Towards Cloud-Based Enactment of Safety-Related Processes

Sami Alajrami[1], Barbara Gallina[2], Irfan Sljivo[2], Alexander Romanovsky[1], and Petter Isberg[2]

[1] Newcastle University, Newcastle upon Tyne, UK
{s.h.alajrami,alexander.romanovsky}@newcastle.ac.uk
[2] Mälardalen University, Västerås, Sweden
{barbara.gallina,irfan.sljivo,petter.isberg}@mdh.se

Abstract. Engineering safety-critical systems is a complex task which involves multiple stakeholders. It requires shared and scalable computation to systematically involve geographically distributed teams. The paper proposes a model-driven cloud-based enactment architecture automating safety-critical processes. This work adapts our previous work on cloud-based software engineering by enriching the architecture with an automatic support for generation of both, product-based safety arguments from failure logic analysis results and process-based arguments from the process model and the enactment data. The approach is demonstrated using a fragment of a process adapted from the aerospace domain.

Keywords: Safety Process Enactment, Argumentation, Cloud Computing

1 Introduction

The malfunctioning of safety-critical systems may lead to catastrophic consequences to the environment and people. Safety-critical systems are identified as complex systems and their engineering has to follow best practices. More specifically, (safety) standards provide guidance in terms of reference process models for the development and assessment of such systems. The complexity of such systems is reflected in their supply chain, which consists of a complex, geographically-distributed and heterogeneous supply network. The suppliers provide software/hardware components or automation of certain activities during the production. This compositional nature is reflected in software-specific, hardware-specific, tool qualification-specific, system-specific and method-specific guidance and/or reference processes recommended by standards. To be released on the market, the integrated systems must be certified. The certification process in various domains is conducted by scrutinizing an argument supporting system safety [16]. In the automotive and rail domains, such argument is known as *safety case*. In the aerospace domain, an explicit safety case is not required however as discussed by Holloway [13] an implicit safety case request is contained within the standards. While the considerations listed in this paper hold for several complex safety-critical systems, we focus on aircraft as an example of such

systems. In particular, we use the Preliminary System Safety Assessment (PSSA) from ARP4761 [1] as example which demonstrates safety process-related requirements. Aircrafts must be accompanied by a safety case that provides assurance about their behaviour as well as about the set of processes that were adopted to develop them. Safety cases can/should also reflect the compositional nature of the systems under examination. The provision of a safety case may follow a reference process [3]. The planning and execution of all the recommended reference processes is a time consuming and costly activity. Moreover, given the compositional and geographically distributed nature of the supply network, different interpretations of the processes may coexist resulting in conflicts and ultimately risk of low-quality products. To reduce time, cost as well as conflicting interpretations we propose to adapt a model-driven, cloud-based process enactment architecture for safety-critical systems. Using cloud computing not only reduces cost (through the pay-as-you-go and on-demand acquisition models), but also provides an accessible platform for the distributed teams involved in the system engineering process. In addition, artefacts from across the different geographical locations can be maintained centrally. Along with the use of a standardized process modelling language, this can reduce the conflicting interpretations. Our vision is that a manufacturer enforces the execution of the planned safety lifecycle as well as of the corresponding argumentation process. To achieve that vision, the paper provides the following contributions: a) an extension of our cloud-based process enactment architecture from [4] for safety-critical systems (Section 3.1), b) automation of product-based safety argument generation from failure logic analysis results and automation of evidence gathering, in particular, detecting sources of failures (if exist) or finding full and partial mitigators (Section 3.2), and c) automation of the generation of process-based arguments directly from the process model and enactment-related provenance data (Section 3.2). We demonstrate the usability and effectiveness of this approach on a portion of a safety process (from PSSA) which we enact on the cloud and generate product and process based safety arguments fragments. These fragments are manually integrated within a single safety case.

The paper is structured as follows: Section 2 provides a background foundation. Section 3 describes our approach to enacting safety processes followed by a case study in Section 4. Section 5 discusses related work and finally, Section 6 draws our conclusions and highlights our future work.

2 Background

In this section, we recall essential information on our previous work and on safety-critical systems engineering and certification.

2.1 General Architecture on the Cloud

To enact software processes on the cloud and ease global software engineering, we proposed a model-driven cloud-based architectural solution [4]. Our solution

consists of the three layers: a) the modelling layer where processes along with their enactment requirements are modelled, b) the enactment service layer which orchestrates the process enactment on the cloud and c) the workflow engines which are deployed on the cloud and host the enactment of individual activities. We implemented a prototype of this architecture consisting of two main components: **The Enactment Service** which consists of subcomponents for: scheduling activities execution, monitoring executions, managing artefacts and monitoring and registering workflow engines. The enactment service maintains a document-oriented database where all artefacts, activities and their meta-data are stored. Interactions with the enactment service are done through a RESTful API. **The Workflow Engine** is where the individual process activities are executed. The execution on the workflow engine is black-boxed: a workflow engine executing an activity of a process does not have information about the rest of the process execution. These two components of the prototype are decoupled (they communicate through asynchronous message queues) and are platform-independent (i.e. they can be deployed on any physical/virtual machine).

2.2 EXE-SPEM

To model the software processes and thier enactment requirements, we proposed an extension, called EXE-SPEM [5], of the OMG standard Software and Systems Process Engineering Meta-model (SPEM2.0). EXE-SPEM permits process engineers to model important information needed for enabling the enactment such as: control flow of process enactment (i.e., order, conditions and loops), the responsible person for enacting each activity (task), and the cloud-specific enactment information such as: the choice of cloud deployment model (private vs. public) and the amount of computational resources required. Table 1 shows the icons of a subset of EXE-SPEM concrete syntax, obtained by decorating the SPEM2.0 icons with the symbol of the cloud. Via model-to-text transformational

Table 1: Subset of EXE-SPEM modelling elements

Process	TaskUse	Activity	WorkProduct
			

rules, EXE-SPEM models are mapped onto XML-based textual representations, compliant with our enactment-oriented XML-meta-model.

2.3 Aircraft Engineering and Certification

To engineer and certify safety-critical systems, various standards are at disposal. Typically, these standards provide requirements that should be followed to define the process to be used during the development and assessment of the aircraft and the software and hardware to be integrated within the aircraft. A document aimed at showing process compliance by providing a process-based argument is typically required. Besides the process requirements, safety standards also

include product requirements aimed at assessing the level of a product's safety. Additional requirements target the assessment process, which, as mentioned, in many application domains is conducted by scrutinising an explicit or implicit safety case. A complete safety case as the final output of the assessment process should contain both the process and the product-based arguments.

ARP4761 [1] defines Airworthiness Safety Assessment Process to handle hazardous events (system and equipment failure or malfunction that may lead to hazard). This process includes: Functional Hazard Assessment (FHA), Preliminary Aircraft Safety Assessment (PASA) and PSSA. PSSA consists of a systematic examination of a proposed system architecture(s). It takes in input the system FHA and the aircraft Fault Tree Analysis. PSSA tasks include: identifying the derived safety requirements, associating them with Development Assurance Levels (DALs) and allocating them to architectural elements. For certification purposes, both process- and product-related behavioural evidence constitute the basis for supporting safety claims. Thus, additional tasks that should be considered are: creation of arguments fragments explaining why the safety claim can be supported. PSSA is conducted according to guidelines contained in Appendix B3 of ARP4761. In this paper, we use GSN [3] and SACM [15] for representing safety case arguments. We refer the readers to [3, 15] for details about these notations.

2.4 Process and Product-Based Arguments Fragments Generation

The product-based argument aims at showing that the product behaves as it should. To automate the generation of such arguments, the analysis and verification results can be exploited. For example, information about the failure behaviour of the system, extracted from the Fault Propagation and Transformation Calculus (FPTC) results, is used to generate an argument that the unacceptable failures have been successfully mitigated [17]. FPTC is a failure logic analysis allowing for the calculation of the system level failure behaviour based on the failure behaviour of the individual components. The propagation of failures from the inputs to the outputs of a component are captured via FPTC rules.

The process-based argument aims at showing that the process mandated by the corresponding standard has been followed. To automate the generation of such arguments, MDSafeCer (Model-driven Safety Certification) [8] is at disposal. Via MDSafeCer, process models compliant with e.g., SPEM 2.0 are transformed into composable process-based argumentation models compliant with e.g., SACM and presented via e.g., GSN. The top level claim of the MDSafeCer generated arguments states that "the process is in compliance with the required standard and integrity level". This claim is decomposed by showing that all the activities have been executed and that in turn for each activity all the tasks have been executed and so on until an atomic work-unit is reached.

3 Cloud-based Engineering of Safety-Critical Systems

Through the introduction and background, we have learnt that safety-critical systems engineering is a complex task which needs to comply with standards

and involves heterogeneous and geographically-distributed stakeholders. Undertaking such a complex engineering task requires extensive support. In this paper, we identify some requirements that a development environment/platform should satisfy to fit for safety-critical systems engineering. These are: **R1. Process enforcement and reuse:** Despite the dynamicity of safety-critical processes, they still need to be enforced (including enforcing change as it happens) to maintain consistency and compliance. To avoid misinterpretations of the process (and its changes) in a distributed setting, the process should be executable. Process customization and reuse across similar projects should also be supported to save time and cost. **R2. Distribution management:** The distribution of stakeholders not only bring communication and time difference challenges, but also brings cultural and language hindrances which might lead to misinterpretations and lack of trust between collaborating teams. This raises the need for synchronization and mutual understanding of the development process in order to minimize failure propagation between sub-systems built by different teams. **R3. Safety artefacts management:** Safety artefacts range from safety requirements to safety cases and safety arguments. In a dynamic and global environment, manually managing safety artefacts and continuously ensuring their consistency and compliance is an expensive task. Artefacts can be physically distributed and co-authored by multiple distributed teams. Capturing artefacts and their meta-data is also related to capturing safety evidence that is used to support safety cases.

This list is not comprehensive and other necessary requirements may exist. Therefore, the development platform should be extensible. In this paper we focus on supporting the ones mentioned above.

3.1 Extended Architecture for Safety-Critical Systems Engineering

In a previous work [4], we focused on supporting global software development using a model-driven cloud-based architecture. In this work, we extend that architecture to fully satisfy the requirements identified in Section 3. The extension affects the following: **Artefacts:** we introduce versioning of artefacts in which each change introduced to an artefact is treated as a new version. The versions (and meta-data) are kept in a central repository on the cloud. This satisfies requirement *R3* as artefacts are unified and versions capture their change. **Execution scheduler:** we enable parallel execution of activities which are ready to execute (i.e. their input artefacts are ready). **EXE-SPEM:** we enable capturing some safety-related elements in the process model. Those elements are: certification information for roles, the qualification of activities and the guidance and standard each activity adheres to. The executability of models ensures that a process is enforced, which satisfies requirement *R1*. As models can be edited/re-enacted, reuse of processes and activities becomes possible. In addition, *R2* is satisfied since a single process model with its enactment semantics is centrally shared between stakeholders. This gives each stakeholder a global awareness of the progress. **Argument generation support:** we extended the enactment service to support generating safety arguments from process models. This is done by capturing artefacts and activities execution meta-data and extract safety cases content from it.

```
S: the set of system components; HE: the set of undesired hazardous events
M: list of mitigators; PM: list of partial mitigators
for each he in HE {
       if(he.criticality > negligible)
              if(he exists on the system output)
                    trace_failure_to_the_source();
              else
                    for each component s in S
                      if(he is present on s.input)
                             if(he is not on s.output){
                               M.add(s);
                               find_the_mitigating_rule();
                              }else
                                    if(the source of he on s.output != s.input)
                                          PM.add(s);}
```

Fig. 1: The pseudo code for analysing the FPTC results.

```
R1: Make CLAIM "All causes of hazardous Failure Modes are acceptable"
R2: For each hazardous event {he} in the set HE, apply the following:
  R2.1: If {he} is negligible, make a CLAIM "Hazardous Failure Mode {he} is negligible"
  R2.2: If {he} is not negligible, make a CLAIM "Hazardous Failure Mode of type {he} absent
        in contributory software functionality" and attach CONTEXT "Known causes of {he}
        failure mode"
    R2.2.1: If {he} is present on the output, make COUNTER-EVIDENCE "The {he} Hazardous
          Failure Mode present in the contributory software functionality. Check traces."
    R2.2.2: If {he} is not present on the system output, make a STRATEGY "Argument over
          failure mechanisms" and attach a JUSTIFICATION "Identified failure mechanisms
          describe all known causes of {he} hazardous Failure Mode"
      R2.2.2.1: make a CLAIM "The known causes of secondary failures of other components are
            acceptably handled" and leave it undeveloped.
      R2.2.2.2: make a CLAIM about the mitigators "Hazardous event {he} has been mitigated by
             {mitigators}" and attach an EVIDENCE "Mitigation details in the textual argument"
            .
```

Fig. 2: Rules for product-based argument generation.

3.2 Argument Generation

Product-based argument: We generate product-based arguments from FPTC analysis results. By analysing if certain failures/hazardous events (HEs) occur or not, we can argue about how the system handles HEs. The analysis starts by parsing the FPTC results and following the pseudo code in Fig. 1. Then the argument is formulated by constructing *Claims* and *Strategies* and supporting them by *Evidences/Counter-Evidences* following the rules in Fig. 2. These rules are adapted from [17] where the generation of product-based argument-fragments is made from contracts translated from FPTC analysis. In this work we provide rules for generation of argument-fragments directly from the FPTC analysis, thus skipping the translation of FPTC specification to contracts. Moreover, we provide more fine-grained analysis of how the system handles HE based on the FPTC specification. If the HE is present in the system, we produce a counter-evidence in the form of a trace to the source(s) of the HE. If it is not, we find the component(s) that mitigated it. We distinguish between partial or full mitigation. Full mitigation is when the failure does not propagate from a component's input to its output, while partial mitigation is when the failure is present on the output, but at least one of the input causes of the output failure has been mitigated by the component.

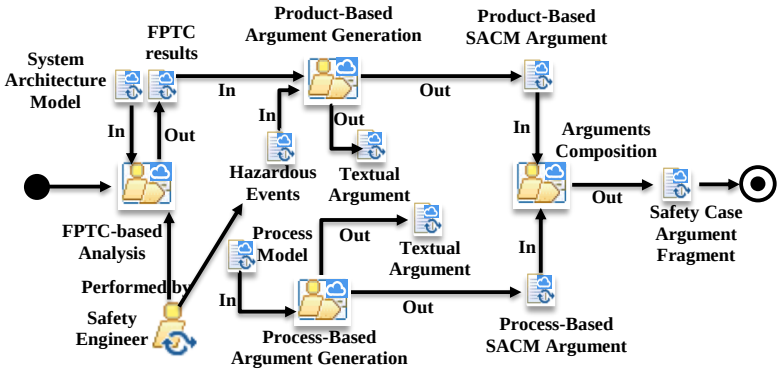

Fig. 3: PSSA augmented with the argument generation process.

Process-based argument: We use the the rules explained in MDSafeCer [8] to structure a process-based safety argument fragment. As explained in Section 2.4, the fragment argues about: the tools used, the roles involved, the guidances/standards followed and the work products generated/consumed in the process. Information about these aspects is extracted from both the process model and provenance data about the process enactment.

Once the product and process-based argument fragments are generated, they are joined with a top claim arguing about the overall system safety to compose the overall safety case argument fragment.

4 Case Study

The purpose of this case study is to demonstrate the cloud-based execution of the augmented PSSA process.

Fig. 3 shows the EXE-SPEM model of the PSSA augmented with the argument generation process. It consists of four activities and involves creation of multiple artefacts. The *FPTC-based Analysis* activity analyses the failure behaviour of a system. It takes as an input the system architecture model and generates as an output the failure behaviour of the system. As mentioned in Section 3.2 this failure behaviour can be used by the next activity (*Product-based Argument Generation*) to verify if the *undesired hazardous events* (identified after performing FHA) have been mitigated. The *Process-based Argument Generation* activity uses the process model and provenance data about the process enactment to populate the process-based safety argument. Finally, the *Arguments Composition* activity combines both the product and the process based arguments into one safety argument fragment.

In this case study, we used the airplane Wheel Braking System (WBS) adopted from ARP4761 [1]. The WBS consists of the Brake System Control Unit (BSCU) and the hydraulics system which is connected to the wheels of the airplane. We limit our attention to the portion of the architecture that comprises the BSCU (shown in Fig. 4). Since the FHA process for the WBS system is out of

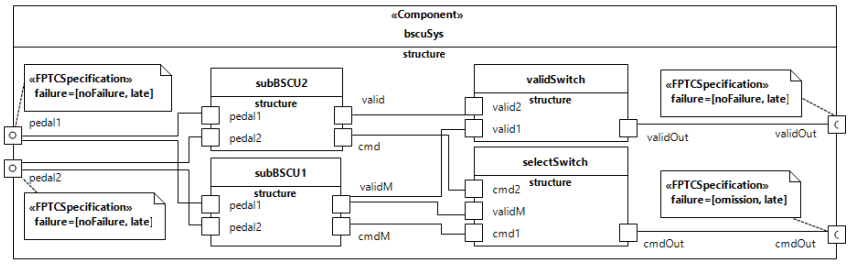

Fig. 4: The Brake System Control Unit (BSCU)[17].

the scope of this case study, we have randomly selected the *undesired hazardous events* that the system should mitigate and provided them as an input to the *Product-based Argument Generation* activity.

4.1 Implementation

After modelling the PSSA augmented with the argument generation process (Fig. 3), the model is mapped onto XML to be enacted on the cloud-based architecture. Below, we describe the implementation of each of the activities used in the safety argument generation process. **FPTC-based Analysis:** This activity uses Concerto-FLA (the extended FPTC implementation from the CONCERTO project [3]) to perform the FPTC analysis. The CONCERTO toolset allows: creating UML-based architectural models of the system; performing FPTC analysis (using Concerto-FLA) including back-propagation of the results on the models. The architectural model is transformed to the *flamm* format (an XML-like format) on which the analysis takes place. The *flamm* model consists of composite components (systems) containing atomic components. The (atomic) components have input and output ports where failures are attached. In addition, each component has a set of rules defining its failure behaviour. For this case study, we have extracted the FPTC analysis part from Concerto-FLA into this standalone activity which embed the analysed failure behaviour of the system into the *flamm* model. **Product-based Argument Generation:** This activity uses the FPTC analysis results to construct the argument concerning the BSCU. Each undesired HE is accompanied by a definition of its criticality level. These levels are mapped to a five-level numerical criticality scale ranging from 1 (lowest criticality) to 5 (highest criticality). For instance, in ARP4754A [2], the levels are 1: *negligible*, 2: *minor*, 3: *major*, 4: *hazardous*, 5: *catastrophic*. The tracing and mitigation details are presented in an extended textual argument following the *Argument Outline* format [12] and is referenced in the SACM/GSN arguments. Fig. 5 shows the generated product-based GSN argument for the BSCU while Fig. 6 shows a snippet of the textual argument. It is worth noting that we use the GSN *solution* notation to represent counter evidences (as in *S1.2* in Fig. 5).

Process-based Argument Generation: This activity generates the argument arguing about compliance with PSSA. Fig. 7 shows an argument for

[3] www.concerto-project.org/

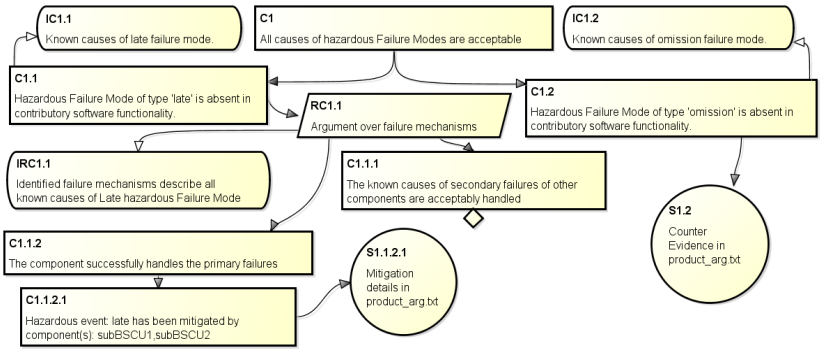

Fig. 5: GSN representation of the generated product-based argument.

```
...
CLAIM 1.1: HAZARDOUS FAILURE MODE OF TYPE 'OMISSION' IS ABSENT IN
     CONTRIBUTORY SOFTWARE FUNCTIONALITY.
         CONTEXT 1.1: Known causes of omission failure mode.
         COUNTER_EVIDENCE 1.1: The omission Hazardous Failure Mode is present
     in the contributory software functionality. Check the traces.
         CONTEXT 1.1: omission CAUSED BY:
         Failure: 'omission' On Output Port: 'cmd' of Component: 'selectSwitch
     '.  CAUSED BY: {Failure: 'omission' On Input Port: 'cmd2' of Component: '
     selectSwitch'.  CAUSED BY: Failure: 'omission' On Output Port: 'cmd' of
     Component: 'subBSCU2'.  CAUSED BY:  ...
```

Fig. 6: The product-based argument represented in text.

FPTC-based Analysis activity from the safety argument generation process we used in this case study. The tools and roles we used, are not qualified. Therefore, undeveloped goal is attached for them. The *failures_list* artefact corresponds to the result of the FTA analysis as required by Appendix B4.1 of ARP4761.

Arguments Composition: This activity combines the process and product-based arguments into one arguing about the overall system safety.

4.2 Execution

We enacted the safety argument generation process model (shown in Fig. 3) in the prototype of our extended cloud-based enactment architecture. We deployed the *Enactment Service* and one *Workflow Engine* on two different Amazon EC2 "t2.small" machines. Using a web browser, we were able to enact the process and retrieve the generated artefacts containing the FPTC analysis results and the safety arguments (separate and combined) in both SACM/ARM XMI and text formats. The SACM/ARM XMI formats were then converted into GSN diagrams (Fig. 5,7) using the Astah GSN editor [4].

4.3 Discussion

By enacting the safety argument generation process on the cloud we demonstrated the application of our cloud-based enactment architecture for safety-

[4] http://astah.net/editions/gsn

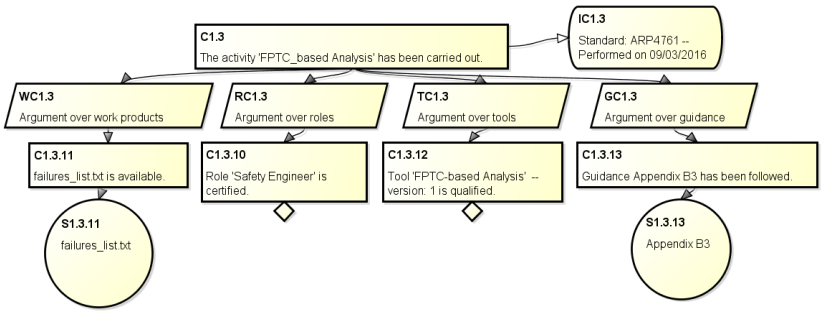

Fig. 7: GSN representation of partial process-based argument.

critical processes. While we have used a process from an aerospace domain standard, processes from other standards can be modelled and enacted similarly. The enactment architecture is our target platform for modelling/development of safety processes. It is a service-oriented architecture and can be deployed into any cloud deployment model (public, private or hybrid). It can also be interfaced with existing platforms as a service call. This flexibility can address security and privacy concerns when using the cloud, i.e. one can use a private cloud to host the process enactment (partially or fully as each activity can be configured differently) and the generated artefacts. Furthermore, the architecture can be extended to support new rising requirements other than the three mentioned in section 3. In this paper we showed how we extended our initial architecture [4] as detailed in Section 3.1. This involved extending the modelling language (EXE-SPEM) to model new requirements and extending the architecture components to incorporate the new required behaviours.

However, there are some limitations to the type of activities that can be supported. Software processes are often long-running and typically would involve human-intensive activities. The implemented prototype of the architecture does not yet support intensive interactions with humans during process execution. Capturing those interactions provides more data that could be integrated into safety arguments. Furthermore, a failure/exception during a long-running process will break the execution and in the current prototype, the process will need to be restarted. It is essential to have support to pause/resume processes in such situations. Since we do not have support to resume process enactment in case of failures, we recommend splitting processes into short-living sub-processes. Sub-processing also means better separation of concerns between teams. Finally, not all activities within a process can be automated and the borders between what can/cannot be automated is not defined yet. The benefits from automation remain, however. The automation of arguments generation saves time and cost and utilizes the enactment architecture to capture and generate supporting evidences for the arguments. The approach we propose does not address the issue of completeness of requirements, hazards etc. As Leveson [14] points out, there will be always hazards that are not considered and that depends on assumptions, the uncertainties and limitations of the used methods.

5 Related Work

As already pointed out by Sljivo et al [17], there has been extensive research of safety case argumentation management and argument generation. For example, Hawkins et al. [11] propose a model-based approach for automated generation of assurance cases from automatically extracted information from the system design, analysis and development models. The approach uses model weaving to capture the dependencies between the reference information models and the assurance argument patterns. The Model Based Assurance Case (MBAC) program is in the heart of the prototype tool that implements the approach [10]. MBAC takes the argument pattern, reference information and weaving models as its input together with the corresponding metamodels, and provides an instantiated argument model as the output. While the weaving approach represents a more generic solution idea, our approach complements that work by looking at the specific information and argumentation pattern models and providing the corresponding model transformation rules. Most of the related approaches to argumentation management (e.g., [6, 7]), however, lack support for distributed and remote safety case development for distributed teams. Moreover, these approaches do not address the potential need for scalable computational power needed for certain tasks in the overall safety certification enactment process. Our work offers a cloud-based solution that allows integrated coproduction of the safety case by geographically distributed teams. Furthermore, we do not only support the product, but also the process-based side of the argument. Gorski et al. [9] present an evidence-based argument management methodology TRUST-IT and a cloud-based software-as-a-service platform called NOR-STA supporting the application of this methodology. Similarly to GSN-goal structures, the TRUST-IT argumentation model represents evidence-based arguments in a tree-like structure. In contrast to NOR-STA, we aim at providing a complete process enactment service on the cloud where argumentation management is not treated as an activity separated from the activities mandated by the standards. Producing the evidence and managing the argumentation on the same platform allows us to automate the creation of the argument fragments that can be later combined in the overall safety case. Furthermore, by generating the argument fragments in a standardised format we support portability.

6 Conclusion and Future Work

This paper starts with listing a set of requirements for a development environment that supports the enactment of safety-critical processes. To meet such requirements we extend our previous model-driven cloud-based software process enactment architecture [4] to support the safety critical processes. We present a fragment of a process adapted from the aerospace domain and demonstrate its executability on the cloud. While our proposal brings the economical benefits of the cloud to safety-critical systems engineering, empirical studies and industrial collaborations are still needed to study the impacts of our proposal at

both the organizational and individual levels and on the quality and safety of the produced systems.

To take this work further, we plan to develop a support for continuous compliance modelling and checking, for enabling extensive human interactions and off-line activities, as well as for sub-processing to allow long-lived processes typical for the aerospace domain.

References

1. ARP4761: Guidelines and Methods for Conducting the Safety Assessment process on Civil Airborne Systems And Equipment. (1996)
2. ARP4754A, Guidelines for Development of Civil Aircraft and Systems. SAE International (2010)
3. GSN: Community Standard Version 1. Origin Consulting (York) Limited (2011)
4. Alajrami, S., Gallina, B., Romanovsky, A.: Enabling global software development via cloud-based software process enactment. Tech. Rep. TR-1494, Newcastle University, School of Computing Science (03 2016)
5. Alajrami, S., Gallina, B., Romanovsky, A.: Exe-spem: Towards cloud-based executable software process models. In: Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (2016)
6. Armengaud, E.: Automated Safety Case Compilation for Product-based Argumentation. In: Embedded Real Time Software and Systems (2014)
7. Denney, E., Pai, G.J.: Automating the Assembly of Aviation Safety Cases. IEEE Transactions on Reliability 63(4), 830–849 (2014)
8. Gallina, B.: A Model-driven Safety Certification Method for Process Compliance. In: 2nd International Workshop on Assurance Cases for Software-intensive Systems. pp. 204–209. IEEE (2014)
9. Górski, J., Jarzebowicz, A., Miler, J., Witkowicz, M., Czyznikiewicz, J., Jar, P.: Supporting Assurance by Evidence-Based Argument Services. In: 1st Workshop on Next Generation of System Assurance Approaches for Safety-Critical Systems. LNCS, vol. 7613, pp. 417–426. Springer (2012)
10. Hawkins, R., Habli, I., Kelly, T.P.: The Need for a Weaving Model in Assurance Case Automation. Ada User Journal 36(3), 187–191 (Sep 2015)
11. Hawkins, R., Habli, I., Kolovos, D., Paige, R., Kelly, T.P.: Weaving an Assurance Case from Design: A Model-Based Approach. In: 16th International Symposium on High Assurance Systems Engineering. pp. 110–117. IEEE (Jan 2015)
12. Holloway, C.M.: Safety case notations: Alternatives for the non-graphically inclined? In: 3rd IET International Conference on System Safety. pp. 1–6 (2008)
13. Holloway, C.M.: Explicate '78: Uncovering the implicit assurance case in do-178c. Tech. Rep. 20150009473, NASA Langley Research Center (2015)
14. Leveson, N.: White paper on the use of safety cases in certification and regulation. Tech. rep., MIT (May 2012)
15. (OMG), O.M.G.: SACM: Structured Assurance Case Metamodel (2013)
16. Rushby, J.: New challenges in certification for aircraft software. In: 9th ACM International Conference on Embedded Software. pp. 211–218. EMSOFT (2011)
17. Sljivo, I., Gallina, B., Carlson, J., Hansson, H., Puri, S.: A Method to Generate Reusable Safety Case Fragments from Compositional Safety Analysis. In: 14th International Conference on Software Reuse. pp. 253–268. LNCS, Springer (2015)