the overall complexity of the prototype user interface, therefore the goal is to keep the editor as easy to use as possible. As the task is non-trivial, there is an inherent risk involved in pursuing this goal due to time constraints. The simple import and export approach serves not only as a fallback solution, it also provided valuable insights into fundamental problems like the aforementioned unresolved model references. It also allowed us to get model data into the common storage for the implementation of the RESTful API.

2.2 Installation Guides & User Manuals

The steps necessary to install the second prototype are exhaustively described in D6.6 and will not be repeated here. Deliverable D6.6 contains all required steps and document references to set up the server and client tools. Note this document is a developer guide of the OPENCOSS tool prototype implementation. The developers can find the source code, installation instructions, step by step, in order to set up the development environment and the workspaces to use the OPENCOSS tools but also to implement new functionalities for the OPENCOSS Prototype. There is currently no pre-packaged distribution.
The user manual for Argumentation Module is detailed in the document “OPENCOSS first prototype user manual” (Date 25/02/2014 - V0.8; Section 7). This document is hosted, with the source code of the first prototype, at https://svn.win.tue.nl/repos/opencoss-code/tags/prototype/0.8/prototype, under the doc branch.

In summary, this document is a user manual of the first OPENCOSS tool prototype implementation. The users can find the installing instructions, the tool environment description, and the functionalities starting for the creation of Reference Frameworks (models representing Standards, Regulations, or Company-specific Processes), Assurance Projects and the associated Baseline (subset of Reference Framework to be applied in a specific assurance project), Evidence models (Artefacts), Process models (Activities), Compliance Maps (so far, compliance maps from Reference Artefacts to Artefacts), and Argumentation models.

Besides the first prototype for modular argumentation structures, there is a separate technical preview for vocabulary support and additional supporting features as markups, syntax highlighting and content assist. The user guide for this prototype can be found at https://svn.win.tue.nl/trac/opencoss/browser/WP-transversal/Implementation/VocabularyPrototype/Margot-UserManual.docx.

2.3 Source Code

The source code of the first prototype can be found in the source code Subversion repository at https://svn.win.tue.nl/repos/opencoss-code. There are tagged baseline versions for each prototype phase under “tags” (for example under “tags/prototype/0.8”). The trunk is reserved for the 3rd prototype phase and future developments.

After installing the prototype and following the steps described in the document “OPENCOSS developer guide”, all the source code can be found under the plugins branch.

Once all the plugins are installed, these are the necessary ones for the Argumentation Management:

- **GSN.figures**
  This plugin provides utilities to draw model elements according to the Goal Structuring Notation (or GSN) standard.

- **org.opencoss.sam.agree**
  In this plugin, the agreement metamodel is defined and stored, and the Java implementation classes for this model are generated.

- **org.opencoss.sam.agree.sdk**
  This plugin includes the Java implementation of the agreement editor. It includes the framework required to create, modify and validate the definition of the structure of an agreement.

- **org.opencoss.sam.agree.ui**
  In this plugin the views and the user interfaces required for defining an agreement are found.

- **org.opencoss.sam.arg**
  In this plugin, the argumentation metamodel is defined and stored, and the Java implementation classes for this model are generated.

- **org.opencoss.sam.arg.diagram**
  This plugin is the diagram editor itself. It manages diagrams and includes a canvas to draw on, a palette with creation tools and default selecting and zooming capabilities, a property view and an outline view.

- **org.opencoss.sam.arg.edit**
  The edit plugin includes adapters that provide a structured view and perform command-based edition of the model objects.
• org.opencoss.sam.arg.editor
  This plugin provides the user interface to view instances of the model using several common viewers and to add, remove, cut, copy and paste model objects, or to modify the objects in a standard property sheet.

• org.opencoss.sam.arg.ui
  This is an additional plugin. It offers several utilities such as drawing model elements not included in the GSN standard, accessing to patterns and modules files.

• org.opencoss.sam.arg.preferences
  This plugin manages the default preferences required by the Argumentation diagram editor. The parameters which can be defined are the Modules Directory (with all argumentation modules stored from previous argumentation phases) and the Patterns Directory (that contains all argumentation patterns templates).

In addition, these plugins are necessary to manage assurance project and to handle the corresponding evidences:

• org.opencoss.apm.assuranceassets
  In this plugin, the assurance assets metamodel is defined and stored, and the Java implementation classes for this model are generated.

• org.opencoss.apm.assuranceassets.edit
  The edit plugin includes adapters that provide a structured view and perform command-based edition of the assurance assets model objects.

• org.opencoss.evm.evidspes
  In this plugin, the evidence metamodel is defined and stored, and the Java implementation classes for this model are generated.

• org.opencoss.evm.evidspec.edit
  The edit plugin includes adapters that provide a structured view and perform command-based edition of the model objects.

• org.opencoss.infra.properties
  This plugin contains the definition of the Property metamodel, and the Java implementation classes for this model.

• org.opencoss.infra.properties.edit
  In relation with the edit plugin for evidence, this plugin contains a provider to display the model in a user interface.

Figure 7 shows all the plugins described above.
Figure 9 Argumentation management plugins
The following plugins contain optional code implementing vocabulary, markup and content assist support. These plugins are currently not based on the above argumentation editor but will be integrated in the 2nd phase:

- **de.ikv.opencoss.vocabulary.[diagram|edit|editor|tests]**
  Contain the CCL vocabulary meta model respective the related EMF based tree editor and GMF based graphical editor to create and edit vocabulary models

- **de.ikv.opencoss.contentassist**
  Contain base support for content assist and syntax highlighting based on markups and on CCL vocabularies.

- **de.ikv.opencoss.gsn.[diagram|edit|editor|tests]**
  Contain a simple GSN editor as a proof of concept for the above add-on features.

Further packages for import and export of argumentation models to the shared database can be found under a separate branch “branches/DAO_Export_Import”. These packages will be merged back to the trunk development in a next step.
3 Research and Investigation

This chapter gives a brief outlook of what will be tackled in the 3rd development phase.

3.1 Evidence Assessment

A prototype tool named EviCA (Evidence Confidence Assessor) was developed to support the evidence assessment framework. Specifically, EviCA allows users to: (1) create and edit safety arguments using GSN, (2) question the various reasons for having confidence in the used in primary argument, (3) automatically build confidence arguments based on a predefined GSN pattern that is customisable, and (4) calculate the confidence and the uncertainty at each level of the argument automatically.

EviCA is written in the Java programming language as a plug-in to the Eclipse IDE. It uses some utilities of the underlying Eclipse framework, notably the Graphical Editing Framework (GEF). We use Microsoft Excel as one of the means to import checklists for reasoning lowest-level factors. We also use Graphviz, an open source graph visualization software to visualize the individual belief functions the user provides and build a model of the confidence argument summarizing the belief functions. The figure below shows the technology stack used for EviCA.

Figure 9. Technology stack of EviCA tool

Figure 10 shows a screenshot of a sample safety argument fragment described in GSN. The pallet to the right of the screen provides users with the various GSN elements (Goals, solutions, strategies, context, etc.) that they need to create a goal structured safety case. The properties of a selected item can be accessed at the bottom of the screen. The node description can be either edited in the properties window or can be edited directly in the canvas. All edits in the elements are reflected in real-time. The nodes can be selected, resized, moved or deleted as required. The pane in the left of the window is a project explorer that displays the different projects and their associated safety case diagrams. The GSN editor developed as part of EviCA is the first of its type that allows users to create and manipulate confidence arguments. Users can click and drag Assertion Claim Points (ACP), between goals and solutions. An ACP is indicated by a black rectangle on the relevant link. Fig. 10 shows ACPs named ACP36 and ACP 37.
The usage of the tool including features as “Confidence Argument Generation”, “Belief Function Collection” and “Evidence Assessment Visualization” are described in the EviCA user guide which is available together with source code on the project repository.

3.2 Vocabulary Support

The support for vocabularies is planned to be extended in the next prototype phase and to go beyond the definition and usage of simple terms as they are currently modeled in the CCL and used in the tools. Different techniques and standards such as SBVR and existing tools to support the definition as well as to support the usage of “electronic” versions of vocabularies in OPENCOSS tools will be further researched and analyzed. Based on rich vocabularies the semantic analysis and validation of claims is supposed to be much more capable. The investigation in existing supporting tools and related projects has been already started in the current phase but no significant results in terms of direct vocabulary tooling that can be incorporated into the platform or that have been developed are available at this stage.

3.2.1 SBVR

Semantics of Business Vocabulary and Business Rules (SBVR) is a standard business-focused specification proposed by the Object Management Group (OMG) in 2008. Recently, OMG published its second version (SBVR 1.1). It defines a metamodel for domain experts to develop semantic models of business vocabulary and business rules, which are two key elements of SBVR meanings. The definitions of some main concepts in SBVR specification are listed as follows:

- **Meaning**: what is meant by a word, sign, statement, or description; what someone intends to express or what someone understands.
- **Vocabulary**: set of designations and verb concept wordings primarily drawn from a single language to express concepts within a body of shared meanings.
- **Concept**: unit of knowledge created by a unique combination of characteristics.
- **Rule**: proposition that is a claim of obligation or of necessity.
• Business rule: rule that is under business jurisdiction.

In this document, all SBVR examples are given in SBVR Structured English (SSE), which is introduced in SBVR Annex C. There are four font styles with formal meaning in SSE: green and underlined font style is used to describe noun concepts, green and double underlined font style is used for individual concepts, blue and italic font style is used to describe verb concepts, then other linguistic symbols used for definitions and statements are represented in orange font style. In our implementation, for the font style of Name, we use the same font style as for Term.

Business rules provide elements of guidance on business structure and actions. SBVR defines deontic and alethic modalities for the formulations of guidance. The deontic modal operators describe behavioral or operative rules, which specify expectations of humans or automated systems. Alethic modal operators enable definitional structural rules, which define features of a model, thus cannot be violated.

3.2.2 Implementation with SBVR support

In OPENCOSS different approaches for extracting the SBVR vocabulary from standards (and other relevant processes and documents) are being investigated. These include:

1. Manual extraction of verbs, nouns and relations from the documentation. This has the advantage that inconsistencies in terminology can be examined in turn, and pragmatic decisions made for trade offs where necessary. It has the disadvantage that it is very time consuming.
2. Another approach is based on the assumption that existing models of certain business domains (or standards as in OPENCOSS) contain the relevant information already, and are rich enough and on a level that allows the extraction of the SBVR vocabulary from them. This has the advantage of being automated (and it should be noted that the models are required for other aspects of the OPENCOSS approach so must be created whichever SBVR approach is taken), however much work is needed to ensure the standards models capture all the subtleties and relationships that may be needed.
3. An alternative approach may be to marry these two together, with partial automation highlighting areas where manual intervention is required.

The method by which the SBVR is extracted is not the primary concern of this task, but we include this information for completeness and consistency with ongoing research in Task 5.2.

![Figure 11 Methodology of the approach](image_url)
Figure 12 shows the methodology of the second approach. It transfers conceptual models from safety standards or existing design documents of projects to SBVR vocabulary, where all the domain concepts will be explicitly defined. Then, in the modeling phase, certification data, by means of safety argumentation, is structured into safety cases represented in GSN. By using the SBVR vocabulary, a link to conceptual models will be added to those safety cases to support assessing and certifying systems.

In the first prototype phase a tool framework to support user create safety case with SBVR vocabulary was developed. It is implemented using Eclipse Modeling Framework with certain plug-ins. Xtext is used for the editing support of SBVR in the GSN editor. By using Xtext, syntax highlighting and content assistant are also provided, shown in Figure 12. The input of this tool is SBVR vocabulary, which includes the keywords for noun concepts and verb concepts used in safety cases. Besides, content assistant enables user to check the original definition of a concept to see if that concept is what they want to use.

3.3 Difference Analysis

In compositional (safety) cases the composed and composing parts are typically created and later edited independently from each other. Thus it is inevitable that at a certain point in time he composed parts need to be updated. However, in order to know what changes have been done in the latest version, to know whether composed and composing parts still do “fit” and whether analyses and further work that has been done on the composed parts are still valid it is essential to be able to compare two versions of a model and present the result in an adequate way.

GSN based argumentation is highly graphically oriented – arguments, evidences and the claim structure are expressed on diagrams which make the comparison and even more the visualization of differences a challenge. The investigation will concentrate on technical and theoretical challenges imposed on the currently implemented tooling. From the theoretical perspective is for example unlikely possible to visualize diagram differences in a single diagram, differently from tables or forms. Whereas technically it is a challenge to use the existing tooling code and show two diagrams within a single graphical GMF editor.
Figure 13 Exemplary simplified difference between argumentation structures
4 Outlook

There are four main topics that will be tackled in the 3rd development phase.

1. Contious further integration of prototype tool parts. Tools that were researched and developed independently are integrated into a single solution if technically possible.

2. Contious integration with other parts of the OPENCOSS infrastructure (for example the common data infrastructure). Investigate further support for rich claims but also in a tighter integration of actual assurance projects and underlying reference frameworks in the tools.

3. Integration with existing safety analysis or evidence tools that are on the market (such as medini analyze for example) is envisaged to achieve a show case that is applicable in the real field. Demo use cases and scenarios will be developed that underpin the integration potentials and possibilities of the OPENCOSS platform towards tool vendors.
5 Appendix A

This appendix presents the detailed requirements specified in D5.2 that have been deployed in the first prototype of the evidence management infrastructure. These requirements are listed in Tables 1, 2, 3, 4 and 5, and grouped by functional area groups. The column labelled as "Deploy" indicates the state of the deployment, and the column labelled as "Comment" is a small explanation of the state of deployment.

### Table 1. Component level requirements for Reuse of existing argumentation

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Deploy</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>01_01</td>
<td>The OPENCOSS Approach shall provide a consistent and constrained means for the expression of safety argument claims</td>
<td>X*</td>
<td>Supported by the argumentation modeling toolset</td>
</tr>
<tr>
<td>01_02</td>
<td>The OPENCOSS Approach shall provide a consistent and constrained means for the expression of safety argument contracts</td>
<td>X</td>
<td>Supported by the argumentation modeling toolset</td>
</tr>
<tr>
<td>01_03</td>
<td>01-03, The OPENCOSS Approach shall provide a consistent and constrained means for the expression of contextual information used in safety arguments</td>
<td>X</td>
<td>Supported by the argumentation modeling toolset</td>
</tr>
<tr>
<td>01-04</td>
<td>The OPENCOSS Approach shall provide a consistent and constrained means for the expression of assumptions used in safety arguments</td>
<td>X</td>
<td>Supported by the argumentation modeling toolset</td>
</tr>
<tr>
<td>01-05</td>
<td>The OPENCOSS Approach shall develop a library of reusable pattern-based structural templates for modular arguments</td>
<td>X</td>
<td>Provides as separately modeled libraries</td>
</tr>
<tr>
<td>02-01</td>
<td>The OPENCOSS Approach shall provide consistent and constrained means for the expression, and identification, of public goals</td>
<td>X</td>
<td>Supported by the argumentation modeling toolset</td>
</tr>
<tr>
<td>02-02</td>
<td>The OPENCOSS Approach shall provide consistent and constrained means for the expression, and identification, of ‘away’ goals</td>
<td>X</td>
<td>Supported by the argumentation modeling toolset</td>
</tr>
<tr>
<td>02-03</td>
<td>The OPENCOSS Approach shall provide consistent and constrained means for the expression, and identification, of evidence characteristics</td>
<td>X</td>
<td>Supported by the evidence modeling toolset</td>
</tr>
</tbody>
</table>

### Table 2. Component level requirements for Specification of Assumptions

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Deploy</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-03</td>
<td>The OPENCOSS Approach shall provide a consistent and constrained means for the expression of contextual information used in safety arguments</td>
<td>X</td>
<td>Supported by the evidence modeling toolset</td>
</tr>
<tr>
<td>02-02</td>
<td>The OPENCOSS Approach shall provide a consistent and constrained means for the expression, and identification, of ‘away’ goals</td>
<td>(X)</td>
<td>Partially supported by the argumentation modeling toolset</td>
</tr>
<tr>
<td>03-03</td>
<td>The OPENCOSS Approach shall provide a means for the overall assessment of confidence within the composed argument</td>
<td>(X)</td>
<td>Supported by the EvICA tool</td>
</tr>
</tbody>
</table>
Table 3. Component level requirements for Validation

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Deploy</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>03-01</td>
<td>The OPENCOSS Approach shall provide a means for the overall assessment of evidence supporting the argument</td>
<td>(X)</td>
<td>Supported by the EviCA tool</td>
</tr>
<tr>
<td>03-02</td>
<td>The OPENCOSS Approach shall provide a means for the overall assessment of risk represented by the composed argument</td>
<td>(X)</td>
<td>Partially supported by the EviCA tool</td>
</tr>
<tr>
<td>03-03</td>
<td>The OPENCOSS Approach shall provide a means for the overall assessment of confidence within the composed argument</td>
<td>(X)</td>
<td>Supported by the EviCA tool</td>
</tr>
</tbody>
</table>

Table 4. Component level requirements for Gap Analysis

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Deploy</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-06</td>
<td>The OPENCOSS Approach shall provide a means for managing change within a modularised argument</td>
<td>(X)</td>
<td>Supported by the EviCA tool</td>
</tr>
<tr>
<td>03-01</td>
<td>The OPENCOSS Approach shall provide a means for the overall assessment of evidence supporting the argument</td>
<td>(X)</td>
<td>Supported by the EviCA tool</td>
</tr>
<tr>
<td>03-02</td>
<td>The OPENCOSS Approach shall provide a means for the overall assessment of risk represented by the composed argument</td>
<td>(R)</td>
<td>To be further researched</td>
</tr>
<tr>
<td>03-03</td>
<td>The OPENCOSS Approach shall provide a means for the overall assessment of confidence within the composed argument</td>
<td>(R)</td>
<td>To be further researched</td>
</tr>
</tbody>
</table>