

Preventing Omission of Key Evidence Fallacy in Process-based Argumentations

Faiz UL Muram, Barbara Gallina and Laura Gómez Rodríguez

School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden

Email: faiz.ul.muram|barbara.gallina@mdh.se

lgz17001@student.mdh.se

Abstract—Process-based argumentations argue that a safety-critical system has been developed in compliance with the development process defined in the standards and provide the evidence for certification of compliance. However, the process-based argumentations cannot ensure that the evidences are sufficient to support the claim. If the argumentations are insufficient (i.e., fallacious) they may result in a loss of confidence on system’s safety. It is thus crucial to prevent or detect fallacies in the process-based argumentations. Currently, argumentations review process to detect fallacies largely depends on the reviewers’ expertise, which is a labour-intensive and error prone task. This paper presents an approach that validates the process models (compliant with Process Engineering Metamodel 2.0), and prevent the occurrence of fallacy, specifically, *omission of key evidence* in process-based argumentations. If fallacies are detected in the process models, the approach develops the recommendations to resolve them; afterwards the process and/or safety engineers modify the process models based on the provided recommendations. Finally, the approach generates the safety argumentations (compliant with Structured Assurance Case Metamodel) from the modified process models by using model-driven engineering principles that are free from the fallacies. The applicability of the proposed approach is illustrated in the context of ECSS-E-ST-40C (Space engineering–Software) standard.

Index Terms—Process models; safety cases; process-based argumentation; argumentation fallacies; model transformation

I. INTRODUCTION

De facto standards (see, for example, ECSS-E-ST-40C [1], ISO 26262 [2], and EN 50128 [3]) provide the guidance, in the form of processes to be followed during the development of safety-critical systems. The compliance of processes is a mandatory requirement for certification of safety-critical systems. To this end, process-based argumentations show that a safety-critical system has been developed in compliance with the process (safety life-cycle) defined in the standards and justify the safety-related decisions. Process-based argumentations (at planning phase) argue about different phases or activities in process planning and provide the convincing evidence that each phase/activity was planned. Once the plans are approved, the real development consisting of the execution of the plans can be started. For instance, DO-178C standard defines the certification liaison process, where the first interaction between the applicant and the certification body are expected to take place at after the initial completion of the planning phase to ensure plan’s approval [4].

It has been recognized that creating argumentations is a very expensive, time consuming and error prone task; because inappropriate, incomplete or inherently faulty reasoning about the evidence introduces the defects in safety argumentations, namely called fallacies. These fallacies could lead to overconfidence in a system and tolerate certain faults, which in turn contribute to safety-related failures of the system. This risk also affects the process-based argumentation. For example, a process-based argumentation, supported by the evidence personnel competency (for performing the model checking task), is weakened in detecting all faults. Because underlying proof attributable to a lack of training in formal methods. The undetected design faults during the development process might lead to the failure of a safety system when it is deployed. Therefore, it is necessary to prevent or detect fallacies in the process-based argumentations.

This paper presents an approach that validates whether the process models contain sufficient information, to prevent the occurrence of fallacy (i.e., *omission of key evidence*) in process-based argumentations. If flaws are detected, the approach provides the process engineers comprehensive feedback regarding the deviations, and therefore, help them to efficiently resolve the fallacies and correct the process models. Afterwards, the approach generates the process-based safety argumentations that are free from the omission of key evidence fallacies. To do that Model-Driven Safety Certification (MDSafeCer) method is used that supports the generation of process-based argumentation by using model-driven engineering principles [5], [6]. More specifically, via MDSafeCer, process models compliant with Process Engineering Metamodel (SPEM) 2.0 [7] are transformed into argumentation models compliant with Structured Assurance Case Metamodel (SACM) [8]. The generated process-based argumentations provide valid justification that the evidence is sufficient to meet the standard’s requirements, and therefore, facilitate greater confidence of the safety of a system. The applicability of the proposed approach is illustrated by detecting fallacies and generating process-based argumentations for Attitude and Orbit Control Subsystem (AOCS) development plan, which follows the ECSS-E-ST-40C (Space engineering–Software) standard [9].

The rest of this paper is organized as follows: Section II presents essential background information. Section III de-

scribes the method for preventing fallacies in process-based argumentations. Section IV illustrates the application of our approach for ECSS-E-ST-40C standard. Section V presents the related work. Finally, Section VI concludes the paper and presents future research directions.

II. BACKGROUND

This section provides the background information on which the presented work is based: in particular, Section II-A recalls the necessary information from ECSS-E-ST-40C standard, which provides guidance and direction of the software development process. Section II-B presents the process modelling language used in this paper, specifically syntax of SPEM 2.0. Section II-C provides essential information concerning safety argument and their possible representation. Section II-D presents the process-based argumentation within the context of process compliance and MDSaferCer method. Finally, Section II-E discusses fallacies in process-based argumentations.

A. ECSS-E-ST-40C

ECSS-E-ST-40C [1] standard is one of the series of European Cooperation for Space Standardization (ECSS) standards, which focuses on space software engineering processes. In this paper, we limit our attention to Section 5.5 (Software Design and Implementation Engineering Process) of ECSS-E-ST-40C standard, which consists of three phases (design of software items, coding and testing, and integration); each of which contains various activities (e.g., detailed design of each software component, and development and documentation of the software interfaces detailed design). Each activity in turn consists of one or more tasks, each task is associated with the expected output showing the evidences of safety requirements. The standard also contains the normative tables and guidelines for the work products in Annexes, for example, Annex E (normative) Interface Control Document (ICD), Annex F (normative) Software Design Document (SDD), Annex H (normative) Software User Manual (SUM) and Annex K (normative) Software Unit/Integration Test Plan (SUITP).

B. Process Modelling










A process defines the structure that is imposed on the development of a system. SPEM 2.0 [7] is the Object Management Group (OMG) standard that provides the necessary concepts for modelling, documenting, and presenting systems and software development processes. The conceptual framework of SPEM 2.0 considers two views, the *Method Content* and the *Process Packages*. *Method Content* offers support for the definition of reusable process content, i.e., partially ordered *tasks*, *work products* (such as artefacts, deliverables, outcomes, etc.), *roles* (such as software architect/designer, engineer, development team leader, etc.) and *guidances*.

Typically, a *task* is assigned to a specific *role*, who is responsible for the execution of work and may generate the *work products* as output. The *guidance* describes additional information of work and is classified into various kinds such as *tool mentor*, *guideline* and *practice*. A *tool mentor* describes

that how to use a specific tool to perform certain activities. A *guideline* provides additional details on how to perform a particular task or grouping of tasks. A *practice* defines a proven way or strategy of doing work to accomplish a goal. Furthermore, a *qualification* documents specific skill or key competency that is used to model and represent the qualifications provided by instances of a role who has the responsibilities to perform relevant tasks correctly and efficiently. In contrast, *process package* focuses on the development processes, which are actually defined using elements that point to elements of the method content package. In order to define a *process*, tasks can be grouped to form an *activity* that in turn can be grouped to form a *phase*. *TaskUse*, *RoleUse* and *WorkproductUse* provide information about method elements *task*, *role* or *work product* in the process, respectively.

SPEM 2.0 supports variability management in the *Method Content* package, which allows elements to modify or reuse elements in other content packages without directly modifying the original content. SPEM 2.0 defines five types of variability relationships: *not assigned* (na)—the default value, *contributes*, *replaces*, *extends*, and *extends and replaces*. In the scope of this paper, we consider *Contributes* variability, which provides a way for elements to contribute their properties into their base element without directly altering any of base existing properties. Table I shows the basic structural elements for defining the process in SPEM 2.0.

TABLE I
PROCESS MODELLING ELEMENTS IN SPEM 2.0

Process	Phase	Activity	TaskUse	RoleUse	WorkProductUse	Guideline	ToolMentor	Practice
								

The EPF (Eclipse Process Framework) Composer¹ has been developed in order to support modelling of safety processes. EPF Composer is an extensible process framework, based on the Unified Method Architecture (UMA) metamodel, which is an evolution of the OMG's SPEM 1.1 [10]. UMA covers the SPEM 2.0 [7] concepts needed for our purposes. Recently, EPF Composer was ported from Eclipse Galileo 3.5.2 to Eclipse Neon 4.6.3 in the context of the AMASS project [11]. IBM Rational Method Composer (RMC)² is the commercial version of EPF Composer. In [12], IBM describes the modelling of standard's requirements with RMC. For this reason, they recommended to create three separate plugins: (i) *Standard requirements plugin*, which captures the standard's requirements, specifically, in *user-defined type*, (ii) *Lifecycle plugin*, which describes the process life-cycle (i.e., content elements and process), and (iii) *Requirements mapping plugin*, which defines the mappings between standard requirement and process elements. In order to define the mapping, standard requirements are copied in this plugin. These copied require-

¹See <https://www.eclipse.org/epf/>

²See https://www.ibm.com/support/knowledgecenter/SSBSK5_7.5.1/com.ibm.rmc.help.doc/topics/a_product_overview.html

ments have a variability relationship *Contributes* with original requirements. In addition, the links between requirements and life-cycle elements have been established through references.

C. Safety Case Representation

The certification process in various domains provides the justification that a system is safe for use in a specific environment under specific conditions. Accordingly, the justification can be provided through a contextualized structured argument that links evidence to claims; it is known as a safety case [13]. To document safety cases, several approaches exist both graphical and textual [14]. SACM [8] is the OMG's standard that unifies broadly used graphical notations for documenting safety cases. Common Assurance and Certification Metamodel (CACM), which includes SACM metamodel is being implemented in the OpenCert³ within the AMASS platform [15]. It might be noted that SACM [16] is evolving based on the contributions achieved in research projects (including AMASS [15]). OpenCert implements the assurance case editor that includes argumentation model and diagram; they are based on the main nodes of Goal Structuring Notation (GSN) [17]. However, internally it uses the SACM metamodel. Some of the main elements of SACM and their semantics (equal to CACM/OpenCert) are given in the following list:

- *Claim* states the safety requirement, plan or objective that needs to be achieved or justified. The property of the claim toBeSupported = "true" means that it requires further development.
- *ArgumentReasoning* describes a method that is used to connect one or more claims (premises) to another claim (conclusion).
- *InformationElementCitation* describes the supporting evidence, context, or additional description (rationale) for the core reasoning of the recorded argument.
- *AssertedContext* relationship is used to declare contextual relationships between contexts and claims or ArgumentReasoning elements.
- *AssertedInference* relationship is used to show the inferential relationships between elements.
- *AssertedEvidence* relationship is used to show the evidential relationship between claim and evidence.

D. Process-based Argumentation and MDSafeCer

The process-based argumentation shows that the system has been developed in compliance with the development life-cycle (process planning) according to the normative space (e.g., standards and regulations). Process-based safety argumentation consists of the claim that can argue about process activities and the evidence(s) that provide justification of compliance. Therefore, the process-based argumentation plays a central role in justifying that the available evidence, in the form of staff competencies, guidelines, work products (e.g., software design document) and tool qualifications has achieved a set of safety requirements, and in turn an acceptable level of safety

has been achieved. Compliance with ECSS-E-ST-40C standard (process planning) mandates the satisfaction of a specific set of objectives or requirements (for planning phase) by the generation of a concrete set of evidences.

MDSafeCer [5] is a method that enables the generation of process-based argumentations from process model using model-driven engineering methodology. By means of MDSafeCer, process models compliant with SPEM 2.0 metamodel are transformed into process-based argumentation models compliant with the SACM 1.1 metamodel and are presented through a concrete syntax, e.g. GSN [17]. Process elements such as phases are mapped into the claims. These claims are decomposed by showing the process activities have been planned (during planning phase) or executed (during the execution phase) and in turn each activity is decomposed into tasks and so on until an atomic work-unit is reached.

E. Argumentation Fallacies

An argumentation fallacy is a mistake or flaw in the reasoning of an argument. In safety arguments, fallacies exist in different forms. Greenwell et al. presented a taxonomy of common fallacies in safety arguments and organized them into three categories namely, relevance, acceptability and sufficiency fallacies [18]. *Relevance fallacies* add no value to an argument and provide irrelevant evidence. The existence of a relevance fallacy in an argument cannot contribute to a failure; rather, these fallacies might mislead/distract the developer or reviewer into accepting an insufficient argument, which, in turn, may contribute to a system failure. *Acceptability fallacies* are those in which an argument provides the unacceptable, contradict or inconsistent evidence to support the claims, for example, an argument contains the evidence that is only the restatement of the claim. *Sufficiency fallacies* are those in which arguments can fail to provide sufficient evidence to support the claims, either provide little or no evidence, biased or weak evidence, or omit crucial types of evidence. In this paper we limit our attention to *sufficiency fallacies*, more specifically, *omission of key evidence* in which arguments fail to provide key evidence that is crucial to support the claim. All instances of omission of key evidence detection are considered in which no or less evidences are provided to support the claim (at least one of the evidence has been omitted) or no valid reasons are given for its omission.

III. A METHOD FOR PREVENTING FALLACIES

This section describes the overall method for preventing the omission of key evidence fallacies in the process-based argumentations. Omission of key evidence fallacies within the context of process argumentation are the flaws or defects in which arguments can fail to provide sufficient evidence (e.g., staff competency) to support the process claim (e.g., claim about designer who is responsible for the design task, which deals with the production of design-related work products). The proposed approach consists of three steps. Firstly, the process engineers use EPF Composer to model the safety processes (see Section III-A). Secondly, the process engineers

³See <https://www.polarsys.org/proposals/opencert>

and/or safety engineers use our *Validation plugin* to detect whether the process contains all the essential information for supporting the key evidence(s) (see Section III-B). In case of omission of crucial information, recommendations are provided to aid process engineers or safety engineers in resolving deviations. Based on the recommendation(s), engineers are expected to modify the process model. Finally, the modified process model is proceeded to generate the process-based argumentation using implemented *Generation plugin* (see Section III-C). The generated arguments are visualized via the assurance editor in OpenCert. Detecting fallacies and generating process-based argumentation are iterative and incremental tasks, in particular, the development project can be continuously re-planned based on found deviations. The overall workflow of our approach is shown in Figure 1, specifically the solid lines show the extended step of MDSafeCer.

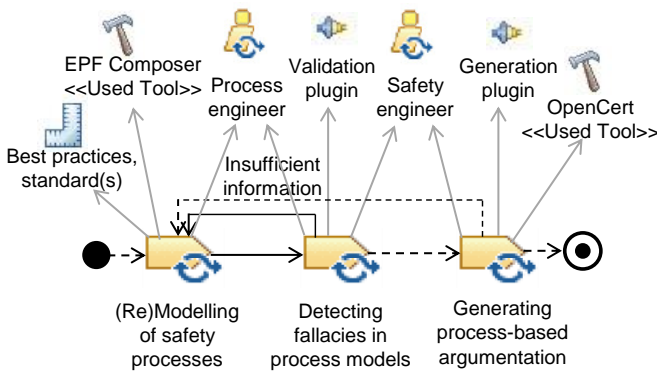


Fig. 1. Overview of the proposed method

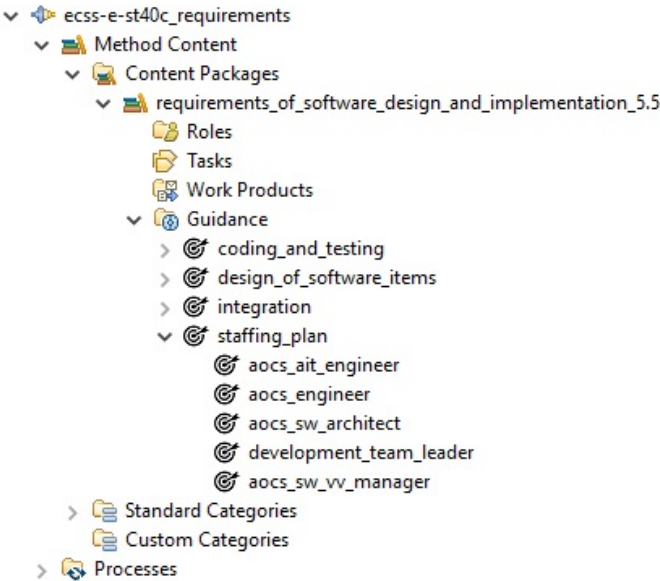


Fig. 2. Requirements modelling in EPF Composer

A. Modelling of Safety Processes

The first step involves modelling of a safety process in EPF Composer according to the best practices as well as according

to the standard(s). There are two possible ways of modelling requirements in EPF Composer. First, the requirements and associate process life-cycle can be modelled by following the IBM approach (see Section II-B). However, EPF Composer does not support the definition of a *user-defined type*. Therefore, the guidance type *Practice* has customized with an icon and variability relationships, as shown in Figure 2. The associated process elements (e.g., tasks, work products and roles) are modelled under the *Method Content* package and development life-cycle under *Processes* in the *process_lifecycle* plugin (see Figure 3).

Second, ECSS-E-ST-40C standard requirements can automatically be imported into the EPF Composer. For this, ECSS Applicability Requirement Matrix (EARM)—ECSS in MS Excel format (i.e., EARM_ECSS_exportDOORS-v0.5Statistics.xlsx⁴) has been parsed and the set of requirements has been filtered. The instructions for converting excel file to xml format compatible with EPF Composer has been described in AMASS [19].

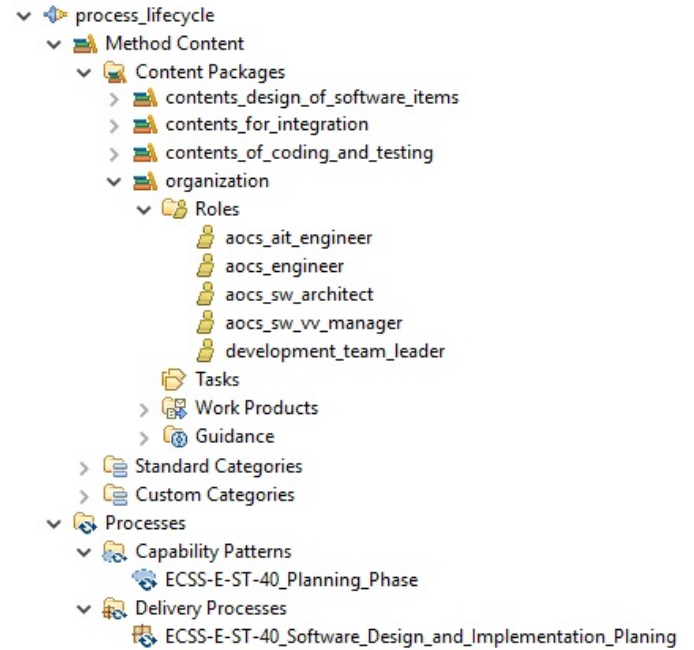


Fig. 3. Process fragment in EPF Composer

B. Detecting Fallacies in Process Models

In this subsection, we explain our algorithmic solution for detecting fallacies, more specifically, omission of key evidence fallacies, in process models. The approach validates whether the safety process contains the sufficient information corresponding to the key evidence for supporting the specific requirement. Algorithm starts by searching the process model (i.e., top-level element) and considers the (decomposed) linked elements such as phases, activities, tasks and so on. In particular, it follows the *Work Breakdown Structure*, as shown

⁴See <http://ecss.nl/standards/ecss-standards-on-line/active-standards/ecss-applicability-requirement-matrix-earm/>

in Algorithm 1. The function `getTopLevelElements()` returns a set of top-level elements of the process. Given a certain element e , its linked element le can be achieved by using the function `getLinkedElements()`. Once the link between the supporting elements has been established, the premises/details related to evidences are parsed and stored in an array. In a similar way, the requirements p and sub-requirements sp modelled as practices are extracted from requirements plugin, as shown in Algorithm 2.

Algorithm 1 Extracting elements from process models

```

1: procedure EXTRACT PROCESS ELEMENTS( $Proc$ )
2:    $Q \leftarrow \emptyset$   $\triangleright Q$  is the queue of non-visited elements
3:    $V \leftarrow \emptyset$   $\triangleright V$  is the queue of visited elements
4:    $Q \leftarrow Q \cup \text{getTopLevelElements}(Proc)$ 
5:   for all  $e \in Q$  do
6:      $V \leftarrow V \cup \{e\}$ 
7:      $Q \leftarrow Q \setminus \{e\}$ 
8:      $E_{breakdownElement} \leftarrow$ 
        $\text{getLinkedElements}(e)$ 
9:     for all  $le \in E_{breakdownElement}$  do
10:      if ( $\text{evidenceDetail}(le) \neq \emptyset$ ) then
11:        split the evidences and
12:        add in array  $evi$ 
13:      end if
14:    end for
15:  end for
16: end procedure

```

Algorithm 3 illustrates the detecting fallacies procedure. In this context, the requirements (*briefDescription*) related to a specific element (e.g., role) are matched with the provided information details (e.g., *Skills* or *MainDescription*). If the evidence details are omitted or the rationale is not provided; it means that process contains the omission of key evidence fallacies. The approach provides validation results including the list of elements containing sufficient and insufficient information (i.e., detected fallacies). In addition, the appropriate recommendations to resolve the particular deviations are presented. These results can be printed on the console, or otherwise the validation reports are generated in the selected folder. Process engineers and/or safety engineers then modified the process by providing required evidences or rationale for omitted information, the revised version will be validated again. Therefore, it ensures that the final argumentation will be generated from the process (which is specified in the next step) is valid.

C. Generating Process-based Argumentation

In this subsection, we explain how we generate the process-based argumentations from process models for arguing about compliance with standards. For this, the mapping between process elements (SPEM/UMA) and argumentation elements (SACM) has been implemented. The mapping is focused on the *Work Breakdown Structure* of processes in EPF Composer.

Algorithm 2 Extracting requirements from requirements plugin

```

1: procedure EXTRACT REQUIREMENTS( $Reqs$ )
2:    $R \leftarrow \emptyset$   $\triangleright R$  is the queue of non-visited elements
3:    $T \leftarrow \emptyset$   $\triangleright T$  is the queue of visited elements
4:    $R \leftarrow R \cup \text{getRequirementPractices}(Reqs)$ 
5:   for all  $p \in R$  do
6:      $T \leftarrow T \cup \{p\}$ 
7:      $R \leftarrow R \setminus \{p\}$ 
8:      $P_{subRequirements} \leftarrow$ 
        $\text{getSubRequirements}(p)$ 
9:     for all  $sp \in P_{subRequirements}$  do
10:      if ( $\text{practiceAdditionalInfo}(sp) \neq \emptyset$ ) then
11:        split the requirements and
12:        add in array  $req$ 
13:      end if
14:    end for
15:  end for
16: end procedure

```

Algorithm 3 Detecting omission of key evidence fallacies

```

1: procedure DETECTING FALLACIES( $Reqs, Proc$ )
2:    $\text{getEvidenceDetails}$  from process
3:   (call Algorithm 1)
4:    $\text{getAdditionalInfo}$  from requirements
5:   (call Algorithm 2)
6:   for all  $le \leftarrow evi$  do
7:     for all  $sp \leftarrow req$  do
8:        $\text{match}(sp.name, le.name) \wedge (req[j], evi[i])$ 
9:       if ( $sp.name = le.name$ )  $\wedge$  ( $req[j] = evi[i]$ )
10:        then returns true
11:      else
12:        returns false
13:        generate omitted information
14:        and provide recommendations
15:        regarding evidences
16:      end if
17:    end for
18:  end for
19: end procedure

```

In particular, the *ProcessComponent* that contains the information of the process is mapped into a *Case*, whereas the planning phases are constituted of the top-level claims stating that the planned process is in compliance with the required standard-level. As recalled in Section II-D, these claims can be decomposed by showing that all the process activities have been planned, in turn, for each activity all the tasks have been planned and so on. The crucial process elements that are associated to a task, namely a set of roles, a set of work products, and a set of guidances are mapped into the sub-claims. *ArgumentReasoning* is created in order to divide the claim into sub-claims. The evidences related to these elements are mapped into the *InformationElementCitation* property type

solution. The main mapping between these metamodels is described in Table II.

TABLE II
MAPPINGS CONCEPTS

SPEM/UMA	SACM	Diagram
ProcessComponent	Case	
Process purpose (Standard)	InformationElementCitation Property type = "context"	Context
Phase, Activity, TaskUse / TaskDescriptor	Claim	Goal
A set of RoleUse / RoleDescriptor, WorkProductUse / WorkProductDescriptor, Guideline and ToolMentor	ArgumentReasoning	Strategy
Requirements for competency of RoleUse	Sub-Claim	Sub-Goal
Evidences associated to WorkProductUse, RoleUse, Guideline and ToolMentor	InformationElementCitation Property type = "solution"	Solution
Relationship between a competency of RoleUse and certification	AssertedEvidencee	SolvedBy
Relationship between a TaskUse and a RoleUse... (not related to evidence)	AssertedInference	SolvedBy
Id, name and description	Id, name and description	Id, name and description

The mapping is achieved by using Epsilon Transformation Language (ETL)⁵. It is a hybrid, rule-based model-to-model transformation language and provides the enhanced flexibility to transform many input to many output models. A plugin has been implemented in the AMASS platform⁶, which automatically transforms the process model into safety argument fragments (i.e., model and diagram) using ETL. The generated argumentation model and diagram are visualized via the assurance editor in OpenCert.

IV. AN ILLUSTRATIVE EXAMPLE

Description: In this section, Attitude and Orbit Control Subsystem (AOCS), will be described to illustrate how our approach works to detect omission of key evidence fallacy and generates the process-based argumentation. It is used in a number of different telecommunication satellite platforms [9]. The *Attitude control* manages the orientation of the satellite, whereas, *Orbit control* regulates the positioning of the satellite in orbit. This illustrative example is based on the AOCS-related Software (SW) development process. Accordingly, it follows a set of recommendations from the ECSS-E-ST-40C standard. AOCS SW requires the high-level of assurance activities and provisions of evidence.

As discussed in Section II-A, ECSS-E-ST-40C standard explicitly describes the requirements and recommendations related to planning process such as activities, tasks and expected outputs. However, requirements related to the staffing

plan (i.e., key competencies or skills required for specific roles) are not provided. Key competencies required for an architect/designer role are adapted from railway standard EN 50128 [3]. The key competencies for other roles, for example, AOCS Assembly Integration and Test (AIT) engineer, AOCS engineer, development team leader, and AOCS SW V&V (Software Validation and Verification) Manager, who have the responsibility for planing process are modelled according to the industrial requirements for AOCS SW development. Specifically, AOCS engineer shall have the following experience and competencies: university degree in engineering, several years of experience in the design, analysis and simulation of AOCS systems in different project phases, excellent hardware knowledge (sensors, computer, actuators), experience in AOCS AIT at subsystem and system level, and working experience with Linux System, Matlab and Satsim.

Method Application: The requirements and process for AOCS are modelled as plugins in EPF Composer by following the guidelines mentioned in Section III-A. Then the detection of omission of key evidence fallacy is performed on the process models, as depicted in Section III-B. Figure 4 shows the process model and the validation result including the list of roles containing sufficient information (enclosed in green box), omitted details of evidences and recommendations (enclosed in red box). The results have been printed on the console, or otherwise the fallacies reports would have been generated.

By looking at the result we find that skill certifications (evidences) against AOCS AIT engineer, AOCS SW Architect and AOCS SW V&V Manager are sufficient. Key evidences associated to AOCS Engineer, particularly, university degree and working experience with Linux System, Matlab and Satsim are omitted (i.e., less evidences are provided). In addition, no evidence of skill concerning Development Team Leader is provided and no valid reason (rationale) is given for its omission. The recommendations for omitting skill certifications are provided. In particular, first fallacy (less evidence) can be resolved by adding the certifications against over university degree, and working experience with Linux System, Matlab and Satsim or by providing rationale for omission. The second fallacy (omission of all evidences) can be addressed by adding the following skill certifications: management of Electra AOCS SW development team, working experience with Matlab/Simulink, knowledge of design analysis and design test methodologies, and good analytical and problem-solving skills. Based on the results engineers either modified the process models or provide the rationale about omitted information.

Once the fallacies are detected, fallacies are eliminated by modifying the process models and rerunning the validation process yielded no further flaws. Then the process-based argumentations are automatically generated from the modified models (see Section III-C). Figure 5 shows generated argumentation model and diagram, compliant to the SACM metamodel that are visualized in assurance case editor in OpenCert. Without the automatic validation of process models, engineers would have to check arguments manually which

⁵See <https://www.eclipse.org/epsilon/doc/etl/>

⁶See Training on the Prototype P1: WP6 Session <https://www.amass-ecsel.eu/content/training>

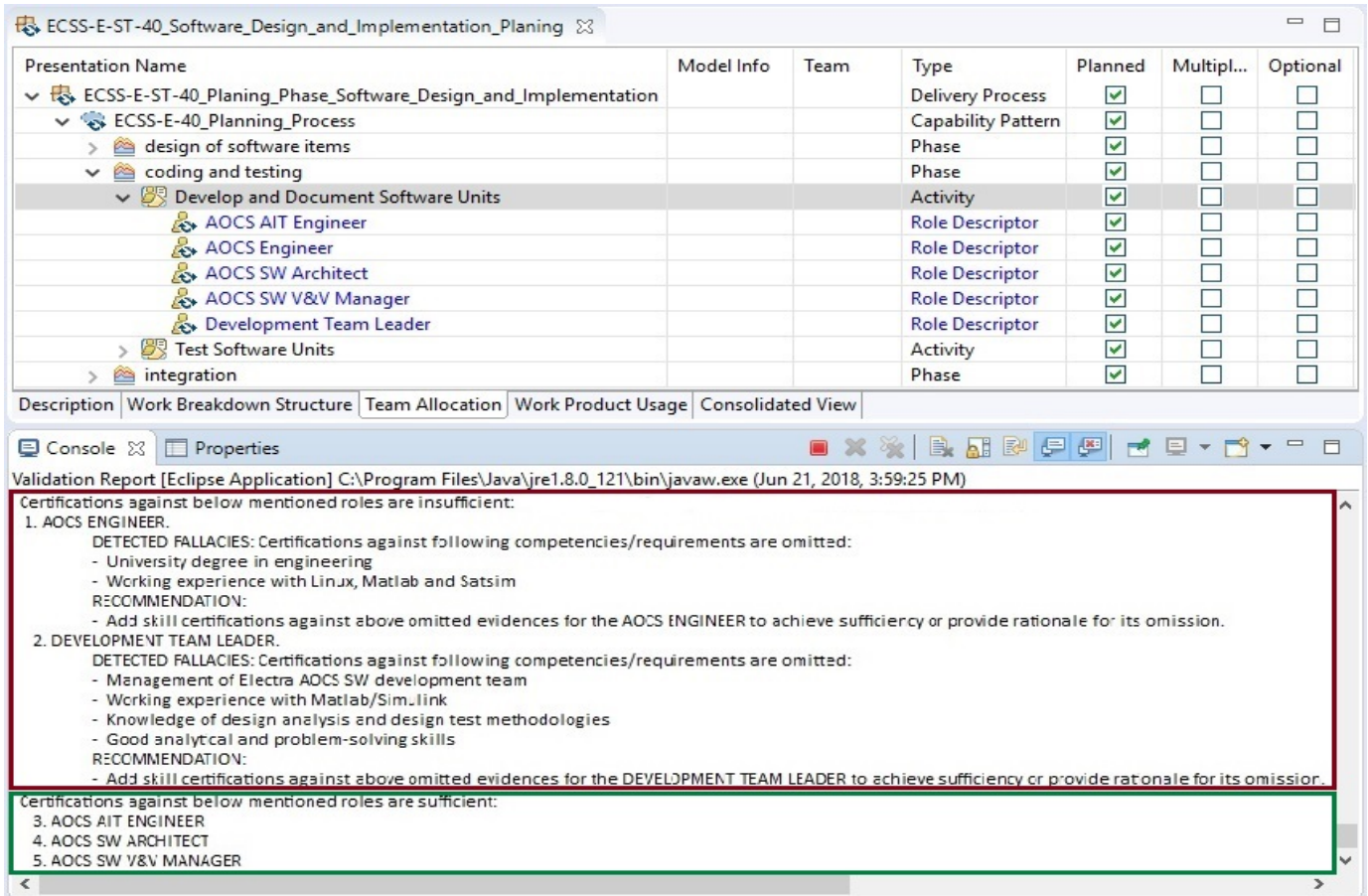


Fig. 4. Result after detecting omission of key evidence fallacies

requires huge effort and knowledge. The automatic detection of fallacies and argumentations generation not only save time and cost but also provide valid justification that the evidence is sufficient to meet the standard's requirements.

V. RELATED WORK

Over the past few years, creation and assessment of safety arguments have been gained much research interest. However, very few of these studies focus on the detection of fallacies in the arguments. These studies mainly focus on the confidence estimation and decision support for the acceptability of the argument, or its improvement. To the best of our knowledge, none of them considers the preventing omission of key evidence fallacy in process-based argumentations. Our approach validates whether the process models contain sufficient information. In case of omitted evidences, the recommendations are provided; afterwards, the process-based argumentations are generated from modified process models.

Greenwell and Knight [20] propose an approach for analyzing the digital system failures based on the concept of safety cases. They have built a tool called Pandora, which extracts evidences from a failure to discover fallacies in the safety argument that might have contributed to the failure. Pandora is a manual process and depends on the completeness of pre-

failure safety case. Yuan et al. [21] develop a dialogue based model and DiaSAR tool to review safety arguments. This tool allows the argument proposer to create, defend or revise the argument, based on the reviews conducted by independent reviewers. However, the quality of review arguments can not be guaranteed because it largely depends on the reviewers' expertise.

Cyra and Górski focus on the concept of belief and uncertainties and their linking with a decision to accept or reject the argument used in trust cases [22]. They propose a method Visual Assessment of Arguments based on the Dempster-Shafer theory of evidence, implemented in a Trust Case Toolbox for decision support. Similarly, Ayoub et al. [23] present an approach to construct confidence arguments and identification of weaknesses in the safety arguments. They structured the collection of common concerns, called the common characteristics map, which is used to identify sources of uncertainties with recursive dependencies. In another study [24], the authors extend prior work by only considering that for each argument element it exists a level for sufficiency. Their approach provides a framework to lead the reviewer through the evaluation process and to combine the reviewer estimates. However, these approaches mainly considered expert opinion to evaluate the argument, the final decision seem not be accurate enough.

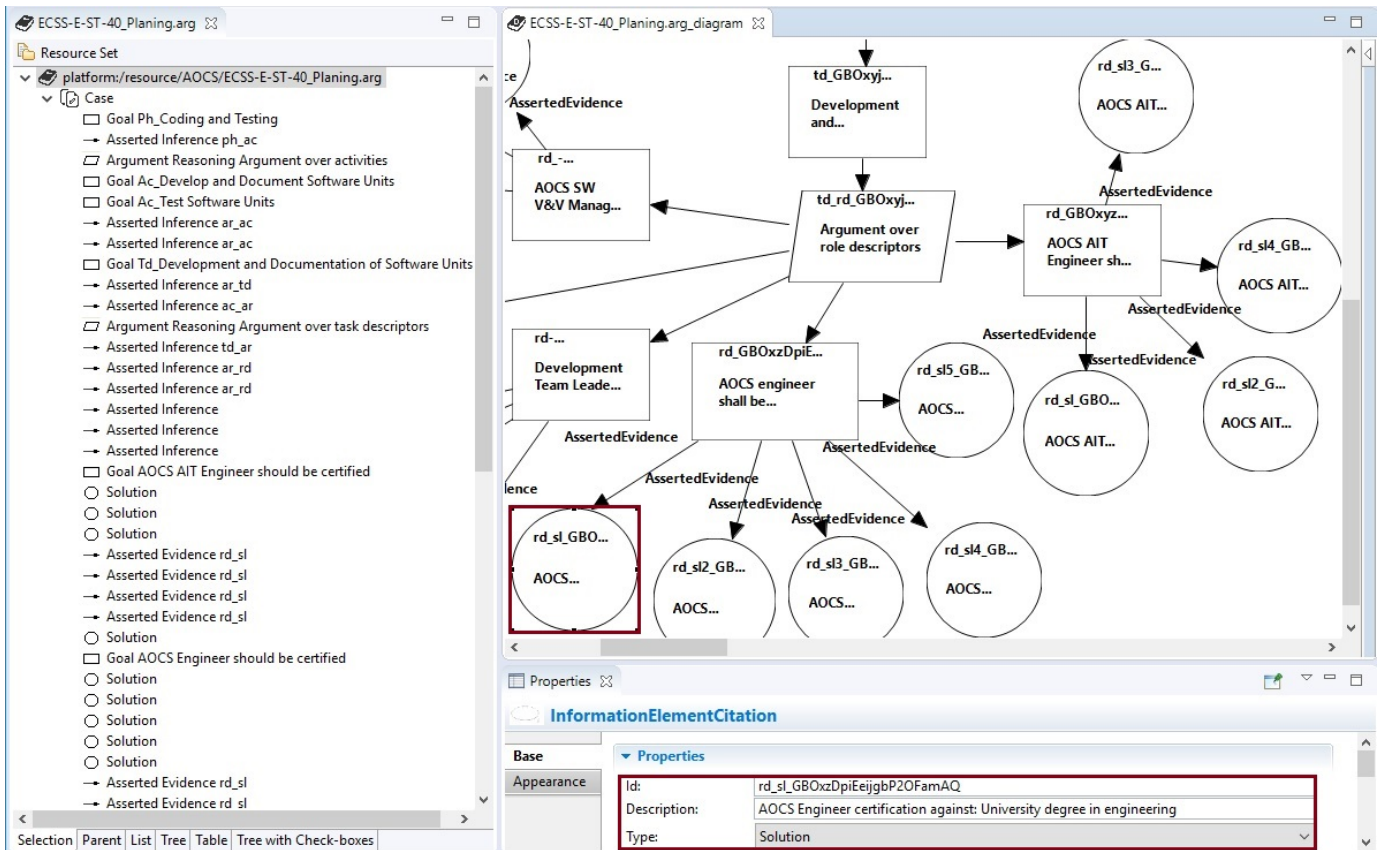


Fig. 5. Generated argumentation model and diagram

As discussed by Mizrahi [25] that arguments from expert opinion are weak arguments (i.e. fallacies) – the premises of such arguments provide probable support rather than logically conclusive support for their conclusions. The author provides plenty of empirical evidence stating that expert opinions “are only slightly more accurate than chance”.

Kokaly et al. [26] present the impact assessment algorithm, GSN-IA that detects the impact type for each safety case element and marks the element accordingly. The approach presumes the existence of safety cases and assesses the impact of system changes on them. Nair et al. [27] introduce an approach to automatically construct confidence arguments for the evidence cited in a primary safety argument and quantify confidence using Evidential Reasoning (ER) algorithm. The proposed approach is supported by a prototype tool Evidence Confidence Assessor (EviCA). Luo et al. [28] propose a safety case assessment process and develop a tool, called AGSN (Assessable-GSN). The authors develop a graphical safety case editor for assessing GSN-based safety case and use the ER algorithm to assess the overall confidence in a safety case. Yuan et al. propose the predicate-based representation of safety arguments [29]. In particular, they build an ontology containing a set of constant symbols, predicate symbols and function symbols, which create the vocabulary for the expressions of GSN nodes. To detect fallacy, these symbols are checked against the database to verify whether

the argument is correct. The search function only works with frequently used keywords that are stored in the database. Moreover, they do not consider the omission of key evidence fallacy. However, the above mentioned approaches presume the existence of the safety cases or trust cases. Denney and Pai [30] develop a toolset, called AdvoCATE for creating and editing the argument fragments. Tool supports automated creation of argument fragments by instantiating a pattern either interactively, or by data extracting from the output tables of hazard and safety requirements analysis tool, or a formal verification tool. AdvoCATE also supports the argument verification, in particular structural constraints (i.e., syntactic checks) for some types of argument properties can be specified in the editor. In contrast, our approach supports detection of fallacy in process models and generation of process-based safety arguments from these models.

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented a method to prevent omission of key evidence fallacy in the process-based argumentations. In this context, the validation is performed to detect whether the safety process modelled in EFP Composer contains the sufficient information corresponding to the key evidence for supporting the specific requirement. In case of omitted crucial evidence detail, the feedback is provided to the process and/or safety engineers regarding detected fallacies and recommen-

dations to resolve them. The process model is modified based on the provided recommendations. Once process model has been modified, the process-based argumentations (model and diagram) are automatically generated from process and are visualized via the assurance editor in OpenCert. The generated process-based argumentations are free from the fallacies and provide valid justification that the evidences are sufficient to meet the standard's requirements. The application of the proposed approach is illustrated for ECSS-E-ST-40C standard used to engineer AOCs. Please note that our approach is not dependent on the ECSS-E-ST-40C standard and is also valid in the context of other standards.

Currently, we supported sufficiency fallacies, specifically, omission of key evidence fallacy. As mentioned in the background, there are other argumentation fallacies that are categorized in [18] but not examined in this paper. Our research agenda includes support for other sufficiency fallacies, for example, ignoring the counter-evidence as well as to investigate relevance fallacies and acceptability fallacies. Another direction for future work is to conduct more comprehensive case studies.

ACKNOWLEDGMENT

This work is supported by EU and VINNOVA via the ECSEL Joint Undertaking under grant agreement No. 692474, AMASS project.

REFERENCES

- [1] European Cooperation for Space Standardization (ECSS), "ECSS-E-ST-40C, Space Engineering Software." [http://www.wis.win.tue.nl/2R690/doc/ECSS-E-ST-40C\(6March2009\).pdf](http://www.wis.win.tue.nl/2R690/doc/ECSS-E-ST-40C(6March2009).pdf), 2009, (Last accessed: June 22, 2018).
- [2] International Standards Organization (ISO), "ISO 26262, Road Vehicles – Functional Safety," 2011.
- [3] European Committee for Electrotechnical Standardization (CENELEC), "EN 50128 - Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems," 2011.
- [4] RTCA Inc, *Software Considerations in Airborne Systems and Equipment Certification, RTCA DO-178C (EUROCAE ED-12C)*, Washington DC, 2013.
- [5] B. Gallina, "A Model-driven Safety Certification Method for Process Compliance," in *2nd International Workshop on Assurance Cases for Software-intensive Systems, joint event of ISSRE, Naples, Italy, November 3-6, 2014*. IEEE, 2014, pp. 204–209.
- [6] B. Gallina, E. Gómez-Martínez, and C. B. Earle, "Deriving Safety Case Fragments for Assessing MBASafe's Compliance with EN 50128," in *16th International Conference on Software Process Improvement and Capability Determination (SPICE), Dublin, Ireland, June 9-10, 2016*, ser. Communications in Computer and Information Science, vol. 609. Springer, 2016, pp. 3–16.
- [7] Object Management Group (OMG), "Software & Systems Process Engineering Metamodel Specification (SPEM), Version 2.0," <http://www.omg.org/spec/SPEM/2.0/>, 2008, (Last accessed: June 22, 2018).
- [8] —, "Structured Assurance Case Metamodel (SACM), Version 1.1," <https://www.omg.org/spec/SACM/1.1/>, 2017, (Last accessed: May 15, 2018).
- [9] Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems (AMASS), "Case studies description and business impact D1.1," https://www.amass-ecsel.eu/sites/amass.drupal.pulsartecnia.com/files/documents/D1.1_Case-studies-description-and-business-impact_AMASS_Final.pdf, (Last accessed: April 10, 2018).
- [10] Object Management Group (OMG), "Software Process Engineering Metamodel Specification (SPEM), Version 1.1," <ftp://ftp.omg.org/pub/spem-rtf/SPEM-CD-20040308.pdf>, 2004, (Last accessed: June 22, 2018).
- [11] M. A. Javed and B. Gallina, "Get EPF Composer back to the future: A trip from Galileo to Photon after 11 years," http://www.es.mdh.se/publications/5091-Get_EPFComposer_back_to_the_future_A_trip_from_Galileo_to_Photon_after_11_years, 2018.
- [12] B. McIsaac, "IBM Rational Method Composer: Standards Mapping," IBM Developer Works, Tech. Rep., 2015.
- [13] J. M. Rushby, "Just-in-Time Certification," in *12th International Conference on Engineering of Complex Computer Systems (ICECCS), Auckland, New Zealand, 10-14 July, 2007*, pp. 15–24.
- [14] S. Nair, J. L. de la Vara, M. Sabetzadeh, and L. C. Briand, "Classification, Structuring, and Assessment of Evidence for Safety - A Systematic Literature Review," in *Sixth IEEE International Conference on Software Testing, Verification and Validation (ICST), Luxembourg, Luxembourg, March 18-22, 2013*, pp. 94–103.
- [15] AMASS, "Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems," <http://www.amass-ecsel.eu/>, 2016.
- [16] Object Management Group (OMG), "Structured Assurance Case Metamodel (SACM), Version 2.0," <https://www.omg.org/spec/SACM/2.0/>, 2017, (Last accessed: June 10, 2018).
- [17] Goal Structuring Notation Community Standard (GSN), "COMMUNITY STANDARD VERSION 2," <https://scsc.uk/r141B:1>, 2018, (Last accessed: June 22, 2018).
- [18] W. S. Greenwell, J. C. Knight, C. M. Holloway, and J. J. Pease, "A Taxonomy of Fallacies in System Safety Arguments," in *24th International System Safety Conference (ISSC), New Mexico, July 31-Aug 4, 2006*.
- [19] Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems (AMASS), "AMASS demonstrators (a) D1.4," https://www.amass-ecsel.eu/sites/amass.drupal.pulsartecnia.com/files/documents/D1.4_AMASS-demonstrators-%28a%29_AMASS_Final.pdf, (Last accessed: 2018-03-09).
- [20] W. S. Greenwell and J. C. Knight, "Framing Analysis of Software Failure with Safety Cases," *Department of Computer Science, University of Virginia, Charlottesville, VA, 2006*.
- [21] T. Yuan, T. Kelly, and T. Xu, "Computer-assisted Safety argument Review – A Dialectics Approach," *Argument & Computation*, vol. 6, no. 2, pp. 130–148, 2015.
- [22] L. Cyra and J. Górski, "Support for Argument Structures Review and Assessment," *Rel. Eng. & Sys. Safety*, vol. 96, no. 1, pp. 26–37, 2011.
- [23] A. Ayoub, B. Kim, I. Lee, and O. Sokolsky, "A Systematic Approach to Justifying Sufficient Confidence in Software Safety Arguments," in *31st International Conference on Computer Safety, Reliability, and Security (SAFECOMP), Magdeburg, Germany, September 25-28, 2012*, pp. 305–316.
- [24] A. Ayoub, J. Chang, O. Sokolsky, and I. Lee, "Assessing the Overall Sufficiency of Safety Arguments," in *Twenty-first Safety-Critical Systems Symposium (SSS), Bristol, UK, February 5-7, 2013*.
- [25] M. Mizrahi, "Why Arguments from Expert Opinion are Weak Arguments," *Informal Logic*, vol. 33, no. 1, pp. 57–79, 2013.
- [26] S. Kokaly, R. Salay, M. Chechik, M. Lawford, and T. Maibaum, "Safety Case Impact Assessment in Automotive Software Systems: An Improved Model-Based Approach," in *36th International Conference on Computer Safety, Reliability, and Security (SAFECOMP), Trento, Italy, September 13-15, 2017*, pp. 69–85.
- [27] S. Nair, N. Walkinshaw, T. Kelly, and J. L. de la Vara, "An Evidential Reasoning Approach for Assessing Confidence in Safety Evidence," in *26th IEEE International Symposium on Software Reliability Engineering (ISSRE), Gaithersbury, MD, USA, November 2-5, 2015*, pp. 541–552.
- [28] Y. Luo, M. van den Brand, Z. Li, and A. K. Saberi, "A Systematic Approach and Tool Support for GSN-based Safety Case Assessment," *Journal of Systems Architecture*, vol. 76, pp. 1 – 16, 2017.
- [29] T. Yuan, S. Manandhar, T. Kelly, and S. Wells, "Automatically Detecting Fallacies in System Safety Arguments," in *Principles and Practice of Multi-Agent Systems - International Workshops: IWEC 2014, Gold Coast, QLD, Australia, December 1-5, 2014, and CMNA XV and IWEC 2015, Bertinoro, Italy, October 26, 2015, Revised Selected Papers, 2015*, pp. 47–59.
- [30] E. Denney and G. Pai, "Tool Support for Assurance Case Development," *Automated Software Engineering*, pp. 1 – 65, Dec 2017.