Towards Increased Efficiency and Confidence in Process Compliance

Julieth Patricia Castellanos Ardila and Barbara Gallina

IDT, Mälardalen University
Box 883, 721 23 Västerås, Sweden
{julieth.castellanos,barbara.gallina}@mdh.se

Abstract. Nowadays, the engineering of (software) systems has to comply with different standards, which often exhibit common requirements or at least a significant potential for synergy. Compliance management is a delicate, time-consuming, and costly activity, which would benefit from increased confidence, automation, and systematic reuse. In this paper, we introduce a new approach, called SoPLE&Logic-basedCM. SoPLE&Logic-basedCM combines (safety-oriented) process line engineering with defeasible logic-based approaches for formal compliance checking. As a result of this combination, SoPLE&Logic-basedCM enables automation of compliance checking and systematic reuse of process elements as well as compliance proofs. To illustrate SoPLE&Logic-basedCM, we apply it to the automotive domain and we draw our lessons learnt.

Keywords: ISO 26262, Automotive SPICE, compliance by design, reuse, defeasible logic, process assessment, software process improvement.

1 Introduction

In the context of safety critical systems engineering, quality (and more specifically safety) standards act as a baseline aimed at contributing to "assuring society at large that deployment of a given system does not pose an unacceptable risk of harm" [1]. Standards impose requirements on the processes to be adopted to engineer the systems as well as on the expected behaviour of the systems. To adhere to the requirements regarding the processes, companies adapt their practices, and provide evidence (e.g., arguments or even proofs of compliance), which to some extent supports the fulfilment of the requirements. Providing such evidence is a time-consuming and costly activity, which risks to steal time and focus from other activities related to e.g., verification of systems behaviour. Since the ultimate goal of our work is to free time for such verification activities, we believe that process compliance would be highly benefit from automation and systematic reuse. Moreover, confidence in the evidence could be increased via logic-based approaches. Safety-oriented Process Line Engineering (SoPLE) [2, 3] permits process engineers to systematise the reuse of process-related information. To argue about or prove compliance, SoPLE is not enough. In a previous work [4, 5], SoPLE was combined with argumentation patterns and model-driven engineering principles to automate the creation of reusable process-based argument fragments aimed at showing compliance. In this paper, we intend to provide an additional layer of confidence by offering a logic-based framework that enables formal proofs of compliance. To do that, we build on top of results stemming from the business process-related community and legal compliance. Specifically, we use defeasible logic, a rule-based approach for efficient reasoning with incomplete and inconsistent information, a typical scenario in normative systems [6]. Our approach represents a novelty which contributes to 1) increasing efficiency (via automation and systematic reuse) and confidence (via formal checking) in process compliance, and 2) cross-fertilising previously isolated communities. In this paper, we do not only present our new approach but we also apply it to the automotive domain. In particular, we consider ASPICE (Automotive Software Process Improvement and Capability Determination) [7], which provides a software process assessment model, and ISO 26262 [8], a safety standard that regulates the development process of safety-critical automotive systems. The motivation for this choice is that it is well-known that process reference models of these two standards overlap and exhibit several similarities [3, 9], specially in process elements related to software and system engineering [10].

The rest of the paper is organised as follows. In Section 2, we provide background information related to our work. In Section 3, we introduce SoPLE&Logic-basedCM for efficient and confidence process compliance. In Section 4, we apply SoPLE&Logic-basedCM to the automotive domain, and based on the application of our approach, we derive our lessons learned. In Section 5, we discuss related work. Finally, in Section 6, we present conclusions and future work.

2 Background

This section provides basic information on which we base our work. In Section 2.1 and 2.2, we present two automotive standards. In Section 2.3, we recall SoPLE. In Section 2.4, we present defeasible logic, and in Section 2.5, we recall an abstract formal framework for regulatory compliance.

2.1 Automotive SPICE

ASPICE [7] is a standard that addresses the software process capability maturity in automotive. To determine maturity, the process assessment model selects the process reference model and augments it with indicators. These indicators are used to identify if the process outcomes (PO), the result of the achievement of the process, and the process attribute outcomes (PA), the result of the achievement of a specific process attribute, are present. Base practices (BP) (activity-oriented PAs), must be evaluated to establish the capability of the process to be achieved. BPs for the process Software Detailed Design and Unit Construction (SWE.3) are: *BP1-Develop software detailed design*, *BP2-Define interfaces of software units*, *BP3-Describe dynamic behavior*, *BP4-Evaluate software detailed* *design*, *BP5-Establish bidirectional traceability*, *BP6-Ensure consistency*, *BP7-Communicate agreed software detailed design*, and *BP8-Develop software units*. These BPs are related to one or more of the POs presented in Table 1.

Table 1. POs for ASPICE SWE.3.

ID	Process outcome description
PO1	A detailed design is developed that describes software units.
PO2	Interfaces of each software unit are defined.
PO3	The dynamic behavior of the software units is defined. NOTE: Not all software units have dynamic behavior to be described.
PO4	Evaluate the software detailed design in terms of interoperability, interaction, criticality, technical complexity, risks and testability.
PO5	Consistency and bidirectional traceability are established between e.g., software requirements and software units. NOTE: Consistency is supported by bidirectional traceability.
PO6	The software detailed design and the relationship to the software architectural design is agreed and communicated to all affected parties.
PO7	Software units defined by the software detailed design are produced.

2.2 ISO 26262

ISO 26262 [8] is a standard that focuses on the functional safety of electrical/-electronical systems in vehicles (gross mass up to 3500 kg). In ISO 26262, ASIL (Automotive Safety Integrity levels) are used to specify applicable safety requirements, but both safety and non-safety requirements are implemented within one development process. Specifically, in the sub-phase Software Unit Design and Implementation (SUDI), described in part 6, clause 8 of the standard, single software units are addressed, and the following activities are included: *A1-Specify the software units*, *A2-Verify the software unit design*, *A3-Implement the software units*, and *A4-Verify the software unit implementation*. These activities are related to one or more of the requirements presented in Table 2.

Table 2. Requirements for ISO 26262 SUDI.

ID	Requirements description
R1	The requirements of this subclause shall be complied with if the software unit is safety-related ("Safety-related" means that the unit implements safety requirements).
R2	Software units are designed by using a notation that depends on the ASIL and the recommendation level.
R3	The specification of the software units shall describe the functional behaviour and the internal design to the level of detail necessary for their implementation.
R4	Design principles for software unit design and implementation shall be applied depending on the ASIL and the recommendation levels to reach properties like consistency of the interfaces, correct order of execution of subprograms and functions, etc.
R5	Software unit design and implementation are verified by applying verification methods according to the ASIL and the recommendation levels to demonstrate, e.g., traceability.
R6	When ASIL and recommendation levels are not followed, a rationale that explains the reasons for this behavior must be provided (Interpretation of tables, ISO 26262-Section 4.2).

2.3 SoPLE

As recalled in the introduction, SoPLE is a methodological framework to systematically model commonalities and variabilities between highly-related processes to facilitate reuse and flexible process derivation. To identify commonalities and variabilities, common terminology that allows the comparisons between the standards is required. In [3], a mapping of common terms between ASPICE and ISO 26262 is provided (see Table 3). These terms are used as follows: if an *activity* in ISO 26262 is equivalent to a *base practice* in ASPICE, the elements are mapped to the common identifier *activity*, and are modeled in SPEM2.0/EPF (Eclipse Process Framework)-Composer with a *TaskUse*. SPEM2.0/EPF-Composer is suggested in the application of SoPLE. SPEM2.0 (Software and Systems Process Engineering Metamodel) [11] is a standard that provides the elements required to define software and systems development process. SPEM2.0 is implemented in EPF Composer [12], a tool able to store reusable core methods separated from its application in processes. One basic method content is the *Task*, which symbolizes an assignable unit of work. *Method content variability* allows adaptation of created content without affecting the original content. We recall one variability type called *contributes*, which provides a way for process elements instances to contribute with their properties into the base variability element. Process structures can be built incorporating method content elements (for example, a task realized as a TaskUse) in a *breakdown structure*. *Commonalities* in processes are usually *partial*, i.e., a process element contains a subset of common aspects. Common aspects constitute the *commonality points (CP)* while *variability points (VP)* are the process elements that are replaced with particular instances of process elements (called variants). It should be noted that in SPEM2.0 there is no notion of variability point, thus, we introduce an empty task, which is made vary via contributes.

Table 3. Mapping of terms in ISO 26262, ASPICE and SPEM2.0/EPF [3].

Common Identifier	ISO 26262	ASPICE	SPEM2.0/EPF
Activity	Activity	Base Practice	TaskUse/ 🔄

2.4 Defeasible Logic

Defeasible logic [13] is a rule-based logic that provides reasoning with incomplete/inconsistent information. A defeasible theory is a knowledge base in defeasible logic, which contains: a) *facts:* indisputable statements; b) *strict rules:* rules in the classical sense, whenever the premises are indisputable, so is the conclusion; c) *defeasible rules:* rules that can be defeated by contrary evidence; d) *defeaters:* rules used only to prevent conclusions; e) *superiority relation:* a relation among rules used to define priorities. Formally, r: A(r) $\hookrightarrow C(r)$, a rule r consists of an antecedent A, the consequence of the rule C, and the rule

$\hookrightarrow = \{\rightarrow (strict), \Rightarrow (defeasible), or \leadsto (defeater)\}$. A defeasible proof requires that we: a) Put forward a supported rule for the conclusion we want to prove; b) consider all possible reasons against the desired conclusion; and c) rebut all counterarguments, by either showing that some premises of the counterargument do not hold, or the argument is defeated by another argument.

2.5 Compliance Checking Approach

In this subsection, we recall the abstract formal framework for modeling *compliance by design* defined in [6], an approach in which compliance of a process with a set of rules is verified before deploying. This approach is based on deontic logic of violations [14], in which deontic notions are modelled using defeasible logics. Deontic notions are present in normative systems e.g., an *obligation* is a deontic effect that arises when a norm bounds the bearer to a specific situation. When a violation occurs, a reparational obligation is in force. For compliance checking, we should: 1) determine the obligations of the rules, 2) determine the state of each task in a process, 3) determine the obligations in force for each task, and 4) check if the obligations in force have been fulfilled or violated. The approach requires that the *traces* of the process (sequence of tasks, in which a process can be executed, respecting the order given by the connectors), and semantic annotations (functions that describe the environment in which a process operates) are defined. The function *Ann(n,t,i)* returns the state of a *trace (n)* obtained after a *task (t)*, in the *step (i)*. The function *Force(n,t,i)* = $\{p\}$ associates to each *task (t)* in a *trace (n)*, in the *step (i)* a set of *obligations (p)*.

3 SoPLE&Logic-basedCM

This section provides an overview of SoPLE&Logic-basedCM (see Fig. 1), our approach for increasing confidence and efficiency in process compliance, by combining safety-oriented process line engineering, the definition of defeasible theories as presented in Section 2.4, and the use of the framework for modelling compliance, presented in Section 2.5. As Fig. 1 depicts, a process engineer is expected to: 1) model a SoPL (which includes manually modelling the skeleton of the process sequence); 2) formalise the standards rules, select the set of rules that overlap, and analyse the compliance of the SoPL commonalities with the overlapping rules; 3) analyze the effects of the tasks that contributes to the variabilities in the in the standard-specific process.

4 Applying SoPLE&Logic-basedCM

In this section, for illustration purposes, we apply SoPLE&Logic-basedCM to the software unit development process part provided by ASPICE and ISO 26262. The remaining part of this section is structured as follows: in Section 4.1, we model a SoPL. In Section 4.2, we define the proofs of compliance. In Section 4.3, we present the lessons learnt.

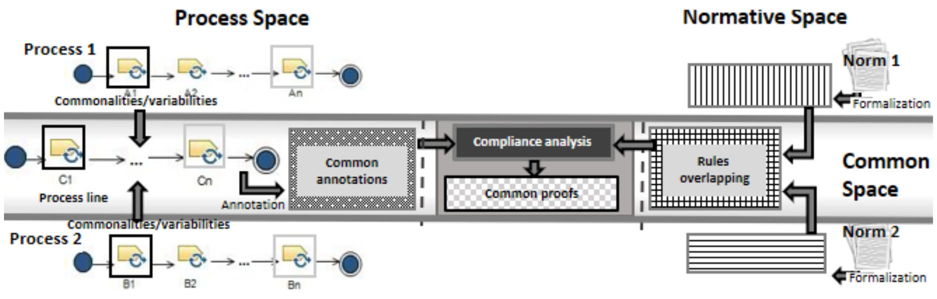

Fig. 1. SoPLE&Logic-basedCM overview.

4.1 SoPL Modeling

In this subsection, we apply SoPLE, recalled in Section 2.3. The scope is ASPICE SWE.3, and ISO 26262 SUDI recalled in Sections 2.1 and 2.2 respectively. In the domain engineering phase, we find the equivalent process activities by applying the terminology mapping presented in Table 3, and by analysing the scope of each activity. Terminological similarity is found between BP1 and A1. However, the scope of A1 (see Table 2 - R4) is broader than the scope of BP1, including also BP2 and BP3. Hence, the commonality point (CP1) is defined as a task called *Define software unit design*, which contains three successive steps, namely *develop software detailed design, define interfaces of software units,* and *describe dynamic behavior*. A similar analysis is done for CP2, where there is a correspondence between A2 (scope determined in Table 2 - R4/R5) with BP4/BP5/BP6. CP3 is a straightforward comparison between A3 and BP8. Our comparison also includes standard-specific variants, for example, ISO 26262 variants are activities that deal with ASIL. Variants of this type are *A1a-Define software unit design concerning safety* derived from A1, and *A2a-Verify the software unit design concerning safety* derived from A2. These and other activities that are standard-specific are represented as variability points (VP) (see Table 4). The result of the domain engineering phase is a SoPL, depicted in Fig. 2.

Table 4. SPICE SWE.3/ISO 26262 SUDI activities mapping.

ID	Step in the trace	ISO 26262	ASPICE	Common Name
CP1	1	A1	BP1, BP2, BP3	Define software unit design
VP1	2	A1a		Define software unit design concerning safety
CP2	3	A2	BP4, BP5, BP6	Verify the software unit design
VP2	4	A2a		Verify the software design concerning safety
			BP7	Communicate agreed software detailed design
CP3	5	A3	BP8	Implement the software units
VP3	6	A4		Verify the software developed units

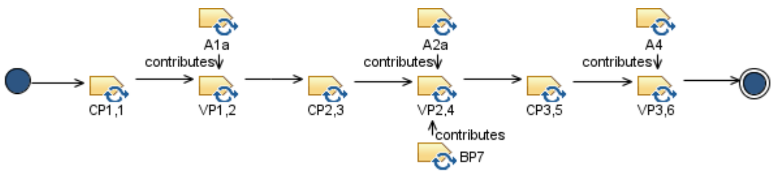

Fig. 2. SoPL model embracing ASPICE SWE.3 and ISO 26262 SUDI.

4.2 Definition of the Proofs of Compliance

In this subsection, we formalize the standards requirements and discover the overlapping set of formal rules. These rules are used to annotate the commonality points of the SoPL model to define common proofs of compliance. Then we define the effects of the rules that apply to the variability points of the SoPL model and analyse their effects in the common proofs. The formalization of the rules includes the definition of defeasible theories, as recalled in Section 2.4. For ASPICE SWE.3 (see Table 5) the rules PO2, PO4, PO5 and PO6 (see Table 1) can be translated into the strict rules RA3, RA5, RA6 and RA7 respectively, since these requirements are indisputable statements that are necessary to achieve for compliance. PO1 and PO7 are also indisputable, but each one can be expressed in a more granular way, RA1, RA2 and RA8, RA9, respectively. PO3, instead, is a defeasible rule (RA4), since the note, *"not all software units have dynamic behaviour to be described"*, presented in the requirement, defeats the rule. This defeasible rule does not have a defeater, so its conclusion is considered provable, as well as the conclusions of the strict rules.

Table 5. Defeasible theories for ASPICE SWE.3.

PO	**ID**	**Rule**	**Rule description**
PO1	RA1	sud $\rightarrow$ d.	software unit design (sud) is developed (d).
	RA2	sud $\rightarrow$ su.	sud describes software units (su).
PO2	RA3	su $\rightarrow$ i.	su has defined interfaces.
PO3	RA4	su $\Rightarrow$ db.	su has usually described dynamic behavior (db).
PO4	RA5	sud $\rightarrow$ v.	sud is verified (v).
PO5	RA6	su $\rightarrow$ tc.	su has established traceability and consistency (bt).
PO6	RA7	sud $\rightarrow$ ac.	sud is agreed and communicated (ac).
	RA8	sui $\rightarrow$ sud.	software unit implementation (sui) is based on sud.
PO7	RA9	sui $\rightarrow$ i.	sui is implemented (i).

For ISO 26262 SUDI, a similar analysis is done (see Table 6). Hence, nine strict rules and three defeasible rules have resulted. The defeasible rules have a defeated rule, namely RI12, which is in favor of the conclusions. For example, if RI7 is not achieved (*sud* $\Rightarrow \neg dp$) then a rationale is provided ($\neg dp \rightarrow r$). Rules of this type are provable, because their counterargument is a strict rule. The mapping of the defeasible theories is presented in Table 7. Direct mapping is done for the strict rules CR1 (RA1/RI2), CR2 (RA2/RI4), CR5 (RA8/RI8) and CR6 (RA9/RI9), since these rules affect the processes in a similar way. CR3 is the mapping between RA3 (definition of the interfaces) and RA4 (description of dynamic behavior) to RI6 (description of the internal design). This mapping is base on the premise that ISO 26262 is not specific on what the software unit should show as internal design. However, definition of interfaces and dynamic behavior are defined as properties that shall be reached by the software unit design (see Table 2 - R4). For CR4 a similar situation occurs, since traceability is considered one of the aspects that have to be demonstrated in ISO 26262 when verification is carried out (see Table 2 - R5). Hence, the mapping for CR4 is RA5 (software unit is verified) and RA6 (software unit has established traceability) to RI10 (software unit design is verified).

Table 6. Defeasible theories for ISO 26262 SUDI.

Req.	**ID**	**Rule**	**Rule description**
R1	RI1	sud $\rightarrow$ sr.	software unit design (sud) is safety related (sr).
R2	RI2	sud $\rightarrow$ d.	sud is design (d).
	RI3	d $\Rightarrow$ n.	d is usually implemented by using a notation that depends on the ASIL and the recommendation level (n).
	RI4	sud $\rightarrow$ su.	sud describes software units (su).
R3	RI5	sud $\rightarrow$ fb.	sud has described functional behavior (fb).
	RI6	sud $\rightarrow$ id.	sud has described internal design (id).
R4	RI7	sud $\Rightarrow$ dp.	sud is implemented by using design principles (dp) that depends on the ASIL and the recommendation level.
	RI8	sui $\rightarrow$ sud.	software unit implementation (sui) is based on sud.
	RI9	sui $\rightarrow$ i.	sui is implemented (i).
R5	RI10	sud, sui $\rightarrow$ v.	sud, sui are verified.
	RI11	v $\Rightarrow$ m.	v is usually done by using a method that depends on the ASIL and the recommendation level (m).
R6	RI12	$\neg n$, $\neg dp$, $\neg vm$ $\rightarrow$ r.	If n, dp or m are not applied depending on the ASIL and the recommendation levels, then rationale (r) is required.

The SoPL model (see Fig. 2) is constituted by one trace *tSoPL*. The effects of its tasks (*tSoPL:<CP1, VP1, CP2, VP2, CP3, VP3>*) are determined with the function *Ann* (defined in Section 2.5) and presented in Listing 1.1.

```
Ann(tSoPL,CP1,1)={CP1}
Ann(tSoPL,VP1,2)=Ann(tSoPL,CP1,1) U {VP1}
Ann(tSoPL,CP2,3)=Ann(tSoPL,VP1,2) U {CP2}
Ann(tSoPL,VP2,4)=Ann(tSoPL,CP2,3) U {VP2}
Ann(tSoPL,CP3,5)=Ann(tSoPL,VP2,4) U {CP3}
Ann(tSoPL,VP3,6)=Ann(tSoPL,CP3,5) U {VP3}
```

Listing 1.1. Annotations for the trace *tSoPL*.

A task determines its state taking its effect and inheriting the previous effects. Once the states are determined, the obligations in force (rules that apply to the tasks) are assigned, using the function *Force* (recalled in Section 2.5). In Table 8, common defeasible theories (Table 7) are assigned to the commonality

Table 7. Rules comparison and commonality identification.

SPICE SWE.3		ISO 26262 SUDI		Common Rule	Description
ID	**Rule**	**ID**	**Rule**		
RA1	sud → d.	RI2	sud → d.	CR1	Software unit design (sud) is developed (d).
RA2	sud → su.	RI4	sud → su.	CR2	sud describe software units (su)
RA3	su → i.	RI6	sud → id˙	CR3	Internal design is described, including interfaces and dynamic behavior
RA4	su ⇒ db.				
RA5	sud → v.	RI10	sud → v.	CR4	su is verified and traceability demonstrated
RA6	su → tc				
RA8	sui → sud.	RI8	sui → sud.	CR5	su implementation (sui) is based on sud
RA9	sui → i.	RI9	sui → i.	CR6	sui is implemented (i)

points (CPs) of the SoPL trace *tSoPL*. In *tSoPL*, rules CR1, CR2, CR3 are effective at CP1, meaning that for the software unit design task (CP1), software unit is designed (CR1), units are described (CR2), and the internal design, including interfaces and dynamic behaviour is described (CR3) (Proof 1). Rule CR4 is effective at CP2, meaning that in the verification of the software design task, the software is verified and traceability is demonstrated (CR4) (Proof 2). Finally, CR5 and CR6 are effective at CP3, meaning that in the develop of the software units activity, implementation is based on design (GR5) and unit implementation is carried out (GR6) (Proof 3). The obligations triggered by the rules are fulfilled in the corresponding step, meaning that the obligation cannot be postponed for other steps. As presented in Section 2.5, this means that the commonality points of the trace *tSoPL* are compliant with the set of rules presented in Table 7.

Table 8. Applicable rules and obligations in force for tSoPL.

Task, Step	Rule	Obligations in force
CP1,1	CR1, CR2, CR3	Force(tSoPL,CP1,1) = {CR1} U {CR2} U {CR3}
VP1,2	(standard-specific)	Force(tSoPL,VP1,2) = Force(tSoPL,CP1,1) U {standard-specific}
CP2,3	CR4	Force(tSoPL,CP2,3) = Force(tSoPL,VP2,2) U {CR4}
VP2,4	(standard-specific)	Force(tSoPL,VP2,4) = Force(tSoPL,CP2,3) U {standard-specific}
CP3,5	CR5, CR6	Force(tSoPL,CP3,5) = Force(tSoPL,VP2,4) U {CR5} U {CR6}
VP3,6	(standard-specific)	Force(tSoPL,VP3,6) = Force(tSoPL,CP3,5) U {standard-specific}

Standard-specific processes are generated when variability points are deployed with specific tasks. ASPICE SWE.3 process is *tA:<CP1, CP2,* ***BP7****, CP3>*, where VP2 is contributed with BP7. ISO 26262 SUDI process is *tI:<CP1,* ***A1a****, CP2,* ***A2a****, CP3,* ***A4****>*, where VP1 is contributed with A1a, VP2 with A2a and VP3 with A4. Table 9 shows the influence of the obligations in force for these tasks.

In ASPICE SWE.3, the proof obtained for CP1 (Proof 1), and the one obtained in CP2 (Proof 2) are not altered, since the variability point (VP1) do not have any corresponding task in the trace. The rule applied in VP2 (replaced by

Table 9. Applicable rules for the variability points.

tSoPL	tA	tI	Rules	Influence of the obligations in force
VP1		A1a	RI1, RI3, RI5, RI7	For the design of the software units concerning safety (A1a), the software is safety-related (RI1), the design is usually implemented by using a notation that depends on the ASIL and the recommendation level (RI3), the design has described functional behavior (RI5), and the design is usually implemented by using design principles that depend on the ASIL and the recommendation level (RI7).
VP2	BP7		RA7	The software unit design communication (BP7) is done (RA7).
		A2a	RI11	The verification of the software unit design concerning safety (A2a) is done according to methods that depends on the ASIL and the recommendation level (RI11)
VP3		A4	RI10, RI11	The verification of the software unit implementation (A4) is done (RI10) according to methods that depends on the ASIL and the recommendation level (RI11)

BP7) is triggered and fulfilled in BP7, so the proof obtained for CP3 is not altered. In ISO 26262 SUDI, the proofs obtained in CP1 (Proof 1), CP2 (Proof 2) and CP3 (Proof 3) corresponds to non-safety related rules, while the rules that apply to the variability points corresponds to safety-related rules. Hence, the proofs obtained in VP1 and VP2 adds information to the trace, and influence the proofs obtained in CP2 (Proof 2) and CP3 (Proof 3) respectively. In this case, we can conclude that the proof can be partially used.

4.3 Lessons learnt

Our automotive SoPL describes commonality and variability points presented in ASPICE and ISO 26262, as a sequence of ordered tasks distributed in a trace. Tasks have assigned states and obligations in force (normative rules) that produce tasks effects. These effects can influence the tasks' behaviors in the trace, and define whether the designed trace is compliant or not with a given set of rules. Our analysis started with the annotation of the commonality points of the SoPL with the overlapping set of rules, obtained from the comparison of the requirements provided by the two standards. These annotations provide the possibility to derive a common set of proofs of compliance. However, the states and obligations in force for the tasks that contribute to the variability points, once the standard-specific processes are deployed, can affect the proof obtained for the commonality points. Hence, proofs of compliance can be fully reused or may be partially reused, depending on the effects produced by the variability points. Fully reused proofs can be applied to commonality points that are not preceded by a variability point, or that are preceded by a variability point that either remains empty after deployment, or its state after deployment does not produce effects that can be spread out in the trace. Partially reused proofs can be applied to variability points that are contributed with standard-specific tasks which states and obligations in force influence the process proofs obtained. Therefore, a classification of the standard-specific tasks that contribute to the standard-specific processes is required to understand whether the common proofs can be fully/partially reused.

5 Related work

Related work regarding increased efficiency via automation and reuse was already discussed in [4, 5]. Thus, in this paper, we limit our attention to automated compliance checking, the novel layer added to SoPLE. Automated compliance checking is not a new research area, specially in business management. An example of a framework that define proofs of compliance by design is presented in [15], where rules are formalized using logics. However, these frameworks do not contemplate the reuse of proofs of compliance. A more closely related work is presented in [16] where business process are augmented with reusable fragments to ensure process compliance by design. In this approach, rules are formalized with temporal logics. In our approach, we seek to establish compliance proofs for safety compliance, using defeasible logic and deontic logic of violations, instead of temporal logics. Approaches for reusing proofs are also found in other areas. For example, in [17, 18], software verification tasks are benefited by the reuse of chunks of proofs. Our reusable chunks of proofs are instead derived from the comparison between the set of rules and the process reference model provided by a normative system.

6 Conclusions and future work

In this paper, we introduced SoPLE&Logic-basedCM, a novel approach for confident and efficient process compliance based on the combination of safety-oriented process line engineering, defeasible logic, and an approach for compliance by design. We have applied SoPLE&Logic-basedCM to the automotive domain to illustrate its potential in terms of reuse of proofs. More specifically, we have limited our illustration to a specific portion of automotive standards (ASPICE and ISO 26262) and we have shown that sets of compliance-related proofs can be fully/partially reused.

The formalization of the approach presented in this paper is limited to process-related activities, and a general view of the deontic notion *obligation*. For future work, other process elements, e.g., work products will be addressed, as well as a broader range or deontic notions classification (permissions and prohibitions). Further validation of the approach on more complex processes is also required, as well as the exploration of tools that can potentially support the automation of our work, like Regorous [19], a compliance checker, and logic reasoners like SPINdle[1] and Deimos[2], programs that are used to compute the consequence of defeasible logic theories.

Acknowledgments. This work is supported by the EU and VINNOVA via the ECSEL JU project AMASS (No. 692474) [20]. We thank Mustafa Hashmi for his valuable comments on an earlier version of this paper.

[1] `http://spindle.data61.csiro.au/spindle/`

[2] `http://www.ict.griffith.edu.au/arock/defeasible/Defeasible.cgi`

References

1. Rushby, J.: New Challenges in Certification for Aircraft Software. In: 9th ACM International Conference on Embedded Software (EMSOFT). (2011) 211–218
2. Gallina, B., Sljivo, I., Jaradat, O.: Towards a Safety-oriented Process Line for Enabling Reuse in Safety Critical Systems Development and Certification. In: 35th Annual IEEE Software Engineering Workshop (SEW). (2012) 148–157
3. Gallina, B., Kashiyarandi, S., Martin, H., Bramberger, R.: Modeling a Safety- and Automotive-Oriented Process Line to Enable Reuse and Flexible Process Derivation. In: IEEE 38th International Computer Software and Applications Conference Workshops (COMPSACW). (2014) 504–509
4. Gallina, B., Lundqvist, K., Forsberg, K.: THRUST: A Method for Speeding up the Creation of Process-related Deliverables. In: IEEE/AIAA 33rd Digital Avionics Systems Conference (DASC). (2014) 5D4–11
5. Gallina, B.: A Model-Driven Safety Certification Method for Process Compliance. 2nd Int. Workshop on Assurance Cases for Software-intensive Systems (ISSREW) (2014) 204–209
6. Hashmi, M., Governatori, G., Wynn, M.T.: Normative Requirements for Regulatory Compliance: An Abstract Formal Framework. Information Systems Frontiers (2016) 429–455
7. Automotive SPICE: Process Assessment/Reference Model (2015)
8. ISO 26262: Road Vehicles-Functional Safety. International Standard (2011)
9. Lami, G., Falcini, F.: Automotive SPICE Assessments in Safety-Critical Contexts: An Experience Report. In: IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). (2014) 497–502
10. Bleakley, G.: How Rational can Help with Compliance to ISO 26262 & ASPICE. Technical report, IBM Software Group (2014)
11. SPEM 2.0: Software & Systems Process Engineering Meta-model (2008)
12. Eclipse Composer Framework: https://eclipse.org/epf/
13. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Representation Results for Defeasible Logic. ACM Transactions on Computational Logic (2) (2000) 255–287
14. Governatori, G., Rotolo, A., Sartor, G.: Temporalised Normative Positions in Defeasible Logic. In: 10th International Conference on Artificial Intelligence and Law (ICAIL). (2005) 25–34
15. Awad, A., Decker, G., Weske, M.: Efficient Compliance Checking Using BPMN-Q and Temporal Logic. International Conference on Business Process Management (BPM) (2008) 326–341
16. Schumm, D., Turetken, O., Kokash, N., Elgammal, A., Leymann, F., van den Heuvel, W.J.: Business Process Compliance through Reusable Units of Compliant Processes. In: International Conference on Web Engineering (ICWE). (2010) 325–337
17. Reif, W., Stenzel, K.: Reuse of Proofs in software verification. In: International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), Lecture Notes in Computer Science (1993) 284–293
18. Beckert, B., Bormer, T., Klebanov, V.: Reusing Proofs when Program Verification Systems are Modified. Long Beach, California, USA (2005) 41
19. Governatori, G.: The Regorous Approach to Process Compliance. In: IEEE 19th International Enterprise Distributed Object Computing Workshop (EDOCW), IEEE (2015) 33–40

20. AMASS: Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems. http://www.amass-ecsel.eu/