Mälardalen University Licentiate Thesis
No.277

# Facilitating Automated Compliance Checking of Processes against Safety Standards

Julieth Patricia Castellanos Ardila

March 2019


Department of Computer Science and Engineering
Mälardalen University
Västerås, Sweden

Copyright © Julieth Patricia Castellanos Ardila, 2019
ISBN 978-91-7485-422-0
ISSN 1651-9256
Printed by E-print AB, Stockholm, Sweden

# Populärvetenskaplig sammanfattning

Runtomkring oss i våra liv idag har vi olika system som anses vara säkerhetskritiska, system som vid eventuella funktionsfel skulle kunna få katastrofala konsekvenser för oss. Dessa system finns i krockkuddar i bilar, medicinsk utrustning som utför röntgenterapi, och system som styr flygplan, för att nämna några av dem. Tillverkare av de här systemen följer en områdesspecifik säkerhetsstandard, som beskriver vad som är en allmänt accepterad nivå av säkerhet. Vissa specifika områden har säkerhetsstandarder som beskriver de krav för processer som man måste tillämpa när man utvecklar säkerhetskritiska system. För att följa en sådan standard måste företag anpassa sina rutiner och kunna uppvisa övertygande säkerhetsbevisning av processen, från och med de första stegen av produktionen. I synnerhet är planeringen av utvecklingsprocessen en väsentlig del av bevisen som används i bedömningen, i enlighet med de krav som ställs i den specifika standarden. Att konstruera en sådan bevisning kan dock vara tidskrävande och innebära en stor felmarginal eftersom det kräver att processingenjörerna måste kontrollera att hundratals krav baserade på deras specifika processer uppfylls. Med tillgång till lämpliga verktygsstödda metoder skulle processingenjörerna kunna utföra sitt jobb både effektivare och på litligare.

I praktiken är det svårt att genomföra automatiserad kontroll av de processer som krävs ur ett säkerhetskritiskt perspektiv. En orsak är att säkerhetsstandards uttrycker krav i naturligt språk, vilket datorer inte kan förstå. Det finns redan metoder som möjliggör att en dator i viss mån kan tolka skriftligt språk men de innehåller inte de koncept som behövs för att följa existerande standarder. Därför föreslår vi ett nytt tillvägagångssätt som

kombinerar dessa tre egenskaper: 1) processmodelleringsfunktioner för att representera system- och mjukvaruprocesspecifikationer, 2) normativ representation för tolkning av kraven i säkerhetsstandarderna i en adekvat maskinläsbar form samt 3) möjlighet att kontrollera att processen överensstämmer med den branschspecifika standarden. Inom vårt tillvägagå ngssätt har vi definierat metodiska riktlinjer som gör det lättare att följa de krav som beskrivs i ISO 26262, vilken är standarden som behandlar säkerhet inom bilindustrin. Slutligen introducerar vi ett tillvägagångssätt för att systematiskt kunna återanvända de vanligast förekommande kontrollerna. Vår metodik utvärderas i denna licentiatavhandling genom akademiska exempel men försatt arbete inkluderar utvärderingar genom industriella fallstudier.

# Abstract

A system is safety-critical if its malfunctioning could have catastrophic consequences for people, property or the environment, e.g., the failure in a car's braking system could be potentially tragic. To produce such type of systems, special procedures, and strategies, that permit their safer deployment into society, should be used. Therefore, manufacturers of safety-critical systems comply with domain-specific safety standards, which embody the public consensus of acceptably safe. Safety standards also contain a repository of expert knowledge and best practices that can, to some extent, facilitate the safety-critical systems engineering. In some domains, the applicable safety standards establish the accepted procedures that regulate the development processes. For claiming compliance with such standards, companies should adapt their practices and provide convincing justifications regarding the processes used to produce their systems, from the initial steps of the production. In particular, the planning of the development process, in accordance with the prescribed process-related requirements specified in the standard, is an essential piece of evidence for compliance assessment. However, providing such evidence can be time-consuming and prone-to-error since it requires that process engineers check the fulfillment of hundreds of requirements based on their processes specifications. With access to suitable tool-supported methodologies, process engineers would be able to perform their job efficiently and accurately.

Safety standards prescribe requirements in natural language by using notions that are subtly similar to the concepts used to describe laws. In particular, requirements in the standards introduce conditions that are obligatory for claiming compliance. Requirements also define tailoring rules, which are actions that permit to comply with the standard in an alternative way. Unfortunately, current approaches for software verification are not furnished with these notions, which could make their use in compliance

checking difficult. However, existing tool-supported methodologies designed in the legal compliance context, which are also proved in the business domain, could be exploited for defining an adequate automated compliance checking approach that suits the conditions required in the safety-critical context.

The goal of this Licentiate thesis is to propose a novel approach that combines: 1) process modeling capabilities for representing systems and software process specifications, 2) normative representation capabilities for interpreting the requirements of the safety standards in an adequate machine-readable form, and 3) compliance checking capabilities to provide the analysis required to conclude whether the model of a process corresponds to the model with the compliant states proposed by the standard's requirements. Our approach contributes to facilitating compliance checking by providing automatic reasoning from the requirements prescribed by the standards, and the description of the process they regulate. It also contributes to cross-fertilize two communities that were previously isolated, namely safety-critical and legal compliance contexts. Besides, we propose an approach for mastering the interplay between highly-related standards. This approach includes the reuse capabilities provided by SoPLE (Safety-oriented Process Line Engineering), which is a methodological approach aiming at systematizing the reuse of process-related information in the context of safety-critical systems. With the addition of SoPLE, we aim at planting the seeds for the future provision of systematic reuse of compliance proofs. Hitherto, our proposed methodology has been evaluated with academic examples that show the potential benefits of its use.

Para mi familia, con profundo amor.

# Acknowledgments

First and foremost, I would like to express my highest appreciation to my supervisory team, Barbara Gallina and Faiz UL Muram, without whom this thesis would not be possible. Thanks to their guidance and encouragements, I have been inspired and motivated during my research. Special thanks to Guido Governatori, Group Leader of the Software Systems Research Group at CSIRO's Data61[1], for sharing his knowledge and expertise.

I also want to take the opportunity to be grateful with the head of our division, Radu Dobrin, as well as Jenny Hägglund and Carola Ryttersson for facilitating all the MDH routines. My gratitude is also for the people, who are, or have been colleagues at MDH. In particular, I thank Jan Carlson, Antonio Cicchetti, Mirgita Frasheri, Irfan Sljivo, Simin Cai, Filip Markovic, LanAnh Trinh, Muhammad Atif Javed, Soheila Sheikh Bahaei, Gabriel Campeanu, Omar Jaradat, Inmaculada Ayala, Luciana Provenzano, Zulqarnain Haider, Mustafa Hashmi, Husni Khanfar and Robbert Jongeling, for taking their time to answer my countless questions, sharing their knowledge and/or interests. Special thanks to Cristina Seceleanu for reviewing my thesis and giving me valuable comments.

I also want to give special thanks to my mother Mercedes for always believing in me, offering her most caring support and enthusiasm. Finally, and most important, I would like to express my gratitude and love to my husband Ola and my son Gabriel. Their company, patience, and unconditional support and love have strengthened me through this challenging experience.

The work in this Licentiate thesis has been supported by EU and VINNOVA via the ECSEL JU project AMASS (No. 692474) [1].

Julieth Patricia Castellanos Ardila
Västerås, March, 2019

[1] https://www.data61.csiro.au/

# List of Publications

## Papers Included in the Licentiate Thesis[2]

**Paper A:** *Enabling Compliance Checking against Safety Standards from SPEM 2.0 Process Models*, Julieth Patricia Castellanos Ardila, Barbara Gallina and Faiz UL Muram. In Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Prague, Czech Republic, August 2018.

**Paper B:** *Transforming SPEM 2.0-compatible Process Models into Models Checkable for Compliance*, Julieth Patricia Castellanos Ardila, Barbara Gallina and Faiz UL Muram. In Proceedings of the 18th International Software Process Improvement and Capability Determination Conference (SPICE), Thessaloniki, Greece, October 2018.

**Paper C:** *Formal Contract Logic Based Patterns for Facilitating Compliance Checking against ISO 26262*, Julieth Patricia Castellanos Ardila and Barbara Gallina. In Proceedings of the 1st Workshop on Technologies for Regulatory Compliance (TeReCom), Luxembourg, Luxemburg, December 2017.

**Paper D:** *Lessons Learned while Formalizing Functional Safety Standards for Compliance Checking*, Julieth Patricia Castellanos Ardila, Barbara Gallina and Guido Governatori. In Proceedings of the 2nd Workshop on Technologies for Regulatory Compliance (TeReCom), Groningen, The Netherlands, December 2018.

**Paper E:** *Towards Increased Efficiency and Confidence in Process*

[2]The included papers have been reformatted to comply with the thesis layout

*Compliance*, Julieth Patricia Castellanos Ardila and Barbara Gallina. In Proceedings of the 24th European Conference on Software Process Improvement (EuroAsiaSPI), Ostrava, Czech Republic, September 2017.

## Additional Peer-reviewed Publications Related to the Thesis[3]

**Paper 1:** *Towards Efficiently Checking Compliance Against Automotive Security and Safety Standards* Julieth Patricia Castellanos Ardila and Barbara Gallina. In Proceedings of the 7th IEEE International Workshop on Software Certification (WoSoCer), Toulouse, France, October 2017.

**Paper 2:** *Compliance of Agilized (Software) Development Processes with Safety Standards: a Vision*, Barbara Gallina, Faiz UL Muram and Julieth Patricia Castellanos Ardila. In Proceedings of the 4th international workshop on Agile Development of Safety-Critical Software (ASCS), Porto, Portugal, May 2018.

**Paper 3:** *Facilitating Automated Compliance Checking of Processes against Safety Standards*, Julieth Patricia Castellanos Ardila. Accepted research abstract at the Doctoral Symposium hosted by the 8th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLa-DocSymp). Limassol, Cyprus. November, 2018

[3]These papers are not included in this thesis

# Contents

# I

# Thesis

# Chapter 1

# Introduction

Our everyday life is surrounded by systems that are considered safety-critical, i.e., systems whose failures could result in death, injury, loss of property, or environmental harm [2]. Such systems can be situated on, e.g., the car airbags, medical devices that perform radiation therapy, the flight control that guides aircrafts, to mention just some of them. It is predicted that the scope of safety-related areas will be expanded with the implementation of technological advances, such as Artificial Intelligence and Machine Learning techniques [3]. The essential feature characterizing the current and future safety technology is that they are more and more dependent on complicated software [4]. The increasing use of software in safety-systems has been considered closely related to the increasing occurrence of systematic failures, which can lead to accidents [5]. However, the world of *"risky technology"* [6] can be controlled by using proved procedures and strategies that permit these systems to be safer deployed into society. Thus, safety-critical manufacturers rely on safety standards, which not only embody the public consensus of acceptable risk [7] but also contain a repository of expert knowledge and best practices that can, to some extent facilitate the safety-critical system's engineering [8].

The intent of compliance with the requirements provided by a safety standard would assure particular qualities of engineering entities, whose focus is often on demonstrating technical properties of that system [8]. However, there is no real consensus on absolutely essential metrics for assuring the safety of products [9]. Thus, the assurance of processes emerges as an accepted approach for safety qualification [10]. In particular, process features

can derive the evidence regarding issues such as the competency of the personnel producing the system, the suitability and reliability of the methods used, and the qualification of tool support [11]. Therefore, standards, such as DO-178C [12], IEC 61508 [13], ISO 26262 [14], EN 50128 [15], were conceived to focus on the development process used to engineer safety-critical systems. Such standards, also known as *"process assurance-based standards"* [11], prescribe a safety lifecycle which is defined in terms of safety integrity levels (SIL) [16]. The higher the SIL, more stringent are the safety requirements that have to be fulfilled in the processes. A safety lifecycle suggests that instead of safety being included in the system after system development, safety should rather be designed into the system from the beginning [2]. Therefore, the planning of the development process, in accordance with the prescribed safety lifecycle and the adoption of the necessary process-related requirements specified in the standard, is an essential piece of evidence required during compliance assessment [17].

Compliance with process assurance-based safety standards requires complete and convincing justifications regarding the processes used to develop systems [18]. According to a survey carried out in [19], compliance checking reports are beneficial during compliance verification since they facilitate the auditor's job in detecting the defects of the inspected process. Besides, compliance checking reports are useful in identifying compliance errors, assisting the creation of process specifications and preventing non-compliance tasks from being performed [20]. However, their manual production may be time-consuming and challenging since it requires that the process engineer checks hundred of requirements based on the information provided by the specification of the development process used to engineer their systems. An approach for facilitating automated compliance checking of processes against safety standards would provide process engineers the means to perform their job efficiently and accurately.

Process modeling languages and their associated runtime structures are available off the shelf to support process engineer's job. These languages provide the means to generate and manage process models [21]. In particular, SPEM 2.0 (Systems & Software Process Engineering Metamodel) [22] is a metamodel that is considered well suited for modeling development processes, not only for the provision of generic process concepts (e.g., activities, tasks, work products, role, and guidance) but also for the provision of extension mechanisms that allows for modeling and documenting a wide range of development projects [23]. SPEM 2.0 specification is the first step towards formalizing the engineering of processes, using the same kind of

language that is used to model software systems [24], i.e., UML (Unified Modeling Language) [25]. Moreover, SPEM 2.0 includes the improvement of human comprehension of the processes and the facilitation of process tailoring and reuse [26]. Besides, SPEM 2.0 is a good candidate to model processes mandated by safety standards [27], and to some extent, it also supports the creation of compliance tables, i.e., the mapping between standards requirements and process elements [28, 29]. SPEM 2.0 assists the process engineer in representing knowledge about plans with the provision of capability patterns. Capability patterns are generic and reusable process pieces that can be used to assemble complete development processes. However, SPEM 2.0, like many other methodologies for modeling processes, lacks mechanisms for reasoning about plans in a flexible way. Thus, automated compliance checking of processes against safety standards is not currently supported by SPEM 2.0.

In practice, automated compliance checking is difficult to implement since process assurance-based safety standards are usually prescribed in natural language, which computers cannot understand [30]. However, Rule-based systems can provide support for building an environment that contributes to the reasoning capabilities required to automatically checking compliance of process plans. Such a system could help to manage the knowledge about compliance requirements, compare this knowledge with the one provided by the elements in a process, and retrieve information regarding the fulfillment of the requirements. There have been efforts in this matter, such as the ones described in [24] and [26], in which process constraints are expressed using the SWRL (Semantic Web Rule Language) [31], and in [32], which instead, OWL (Web Ontology Language) [33] is used. However, semantic web methods for deriving proofs are not expressive enough for modeling compliance notions [34]. LTL (Linear Temporal Logics) is also used in [35] and [36] for formalizing process properties. However, safety requirements are fundamentally expressed with concepts and terms that are more alike to those used in law, namely, the normative provisions. Normative provisions are the legally binding notions that are anchored to the structure of legislative text [37], which related to rights and obligations, privileges and liabilities [38]. These notions are difficult to implement in languages of the family of Temporal Logics [39].

Thus, a language that offers concepts close to the notions of interest needs to be selected. From the compliance perspective, the normative provisions of importance are the *deontic notions*, which are indicators of states that are legal or illegal [40]. There are three basic deontic notions [41]. *Obligation*, which

is a deontic notion for a state, an act, or a course of action to which a bearer is legally bound, and which, if it is not achieved or performed, results in a violation. *Prohibition*, which is a deontic notion for a state, an act, or a course of action to which a bearer is legally bound, and which, if it is achieved or performed, results in a violation. *Permission*, which is a deontic notion for a state, an act, or a course of action where the bearer has no obligation or prohibition to the contrary. Thus, Deontic Logic, which has traditionally been used to analyze the structure of normative law and normative reasoning, and it has been specially used in computer science in the area of legal applications [42], could also be used to provided the modeling capabilities required to represent safety standards. We also need to be able to provide reasoning about violations of the requirements, which is the failure in fulfilling a requirement within the constraint of the warranting situation [43]. In addition, we need to be able to model imprecise requirements, which is a common situation found in standards requirements [44], that may derive in inconsistencies [45]. Therefore, a suitable approach for formalizing the process-based requirements prescribed by the safety standards must be based on Defeasible Logic [46] and Deontic Logic of Violations [47]. In one hand, Defeasible Logic allows that contrary evidence defeats earlier reasoning, supporting the management of inconsistencies. On the other hand, Deontic Logic of violations allows to encode normative provisions as implications in which the antecedent is read as a property of a state of affairs, and the conclusion has a deontic nature [48]. The language that meets these requirements is Formal Contract Logic (FCL) [47], which is a deontic defeasible reasoning formalism, designed and implemented in the legal compliance context. FCL has also been proved in business process compliance checking.

Regorous [49, 50], which is a compliance checker available on the shelf, supports reasoning with FCL rules. Regorous takes as inputs, the ruleset that contains the requirements formalized in FCL, and the model of the process which should be enriched with compliance effects annotations. Compliance effects annotations are effects that describe the cumulative interactions between process tasks [51]. Compliance effects annotations are derived from the formulas of the logic (FCL rules) and describe the set of permissible states (according to the standards requirements) of the process tasks. With this information, Regorous can automatically check compliance and provide a compliance report, which could help process engineers to understand the reasons why a process does not comply with a specific standard. A compliance report is based on a set of constructive proofs, i.e., for any

conclusion it is possible to have a trace of its derivation [52].

This Licentiate thesis aims at facilitating automated compliance checking of processes against standards in the context of safety-critical systems. For this, we combine and enhance existing tool-supported methodologies. In particular, we propose an automated compliance checking vision [53], consisting of the combination of the three components. The first component is a language to model process that provides process modeling and annotation capabilities. The language selected is SPEM 2.0, which tool-support can be facilitated with the implementation provided by EPF (Eclipse Process Framework) Composer [54] of the SPEM 2.0 reference metamodel called UMA (Unified Method Architecture) [55]. The second component is a rule-based formalism that provides normative representation capabilities, to permit the interpretation of the standards requirements in an adequate machine-readable form, and the generation of the compliance effects required for annotating process models. FCL is the selected rule-based formalism. Finally, the third component is a compliance checker that provides the reasoning capabilities necessary to conclude whether the annotated process model corresponds to a model with compliant states. Regorous provides this component. Within this vision, we have also identified the essential elements required to generate process models checkable for compliance in SPEM 2.0, and the transformations necessary to automatically generate the models that can be processed by Regorous [56]. Since we are aware that the formalization process of safety requirements into FCL rules requires skills which cannot be taken for granted, we have also started an exploration of safety compliance patterns [57] and methodological guidelines [58], which should facilitate the interpretation of safety requirements. These initial attempts are primarily oriented to the automotive functional safety standard called ISO 26262. Finally, we offered the design of a framework for incrementing efficiency in process compliance, called SoPLE&Logic-basedCM [59]. This framework aims at planting the seeds for future provision of systematic reuse of compliance proofs. Hitherto, our proposed methodology has been evaluated with academic examples that show the potential benefits of its use.

## 1.1 Thesis Outline

We organize this thesis in two parts. In the first part, we summarize the research as follows: In Chapter 2, we recall essential background information used throughout this thesis. In Chapter 3, we describe our research

methodology and the thesis research goals. In Chapter 4, we describe the specific research contributions of this thesis. In Chapter 5, we discuss related work. Finally, in Section 6 we present conclusions and future work.

The second part is a collection of the papers included in this thesis. We now present a brief overview of the included papers.

**Paper A:** ***Enabling Compliance Checking against Safety Standards from SPEM 2.0 Process Models***, Julieth Patricia Castellanos Ardila, Barbara Gallina, and Faiz UL Muram. In Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA-2018), Prague, Czech Republic, August 2018.

**Abstract:** Compliance with process-based safety standards may imply the provision of a safety plan and its corresponding compliance justification. However, the provision of this justification is time-consuming since it requires that the process engineer checks the fulfillment of hundred of requirements by taking into account the evidence presented in the process entities. In this paper, we aim at supporting process engineers by introducing our compliance checking vision, which consists of the combination of process modeling capabilities via SPEM 2.0 (Systems & Software Process Engineering Metamodel) reference implementations and compliance checking capabilities via Regorous, a compliance checker, used for business processes compliance checking. Our focus is on the identification and exploitation of the appropriate (minimal set of) SPEM 2.0-like elements, available in the selected reference implementation, which can be used by Regorous for compliance checking. Then, we illustrate our vision by applying it to a small excerpt from ISO 26262. Finally, we draw our conclusions.

**My contribution:** I was the primary driver of the paper under the supervision of the coauthors. My specific contribution included the description of a compliance checking vision supported by preexisting tool-supported methodologies. I also modeled an example from the automotive context to illustrate the vision and wrote the paper. Both co-authors contributed equally with ideas for defining the compliance checking vision as well as reviews and comments for improving the paper.

**Paper B:** ***Transforming SPEM 2.0-compatible Process Models into Models Checkable for Compliance***, Julieth Patricia Castellanos Ardila, Barbara Gallina, and Faiz UL Muram. In Proceedings of the 18th International SPICE

Conference (SPICE-2018), Thessaloniki, Greece, October 2018.

**Abstract:** Manual compliance with process-based standards is time-consuming and prone-to-error. No ready-to-use solution is currently available for increasing efficiency and confidence. In our previous work, we have presented our automated compliance checking vision to support the process engineers work. This vision includes the creation of a process model, given by using a SPEM 2.0 (Systems & Software Process Engineering Metamodel)-reference implementation, to be checked by Regorous, a compliance checker used in the business context. In this paper, we move a step further for the concretization of our vision by defining the transformation, necessary to automatically generate the models required by Regorous. Then, we apply our transformation to a small portion of the design phase recommended in the rail sector. Finally, we discuss our findings, and present conclusions and future work.

**My contribution:** I was the primary driver of the paper under the supervision of the coauthors. My specific contribution included the definition of the transformations required to concretize the automatic compliance checking vision described in Paper A. I also illustrated the transformation by creating a model checkable for compliance from the rail sector, and I wrote the paper. The co-authors contributed equally with reviews and comments for improving the paper.

**Paper C:** ***Formal Contract Logic Based Patterns for Facilitating Compliance Checking against ISO 26262***, Julieth Patricia Castellanos Ardila and Barbara Gallina. In Proceedings of the 1st Workshop on Technologies for Regulatory Compliance (TeReCom-2017), Luxembourg, Luxemburg, December 2017.

**Abstract:** ISO 26262 demands a confirmation review of the safety plan, which includes the compliance checking of planned processes against safety requirements. Formal Contract Logic (FCL), a logic-based language stemming from business compliance, provides means to formalize normative requirements enabling automatic compliance checking. However, formalizing safety requirements in FCL requires skills, which cannot be taken for granted. In this paper, we provide a set of ISO 26262-specific FCL compliance patterns to facilitate rules formalization. First, we identify and define the patterns, based on Dwyer' et al.'s specification patterns style. Then, we instantiate the

patterns to illustrate their applicability. Finally, we sketch conclusions and future work.

**My contribution:** I was the primary driver of the paper under the supervision of the coauthor. My specific contribution included the definition of safety compliance patterns as well as the identification of ISO 26262-related compliance patterns and their instantiation. I also wrote the paper. Both authors contributed equally in discussions and developing the paper contribution. The co-author contributed with reviews and comments for improving the paper.

**Paper D:** ***Lessons Learned while Formalizing Functional Safety Standards for Compliance Checking***, Julieth Patricia Castellanos Ardila, Barbara Gallina, and Guido Governatori. In Proceedings of the 2nd Workshop on Technologies for Regulatory Compliance (TeReCom-2018), Groningen, The Netherlands, December 2018.

**Abstract:** A confirmation review of the safety plan is required during compliance assessment with ISO 26262. Its production could be facilitated by creating a specification of the standard's requirements in FCL (Formal Contract Logic), which is a language that can be used to automatically checking compliance. However, we have learned, via previous experiences, that interpreting ISO 26262 requirements and specifying them in FCL is complex. Thus, we perform a formalization-oriented pre-processing of ISO 26262 to find effective ways to proceed with this task. In this paper, we present the lessons learned from this pre-processing which includes the identification of the essential normative parts to be formalized, the identification of SCP (Safety Compliance Patterns) and its subsequent documentation as templates, and the definition of a methodological guideline to facilitate the formalization of normative clauses. Finally, we illustrate the defined methodology by formalizing ISO 26262 part 3 and discuss our findings.

**My contribution:** The content of this paper is the result of intensive discussions performed to formalize the automotive safety standard ISO 26262 into FCL. In the discussions, all the three authors were involved. I was the primary writer of the paper, and the coauthors contributed with reviews and comments to improve the paper.

**Paper E:** ***Towards Increased Efficiency and Confidence in Process Compliance***, Julieth Patricia Castellanos Ardila and Barbara Gallina. In Proceedings of the 24th European Conference on Software Process Improvement (EuroAsiaSPI-2017), Ostrava, Czech Republic, September 2017.

**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 learned.

**My contribution:** I was the primary driver of the paper under the supervision of the coauthor, who provided the initial idea. My specific contribution included the specification of the approach presented in this paper, which is called SoPLE&Logic-basedCM. I also illustrated the approach by defining an example for the automotive domain, and I wrote the paper. Both co-authors contributed with ideas for defining the approach. The co-author also contributed with reviews and comments for improving the paper.

# Chapter 2

# Background

In this chapter, we introduce essential background and highlight specific definitions which are required by the conducted research. In particular, in Section 2.1, we recall basic information about safety-critical systems. In Section 2.2, we recall information about software process modeling languages. In Section 2.3, we recall essential information regarding process-based compliance management. Finally, in Section 2.4, we recall the basis of property specification patterns.

## 2.1 Safety-Critical Systems

In this section, we present an overview of safety-critical systems. We also present basic information regarding process assurance-based standards. Finally, we recall the functional safety standard ISO 26262 and the cybersecurity guidebook SAE J3061.

### 2.1.1 Safety-Critical Systems Overview

Safety-critical systems are systems that have to perform high-risk functions [60]. An error or failure in these functions could result in death, injury, loss of property, or environmental harm [2]. The scope of safety-critical systems is broad. We can find safety-critical systems in cars, medical devices, aircraft flight control, weapons, and nuclear systems. Besides, future systems are likely to implement new advances in technology such as Artificial Intelligence and Machine Learning techniques [3], defining

a new era of safety technology. For these applications, software is crucial. However, the increasing use of software is closely related to the increasing occurrence of accidents [5]. Therefore, the software development for supporting safety technology needs to follow rigorous development processes to reach adequate levels of functional safety.

**Definition 2.1.1.** *Functional safety is part of the overall safety of a system and generally focuses on electronics and related software [61].*

Functional safety aims at bringing risk down to a tolerable level (it is not possible to eliminate risk completely [62]), by measuring how likely a given event will occur and how severe it would be, namely, how much harm it could cause [61]. The functional safety assurance is guided by the application of functional safety standards.

### 2.1.2 Process Assurance-based Safety Standards

In some safety-critical domains, the applicable safety standards (also called *process assurance-based standards*) prescribe requirements that regulate processes [11]. Specifically, these kind of requirements specify the process to be used for producing the system or for maintaining/changing it rather than specific design features of the system itself [63]. In particular, process assurance-based standards prescribe a safety lifecycle, which requirements are identified with Safety Integrity Levels (SILs) and recommendation levels, e.g., highly recommended or recommended [11, 16].

**Definition 2.1.2.** *A safety lifecycle corresponds to that part of the lifecycle during which activities related to assuring the safety of the system take place [61].*

In a safety lifecycle, which is normally defined in a task flow graph with a fixed structure, other tasks may appear if they are prerequisites for tasks associated with assuring the safety of the system [61].

### 2.1.3 Safety Standard ISO 26262

ISO 26262 [14] is a standard that addresses functional safety in a specific class of road vehicles. ISO 26262 also introduces the notion of Automotive Safety Integrity Level (ASIL), which represents a criterion to specify the item's necessary safety requirements, needed to ensure a certain level of confidence.

**Definition 2.1.3.** *ASIL corresponds with one of four levels to specify the item's or element's necessary requirements of ISO 26262 and safety measures to apply for avoiding an unreasonable residual risk, with D representing the most stringent and A the least stringent level [14].*

Functional safety (see Definition 2.1.1) is influenced by the development lifecycle process. Therefore, ISO 26262 specifies a safety lifecycle (see Definition 2.1.2) that comprises the entirety of phases from concept through decommissioning of the system. ISO 26262 safety lifecycle is based upon a V-model. Planning, coordinating and documenting the *safety activities* of all phases of the safety lifecycle are key management tasks during the implementation of ISO 26262.

**Definition 2.1.4.** *A V-model splits the process lifecycle into two branches: the left-hand branch contains the requirements, analysis, and design tasks, leading to coding at the bottom of the "V." In the right-hand branch, the phases regarding the system integration, testing, and verification [64].*

ISO 26262 is a large document, which is structured in ten parts. The first part is dedicated to the description of the vocabulary that will be used in the other nine parts, which contain the requirements. These nine parts are structured similarly, containing, a foreword, introduction, bibliography, annexes, and clauses. Some of the clauses represent phases of the safety plan. Inside the phases, activities and tasks are also described. Clauses, in general, describe normative requirements. However, the first three clauses, which are similar in all the parts of the standard, only have an informative nature. Clause 1 recalls the general scope of the standard and situate the particular part in this scope. Clause 2, recalls the normative references indispensable for the adoption of the specific part. Clause 3 recalls the reference for terms, definitions, and abbreviated terms. Clause 4 is of particular importance since it describes two compliance conditions required along all the standard. First, the general requirements for compliance which are recalled in Table 2.1. Second, the interpretation of tables which is recalled in Table 2.2.

Table 2.1: General requirements for compliance with ISO 26262 [14]

| |
|---|
| Each requirement of ISO 26262 shall be fulfilled unless:<br>a) tailoring of the safety activities has been planned, or<br>b) an assessed rationale is available that the non-compliance is acceptable. |

Table 2.2: Interpretation of tables recommending methods in ISO 26262 [14]

| |
|---|
| Each method in a table is either a consecutive entry or an alternative entry<br>a) For consecutive entries, all methods shall be applied as recommended in accordance with the ASIL. If methods other than those listed are to be applied, a rationale shall be given that these fulfill the corresponding requirement.<br>b) For alternative entries, an appropriate combination of methods shall be applied in accordance with the ASIL indicated, independent of whether they are listed in the table or not. If methods are listed with different degrees of recommendation for an ASIL, the methods with the higher recommendation should be preferred. A rationale shall be given that the selected combination of methods complies with the corresponding requirement. |

The rest of the clauses states the objectives, general information of the clause, inputs for the clause, requirements and recommendations to be fulfilled, and finally the work products that are to be generated (see Table 2.3)

Table 2.3: ISO 26262:2011 part 3

| |
|---|
| **5 Item definition**<br>**Objectives.** The first objective is to define and describe the...<br>**General.** This clause lists the requirements and recommendations for...<br>**5.3 Inputs of this clause.**<br>*5.3.1. Prerequisites.* None.<br>*5.3.2. Further supporting information.* Any information that already exists concerning the item, ...<br>**5.4 Requirements and recommendations**<br>5.4.1 Functional and non-functional requirements shall be made available, including:<br>a) functional concept<br>...<br>5.4.2 ...<br>...<br>**5.5 Work products**: Item definition resulting from the requirements of 5.4. |

Notes are also included, but they have informative character, i.e., they are expected to help the applicant in understanding and interpreting the requirements. The requirements and recommendations section describes not

only the activities and the tasks required during the engineering process but also specific conditions required for compliance.

### 2.1.4 Cybersecurity Guidebook SAE J3061

SAE J3061 [65] consists of a guidebook that provides a process reference model, high-level guiding principles and information on existing tools, and methods to help organizations identify and assess cybersecurity threats, and design cybersecurity into cyber-physical vehicle systems. The current version of SAE J3061 was release in January 2016[1], but the definition of Automotive Cybersecurity Integrity Level (ACsIL)[2] is still a work in progress. A *cyber-physical vehicle system* is a vehicle embedded control systems where there exists a tight coupling between the computational elements, the physical elements of the system and the environment around the system. *Cybersecurity* is an attribute of cyber-physical systems. A Cybersecure system is a system protected against unauthorized access or attacks. A *threat* is a circumstance or event with the potential to cause harm, where *harm* may be financial, reputation, privacy, safety or operational. Cybersecurity should be built into the design. Therefore an appropriate lifecycle, which addresses threats from concept to decommissioning is required. SAE J3061 proposes a *lifecycle* for handling cybersecurity which is based on ISO 26262's safety lifecycle. Therefore, the cybersecurity process can be integrated to a safety process tailored from ISO 26262 by simply including the cybersecurity activities for each product lifecycle phase, with the corresponding activities for each product lifecycle phase described in the safety process.

## 2.2 Software Process Modeling Languages

In this section, we first recall some aspects regarding software process models. Then, we recall essential information regarding SPEM 2.0, a language specially created for modeling development processes. Then, we recall basic information about EPF Composer, which implements a SPEM 2.0-like specification called UMA metamodel. Finally, we recall the essentials of a methodology for systematizing reuse in process models called SoPLE.

[1] http://standards.sae.org/j3061_201601/
[2] http://standards.sae.org/wip/j3061-1/

### 2.2.1 Software Process Models Overview

A software process model is the representation of a process that leads to the production of a software product [66]. A software process model facilitates not only human understanding and communication, but also process management support and improvement [64].

**Definition 2.2.1.** *A process is a sequence of units of work (phases, activities, tasks or steps) that consume resources such as employee energy, time infrastructure, machines, and money to transform inputs, such as data or material, into value-added outputs, such as products services or information [67].*

Appropriate software process models are often used as a mechanism to convince third parties, such as regulatory bodies, about the quality of the software [64]. Thus, standards discussing the reliability, safety or security of (software) systems usually include a number of requirements on the processes used to develop these systems [64]. Currently, the modeling of software process can be supported by tools. To use these tools, a description of the concepts used to define the objects on the process description level is required. In the context of this thesis, we consider to describe the processes with SPEM 2.0, a process metamodel created specifically for modeling systems and software process (see Section 2.2.2).

**Definition 2.2.2.** *A software process metamodel defines the notation to be used for modeling a software process [64].*

### 2.2.2 SPEM 2.0

SPEM 2.0 (Software & Systems Process Engineering Metamodel) [22] is a standard created with the set of elements necessary to define any software and systems development processes. SPEM 2.0 provides four main capabilities. First, there is a clear separation of reusable method content, which describes the concepts used to build up a knowledge base that is used to built processes, i.e., tasks, roles, work products definition from their usage in processes. Second, there is possible to maintain consistency between different development processes. Third, SPEM 2.0 allows flexible process variability and extension mechanisms, i.e., an element can be extended from a base element in different ways without directly altering any of its existing properties. There are four types of variability management, i.e., an element can be extended from a base element in different ways without directly

altering any of its existing properties. *Contributes* is a variability type that allows extending a base in an additive fashion. *Replaces* is a variability type that allows the substitution of (some) properties of an element in the extended element. *Extends* allows to easily reuse elements from a base plugin by providing a kind of inheritance for the special variability element. *Extends-replace* combines the effects of extends and replace variability into one new variability type. Fourth, it is also possible to generate replaceable and reusable process chunks, by using Capability Patterns. A *Delivery Process* in SPEM 2.0 is a structure used to define a complete and integrated approach for performing a specific project type using the elements specified in the Method Content. A delivery process contains a *breakdown structure*, which allows the nesting of units of work and the definition of predecessors and dependencies among them. Some of the elements described in SPEM 2.0 are recalled in Table 2.4.

Table 2.4: Basic Elements contained in SPEM 2.0 [22].

| **Element** | **Description** | **Icon** |
|---|---|---|
| Task Definition | Describes an assignable unit of work |  |
| Work Product Definition | Describes the expected inputs and outputs of the units of work |  |
| Role Definition | Defines a set of related skills, competencies, and responsibilities of an individual or a set of individuals |  |
| Tool Definition | Describes the capabilities of a any specific tool that supports the associated roles in performing the work defined by a task |  |
| Guidance | It is a special element that provides additional information to describe elements and the entire processes |  |
| Custom Category | It is a special element that is used to group elements in a recursive way |  |
| Delivery Process | It is a special process that describes a complete and integrated approach for performing a specific project type. |  |

### 2.2.3 EPF Composer

EPF (Eclipse Process Framework) Composer [68, 55] is a stand-alone java application that uses the Eclipse Rich Client Platform (RCP) to implement UMA (Unified Method Architecture) Metamodel [55].

**Definition 2.2.3.** *UMA is a metamodel that has been developed to provide the concepts and capabilities of different method and process engineering languages, such as SPEM 2.0 [55].*

EPF Composer is based on open standards and specifications to allow exchange of process models specifications between different tools. In particular, XMI (XML Meta Interchange) is used to store and exchange metadata in XML format. In addition, UML 2.0 Diagram Interchange Specification is used to provide EPF Composer with a proprietary activity diagram, which can be used to generates the execution semantics of a process from a process definition. The functionality of EPF Composer mainly offers two capabilities, as follows.

**Method Authoring:** Functionality used to capture a set of reusable building blocks, i.e., roles, tasks, work products, guidance. Figure 2.1 has an example of a plugin, called *Process Elements*.

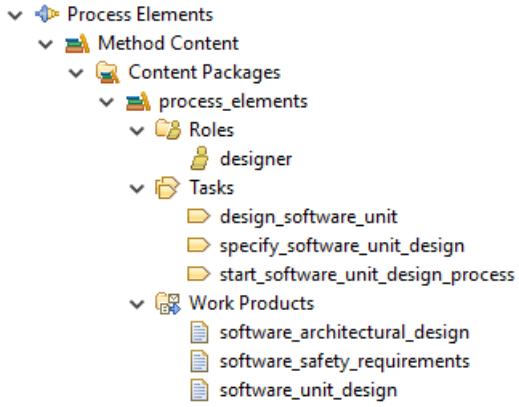


Figure 2.1: An example of Method Content Elements in EPF Composer [56].

As Figure 2.1 depicts, the plugin contains the definition of a role, i.e., *designer*, three tasks, i.e., *design software unit, specify software unit design*

*and start software unit design process* and three work products, i.e., *software architectural design, software safety requirements and software unit design*. These elements are defined by using the subset of elements defined in SPEM 2.0 (see Table 2.4), which are also described by UMA metamodel.

**Process Authoring:** Functionality used to organize reusable process building blocks into processes by defining *Work Breakdown Structures* that describe the order of the process workflow. The workflow can also be represented as an *Activity Diagram* as depicted in Figure 2.2

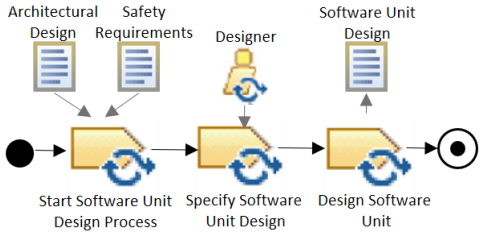


Figure 2.2: An example of an activity diagram of a process. Taken from [56].

EPF composer implements the method plugin package, which defines the capabilities of modularization and extensibility required to manage libraries of method content and processes, allowing the reuse of process and process elements defined in previous projects.

### 2.2.4 Safety-oriented Process Line Engineering

Safety-oriented Process Line Engineering (SoPLE) [27] is a methodological approach that builds on top of an approach called *Product Line Engineering (PLE)* [69]. PLE is a reuse-oriented engineering method that consists of the systematisation of commonalities and variabilities characterising a set of products belonging to the same family/product line. A process line is a set of processes that capture commonalities and controlled variabilities and each of these processes is developed from a common set of core assets (features) in a prescribed way [70].

**Definition 2.2.4.** *Safety-oriented Process Line Engineering (SoPLE) is a methodological approach that permits process engineers to systematise the reuse of process-related information in the context of safety-critical systems [27].*

SoPLE is constituted of two phases. The first phase is aimed at engineering reusable safety process-related commonalities and variabilities. For example, Table 2.5 presents the comparison of activities between the description of the software unit design and implementation phase presented in the functional safety standard ISO 26262 (recalled in Section 2.1.3) and its counterpart presented in the cybersecurity guidebook SAE J3061 (recalled in Section 2.1.4). We called the activities that are common *Commonality Points (CP)*, and the ones that varies *Variability Points (VP)*. In the table we can find four commonality points, which are marked with the unique identifier CP1, CP2, CP3 and CP4. These commonality points can be extended with the variants that belong to the corresponding standard. For example, there is a variability point called VP1a, which contains the information regarding the ISO 26262 requirement (IR), that corresponds to the first activity (IA1), the design concerning safety. In a similar manner, there is a variability point called VP1b, which contains the information regarding the SAE J3061 requirements (JR), that corresponds to the first activity (JA1) the design concerning cybersecurity.

Table 2.5: Activities comparison ISO 26262/SAE J3061.

| ID | IR | JR | Common Name |
|---|---|---|---|
| CP1 | IA1 | JA1 | Unit design |
| VP1a | IA1 | | Design concerning safety |
| VP1b | | JA1 | Design concerning cybersecurity |
| CP2 | IA2 | JA3 | Unit design review |
| VP2a | IA2 | | Design review concerning safety |
| VP2b | | JA3 | Design review concerning cybersecurity |
| CP3 | IA3 | JA2 | Unit implementation |
| VP3a | IA3 | | Unit implementation concerning safety |
| VP1b | | JA2 | Unit implementation concerning cybersecurity |
| CP4 | IA4 | JA4 | Unit implementation review |
| VP4a | IA4 | | Implementation review concerning safety |
| VP4b | | JA4 | Implementation review concerning cybersecurity |

The second phase is aiming at engineering single safety processes via selection and composition of previously engineered reusable process elements. For this, we need to initially describe the skeleton that conforms the software process line. As presented in Figure 2.3, the skeleton is conformed by the Commonality Points, CP1, CP2, CP2 and CP4, which conform its main

part. The variants are added to the commonality points by using the variability type called *contributes*, which is provided by SPEM 2.0 (recalled in Section 2.2.2). Recalling, contributes is used to extend a base in an additive fashion, meaning that we extend the content of the commonality point with the specific elements provided by the variant.

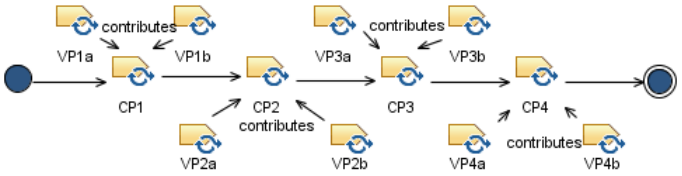


Figure 2.3: A process line skeleton. Taken from [71].

Currently, SoPLE is supported by the integration of EPF Composer (recalled in Section 2.2.3), which is used to model the base process and its related library, and Base Variability Resolution (BVR) Tool [72], which allows users to model the variability, make choices at variation points and bind the conceptual representation of the variable elements. The integration of EPF Composer and BVR Tool is described in more details in [73].

## 2.3 Process-based Compliance Checking

In this section we present an overview of process-based compliance in the safety-critical context. Then, we recall brief information about norms representation, and FCL (Formal Contract Logic), which is a language created to represent legal and normative requirements. Finally, we summarize the main capabilities of Regorous, which supports FCL-based compliance checking..

### 2.3.1 Compliance of Processes in the Safety-Critical Context

In the assurance of safety for safety-critical systems, the focus is often on demonstrating technical properties of that system [18]. However, there are no real consensus on absolutely essential metrics for assuring the safety of products [9]. Therefore, complete, compelling justifications regarding the processes used to develop the systems is also required is some safety-critical domains [18]. Thus, process-based compliance deals with the compliance

management that is required with process assurance-based safety standard's (recalled in Section 2.1.2).

**Definition 2.3.1.** *Process-based compliance can be understood as the consistency between the actual development process and the normative reference model embedded in the standard [74].*

One key consideration when conducting regulatory safety assessment is that regulations are obligations on licensee to fulfill in order to get an authorization [75]. To assure safety, it is common to adopt a highly prescriptive approach, where safety assurance is demonstrated by showing compliance with the requirements set out as a prescribed process in a safety standard, during audits [11].

**Definition 2.3.2.** *An audit is an examination of an implemented process [14], which is typically conducted to ascertain compliance with policies and standards [76].*

A prescriptive regulatory approach requires comprehensive regulatory guides prescribing detailed acceptance criteria [75]. In particular, a key evidence for process-based compliance management is the safety plan, which represents that a plan has been conceived and documented in compliance with the prescribed process-related requirements.

**Definition 2.3.3.** *A safety plan is used to manage and guide the execution of the safety activities of a project including dates, milestones, tasks, deliverables, responsibilities and resources [14].*

A safety plan is a piece of evidence used for safety demonstration; more precisely, a plan identifies the types of evidence that will be used, and how and when this evidence shall be produced [77]. However, the provision of a development safety plan is not sufficient during the compliance assessment process. A compliance justification, which is expected to be scrutinized by a safety auditor, should be produced [17].

**Definition 2.3.4.** *A compliance justification is a document given in term of either a checklist, or an argument, or some proof (e.g. a verification report) which can show/argue/prove that the development plan comply with the requirements [17].*

The safety plan and the compliance justification should be agreed upon at the beginning of the project between the regulatory body and the applicant [77]. A regulatory body is an organization or individual authorized to conduct safety assessments during regulatory processes [78].

### 2.3.2 Norms Representation

Normative systems are systems of norms, namely, the rules of a game or of a language, the laws of a country or the regulations and rules of a social club as forming a system [79].

**Definition 2.3.5.** *Norms are documents written in natural language that arise from different sources (regulations, laws, standards, branch-specific guidelines, internal code of conduct, social and moral rules) [80]. The scope of norms is to regulate the behaviour of their subjects and to define what is legal and what is not [81]. In addition, norms prescribe the conditions under which they are applicable, i.e., the meaning of the terms or concepts where the norms are valid, and the normative provisions they cause when applied. [40].*

**Definition 2.3.6.** *Normative provisions are the legally binding notions that are anchored to the structure of legislative text, laws and regulations [37].*

The intention of states and international organizations which make use of norms, upon recognition of such conduct complying with international standards, is to influence their mutual conduct in a normative, permissive or prohibitive manner [82]. Therefore, as described in [83], the normative provisions required for compliance with such norms are provided by the *Deontic Logic*.

**Definition 2.3.7.** *The system of Deontic logic studies propositions about the obligatory, the permitted and the forbidden [84].*

Deontic logic has traditionally been used to analyze the structure of norms and normative reasoning [42]. As presented in [83], the deontic notions regarding obligations and prohibitions are constraints that limit the behavior of processes. The difference between obligations and prohibitions and other types of constraints is that they can finish in a violation. Instead, a permit is the lack of the obligation to the contrary (and a violation requires an obligation it violates). Thus permissions cannot be violated. A violation could be compensated. A compensation is a set of obligations in force after a violation of an obligation. Since the compensations are obligations themselves, they can be violated, and they can be compensable as well. Thus we need a recursive definition for the notion of compensated obligation. The classification of normative provisions required for compliance is presented in Figure 2.4.

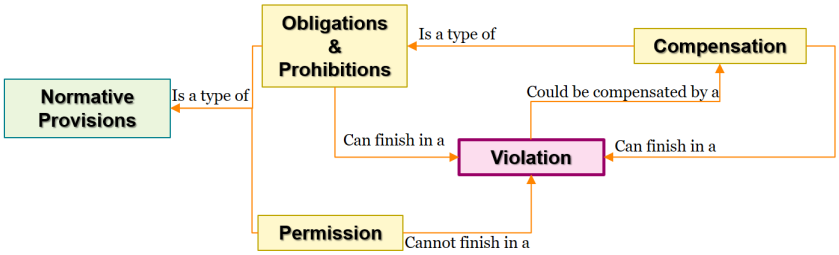


Figure 2.4: Normative provisions classification (Adapted from [40]).

### 2.3.3 Formal Contract Logic

Formal Contract Logic (FCL) [47] is a language based on the system of deontic logic (see Definition 2.3.7) and Defeasible Logic (see Definition 2.3.8), which was designed to represent the norms (see Definition 2.3.5) required for compliance. FCL was initially used for representing business contracts, and then it was used to generate automatic support for compliance checking of business processes.

**Definition 2.3.8.** *Defeasible Logic is a rule-based logic that provides reasoning with incomplete and inconsistent information [85].*

In general, law and legal reasoning largely admits that norms can be represented as conditional rules.

**Definition 2.3.9.** *Rules describe the general association of causes with effects (laws), situations with actions (triggers), premises with conclusions (implications) [86].*

FCL rules has the form presented in formula 2.1.

$$r : a_1, ..., a_n \Rightarrow c, \tag{2.1}$$

where the antecedent $a_1, ..., a_n$ represents the conditions of the applicability of the norm and the precedent $c$ represents the normative effects. FCL is an skeptical non-monotonic logic, meaning that it does not support contradictory conclusions but seeks to resolve conflicts. In case there is sustainable support to conclude both c and $-c$, FCL does not conclude any of them. However, if the support for c has priority for the support of $-c$, then c is concluded. This means that a designer of FCL rules has to identify pairs of incompatible

literals, namely, literals that contradict each other (such as c and $-c$). Once defined the incompatible literals, a *superiority relation* ($>$) among rules is used to determine priorities, as presented in the Formula 2.2.

$$\begin{aligned} r : a_1, ..., a_n &\Rightarrow c, \\ r' : b_1, ..., b_n &\Rightarrow -c, \\ r' &> r \end{aligned} \tag{2.2}$$

Normative effects, also called normative provisions (see Definition 2.3.6) can be of two types. One type describes the environment in which the process will be executed (constitutional rules). The second type triggers deontic effects, such as *Obligations*, which are mandatory situations, *Prohibitions*, which are forbidden situations and *Permissions*, which are allowed situations. In addition, if something is permitted the obligation to the contrary does not hold. A norm can specify that an obligation is in force at a particular time point *n* only, i.e., a norm indicates when an obligation is active. As presented in Figure 2.5, the obligation *O* is in force at n in t. An obligation is considered to remain in force until it is terminated or removed.

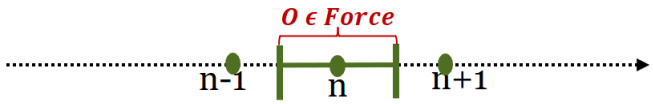


Figure 2.5: Obligation in Force [40].

FCL provides a classification model of normative requirements based on temporal validity of obligations, specifically the concept of *Obligation in Force* (presented in Figure 2.5), and the effects of violations on obligations. If an obligation needs to be obeyed for the whole duration within the interval in which it is in force, it is categorised as a *maintenance obligation* (see Figure 2.6).

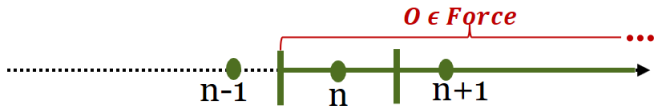


Figure 2.6: Maintenance Obligation [40].

If achieving the content of the obligation at least once is enough to fulfill it, it is called *achievement obligation*. An achievement obligation is *Preemptive*

if it could be fulfilled even before the obligation is in force. An achievement obligations is *non-preemptive* if it only can be fulfilled after it is in force (see Figure 2.7).

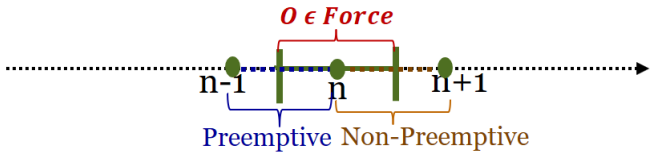


Figure 2.7: Achievement Obligation (Preemptive and Non-Preemptive) [40].

An achievement obligation is *Perdurant* if after being violated, the obligations is still required to be fulfilled (see Figure 2.8). An achievement obligation is *Non-Perdurant* if after being violated, the obligation does not require to be fulfilled.

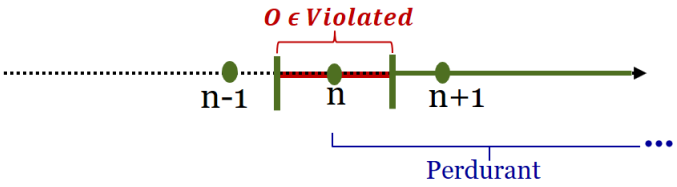


Figure 2.8: Achievement-Perdurant Obligation [40].

FCL normative effects are summarized in Table 2.6. Following, we present examples of statements that could be modeled in FCL by using the types of obligations presented in Table 2.6.

1. **For the party in John's house, you can dress informally.** This is an statement that could be interpreted as a permission since it uses the modal *can*. Therefore, a representation of the statement in FCL, could have as an antecedent the proposition that refers to the fact that there is a party in John's house, which conclusion is that there is a permit for dressing informally (see Formula 2.3).

$$r : PartyInJohn's House \Rightarrow [P]dressInformally \tag{2.3}$$

2. **You must not walk the dog in the avenue.** This statement could be represented as a prohibition since it uses the modal *must not*. In FCL a

Table 2.6: FCL rule notations [47]

| Notation | Description |
|---|---|
| [P]P | P is permitted |
| [OM]P | There is a maintenance obligation for P |
| [OAPP]P | There is an achievement, preemptive, and non-perdurant obligation for P |
| [OANPP]P | There is an achievement, non-preemptive and perdurant obligation for P |
| [OAPNP]P | There is an achievement, preemptive and non-perdurant obligation for P |
| [OANPNP]P | There is an achievement, non-preemptive and non-perdurant obligation for P |

prohibition is formulated as the negation of the content of a maintenance obligation. Therefore, a representation of the statement in FCL could have as an antecedent the proposition that refers to the action *walk the dog*, which conclusion is the prohibition to do it in the Avenue (see Formula 2.4).

$$r : walkTheDog \Rightarrow [OM] - walkTheDogInAvenue \tag{2.4}$$

3. **You have to pay the loan fee every month**. This statement could be represented as an obligation since it uses the modal *have to*. Moreover, this obligation is considered as an *achievement obligation* because doing it once in the month is enough to fulfil the obligation. It is also a *preemptive obligation*, because the fee could even be paid before the deadline. It is also perdurant because the no payment means a violation that continues in the future. Therefore, a representation of the statement in FCL could have as an antecedent the fact that the month starts, which conclusion is the *Obligation, Achievement, Perdurant, Preemtive* of pay the loan (see Formula 2.5).

$$r : startTheMonth \Rightarrow [OAPP]payLoanFee \tag{2.5}$$

4. **When the package arrives it has to be register.** This statement could be represented as an achievement obligation since the package has to be register only once. However, you cannot register it before it arrives.

Thus, the achievement obligation is *non-preemptive*. In addition, if the package is not registered as soon as it arrives, there is a violating of the norm, but it does not mean that the packages should not be registered anyway. Therefore, the obligation is also *perdurant*. A representation of the statement in FCL is presented in Formula 2.6.

$$r : PackageArrives \Rightarrow [OANPP]registerPackage \tag{2.6}$$

5. **You should buy the machine with discount. Discounts are only today.** In this case, buying the machine only once is enough to fulfil the obligation. However, you cannot buy before today, so, it is a non-preemptive achievement obligation. Moreover, if you do not buy today, you cannot buy tomorrow with discount. Thus the obligation is also non-perdurant. A representation of the statement in FCL is presented in Formula 2.7.

$$r : BuyMachine \Rightarrow [OANPNP]buyingWtihDiscount \tag{2.7}$$

### 2.3.4 Regorous

Regorous Process Designer (for simplicity called only Regorous) [49, 50] is a process compliance checker, which is part of the *NICTAs Regorous Toolsuite*[3]. Regorous architecture is depicted in Figure 2.9.

**Definition 2.3.10.** *A process-based requirement is checkable for compliance if there is information in the process that corroborate that the requirement is fulfilled [28].*

Regorous assists process engineers during the design of the processes with mapping normative systems to specific process and process steps to produce constructive proofs.

**Definition 2.3.11.** *Constructive proofs are conclusions from which it is possible to have a trace of its derivation [52].*

Regorous uses the constructive proofs to report the traces, tasks, rules and obligations involved in the non-compliance issues, so that processes can be designed or re-designed in a compliant way. Regorous is the result of the implementation of *the compliance by design approach* proposed in [52].

[3]Regorous Process Designer is available under an evaluation license in `https://digital-legislation.net/`

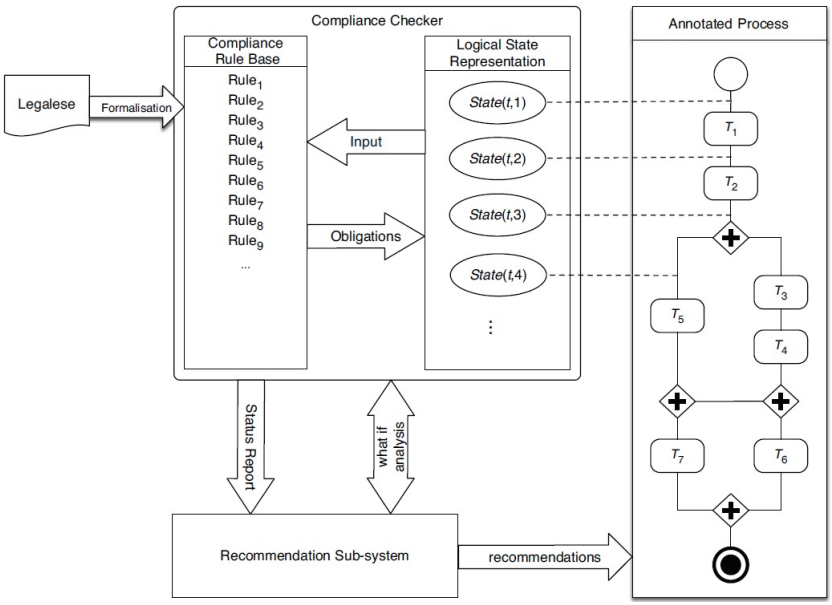


Figure 2.9: Regorous Architecture [50].

**Definition 2.3.12.** *Compliance by design is an approach that provides the capability of capturing compliance requirements through a generic modeling framework, and subsequently facilitate the propagation of these requirements into the process models [52].*

To check whether a process is compliant with a relevant regulation, Regorous requires two elements. The first one is the formal representation of the regulation in Formal Contract Logic (FCL), which was recalled in Section 2.3.3.

**Definition 2.3.13.** *Formal specification is a subcategory of formal methods and is a specification expressed in a language whose vocabulary, syntax and semantics are formally defined [87].*

The second element is a semantically annotated process model. An annotation is extra information that is attached to supplement a concept [88]. Semantic annotations on process elements are literals, that record data, resources and other information that are used by machines to refer, compute

and align information [40]. In particular, the recorded information represent the effects caused by the tasks, which are used by Regorous to perform the analysis of compliance [40].

**Definition 2.3.14.** *Compliance effects are caused by the cumulative interaction between process tasks that are adhered to the standard requirements influences [51].*

In particular, for the n-th element in a trace t, it is used the annotation form *State (t,n)* to semantically annotate the set of facts in the computation to determine which rules fire.

**Definition 2.3.15.** *A trace is a sequence of tasks, in which a process can be executed.*

Consequently, obligations are in force after rules fire. Rules in force are annotated with *Force(t,n+1)*. In addition, the semantic annotation Force (t,n) contains the obligations that are in force but are not terminated in n. After compliance checking with Regorous, a process can be deemed fully or partially compliant [50].

**Definition 2.3.16.** *Let be N a normative system (in this cases a normative system corresponds to the safety standard)*

- *A process P fully complies with N if and only if every trace t complies with the normative system N.*
- *A process P partially complies with N if and only if there is a trace t that complies with the normative system N.*

To check compliance of an annotated process model against a relevant normative system, the procedure executed is the following [50]:

1. Generate an execution trace of the process.
2. Traverse the trace:
   - For each task in the trace, cumulate the effects of the task. Remark: if an effect in the current task conflicts with a previous annotation, update using the effects of the current task.
   - Use the set of cumulated effects to determine which obligations enter into force at the current task. This is done by a call to of FCL reasoner.

- Add the obligations obtained from the previous step to the set of obligations carried over the previous task.
- Determine which obligations have been fulfilled, violated or a pending, and if there are violated obligations, check whether they have been compensated.

3. Repeat for all traces.

An obligation can be terminated if the deadline is reached, the obligation has been fulfilled, or if the obligation has been violated and it is not perdurant. A process is fully compliant if all its traces are compliant (all obligations have been fulfilled, or if violated, they have been compensated). A process is partially compliant if there is at least one trace that is compliant.

## 2.4 Property Specification Patterns

Patterns are abstractions from concrete forms which keeps recurring in specific non-arbitrary context [89].

**Definition 2.4.1.** *Property specification patterns are generalized descriptions of a commonly occurring requirement on the permissible state/event sequences in a finite state model of a system [90].*

A property specification pattern [90] not only describes the essential structure of some aspect of a systems behavior but also provides expressions of this behavior in a range of common formalisms. Each pattern has a scope, which is the extent of the program execution over which the pattern must hold. There are five kinds of scope:

- Global: the entire program execution.
- Before: the execution up to a given state.
- After: the execution after a given state.
- Between: Any part of the execution from one given state to another given state
- After-until: Any part of the execution from one given state to another given state, but the designated part of the execution continues even if the second state does not occur.

# Chapter 3

# Research Summary

In this chapter, we present a summary of the research performed in this thesis. In Section 3.1, we describe the research methodology applied. In Section 3.2, we describe the specific research problem we aim at solve in this thesis.

## 3.1 Research Methodology

The research presented in this thesis is based on the methodology created in [91], which incorporates principles, practices, and procedures required for conducting design science (DS) research in information systems (IS). The DS process includes six tasks: problem identification and motivation, the definition of the objectives for a solution, design and development, demonstration, evaluation, and communication. This methodology has been adapted to the needs of this specific research. The adaptation, which is presented in 3.1, includes the work regarding the subgoals that are derived from an initial main goal (instead of objectives). The definition of subgoals required the definition of a loop that finishes when all subgoals have been tackled. To describe the research process, we use SPEM 2.0 elements, described in Section 2.2.2.

The methodology starts with identifying a relevant real-world problem which has the potential for theoretical contribution. Thus, the first task is called **identify and motivate the problem**. In this task, the researcher should define the overall problem and formulate a motivation about the need to solve the problem. The resources required for this task include the knowledge of *the*

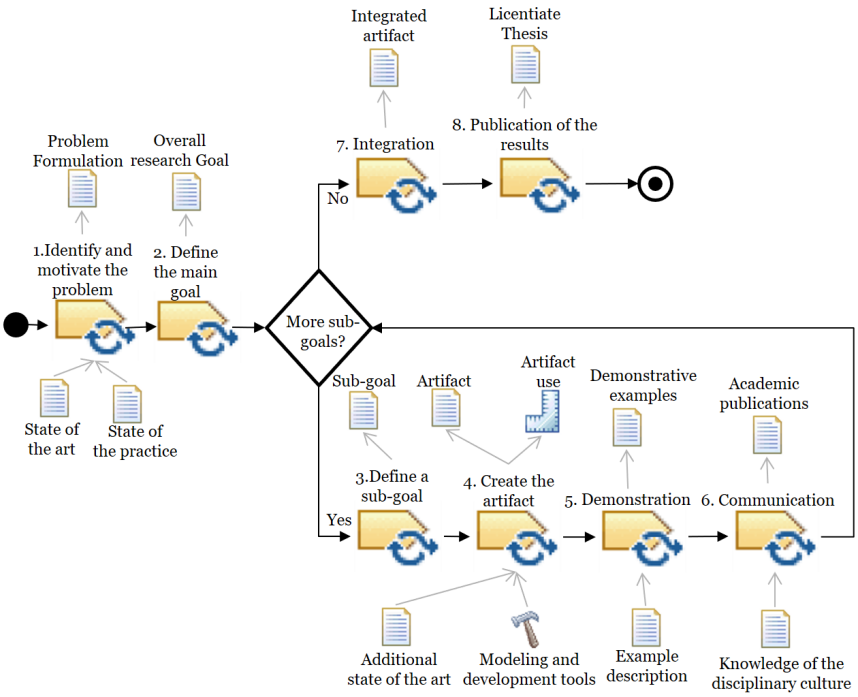


Figure 3.1: Research Methodology.

*state of the art* and the *state of the practice*. The output of this task is the *problem formulation* in which the research problem aimed at being solved is clarified as well as to whom and where it is relevant. The problem formulation should have the potential to be investigated through the research process, understandable and formulated in a logical way. The second task is called **define the main goal**. In this task, the overall research goal that shapes all the research is defined. The goal is inferred from the *problem formulation* and the knowledge of what is possible and feasible. The output of this task is the *overall research goal*. The third task is called **define a subgoal**. This task considers the *problem formulation* and the *overall research goal*, which are the work products of the previous tasks. The output of this task is the formulation of a *subgoal*. The fourth task is called **create the artifact**, in which the type of artifact, i.e., constructs, models, methods, or instantiations,

new properties of technical, social, and/or informational resources, that solves the problem is produced. This task includes determining the artifact's desired functionality, its architecture and its actual development. Resources required for moving from the goal to creation of the artifact include knowledge of the theoretical approaches, i.e., *additional state of the art*, that can be brought to bear in a solution as well as the *tools* required for modeling and development. The output of this task is the produced *artifact* and the guidance required for the *artifact use*. Once the artifact is created, we carry out the fifth task, which is called **demonstration**. In this task, the use of the artifact to solve the specific instance of the problem is demonstrated. The demonstration could involve the use of examples or other appropriate actions. Resources required for the demonstration include adequate knowledge of how to use the artifact, which is given by the *artifact use guidance*, and the actual *artifact*. Besides, the description of the *example* that would be used. The output of this task is the *demonstrative example*. After a subgoal is reached, the sixth task, called **communication** is performed. In the communication tasks, the goal defined and its importance, the artifact, its utility and novelty, the rigor of its design, and its effectiveness is communicated to the research community and practitioners. In the case of this research, the main communication channels are *academic publications*. Thus, the *knowledge of the disciplinary culture* is a basic input for this task. These four steps are repeated for every subgoal that is described in the research. Once we consider that the main problem has been reached and there are not more subgoals to be defined, we carry out the seventh task, which is called **integration**, in which all the solutions that contribute to reaching subgoals are combined in a general solution that supports the procurement of the overall research goal. The output of this task is the *integrated artifact*. The eighth task is called **Publication of the results**, which is a task defined to collect the overall work, in the main output, for the performed research, which in our case is the *the Licentiate thesis*.

## 3.2 Research Problem

In this section, we initially identify the problem to be solved within this thesis. Based on the identified problem, we present a research motivation, in which we describe the reasons why we do our research and what are the specific outcomes we are pursuing. Later, we use the research motivation to describe the overall research goal and the required concrete subgoals.

### 3.2.1 Problem Identification

Manufacturers in the safety-critical context comply with domain-specific safety standards to demonstrate that the deployment of their systems is acceptably safe. In some domains, the applicable safety standards establish the accepted procedures that regulate the development processes used to engineer safety-critical systems. Companies aiming at complying with such standards should adapt their practices, and provide evidence that demonstrates the fulfillment of the requirements. In particular, compliance checking of process plans against safety standards is a mechanism that can be used to demonstrate the adherence of the safety plan (see Definition 2.3.3) to the standard requirements regarding processes. The result of this demonstration, which can take the form of a compliance checking report, can support the provision of the compliance justification (see Definition 2.3.4), which is required during the interaction with the certification bodies in the planning phase. Compliance checking may involve several steps. Initially, a process engineer should know and understand the range of the criteria provided in the standard's requirements, i.e., what are the process entities and their associated properties that can fulfill the requirements. Then, a careful examination of the process description (which could already exist from older projects or being designed for the current project) and the interactions between process elements, should be done to identify whether the elements involved in the planning of the process conforms to the standards prescriptions. Fulfilled requirements can be considered checkable for compliance, in the sense that is presented in Definition 2.3.10. However, the checking mark is not enough. It is expected that a compliance checking report informs not only the fulfillment of the requirements but also how they are fulfilled (what is the evidence collected that demonstrates that the process satisfies the requirements). Thus, information regarding the identified elements is also considered evidence that demonstrates compliance, and should be documented within the checking mark, to produce a proper compliance checking report. The process engineer can use the compliance checking report to identify areas in the process that are uncompliant and, if needed, improve the process. The improvement can be done by modifying or deleting existing process elements, or by adding new process elements, according to the compliance checking report recommendations. However, improving some process elements may affect the behavior of others, resulting in new uncompliant situations. Therefore, a complete re-checking may be required. Once fully compliance is reached, the compliance checking report itself can be used as the evidence required for the

certification bodies to justify process compliance. However, manually performing all the steps described before can be time-consuming and prone-to-error since standards are large documents with hundreds of process-related requirements. Besides, a company can have many safety-critical-related processes to be checked.

### 3.2.2 Research Motivation

As presented in Section 3.2.1, compliance checking requires that the process management area in the organizations, headed by the process engineer, performs several, time-consuming and repetitive steps with a lot of focus. This kind of activities are typically considered tedious and error-prone, in which the required focus may be lost leading to mistakes. Mistakes in compliance reports may endanger audits (see Definition 2.3.2), leading to delays in the production and thus, economically lost. It is clear that the process engineer requires support for performing compliance checking. Thus, in this thesis, we aim at providing an approach for automated compliance checking of processes against safety standards to facilitate the production of the compliance checking report required during planning phases. More specifically, this thesis aims at identifying the aspects that are required to provide automatic support for reasoning from standard's requirements and the description of the process they regulate. Once these aspects are identified, an approach for supporting the process engineers in their compliance checking tasks is proposed. Moreover, patterns and methodological guidelines that facilitate the use of the proposed approach are offered, to guarantee its understanding and its use. Finally, the reuse of compliance checking results between highly-related safety-oriented process plans that could improve efficiency in the production of compliance checking reports is studied.

### 3.2.3 Research Goals

Given the problem identified in Section 3.2.1, and the research motivation presented in Section 3.2.2, we present our overall research goal and the concrete subgoals in this section. Our overall research goal is formulated as follows:

**Overall Goal:** ***Provide an approach that facilitates compliance checking of the processes used to engineer safety-critical systems against the standards mandated (or recommended) in the safety-critical context.***

In order to address the overall research goal, we define concrete subgoals that address specific challenges. The subgoals are described as follows:

**Subgoal 1: Elicit the requirements to be met to support the automation of process-based compliance checking in the safety-critical context.**

The challenge with this goal consists of identifying the specific conditions required to provide an appropriate methodology that permits automated compliance checking in the safety-critical context.

**Subgoal 2: Identify methodologies that contribute to automate the compliance checking of planned process against process-based safety standards.**

The challenge with this goal consists of discovering existing approaches that meet the specific conditions required to automate compliance checking. Besides, this goal aims at determining how existing approaches can be appropriately combined to offer the required support for compliance checking of process plans against the safety standards.

**Subgoal 3: Facilitate the creation of formal specifications of the process-based requirements prescribed by safety standards.**

The challenge with this goal consists of understanding how the standards are structured and how these structures can be managed to facilitate the formalization required for using approaches for compliance checking.

**Subgoal 4: Analyse existing methodological approaches that could be used for increasing efficiency in process compliance.**

The challenge with this goal consists of understanding the way in which systematic reuse of proofs of compliance can be performed between highly related-processes in order to improve the efficiency of the productions of compliance checking reports.

# Chapter 4

# Thesis Contributions

In this chapter, we present a brief description of the five technical contributions provided by this thesis. In particular, in Section 4.1, we describe the first contribution, which is called *Conditions for automatically checking compliance in the safety-critical context*. In Section 4.2, we describe the second contribution, which is called *Automated Compliance Checking Vision*. In Section 4.3, we describe the third contribution, which is called *ISO 26262-related Safety Compliance Patterns*. In Section 4.4, we describe the fourth contribution, which is called *Methodological Guidelines for Formalizing ISO 26262*. Finally, In Section 4.5, we describe the fifth contribution called *Logic-based Framework for Enabling Reuse of Compliance Proofs*.

## 4.1 Conditions for Automatically Checking Compliance in the Safety-Critical Context

As recalled in Section 2.1.1, to support the production of safety technology, rigorous development processes prescribed by safety standards should be followed. Prescribed process usually include the definition of the activities regarding safety specifically compiled in a *safety lifecycle* (see Definition 2.1.2). To comply with these requirements, it is common to adopt a highly prescriptive regulatory approach that have to be achieved from the initial steps of the development process. Therefore, the *safety plan* (see Definition 2.3.3) should get an initial approval from regulatory bodies and should be used to manage the execution of safety activities during the

engineering of safety-critical systems. To be able to get the approval, a *compliance justification* (see Definition 2.3.4) should be available during the *safety audit* (see Definition 2.3.2). As the definition recalls, a compliance justification contains information that shows/argues/prove that the development plans fulfill the requirements imposed by the standards. The production of the compliance justification requires that the process engineer checks whether the standards requirements are fulfilled via the model of the process plan. However, the manual production of the compliance checking report may be time-consuming since the standards are constituted by hundreds of requirements. Besides, a company may have planned multiple processes. Therefore, automatizing the compliance checking is considered useful to facilitate the procurement of the compliance justification report. Since we are conceiving compliance at the initial stages of the development process, i.e., the planning phase, we have selected the *compliance by design approach* (see Definition 2.3.12). As the definition recalls, for performing compliance by design we need to model two components: the model that describes the *norms* (see Definition 2.3.5), which will be propagated into the model that describes the *process* (see Definition 2.2.1). This propagation is possible by a mechanism called *compliance effects* (see Definition 2.3.14) annotation. This mechanism consists of recording the information that represents the effects caused by the tasks that are aligned with the requirements influences. Giving this appreciation, we could assume that the compliance effects unlike other effects caused by the process tasks, corresponds to the permissible states (according to the standard's requirements) of the process tasks. The permissible states trigger other (possible) permissible states that describe a model with compliant states. When permissible states are possible to be annotated into a process model, the requirements that represent are considered to be *checkable for compliance* (see Definition 2.3.10), because they have the possibility to occur in the process model. Thus, we can assign a boolean function to the requirements that is true when it occurs and false otherwise. Based on the previous reasoning, we have defined the conditions for automatically checking compliance in the safety-critical context as follows (see the introduced Definition 4.1.1):

**Definition 4.1.1.** *Automatic compliance checking of a safety plan involves the annotation of the process elements defined to manage and guide the execution of safety activities with compliance effects, which correspond to the permissible states provided by the standards requirements, to describe a model with standard-compliant states.*

The conditions for automatically checking compliance in the safety-critical context, described in Definition 4.1.1, require the association of three components as depicted in Figure 4.1. The first component is a *language to model processes* that provides not only the *process modeling capabilities* but also the *annotation capabilities* that allows the enrichment of process tasks with compliance effects. The second component is a *language to encode requirements* that provides *normative representation capabilities*, to permit the interpretation of the standard's requirements in an adequate machine readable form, and the generation of the permissible states that will be used as the compliance effects required for the annotation process. Finally, the third component is a *compliance checker* that provides the *reasoning capabilities* necessary to conclude whether the annotated process model corresponds to a model with compliant states.

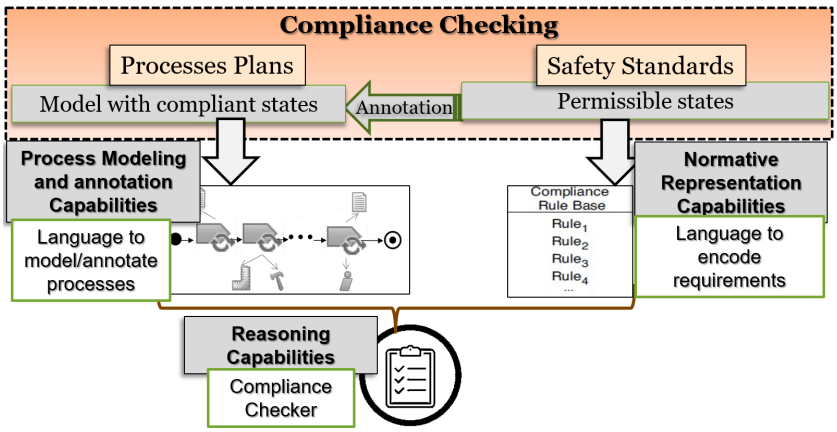


Figure 4.1: Required Components for Automating Compliance Checking.

This contribution addresses the first subgoal presented in Chapter 3, specifically in Section 3.2.3 that says: *Elicit the requirements to be met to support the automation of process-based compliance checking in the safety-critical context*, which challenges is related to the understanding of the specific conditions that define the required components for automatically checking compliance in the safety critical context. This contribution has been particularly discussed in papers A and E.

## 4.2 Automated Compliance Checking Vision

In Section 4.1, we have described the conditions for automatically checking compliance (see Definition 4.1.1). We also described the characteristics of the required components to fulfil these conditions. In this part of the research, we describe the identified tool-supported methodological approaches that provide the capabilities required by the mentioned components, and describe our automated compliance checking vision.

### Process Modeling and Annotation Capabilities

A Process Engineer deals with the representation of the process (more or less formal), which can be composed of hundreds of entities. Software process metamodels (see Definition 2.2.2) have been developed to provide process engineers the possibility to model and have more control over their processes. In particular, SPEM 2.0 (Systems & Software Process Engineering Metamodel, which was recalled in Section 2.2.2, is considered well suited for modeling software and systems processes, not only for the provision of generic process concepts (e.g., tasks, work products and roles) but also for the extension mechanisms for modeling and documenting a wide range of processes. Besides, SPEM 2.0 is a good candidate to model processes mandated by safety standards, as demonstrated in the description of process lines for the safety-critical context (see Section 2.2.4). To some extent, SPEM 2.0 also supports the creation of compliance tables, and is the first step towards formalizing the engineering of processes, as presented in our introductory part (see Chapter 1). Moreover, SPEM 2.0 process elements can be documented with special types of guidance (See Table 2.4). This capability can be enforced and used to add the extra information required for checking compliance, i.e., compliance effects (see Definition 2.3.14). Furthermore, SPEM 2.0 is tool-supported. In particular, the provision of SPEM 2.0-like process model is feasible with EPF Composer. As recalled in Section 2.2.3, EPF Composer implements UMA metamodel (see Definition 2.2.3), which has a good coverage of SPEM-2.0 elements. EPF Composer also provides the option for modeling activity diagrams of the process models, which can be used to supply the creation of the structure of the process, as well as a graphical view of the flow of the process. In addition, EPF Composer facilitates process tailoring and reuse. Specifically, EPF Composer allows the representing of the knowledge about plans, with the provision of capability patterns, which are generic and reusable process pieces which can be used to

assemble complete development processes. The annotations and the process flow can be used to describe the process checkable for compliance required to automatically checking compliance.

## Normative Representation Capabilities

The requirements provided by the safety standards, should be encoded in formal notations that can express not only their contradictory, incomplete and inconsistent nature, but also their *normative provisions* (see Definition 2.3.6). From the compliance perspective, as recalled in Section 2.3.2, the normative provisions of importance are those related to the obligations, prohibitions and permissions. Therefore, a promising approach for formalizing requirements could be based on defeasible logic (see Definition 2.3.8), in which contrary evidence defeats earlier reasoning, supporting the management of inconsistencies. Also, normative provisions should be encoded as implications in which the antecedent is read as a property of a state of affairs, and the conclusion has a deontic nature as provided by the Deontic Logic (see Definition 2.3.7). Moreover, we need to be able to provide reasoning about violations of the requirements, which is the failure in fulfilling a requirement. Thus, we argue that deontic defeasible reasoning formalisms, such as Formal Contract Logic (FCL) (recalled in Section 2.3.3), can be used to represent the requirements prescribed by the safety standards, in an adequate machine-readable form.

## Reasoning Capabilities

As recalled in Section 2.3.4, Regorous is a tool-supported methodology for compliance checking designed and implemented in the legal compliance context, and also proved in the business context. Regorous is of particular interest since it implements *compliance by design* (see Definition 2.3.12), which, as described in Section 4.1, is a methodology that is adequate for compliance checking of process plans, which are mandatory pieces of evidence required by the majority of standards. In the process of compliance checking, Regorous provides *constructive proofs* (see Definition 2.3.11) which helps process engineers to understand why a process does not comply. Regorous methodology is process modeling language agnostic, i.e., only requires a process description that contains compliance effects (see Definition 2.3.14). Therefore, we argue that Regorous is the appropriate tool

for generating automatic support to reason from standards requirements and the description of the process they regulate.

## Our Compliance Checking Vision

Our compliance checking vision (see Figure 4.2), which has the potential to automatize the compliance checking in the safety-critical context, considers the combination of the tool-supported methodological approaches that provides the required capabilities previously described. In particular, the vision includes the provision of a compliance rule base in FCL, which provides the normative representation capabilities required for annotating the process models and check compliance. Moreover, we include EPF Composer, which provides the process modeling and annotation capabilities, as well as a basic platform for FCL rule edition. Finally, we include Regorous, which provides reasoning capabilities with FCL rules required for compliance checking. In the vision is also included the two main roles required, i.e., a Process engineer, which should support the interpretation of the standards requirements, model, annotate the process, and analyze the compliance report, and the FCL expert, which should interpret standard's requirements and formalize them in FCL.

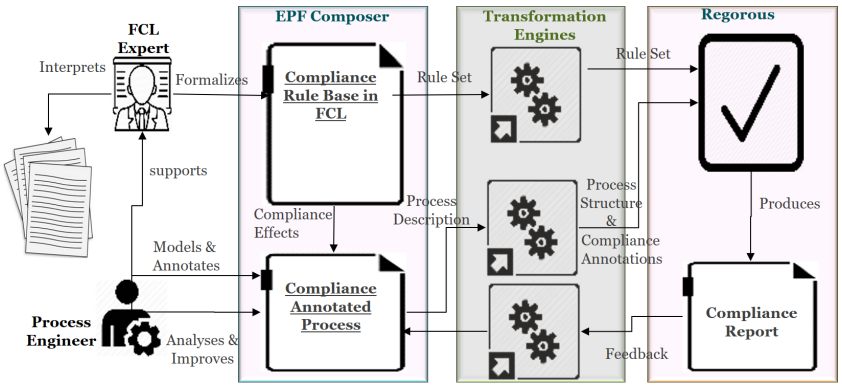


Figure 4.2: Automated Compliance Checking Vision [53].

The tool-support previously described is conceived in three steps. First, we consider the definition of the mechanisms to annotate process models, to support the process engineers. Then, we consider the definition of the

facilities required for editing FCL rules to produce the rule set supporting FCL experts. Finally, we created the mechanisms to ensure EPF Composer and Regorous compatibility. These mechanisms consist of a series of transformations that take the models produced by EPF Composer and convert them into the models that Regorous can process. During the production of the transformations, we realize that the tool-support provided by Regorous is not process modeling language agnostic, as the Regorous methodology. In particular, Regorous depends on a specific process modeling language, i.e., BPMN (Business Process Model and Notation) to produce the compliance report. We need to detach the compliance report from the modeling language to be able to backpropagate the compliance results into EPF composer. The result of this discovering is that Regorous has entered a refactoring period, from which we expect to concretize our automated compliance checking vision in the future.

The automated compliance checking vision and the identification of the modeling capabilities in EPF Composer were presented in the paper A, while the transformation of models from EPF Composer to Regorous was presented in paper B. This contribution addresses the second subgoal presented in Chapter 3, specifically in Section 3.2.3 that says: *Identify methodologies that contribute to automate the compliance checking of planned process against process-based safety standards*.

## 4.3 ISO 26262-related Compliance Patterns Definition

Formal Contract Logic (FCL) (recalled in Section 2.3.3), is a logic-based language stemming from business compliance which provides the normative representation capabilities necessary to formalize the requirements provided by the safety standards and supports compliance checking in the safety-critical context, as discussed in Section 4.2. However, formalizing safety requirements in FCL is not an easy task, since it requires skills, which cannot be taken for granted. Patterns could represent a solution. As presented in Section 2.4, the idea of design patterns is extended into *property specification patterns* (see Definition 2.4.1). Property specification patterns (as recalled in Definition 2.4.1) were created to ease the formalization of systems requirements for finite state system model verification. We follow property specification patterns style, to draw a general definition of safety compliance pattern as presented in introduced Definition 4.3.1.

**Definition 4.3.1.** *Safety compliance patterns describe commonly occurring normative safety requirements on the permissible state sequence of a finite state model of a process.*

With this definition, we can develop a mapping between specification patterns and safety compliance patterns, as presented in the Observation 4.3.1.

**Observation 4.3.1.** *The mapping between specification patterns and safety compliances patterns permits that the state of the obligation imposed to an element in the process is considered in a similar way as the presence of a state in a system, and that the scope corresponds to the interval in a process when the obligations formulated by the pattern are in force.*

For the identification of ISO 26262-related compliance patterns, we have delineated five methodological steps, which are depicted in Figure 4.3.

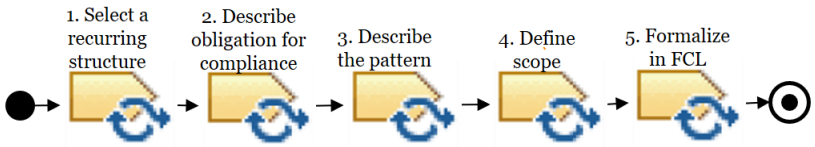


Figure 4.3: Methodological steps for identifying safety compliance patterns.

The first step consists of the selection of a recurring structure in the standard since, as recalled in Section 2.1.3, safety requirements in ISO 26262 have implicit and explicit structures. The second step is the description of the obligation for compliance, namely, the reasons why the structure is required for safety compliance. The third step is the pattern description, based on similar (or a combination of) behaviors of the property specification patterns described in Section 2.4. This description is contextualized to safety compliance, based on the mapping presented in Observation 4.3.1. In this step, we also assign a name for the safety compliance pattern, which reflects the related obligation for compliance. The fourth step is the definition of the scope of the pattern, which we also based on the scopes defined to the property specification patterns. The fifth step is the formalization in FCL. To formalize the pattern, the scope defined for the pattern (see Section 2.4) requires being mapped into the rule notations provided by FCL (see Section 2.3.3). Therefore, a *global scope*, which represents the entire process model execution, can be mapped to *maintenance obligation*, which represents that an obligation has to be obeyed during all instants of the process interval.

A *before scope*, which includes the execution of the process model up to a given state, can be mapped to the concept of *preemptive obligation*, which represents that an obligation could be fulfilled even before it is in force. An *after scope*, which includes the execution of the process model until a given state, can be mapped to the concept of *non-preemptive obligation*, which represents that an obligation cannot be fulfilled until it is in force. It should be noted that, in safety compliance, it is possible to define exceptions for the rules. Therefore, if the obligation admits an exception, the part of the pattern that corresponds to the exception is described as a permission, since, as recalled in Section 2.3.3, *if something is permitted the obligation to the contrary does not hold.* The obligation, to which the exception applies, is modeled as *non-perdurant*, since the permission is not a violation of the obligation, and therefore the obligation does not persist after the permission is granted. In this case, the obligation and the permission have contradictory conclusions, but the permission is superior since it represent an exception.

The definition of safety compliance pattern as well as the identification, definition and instantiation of an initial set of ISO 26262-specific FCL compliance patterns is presented in Paper C. This contribution addresses the third subgoal presented in Chapter 3, specifically in Section 3.2.3 that says: *Facilitate the creation of formal specifications of the process-based requirements prescribed by safety standards.*

## 4.4 Methodological Guidelines for Formalizing ISO 26262

Our initial efforts to formalize ISO 26262 into FCL, as recalled in Section 4.3, gave us some insights about the complexity that this task entails. As recalled in Section 2.1.3, ISO 26262 is structured in a specific way, i.e., it is composed of parts, which are subdivided into very structured clauses. We encounter that not all the structures are required to be formalized. We also find that some structures are repetitive, and can be represented as **Safety Compliance Patterns** (see Definition 4.3.1). Therefore, to be able to formalize effectively, we consider that doing a pre-processing of ISO 26262 was necessary. The pre-processing, which is depicted in Figure 4.4, includes three tasks. Initially, we identify the essential normative structures, namely those structures that define the safety process to be adopted for developing the cars safety-critical systems. Then, we identified the repetitive structures of the standard that can be considered Safety Compliance Patterns. With the identified Safety

Compliance Patterns, we create templates to consolidate a reusable knowledge base for future formalization jobs. Finally, the knowledge gathered in the pre-processing is used to define a methodological guideline for facilitating the formalization of normative clauses in ISO 26262.


Figure 4.4: Pre-processing.

From the pre-processing tasks described above, we got an understanding of what to formalize and how we could proceed in the formalization process. The parts to be formalized are those that determine the safety lifecycle, namely, those clauses that start from Clause 5 in every part of the standard ISO 26262. To formalize these clauses, we have described a methodological guideline, which we depict in Figure 4.5.

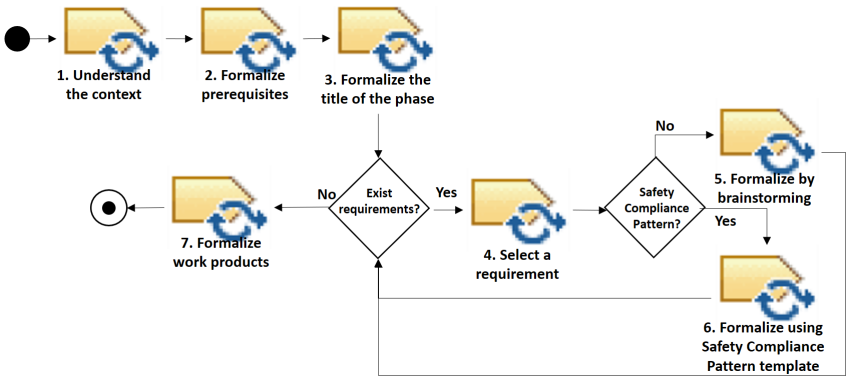


Figure 4.5: Methodological Guidelines.

As Figure 4.5 depicts, initially, the given context of the phase, which is described in the safety standard, should be understood. For this, the reading and the analysis of the objectives and the main general information of the clause to be formalized are required. Then, the formalization process initiates with the prerequisites and followed by the title. These two formalizations can be done by following the Safety Compliance Patterns called Prerequisites and

Initiation of a phase (see those patterns in paper D). After, one requirement is selected from the list of Requirements and Recommendations. We suggest that the requirements are selected in the order they are presented and that the rules are named following the requirement numeration to ensure consistency and traceability. For instance, if a textual requirement is marked with the label 5.1, the corresponding rule should be called r.5.1. During the formalization of the requirements, Safety Compliance Patterns templates could be used to facilitate this task. However, if there are no templates, brainstorming sessions are required. The brainstorming session can be carried out in different ways, but the most relevant is that the group takes one requirement at the time, discuss its importance in the compliance process (e.g., related requirements or permits for tailoring), divide the requirement into smaller sentences that have only one idea, and discuss every sentence. If the requirement has to be divided into several rules, the name of the rule has to be named with the number that accompanies the requirement plus a letter, i.e., r.5.1.a, r.5.1.b. Finally, when all requirements available in Requirements and Recommendations are covered, the work products can be formalized by using the Safety Compliance Pattern template called Work Product. The generated rule set should be verified to avoid inconsistencies and typos in the rules since Regorous do not recognize incorrectly formed rule sets. The preprocessing tasks as well as the methodological guidelines derived from the preprocessing, are presented in Paper D. This contribution addresses the third subgoal presented in Chapter 3, specifically in Section 3.2.3 that says: *Facilitate the creation of formal specifications of the process-based requirements prescribed by safety standards.*.

## 4.5 Logic-based Framework for Enabling Reuse of Compliance Proofs

The engineering of (software) systems (as recalled in Section 2.1.2) 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 that occupies an enormous amount of time from the process engineers. Since the ultimate goal of our work is to free time for activities, such as verification, and validation of the systems, we believe that process compliance would 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), recalled in Section 2.2.4, permits process engineers to systematize the reuse of process-related information. However, to argue about or prove compliance, SoPLE is not enough. Therefore, we intend to provide a 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 legal compliance and business process-related community. Specifically, we use defeasible logic (see Definition 2.3.8), a rule-based approach for efficient reasoning with incomplete and inconsistent information, a typical scenario in normative systems. Our approach, which is called *SoPLE&Logic-basedCM*, is depicted in Figure 11.1.

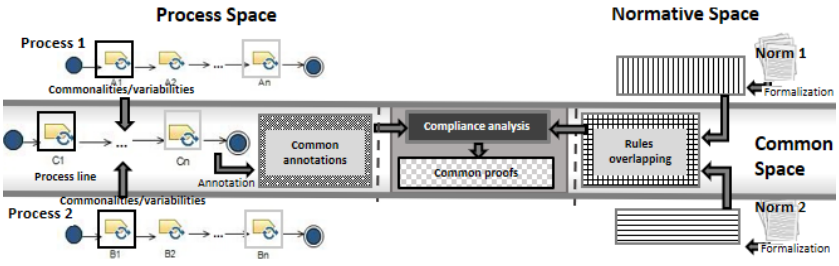


Figure 4.6: SoPLE&Logic-basedCM Framework.

SoPLE&Logic-basedCM is the result of the combination of SoPLE (recalled in Section 2.2.4), the compliance by design methodology (see Definition 2.3.12) and defeasible logic (see Definition 2.3.8). As Figure depicts, a process engineer is expected to:

1. Model a SoPL, which includes manually modeling the skeleton of the process sequence (see Figure 2.3).

2. Formalize the standards rules, select the set of rules that overlap, and analyze the compliance of the SoPL commonalities with the overlapping rules.

3. Analyze the effects of the tasks that contribute to the variabilities in the in the standard-specific process.

SoPLE&Logic-basedCM was presented in the paper E, and it contributes to addressing the fourth subgoal presented in Chapter 3, specifically in Section 3.2.3 that says: *Analyse existing methodological approaches that could be used for increasing efficiency in process compliance*.

# Chapter 5

# Related Work

In this chapter, we extend the related work already done in the papers. In particular, we perform an extended state of the art on approaches for compliance checking in Section 5.1, approaches for formal specifications of requirements in Section 5.2, and approaches for reuse of proofs in 5.3.

## 5.1 Approaches for Compliance Checking

Compliance to normative frameworks (such the ones proposed by the process-based standards) is a matter of decision-making, in which managers should consider the selection of a strategy that fits within the boundaries allowed by the norm [38]. Supporting that decision-making process requires the provision of the right level of abstraction of those boundaries in a way that the conditions for compliance can be evaluated. It may be suitable, specially for small companies, to have process-based assessments supported by tools, which can provide, e.g., periodic checks [92]. It may be also convenient for companies to have methodological tool-support when they adopt the usage of agile development methods/practices in the context of regulatory domains [17]. Tool-supported approaches are developed to follow these premises. In [93], the authors propose a semi-automatic compliance process to support the definition of a formal specification of software requirements. In [94], the authors present an approach to reason about the correctness of the process structure, which is based on the combination of CTN (Composition Tree Notations) [30] and Description Logic (DL). An extension of [94], which

includes ontological modeling of ISO/IEC 15504 [95] for defining capability levels, is presented in [96]. A similar approach to [94] and [96] is used in [97] and [98] to enable the definition process capability levels, according to ISO/IEC 15504 and CMMI (Capability Maturity Model Integration) v1.3 [99]. In [100], the author presents a formalization of data usage policies in a fragment of OWL (Web Ontology Language) [33], which is a DL-based modeling language. All the previous approaches, i.e., [93], [94], [96], [97], [98] and [100], consider the use of DL to reason about the compliance of the process structure. One of the problems of DL, as presented in [101], is its relative expressiveness, which makes more difficult the modeling of certain concepts. Besides, the previous approaches only consider the analysis of the process structure. Instead, our approach considers the use of a mechanism that permits the recording of the information that represents the effects caused by the tasks, which is called compliance effects annotation (see Definition 2.3.14). This mechanism is not only useful for checking the compliance of a process structure, but also its behavior. Other difference, we have included in our approach, is the use of an SPEM 2.0-compatible software process modelling language, called UMA metamodel (see Definition 2.2.3), which is a tool-supported language that provides the modeling, annotation and reuse capabilities that suits our purposes.

SPEM 2.0 community is interested in addressing checking and monitoring capabilities. In [36], the authors propose a framework that uses LTL (Linear Temporal Logics) on top of SPEM 2.0 for adding the ability to monitor and control a real process according to its defined process model. The methodology provided in [36] is also used in [35], to ensure process compliance during execution time, by comparing the executed process with the process reference model specified by a standard. The work presented in [24] aims at facilitating the checking of constraints that can be defined as part of a specific process model (e.g., standards requirements, metrics) by using SWRL (Semantic Web Rule Language) [31]. The approach in [24] is also used in [26], to permit that the description of IT (Information Technology) process models are checked with the constraints provided by the business perspective. An approach for representing SPEM 2.0 process models in DL, to provide process analysis such as reasoning and consistency checks, is presented in [102]. The generation of the tailored process, in the automotive domain, is done by using ontologies created in OWL, which outputs are transformed into SPEM 2.0 process models [32]. There are also approaches that systematically exploit the modeling capabilities of SPEM 2.0 for collecting the compliance elements required by specific standards. Examples

of those approaches are presented in [103], which collects compliance information for EN 50128 [15], and in [3], which collects evidence for supporting compliance with DO-178C [12]. Models for representing Deployment Packages and Implementation Guides for the Standard ISO/IEC 29110 [104] in EPF Composer, are presented in [105]. The use of SPEM 2.0 in the previous approaches, i.e., [36], [35], [24], [26], [102] [32], [103], [3] and [105], are limited to the modeling of the elements required to create the process structure. There are other approaches that, in addition to the structure of the process, use other concepts provided by SPEM 2.0 for modeling other elements required for compliance. An example of these approaches can be found in [29], in which a customization of the elements defined in SPEM 2.0 is performed to give the possibility to generate compliance tables. The modeling of standard's requirements in SPEM 2.0, presented in [106], is used to detect whether the process model contains sufficient evidence for supporting the requirements, providing feedback to the safety engineers regarding detected fallacies and recommendations to solve them. As in [29] and in [106], our approach combines the modeling capabilities for modeling standard's requirements, plus customization of preexisting modeling concepts to generate a centralized compliance-related knowledge base. In addition, we add a layer of confidence by considering the use of methods that allow us to derive proofs of compliance. However, we do not use semantic web methods for deriving our proofs since they are not expressive enough for modelling compliance notions [34]. In addition, semantic web methods are computational methods that deal with ontologies and rules, whose combination could be undecidable [107].

Approaches for compliance checking have been widely studied in the business context. For instance, in [108], the authors propose to capture high-level policies with a compliance metamodel called REALM (Regulations Expressed As Logical Models), to support the formalization of compliance requirements in Real-time Temporal Object Logic [109]. In [110], an object life cycle approach is used to generate a set of actions for the generation of process models, in which the order of the model of the process actions is determined and then combined into process fragments that are connected to decision and merge nodes. In [111], the authors propose an SOA (Service Oriented Architecture)-based compliance governance, called COMPAS, to define compliant process fragments. In [112], authors propose a compliance checking method for business process models, in which norms are expected to be modeled in BPSL (Business Property Specification Language) and then formalized in LTL (Linear Temporal Logic). Once the two formal

specifications are given, model checking via the NuSMV2-based OpenProcessAnalyser is performed for checking compliance. In [113], the authors propose a solution for ensuring compliance by using a formal language for specifying a subset of business rules, called semantic constraints, and the necessary mechanisms for parsing the constraints and ensuring compliance of process management systems. There are also several compliance checking frameworks that combine the modeling capabilities provided by BPMN (Business Process Model and Notations) [114] and Temporal Logics for the modeling of regulations. Examples of these kind of frameworks can be found in [115], [116], and [117]. In our approach, we are using a similar methodology that the presented in the previous mentioned approaches, i.e., [108], [109], [110], [111], [112], [113], [115], [116], and [117]. In particular, we defined a formal specification of the standards requirements which is contrasted to a model of the processes. The main differences in our approach with the previous ones is the use of the modeling languages. In particular, we use FCL (see Section 2.3.3), and not any language derived from the family of Temporal Logics, for creating the formal specification of the standards requirements. The reason for not using Temporal Logics is that this logic is not able to provide conceptually sound representations of the regulatory requirements governing a process [39].

## 5.2 Facilitating Formal Specification of Requirements

As is expressed in [118], writing formal specifications is very difficult, and the common failure is to suggest an implementation rather than a specification of what is required. One of the major challenges associated to the formalization of regulatory texts is the regulatory interpretation, which can include disambiguations, clarifications, and the accounting for exceptions [119]. In this respect, the research found in the state of the art mainly aims at bridging the gap between the normative expertise required to interpret the regulatory text, and the modeling skills required to build the knowledge base. In particular, in [120], the authors focus on the representation of constitutive rules in a machine-readable form. In [121], the authors propose a framework for identifying terms and refining the goals proposed by a privacy legislation. In [122], the authors present a pattern-based approach to capture the knowledge of domains experts. In [43], the authors present a methodology to formalize norms as logic programs. In [123], the

authors uses SBVR (Semantics of Business Vocabulary and Business Rules), which is a controlled natural language, to develop business vocabulary and business rules. In [124], the authors use high-Level methodological approaches for the extraction and representation of compliance objectives, rules and constraints by using semantic web technologies. In [125], the authors specify a conceptual model for representing standards in the safety-critical context. In [126], the authors also use conceptual models to extract information from standards that apply to agricultural production. In [127], the authors describe a methodology for requirements formalization for the flight-critical systems verification. Guidelines are also widely used to spread the used of novel methods in engineering tasks. In [128], the authors describe the Oracle Policy Modeling best practices guidelines for business rules modeling. Similarly, in [129], the authors describe a methodology to guide companies to establish Cyber-Physical Social System data subjects' consent and data usage policies. In [130], the authors present guidelines for supporting the formal representation of safety regulatory requirements. In [131], the authors present the use of tabular expressions to generate formal models of system requirements. The authors of FCL have also published explicative examples of the modeling of FCL rules within the business context, e.g., [132, 50], which can be used as a guideline for learning the language. In [133], the authors present a methodology for extracting models from ISO 26262, called "snowball", in which high-level requirements (objectives section of the standard) are modeled first, and then the low-level requirements (requirements and recommendations) are added as rolling a snowball in the snow. The use of FCL for supporting compliance management tasks in automotive is a novelty. We did not find specific examples or guidelines that apply to the domain yet. Therefore, the results of the formalization-oriented pre-processing of ISO 26262 presented in this thesis may be of interest for process engineers involved in the road vehicles manufacturing. Additionally, we consider that this work can be used as a starting point to derive domain-specific guidance applicable to process-based safety standards beyond ISO 26262.

## 5.3 Reuse of Proofs

"State explosion phenomenon" [134] refers to the exponential grow of the state space required to be verified by automatic checking methods. Essentially, research impacting the reduction of this phenomenon is studied in

software verification. In [135], the authors present an approach for exploiting the evolutionary nature of programs for reusability of proofs, in which a tree of connected proved fragments of a program is built. In [136], the authors present a reuse method, that uses incremental proof construction for reusing correctness proofs of Java programs. In [137], the authors give an overview of techniques for the reuse of information obtained in verifications results from past trials. In [138], the authors present a family-based strategy for minimizing the efforts in verification of product lines. In particular, this verification can be achieved by combining all code of all features in a single product simulator, which checks all valid execution paths of all products without the need of generating and checking any individual product. In the area of process compliance, only few approach that consider reuse of proofs are presented. In [111], the authors consider that after doing many customization steps, which can include techniques, such as abstraction or parametrization, a process fragment can be employed for the reuse of functionality. However, for the application of process fragments to perform compliance checks, special formalizations have to be performed. In [116], business process are augmented with reusable fragments, called process fragments for compliance, which are stored in a fragment repository that supports versioned storage and retrieval. The intention is to ensure process compliance to the corresponding compliance requirements. In this approach, the process is modeled in BPMN (Business Process Model and Notation) and the rules are formalized in LTL (Linear Temporal Logic). According to the authors, the selection of LTL is justified by the fact that Temporal Logics have great trajectory in system property specification, and thus, well tool-support. In our case, we do not base the formalization of the rule in any language of the temporal logics family. Moreover, our approach, which is oriented to the modelling of processes used in engineering tasks, does not uses BPMN (Business Process Model and Notation). Finally, our approach is not considering the uses of process fragments, but the proof of the process line, namely the common elements that conform the family.

# Chapter 6

# Conclusions and Future Work

This chapter finalizes the work presented in this thesis. In particular, we present concluding remarks in Section 6.1, and future work in Section 6.2.

## 6.1 Conclusions

Compliance with process assurance-based safety standards requires that companies provide complete and convincing evidence regarding the processes used to engineer safety-critical systems. The production of compliance checking reports appears as a useful alternative not only for compliance assessment but also for assisting the creation of process specifications. However, the production of compliance checking reports requires that the process engineer performs several, time-consuming and repetitive steps with much focus. This kind of activities are typically considered tedious and error-prone, in which the required focus may be lost leading to mistakes and delays is compliance assessments. For this reason, facilitating automated means for compliance checking is an existing research problem in the safety-critical context.

In this thesis, we aim at providing an approach that facilitates automated compliance checking of the processes used to engineer safety-critical systems against the standards mandated (or recommended) in the safety-critical context. For reaching this goal, we have defined the conditions for

automatically checking compliance based on the application of the compliance by design methodology (see Definition 2.3.12), as follows:

> Automatic compliance checking of a safety plan involves the annotation of the process elements defined to manage and guide the execution of safety activities with compliance effects, which correspond to the permissible states provided by the standards requirements, to describe a model with standard-compliant states.

With this definition, we proposed an automated compliance checking vision that suits the needs in the safety-critical context. The compliance checking vision combines:

1. process modeling and process annotation capabilities that are required for defining process models checkable for compliance,
2. normative representation capabilities that permit the interpretation of the standards requirements in an adequate machine-readable form, and the generation of the compliance effects, which are the permissible states required for the annotation process, and
3. reasoning capabilities necessary to conclude whether an annotated process model corresponds to the model with the compliant states described in the standards requirements.

These capabilities are tool-supported, as follows: SPEM 2.0 (Software and Systems Process Engineering Metamodel) provides the modeling and annotation capabilities, FCL (Formal Contract Logic) provides the normative representation capabilities, and Regorous provides compliance checking capabilities. To support the compliance checking vision, we identified the essential elements required to generate process models checkable for compliance in SPEM 2.0, and the transformations necessary to automatically generate the models that can be processed by Regorous. We have also explored techniques for facilitating the formalization of the process-based requirements provided by ISO 26262, which is the functional safety standard applicable to the automotive industry. The result of this exploration was the characterization of safety compliance pattern as follows:

> Safety compliance patterns describe commonly occurring normative safety requirements on the permissible state sequence of a finite state model of a process.

With this characterization, we identify, define, and instantiate ISO 26262-related compliance patterns, which may be useful for supporting automated compliance checking in automotive. We also generate a methodological guideline for facilitating the formalization of normative clauses in ISO 26262. Finally, we introduce an approach for mastering the interplay between highly-related standards. This approach, called SoPLE&Logic-basedCM, includes the reuse capabilities provided by SoPLE (Safety-oriented Process Line Engineering), which is a methodological approach aiming at systematizing the reuse of process-related information in the context of safety-critical systems. With the addition of SoPLE, we aim at planting the seeds for the future provision of systematic reuse of compliance proofs.

Hitherto, our proposed methodology has been evaluated with academic examples that show the potential benefits of its use. Our work represents a novelty in process-based compliance in the safety-critical context, which may contribute to increasing efficiency, via automation, and confidence, via formal checking. It also contributes to cross-fertilize previously isolated communities, i.e., the safety-critical and the legal contexts.

Figure 6.1 presents the way in which the contributions (described in Chapter 4), the subgoals (presented in Chapter 3, Section 3.2.3), and the papers published (described in Chapter 1, Section 1.1) are connected.

## 6.2 Future Work

The results of our thesis can be improved in several directions. Here, we present the suggested areas of research in the future.

- The mapping of regulations to the process tasks, i.e. the annotation of the compliance effects, is done manually, by deducing the effects that can eventually be caused by the tasks in the general compliance status. When the processes are small, this mapping is straightforward. However, when processes are extensive, the mapping may be difficult to achieve. Therefore, methodologies and tactics should be investigated so that the process annotation does not become a burden for the application of the compliance checking approach.

- The automated compliance checking vision, described in this Licentiate thesis, only permits that the analysis of compliance is performed in the sequence of tasks assigned to a process plan. However, a process plan is not

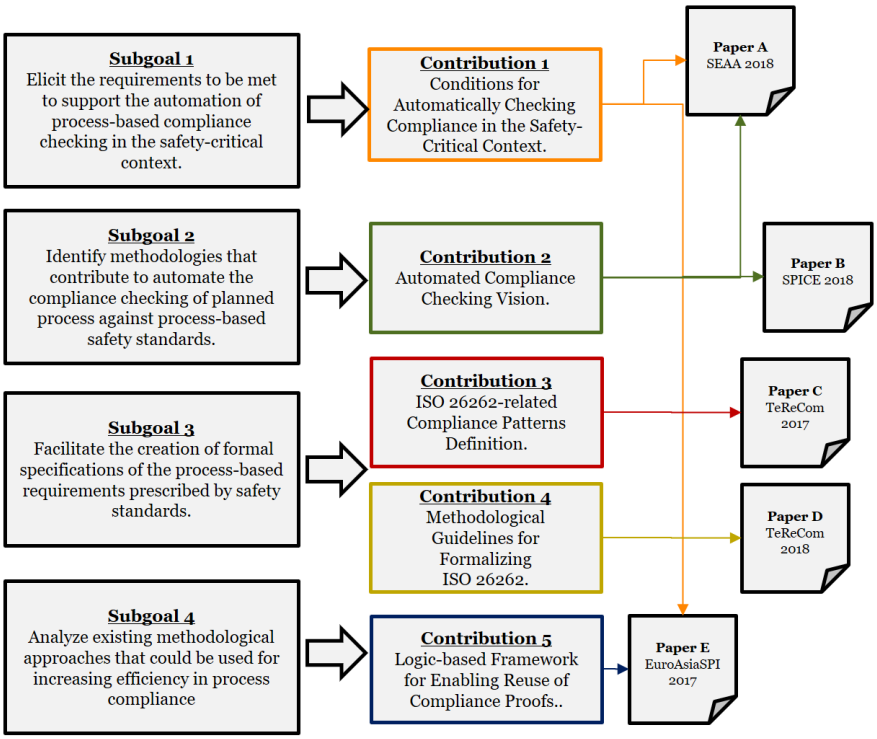


Figure 6.1: Connection between subgoals, contributions and the published papers.

only comprised by tasks but also it contains other process elements, such as roles and work products. We aim at extending our approach for permitting that compliance effects annotated to process elements beyond tasks are also included in the analysis of compliance.

- The compliance checking methodology used in our approach is, undoubtedly, process modeling language agnostic. However, the current tool-support lacks agnosticism, i.e., it depends on a specific modeling tool to provide compliance checking results. This characteristic impedes the back-propagation of the compliance results in our selected process modeling language. Therefore, we need to investigate methods and strategies that allow us to represent the compliance checking results in an

agnostic way so that we can concretize our compliance checking vision.

- We have limited our analysis of patterns and methodological guidelines to the functional safety standard ISO 26262. This restriction may also limit the applicability of our approach. To expand our horizon, we need to generalize the use of patterns and methodological guidelines so that we can incorporate a wide range of standards. Therefore, comparative studies between standards and definition of generalized patterns, as well as standard-specific patterns could be investigated.

- The reuse of proofs of compliance may increase efficiency and confidence in compliance checking. Thus, we aim at studying in deep the conditions that are required for compositionality of proofs of compliance. We also need to provide metrics for measuring increased confidence and increased efficiency.

- Our work has only be evaluated with academic examples. Therefore, we require to further validating the approach with more complex cases, i.e., industrial cases.

- We need to better situate our work in the context of the state of the art. Therefore, an extended and systematic literature review will be performed.

- To augment the impact for our results, we plan to integrate our automated compliance checking approach to the platform created by the European project AMASS (Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems) [1]. We also plan to develop a course module in which students can learn about automated compliance checking with our approach.

# Bibliography


[1] AMASS.: Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems

[2] Leveson, N.: Safety : Why , What , and How. ACM Computing Surveys (CSUR) **18**(2) (1986) 125–163

[3] Gannous, A., Andrews, A., Gallina, B.: Toward a Systematic and Safety Evidence Productive Verification Approach for Safety-Critical Systems. The 29th IEEE International Symposium on Software Reliability Engineering (ISSRE 2018) (2018) 329–336

[4] Tordrup, L., Nielsen, P.: A conceptual model of agile software development in a safety-critical context: A systematic literature review. Information and Software Technology (2018)

[5] Leveson, N.: System Safety in Computer-Controlled Automotive Systems. Technical report, SAE Technical Paper (2000)

[6] Perrow, C.: Normal Accidents: Living with High Risk Technologies. Princeton University Press (2011)

[7] Dunn, W.: Designing Safety-Critical Computer Systems. Computer **36**(11) (2003) 40–46

[8] Knight, J., Rowanhill, J.: The Indispensable Role of Rationale in Safety Standards. In: 35th International Conference SAFECOMP. Volume 2788., Springer (2016) 39–50

[9] Gallina, B.: How to increase efficiency with the certification of process compliance. In: The 3rd Scandinavian Conference on Systems & Software Safety. (2015)

[10] Varkoi, T., Nevalainen, R., Makinen, T.: Process Assessment in A Safety Domain. In: 10th International Conference on the Quality of Information and Communications Technology. (2016) 52–58

[11] Kelly, T.: Can process-based and product-based approaches to software safety certification be reconciled? Improvements in System Safety (2008) 3–12

[12] RTCA/DO-178C: Software Considerations in Airborne Systems and Equipment Certification. (2011)

[13] IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems (2010)

[14] ISO 26262: Road vehicles Functional safety. (2011)

[15] EN50128, B.: Railway applications-Communication, signaling and processing systems Software for railway control and protection systems. British Standards Institution (2011)

[16] McDermid, J., Rae, A.: Goal-Based Safety Standards: Promises and Pitfalls. Achieving Systems Safety (2012) 257–270

[17] Gallina, B., Ul Muram, F., Castellanos Ardila, J.: Compliance of Agilized (Software) Development Processes with Safety Standards: a Vision. In: 4th international workshop on Agile Development of Safety-Critical Software (ASCS). (2018)

[18] Hawkins, R., Richardson, T., Kelly, T.: Using Process Models in System Assurance. In: Computer Safety, Reliability, and Security, Springer International Publishing (2016) 27–38

[19] Jiménez, J., Amelio, J., Merodio, M., Sanz, L.: Computer Standards & Interfaces Checklists for compliance to DO-178C and DO-278A standards. Computer Standards & Interfaces **52** (2017) 41–50

[20] Chung, P.W.H., Cheung, L.Y.C., Machin, C.H.C.: Compliance Flow - Managing the compliance of dynamic and complex processes. Knowledge-Based Systems **21**(4) (2008) 332–354

[21] Fuggetta, A., Di Nitto, E.: Software process. In: Future of Software Engineering. (2014) 1–12

[22] Object Management Group Inc.: Software & Systems Process Engineering Meta-Model Specification. Version 2.0. OMG Std., Rev (2008) 236

[23] Koudri, A., Champeau, J.: MODAL: A SPEM extension to improve co-design process models. New Modeling Concepts for Today's Software Processes **6195** (2010) 248–259

[24] Rodríguez, D., Garcia, E., Sanchez, S., Rodríguez-Solano Nuzzi, C.: Defining software process model constraints with rules using OWL and SWRL. International Journal of Software Engineering and Knowledge Engineering **20**(04) (2010) 533–548

[25] Object Management Group: Unified Modeling Language Specification Version 2.5.1 (2017)

[26] Valiente, M., García-Barriocanal, E., Sicilia, M.: Applying Ontology-Based Models for Supporting Integrated Software Development and IT Service. IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews **42**(1) (2012) 61–74

[27] 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

[28] McIsaac, B.: IBM Rational Method Composer: Standards Mapping. Technical report, IBM Developer Works (2015)

[29] ECSEL Research and Innovation actions (RIA) - AMASS: D6.5 Prototype for Cross/Intra-Domain Reuse (b) (2017)

[30] Wen, L., Tuffley, D., Rout, T.: Using Composition Trees to Model and Compare. In: International Conference on Software Process Improvement and Capability Determination. Number March 2014, Springer (2011) 1–15

[31] Horrocks, I., Patel-schneider, P., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member submission **21**(79) (2004) 1–31

[32] Jost, H., Köhler, S., Köster, F.: Towards a Safer Development of Driver Assistance Systems by Applying Requirements-Based Methods. In: 14th International IEEE Conference on Intelligent Transportation Systems (ITSC), IEEE (2011) 1144–1149

[33] OWL Working Group: Web Ontology Language (OWL)

[34] Palmirani, M., Governatori, G.: Modelling Legal Knowledge for GDPR Compliance Checking. Frontiers in Artificial Intelligence and Applications **313** (2018) 101–110

[35] Golra, F., Dagnat, F., Bendraou, R., Beugnard, A.: Continuous Process Compliance Using Model Driven Engineering. In: International Conference on Model and Data Engineering, Springer (2017) 42–56

[36] Bendraou, R., Combemale, B., Crégut, X., Gervais, M.: Definition of an executable SPEM 2.0. In: 14th Asia-Pacific Software Engineering Conference (ASPEC). (2007) 390–397

[37] Francesconi, E.: Semantic model for legal resources: Annotation and reasoning over normative provisions. Semantic Web **7**(3) (2016) 255–265

[38] Siena, A., Mylopoulos, J., Perini, A., Susi, A.: From Laws to Requirements. In: Requirements Engineering and Law (RELAW'08). (2008) 6–10

[39] Governatori, G., Hashmi, M.: No Time for Compliance. IEEE 19th International Enterprise Distributed Object Computing Workshop, (EDOCW) (2015) 9–18

[40] Hashmi, M., Governatori, G., Wynn, M.: Normative requirements for regulatory compliance: An abstract formal framework. Information Systems Frontiers. **18**(3) (2016) 429–455

[41] Oasis: LegalRuleML Core Specification Version 1.0 (2017)

[42] Wieringa, R., Meyer, J.: Applications of Deontic Logic in Computer Science : A Concise Overview. Deontic logic in computer science (1993) 17–40

[43] Akinkunmi, B., Babalola, M.: Knowledge Representation for High-Level Norms and Violation Inference in Logic Programming. arXiv preprint arXiv:1801.06740 (2018)

[44] Fenton, N., Neil, M.: A Strategy for Improving Safety Related Software Engineering Standards. IEEE Transactions on Software Engineering **24**(11) (1998) 1002–1013

[45] de la Vara, J., Ruiz, A., Attwood, K., Espinoza, H., Panesar-Walawege, R., López, A., del Río, I., Kelly, I.: Model-based specification of safety compliance needs for critical systems: A holistic generic metamodel. Information and Software Technology **72** (2016) 16–30

[46] Nute, D.: Defeasible Logic. In: International Conference on Applications of Prolog, Springer (2001) 151–169

[47] Governatori, G.: Representing business contracts in RuleML. International Journal of Cooperative Information Systems **14**(02n03) (2005) 181–216

[48] Alberti, M., Gavanelli, M., Lamma, E., Riguzzi, F., Zese, R.: Dischargeable Obligations in Abductive Logic Programming. In Springer, ed.: International Joint Conference on Rules and Reasoning. (2017) 7–21

[49] Governatori, G., Shek, S.: Regorous: A Business Process Compliance Checker. In: 14th International Conference on Artificial Intelligence and Law (ICAIL), ACM (2013) 245–246

[50] Governatori, G.: The Regorous Approach to Process Compliance. In: IEEE 19th International Enterprise Distributed Object Computing Workshop (EDOCW), IEEE (2015) 33–40

[51] Koliadis, G., Ghose, A.: Verifying Semantic Business Process Models in Inter-operation. In: IEEE International Conference on Service-Oriented Computing (SCC). (2007) 731–738

[52] Sadiq, S., Governatori, G., Namiri, K.: Modeling Control Objectives for Business Process Compliance. In: International Conference on Business Process Management. (2007) 149–164

[53] Castellanos Ardila, J.P., Gallina, B., Ul Muram, F.: Enabling Compliance Checking against Safety Standards from SPEM 2.0 Process Models. In: 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). (2018)

[54] The Eclipse Foundation: Eclipse Composer Framework

[55] IBM Corporation: Key Capabilities of the Unified Method Architecture (UMA)

[56] Castellanos Ardila, J.P., Gallina, B., UL Muram, F.: Transforming SPEM 2.0-compatible Process Models into Models Checkable for Compliance. In: 18th International SPICE Conference. (2018)

[57] Castellanos Ardila, J.P., Gallina, B.: Formal Contract Logic Based Patterns for Facilitating Compliance Checking against ISO 26262. In: 1st Workshop on Technologies for Regulatory Compliance (TeReCom). (2017) 65–72

[58] Castellanos Ardila, J., Gallina, B., Governatori, G.: Lessons Learned while formalizing ISO 26262 for Compliance Checking. In: 2nd Workshop on Technologies for Regulatory Compliance (TeReCom), CEUR-Workshop Proceedings (2018) 1–12

[59] Castellanos Ardila, J.P., Gallina, B.: Towards Increased Efficiency and Confidence in Process Compliance. In: Systems, Software and Services Process Improvement (EuroAsiaSPI). Volume 748., Springer International Publishing (2017) 162–174

[60] Knight, J.C.: Safety critical systems: challenges and directions. In: 24rd International Conference on Software Engineering, ACM (2002) 547 – 550

[61] International Electrotechnical Commission: Functional safety. Essential to overall safety (2015)

[62] Leveson, N.: Safeware: system safety and computers. Addison Wesley (1995)

[63] Leveson, N.: The use of safety cases in certification and regulation. Massachusetts Institute of Technology. Engineering Systems Division (November) (2011)

[64] Kneuper, R.: Software Processes and Life Cycle Models. An Introduction to Modelling, Using and Managing Agile, Plan-Driven and Hybrid Processes. Springer, Cham (2018)

[65] SAE: Surface Vehicle Recommended Practice. Technical report (2016)

[66] Sommerville, I.: Software Processes. In: Software Engineering. 9th editio edn. Addison-Wesley (2011) 43–55

[67] Boutros, T., Purdie, T.: The process improvement handbook: A blueprint for managing change and increasing organizational performance. (2014)

[68] Tuft, B.: Eclipse Process Framework (EPF) Composer: Installation, Introduction, Tutorial and Manual

[69] Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley Reading (2002)

[70] Ternité, T.: Process lines : a product line approach designed for process model development. In: 35th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). (2009) 173–180

[71] Castellanos Ardila, J.P., Gallina, B.: Towards Efficiently Checking Compliance Against Automotive Security and Safety Standards. In: The 7th IEEE International Workshop on Software Certification (WoSoCer). (2017)

[72] Haugen, Ø., Øgård, O.: BVR Better Variability Results. In: International Conference on System Analysis and Modeling. (2014) 1–15

[73] Javed, M., Gallina, B.: Safety-oriented Process Line Engineering via Seamless Integration between EPF Composer and BVR Tool. In: 22nd International Systems and Software Product Line Conference, ACM (2018) 23–28

[74] Emmerich, W., Finkelstein, A., Montangero, C., Antonelli, S., Armitage, S., Stevens, R.: Managing Standards Compliance. IEEE Transactions on Software Engineering **25**(6) (1999) 836–851

[75] International Atomic Energy Agency: Approaches and Methods to Conduct Regulatory Safety Review and Assessment (2013)

[76] IEEE Computer Society: Guide to the Software Engineering Body of Knowledge Version 3.0 (SWEBOK Guide V3.0). (2014)

[77] Bel V, BfS, CSN, ISTec, ONR, SSM, STUK: Licensing of safety critical software for nuclear reactors. Common position of seven European nuclear regulators and authorised technical support organisations. Technical report (2010)

[78] Vilkomir, S., Ghose, A.: Development of a Normative Package for Safety-Critical Software Using Formal Regulatory Requirements. In: International Conference on Product Focused Software Process Improvement. (2004) 523–537

[79] Raz, J.: Practical Reason and Norms. Oxford Scholarship Online (1999)

[80] Koetter, F., Kochanowski, M., Renner, T., Fehling, C., Leymann, F.: Unifying compliance management in adaptive environments through variability descriptors. Proceedings - IEEE 6th International Conference on Service-Oriented Computing and Applications, SOCA 2013 (2013) 214–219

[81] Governatori, G., Hashmi, M., Lam, H., Villata, S., Palmirani, M.: Semantic Business Process Regulatory Compliance Checking using LegalRuleML. In: European Knowledge Acquisition Workshop, Springer (2016) 746–761

[82] Diallo, B.: The Binding Force of International Legal Standards in the Face of the Recurrent Practice of Soft Law. Adam Mickiewicz University Law Review **7** (2017) 79–91

[83] Hashmi, M.: Evaluating Business Process Compliance Management Frameworks. Doctoral dissertation, Queensland University of Technology (QUT) (2015)

[84] Von Wright, G.H.: Deontic logic. Mind **60**(237) (1951) 1–15

[85] Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Representation Results for Defeasible Logic. ACM Transactions on Computational Logic (2) (2000) 255–287

[86] RuleML: Uses of Rules. http://wiki.ruleml.org/index.php/RuleML_Home

[87] Barlas, K., Berki, E., Stefaneas, P., Koletsos, G.: Towards formal open standards : formalizing a standard ' s requirements. Innovations in Systems and Software Engineering **13**(1) (2017) 51–66

[88] Lin, Y.: Semantic Annotation for Process Models: Facilitating Process Knowledge Management via Semantic Interoperability. Doctoral thesis, Norwegian University of Science and Technology (2008)

[89] Riehle, D., Züllighoven, H.: Understanding and using patterns in software development. Tapos **2**(1) (1996) 3–13

[90] Dwyer, M., Avrunin, G., Corbett, J.: Property Specification for Finite-State Verification. In: 2nd Workshop on Formal Methods in Software Practice. (1998) 7–15

[91] Peffers, K., Tuunanen, T., Rothenberger, M., Chatterjee, S.: A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems **24**(3) (2007) 45–77

[92] O'Connor, R., Laporte, C.: An innovative approach to the development of an international software process lifecycle standard for very small entities. Volume III. IGI Global (2018)

[93] Engiel, P., Sampaio do Prado Leite, J., Mylopoulos, J.: A Tool-Supported Compliance Process for Software Systems. In: 2017 11th International Conference on Research Challenges in Information Science (RCIS), IEEE (2017) 66–76

[94] Kabaale, E., Wen, L., Wang, Z., Rout, T.: Representing Software Process in Description Logics: An Ontology Approach for Software Process Reasoning and Verification. In: Software Process Improvement and Capability Determination, Springer (2016) 362–376

[95] SPICE Project Organization: ISO/IEC 15504 -Software Process Assessment (SPICE).

[96] Kabaale, E., Wen, L., Wang, Z., Rout, T.: Ensuring Conformance to Process Standards Through Formal Verification. In: International Conference on Software Process Improvement and Capability Determination. Volume 2., Springer International Publishing (2018) 248–262

[97] Proença, D., Borbinha, J.: Formalizing ISO/IEC 15504-5 and SEI CMMI v1.3 Enabling automatic inference of maturity and capability levels. Computer Standards and Interfaces (2018)

[98] Soydan, G., Kokar, M.: A Partial Formalization of the CMMI-DEV A Capability Maturity Model for Development. Journal of Software Engineering and Applications **5**(10) (2012) 777–788

[99] Software Engineering Institute, C.M.: CMMI® for Development, Version 1.3 CMMI-ACQ, V1.3. Technical Report November, Software Engineering Institute, Carnegie Mellon (2010)

[100] Bonatti, P.: Fast Compliance Checking in an OWL2 Fragment. In: 27th International Joint Conferences on Artificial Intelligence Organization (IJCAI). (2018) 1746–1752

[101] Borgida, A.: On the relative expressiveness of description logics and predicate logics. Artificial intelligence **82**(1-2) (1996) 353–367

[102] Wang, S., Jin, L., Jin, C.: Represent S oft w are Process Engineering Metamode l in Description Logic. World Academy of Science, Engineering and Technology **11** (2006) 109–113

[103] Gallina, B., Gómez-Martínez, E., Earle, C.: Deriving Safety Case Fragments for Assessing MBASafe ' s Compliance with EN 50128. In: International Conference on Software Process Improvement and Capability Determination. Volume 1. (2016) 3–16

[104] ISO/IEC TR 29110: Systems and software engineering – Lifecycle profiles for Very Small Entities (VSEs). **2** (2016) 23

[105] École de technologie supérieure of Canada.: Deployment Packages for the Generic Profile Group for VSEs Developing Systems and/or Software (2010)

[106] Ul Muram, F., Gallina, B., Gomez Rodriguez, L.: Preventing Omission of Key Evidence Fallacy in Process-based Argumentations. In: 11th International Conference on the Quality of Information and Communications Technology (QUATIC). (2018)

[107] Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman & Hall/CRC, (2009)

[108] Giblin, C., Müller, S., Pfitzmann, B.: From Regulatory Policies to Event Monitoring Rules: Towards Model-Driven Compliance Automation. Technical report, IBM Research Laboratory, Zurich (2006)

[109] Giblin, C., Liu, A., Müller, S., Pfitzmann, B., Zhou, X.: Regulations Expressed As Logical Models (REALM). Technical report, IBM China Research Lab (2005)

[110] Küster, J., Ryndina, K., Gall, H.: Generation of Business Process Models for Object Life Cycle Compliance. In: International Conference on Business Process Management, Springer (2007) 165–181

[111] Daniel, F., Casati, F., Mulo, E., Zdun, U., Strauch, S., Schumm, D., Leymann, F., Sebahi, S., De Marchi, F., Hacid, M.S.: Business compliance governance in service-oriented architectures. In: International Conference on Advanced Information Networking and Applications (AINA). (2009) 113–120

[112] Awad, A., Decker, G., Weske, M.: Efficient Compliance Checking Using BPMN-Q and Temporal Logic. International Conference on Business Process Management (2008) 326–341

[113] Ly, L., Göser, K., Rinderle-ma, S., Dadam, P.: Compliance of Semantic Constraints A Requirements Analysis for Process Management Systems. In: 1st Int'ernational Workshop on Governance, Risk and Compliance - Applications in Information Systems (GRCIS). (2008)

[114] Object Management Group: Business Process Model and Notation Version 2.0. (2011)

[115] Elgammal, A., Turetken, O., van den Heuvel, W., Papazoglou, M.: Formalizing and applying compliance patterns for business process compliance. Software and Systems Modeling. (2016) 119–146

[116] Schumm, D., Turetken, O., Kokash, N., Elgammal, A., Leymann, F., van den Heuvel, W.: Business Process Compliance through Reusable Units of Compliant Processes. In: International Conference on Web Engineering. (2010) 325–337

[117] El Kharbili, M.: Business Process Regulatory Compliance Management Solution Frameworks: A Comparative Evaluation. In: 8th Asia-Pacific Conference on Conceptual Modelling. (2012) 23–32

[118] Rushby, J.: Formal methods and the certification of critical systems. Technical report, SRI International (1993)

[119] Abi-lahoud, E., Brien, L., Butler, T.: On the Road to Regulatory Ontologies Interpreting Regulations with SBVR. AI Approaches to the Complexity of Legal Systems (2014) 188–201

[120] Ceci, M., Butler, T., Brien, L., Al Khalil, F.: Legal Patterns for Different Constitutive Rules. AI Approaches to the Complexity of Legal Systems (2015) 105–123

[121] Islam, S., Mouratidis, H., Wagner, S.: Towards a Framework to Elicit and Manage Security and Privacy Requirements from Laws and Regulations. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. (2010) 255–261

[122] Faßbender, S., Heisel, M.: A Computer-Aided Process from Problems to Laws in Requirements Engineering. In: Software Technologies, Springer Berlin Heidelberg (2014) 215–234

[123] Bouzidi, K., Faron-Zucker, C., Fies, B., Le Thanh, N.: An ontological approach for modeling technical standards for compliance checking. In: International Conference on Web Reasoning and Rule Systems, Springer (2011) 244–249

[124] Bala, S., Cabanillas, C., Haselböck, A., Havur, G., Mendling, J., Sperl, S., Steyskal, S.: A Framework for Safety-Critical Process Management in Engineering Projects. In: International Symposium on Data-Driven Process Discovery and Analysis. Volume 1. (2015) 1–27

[125] de la Vara, J., Gómez, A., Gallego, E., Génova, G., Fraga, A.: Representation of Safety Standards with Semantic Technologies Used in Industrial Environments. In: International Conference on Computer Safety, Reliability, and Security. (2017) 265–272

[126] Nash, E., Wiebensohn, J., Nikkilä, R., Vatsanidou, A., Fountas, S., Bill, R.: Towards automated compliance checking based on a formal representation of agricultural production standards. Computers and Electronics in Agriculture **78**(1) (2011) 28–37

[127] Brat, G., Bushnell, D., Davies, M., Giannakopoulou, D., Howar, F., Temesghen, K.: Verifying the Safety of a Flight-Critical System. In: International Symposium on Formal Methods, Springer (2015) 308–324

[128] Lee, J.: Oracle Policy Automation (OPA). Best Practice Guide for policy Modelers. (2018)

[129] Fernandez, J.: Deliverable 6.1: Privacy policy formalization (v. 1) (2018)

[130] Vilkomir, S., Bowen, J., Ghose, A.: Formalization and assessment of regulatory requirements for safety-critical software. Innovations in Systems and Software Engineering **2**(3-4) (2006) 165–178

[131] Singh, N., Lawford, M., Maibaum, T., Wassyng, A.: Use of Tabular Expressions for Refinement Automation. In: International Conference on Model and Data Engineering. (2017) 167–182

[132] Governatori, G.: Practical Normative Reasoning with Defeasible Deontic Logic. In: Reasoning Web International Summer School. (2018) 1–25

[133] Luo, Y., Van Den Brand, M., Engelen, L., Favaro, J., Klabbers, M., Sartori, G.: Extracting models from ISO 26262 for reusable safety assurance. Lecture Notes in Computer Science **7925 LNCS** (2013) 192–207

[134] Giannakopoulou, D., Namjoshi, K., Pasareanu, K.: Compositional reasoning. In: Handbook of logic and language. Springer (2018) 345–383

[135] 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

[136] Beckert, B., Klebanov, V.: Proof reuse for deductive program verification. In: Proceedings of the Second International Conference on Software Engineering and Formal Methods (SEFM ), IEEE (2004) 77–86

[137] Beyer, D., Wendler, P.: Reuse of verification results conditional model checking, precision reuse, and verification witnesses. Model Checking Software **7976** (2013) 1–17

[138] Apel, S., Von Rhein, A., Wendler, P., Groslinger, A., Beyer, D.: Strategies for product-line verification: Case studies and experiments. International Conference on Software Engineering (2013) 482–491

# II

# Included Papers

# Chapter 7

# Paper A: Enabling Compliance Checking against Safety Standards from SPEM 2.0 Process Models

Julieth Patricia Castellanos Ardila, Barbara Gallina and Faiz UL Muram.
In Proceedings of the Euromicro Conference on Software Engineering and Advanced Applications (SEAA-2018), Prague, Czech Republic, August 2018.

**Abstract**

Compliance with process-based safety standards may imply the provision of a safety plan and its corresponding compliance justification. However, the provision of this justification is time-consuming since it requires that the process engineer checks the fulfillment of hundred of requirements by taking into account the evidence presented in the process entities. In this paper, we aim at supporting process engineers by introducing our compliance checking vision, which consists of the combination of process modeling capabilities via SPEM 2.0 (Systems & Software Process Engineering Metamodel) reference implementations and compliance checking capabilities via Regorous, a compliance checker, used for business processes compliance checking. Our focus is on the identification and exploitation of the appropriate (minimal set of) SPEM 2.0-like elements, available in the selected reference implementation, which can be used by Regorous for compliance checking. Then, we illustrate our vision by applying it onto a small excerpt from ISO 26262. Finally, we draw our conclusions.

## 7.1 Introduction

Compliance with process-based safety standards, which impose requirements on the processes to be adopted to engineer safety-critical systems [1], may imply the provision of a safety plan (used to manage and facilitate the execution of safety activities) together with a compliance justification [2]. The provision of this justification is time-consuming since it requires that the process engineer checks the fulfilment of hundred of requirements by taking into account the evidence provided by the process entities. In order to support compliance checking, existing tools and their methodologies are available. On the one hand, the software processes can be modeled by using SPEM 2.0 [3], a metamodel that provides generic process concepts and extension mechanisms for modeling and documenting software processes [4]. SPEM 2.0 characteristics have been used to support the creation of compliance tables (mapping between standard's requirements and process entities [5]). On the other hand, there is Regorous [6], a tool-supported methodology for automated compliance checking used in the business and legal contexts. Regorous provides a generic framework in which formally captured normative requirements can be propagated in the process models as compliance effects to derive proofs of compliance. Compliance effects are those effects that are caused by the cumulative interaction between the process tasks that are adhered to the standard requirements influences [7]).

Compliance checking of software process plans against safety standards may add value during the negotiation with the certification bodies in the planning phase. Therefore, In this paper, we aim at supporting process engineers by introducing our compliance checking vision, which consists of the combination of process modeling capabilities via SPEM 2.0 (Systems & Software Process Engineering Metamodel) reference implementations and compliance checking capabilities via Regorous, a compliance checker, used for business processes compliance checking. Our focus is on the identification and exploitation of the appropriate (minimal set of) SPEM 2.0-like elements, available in the selected reference implementation, which can be used by Regorous for compliance checking. We illustrate our vision by applying it onto a small excerpt from ISO 26262 [8], and show how the report with compliance results can help the process engineer to trace the unfulfilled requirements.

The rest of the paper is organized as follows. In Section 11.2, we provide background. In Section 7.3, we present our compliance checking vision and the mechanism to annotate software process. In Section 7.4, we examine an ISO

26262-based small example. In Section 7.5, we present related work. Finally, in Section 7.6, we provide conclusions and future work.

## 7.2 Background

In this section, we provide essential background on which we base our work.

### 7.2.1 SPEM 2.0

SPEM 2.0 [3] is a standard that describes *Method Content* (knowledge base for describing process) and *Processes*. We recall some elements used in this paper. A *task definition* is an assignable unit of work which has expected input/output *work products*. *Guidance* provides additional descriptions to method content elements, e.g., *Concept* and *Reusable Asset*. *Custom Category* is a way to organize elements. A *Delivery Process*, which is an integrated approach for performing a project, contains a *Breakdown Structure*, which allows the nesting of units of work (as task use). SPEM 2.0 supports variability management, e.g., *Contributes*, which allows extending a base in an additive fashion without altering its existing properties. The open-source tool *EPF (Eclipse Process Framework) Composer* [9], implements UMA (Unified Method Architecture), a metamodel that exhibits a good coverage of SPEM 2.0 concepts. Also, EPF Composer has a proprietary activity diagram which partially generates the execution semantics of a defined process, and permits importing and exporting libraries with projects (a.k.a. plugins) allowing reusability. Some of the concepts mentioned are described with icons (see Table 7.1).

Table 7.1: Subset of Icons used in SPEM 2.0/EPF Composer.

| SPEM 2.0/EPF Composer | Icon |
|---|---|
| Task Definition/Use |  |
| Work Product |  |
| Delivery Process |  |
| Custom category |  |

### 7.2.2 IBM Standards Mapping Method

Within AMASS project [10], the IBM approach for mapping software processes to standard's requirements [5] was adopted and adapted resulting in an approach to model standard's requirements with EPF Composer [11]. It requires the definition of three plugins. First, the *Standard's requirements plugin*, in which requirements are captured in a *user-defined type* and grouped into a nested table of content by using *custom categories*. Second, a *Lifecycle elements plugin* which contains the documented process elements. Third, a *Requirements mapping* which contains an extended copy (by using a *contributes* relationship) of the standards requirements to be mapped to the process elements that fulfil them. Within

### 7.2.3 Regorous

Regorous [6] is a tool-supported methodology that implements *compliance by design*, an approach that helps process engineers to reach compliance by providing, in a compliance report, the causes of regulations violations and reparation policies. For this, Regorous defines a logical state representation of the process model to be contrasted with a compliance rule set. Three are the main inputs of Regorous. First, the *rule set*, which is obtained from the formalization of the normative requirements in FCL (Formal Contract Logic, the underlying rule-base language used by Regorous). Second, the *execution semantics of the process*, which is obtained from the modeling of the process. Third, the *compliance effects* annotated in the process tasks, which are effects extracted from the set of formulas of the logic that represent the regulations.

### 7.2.4 ISO 26262

ISO 26262 [8] is a standard that addresses functional safety in automotive. ISO 26262 prescribes a safety lifecycle and uses ASIL (Automotive Safety Integrity Levels) to specify applicable safety requirements. In this section, we present a set of rules extracted from the requirements presented in Table 11.2. The interested reader may refer to our previous work [1] for the complete explanation of the formalization process. However, we briefly explain the meaning of the rules. Initially, a rule ($r_1$ in the Ruleset 1) indicates initiation of the Software Unit Design process (R1 in Table 11.2). The expression *in accordance with* (R2) recalls the concept of precondition, namely a task is prohibited (specification of the software units) until the previous tasks or

elements are provided (architectural design and safety requirements) (rules $r_2$ and $r_2'$). The use of mandatory methods in the description of software units which are conditioned by the ASIL and the recommendation levels (R3) can be tailored by providing a rationale that the selected methods comply with the corresponding requirement (see $r_3$, $r_3'$). Conflicts between $r_2$ and $r_2'$ as well as $r_3$ and $r_3'$ due to the presence of contradictory conclusions can be solved by adding superiority relations. Superiority relations give high priority to a rule over the other allowing the checker to derive conclusions without contradictions.

Table 7.2: Requirements for ISO 26262:6 clause 8.

| ID | Ref | Description |
|---|---|---|
| R1 | 8 | Software unit design phase initiation. |
| R2 | 8.1 | Specify software units in accordance with the architectural design and the associated safety requirements. |
| R3 | 8.4.2 | The software unit design shall be described using specific notations, according to ASIL and recommendation levels. |

RuleSet 1: ISO 26262-Software Unit Design Process

$$
\begin{aligned}
r_1 :\Rightarrow\ & [OM]addressSwUnitDesignProcess \\
r_2 :\ & addressSwUnitDesignProcess \\
\Rightarrow\ & [OANPNP] - performSpecifySwUnit \\
r_2' :\ & performProvideSwArchitecturalDesign, \\
& performProvideSwSafetyRequirements \\
\Rightarrow\ & [P]performSpecifySwUnit \\
r_3 :\ & performSpecifySwUnit \\
\Rightarrow\ & [OANPNP]selectMandatoryNotationsforSwDesign \\
r_3' :\ & provideRationaleForNotSelectMandatoryNotationsforSwDesign \\
\Rightarrow\ & [P] - selectMandatoryNotationsforSwDesign \\
& r_2' > r_2, r_3' > r_3
\end{aligned}
\tag{7.1}
$$

## 7.3 Automated Compliance Checking Vision

Our automated compliance checking vision uses SPEM 2.0 elements that implemented in EPF Composer can be used to provide the minimal set of elements to be process by the compliance checker Regorous. As depicted in Figure 8.1, a *process engineer* should support an *FCL expert* in the formalization of the rules, as well as model and annotate the software processes, by using EPF Composer. The interaction between these two tools requires data transformation since EPF Composer and Regorous have different data schemas. In particular, EPF Composer produces the standards description and their formalization, the annotated software process and a diagram model, which should be transformed into the rule set, the compliance annotated tasks and the process execution semantics required by Regorous. Regorous produces a compliance report, which includes rules violations and reparation policies. The information provided by the compliance report can be, through back-propagation into EPF Composer, support the analysis and improvement of the software process.

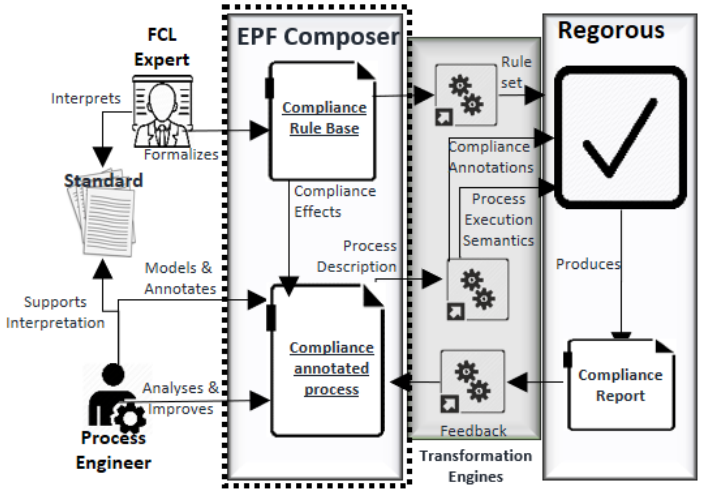


Figure 7.1: Automated Compliance Checking Vision.

The rest of the content of this paper focuses on the region delimited by the dotted line depicted in Figure 8.1, which has a twofold function. First, we identify which modeling capabilities should be used for capturing standard's

information, i.e., we use the guidance kinds offered by SPEM 2.0 called *Reusable Asset* to capture superiority relations between rules, *Concept* to capture compliance effects, and *custom categories*, to organize a nested list of the standard requirements. These elements are customized with the icons depicted in Table 7.3.

Table 7.3: EPF Composer Customization

| EPF Composer | Compliance Information | Suggested Icons |
|---|---|---|
| Reusable Asset | Rule Set |  |
| Concept | Compliance Effect |  |
| Custom category | Standard requirement |  |

Second, we provide mechanisms to model and annotate the process with compliance effects. This mechanism includes the creation of three plugins in a similar way as presented in Section 7.2.2. The first plugin captures *standard's requirements* by using the customized SPEM 2.0 elements presented in Table 7.3. The second plugin captures the *process elements* required to support the software process description. The third plugin captures *the annotated process*, in which compliance effects are added (in the guidance part) to the process tasks that shows adherence to the rules that they represent. Process tasks are an extended copy of the tasks defined in the plugin that contains the process elements (the extension is done by using a *contributes* relationship to the original ones). The delivery process and its corresponding activity diagram are created with the annotated tasks. Finally, we export our three plugins.

## 7.4 Modeling and Annotating a Small Example from ISO 26262

In this section, we apply the mechanism to model and annotate software process using EPF Composer (described in Section 7.3) by modeling a simple example from ISO 26262. Initially, we create the plugin for capturing standard's requirements. For this, we define a custom category root called *Standard Requirements ISO 26262 Software Unit Design*, to which we associate the requirements presented in Table 11.2, with a short but descriptive name, in a nested list of custom categories. Then, we create the rules that are associated to the requirement. For example, R3 is a requirement

that has two associated rules r3.1 and r3.2. Then, we associate to the rules the corresponding compliance effects, which are presented in the precedent and consequent of the rules described in RuleSet 1. The customized list of standard's requirement, the rules and compliance annotations are depicted in Figure 7.2.

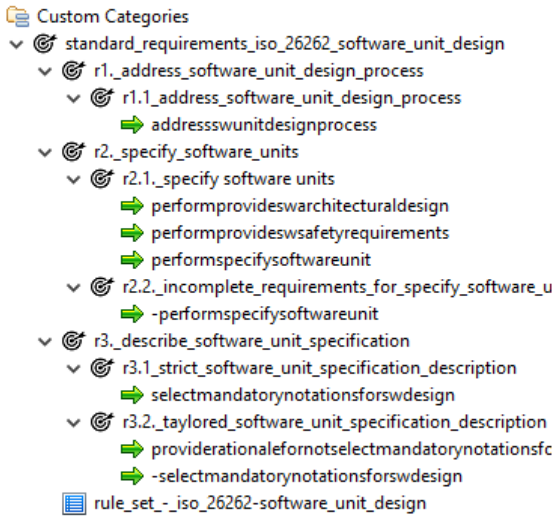


Figure 7.2: Requirements and their Associated Elements.

The actual rule is written in the *main description* field of the compliance effect (See Figure 7.3).

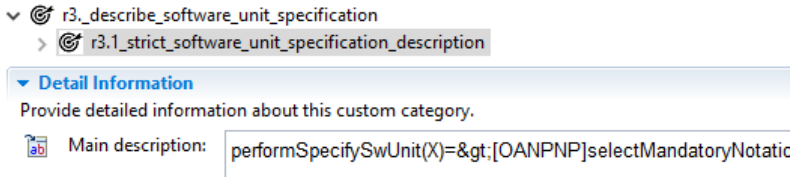


Figure 7.3: Specification of Rule r3.1

The rule set is defined in a customized *reusable asset* (called *Rule Set-ISO 26262-Software Unit Design*), which contains the superiority relations between rules (See Figure 7.4).

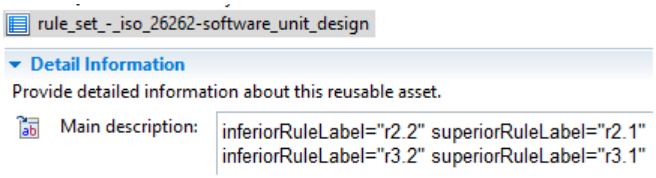


Figure 7.4: Rule Set Specification

Then, we create the plugin for capturing process elements. In this example (See Figure 7.5), the definition of the process elements is based on our interpretation of the requirements provided in Table 11.2. From R1, we deduce that there is one task called *Start Software Unit Design Process* in which the requirements for the process are collected, namely, the *Software Architectural Design* and the *Software Safety requirements*, which are work products resulting from previous phases. From R2 we deduce the existence of the task *Specify Software Unit* and from R3, we deduce that we have a task called *Design Software Unit* which output is the *Software Unit Design*.

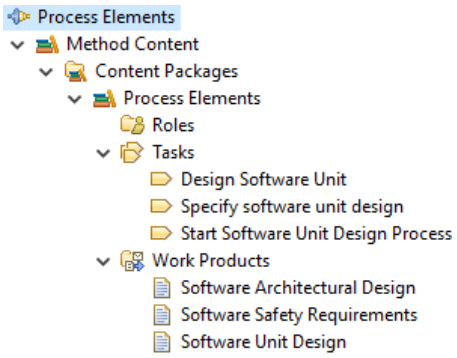


Figure 7.5: Process Elements Plugin.

Finally, we create the third plugin, in which we copy and extend (with *contributes*) the tasks that are part of the delivery process. Then, we annotate the tasks by deducing the compliance effects that they produce. For example, the task *Start Software Unit Design Process*, produces the compliance effect *addressSoftwareUnitDesignProcess*, since with this task we initiate addressing the process. This task has two inputs, i.e., *the software safety*

*requirements* and *the architectural design*. Thus, it also produce the compliance effects *performProvideAssociatedSwSafetyRequirements* and *performProvideSwArchitecturalDesign*. The annotation can be seen in the section called *Concepts* (See Figure 7.6).

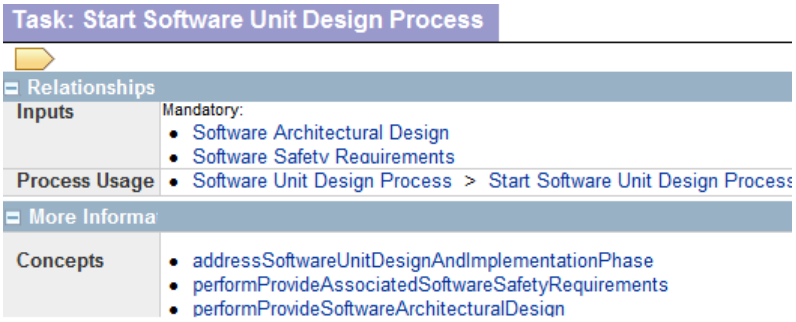


Figure 7.6: Start Software Unit Design Process Task

Then, we create the delivery process, in which the annotated tasks (storage in the method content of the plugin) are used to describe the breakdown structure and the activity diagram. Once created, we export the plugins. We get two files that we briefly describe (for space reasons, we do not provide code). First, we get an XMI file (usually called *diagram*), which contains enough elements for defining the process execution semantics required by Regorous and described in the activity diagram (See Figure 7.7).

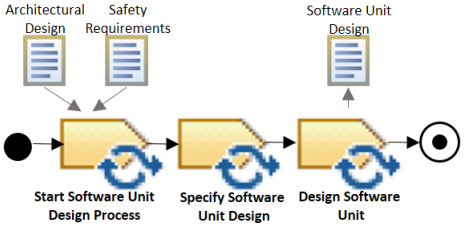


Figure 7.7: Activity Diagram of the Software Unit Design Process.

The elements of interest are an *Activity* that provides the name of the process, an *inital node* and a *final node* that represent the start and the end event respectively, one *Activity parameter node* for every task and one control flow for every sequence. We did not model other process elements in this

example, but the file can also provides a *decision, merge, fork and join nodes* for modeling exclusive and parallel gateway respectively, which can be useful for complex processes. Second, we get an XML file, which provides the compliance annotated process information (see Table 7.4). As the table shows, the activity name corresponds to the process name. Tasks has associated concepts which correspond to the compliance effects. We can also create the rule set since every concept is described with the actual rule and the reusable asset with the superiority relations (not represented in the table for space reasons).

Table 7.4: Process Description

| **Element** | **Information** |
|---|---|
| **Activity name** | Software Unit Design Process |
| **task use name** | Start Software Unit Design Process |
| -concept | addressSwUnitDesignProcess |
| | performProvideAssociatedSwSafetyRequirements |
| | PerformProvideSwarchitecturalDesign |
| **task usename** | Specify Software Unit Design |
| -concept | performSpecifySoftwareUnit |
| **task use name** | Design Software Unit |
| -concept | selectMandatoryNotationsForSwDesign |

The process previously described is manually modeled in Regorous and checked for compliance. To identify how back-propagation of standard's requirements violations should be done, we have purposely introduced a fault in the description, i.e., the compliance annotation *selectMandatoryNotationsForSwDesign* was eliminated. Regorous report is presented in Table 7.5. As expected, Regorous reports the compliance violation that occurs in the process model. Specifically, it says that the

Table 7.5: Regorous Report

| |
|---|
| **Compliance Check Results:** Process is non-compliant.<br>**Description:** Unfulfilled obligation to 'selectMandatoryNotationsForSwDesign' (Achievement, non-preemptive, non-persistent).<br>**Element name:** Specify Software Unit. |

obligation *selectMandatoryNotationsForSoftwareDesign* is unfulfilled. Tracing back (manually) this compliance effect in the plugin that describes the standard information (see Figure 7.2), we can find that the violation is related to the rule r3.1 which is related to the requirement R3. With this information, the process engineer can refer to the requirement description and understand how the process could be improved.

## 7.5 Related Work

To the best of our knowledge, there are no attempts for enabling compliance checking from SPEM2.0 process models. Apart from the IBM method presented in Section 7.2.2, works related to **mapping regulations** have been proposed to facilitate process engineers work. In [12], the authors examine techniques to map a single taxonomy to multiple regulations. In [13], authors, use Semantics of Business Vocabularies and Rules (SBVR) to represent similarity between concepts from regulations and organization operational specifics concepts. **Ontological approaches to map regulations** can be seen in [14], in which the authors propose a metamodel (SafetyMet) that includes concepts and relationships for safety targeted for facilitating safety compliance. In [15], the authors present a method for mapping the information security knowledge of the French EBIOS (Expression des Besoins et Identification des Objectifs de Scurit - Expression of Needs and Identification of Security Objectives) standard and the German IT Grundschutz Manual to an OWL-DL security ontology. In our work, we based our mapping on SPEM 2.0 metamodel, and we do not design a specific ontology or schema to map standards concepts. The **usage of SPEM 2.0 elements to map standards** can be seen in [16], in which the concepts involved in the Capability Maturity Model Integration (CMMI) standard are mapped to SPEM 2.0. As in [16], we have used *Category* to classify standard's requirements. Approaches for **verifying software process models** are presented in [17] and [18]. In [17], BPMN is used to formally specify the software project management and the software process, to deploy and execute agile avionics software development process, adopting the idea of model checking to enable detection and elimination of inconsistencies in process interaction. However, this approach does not explicitly address the checking of software process model against safety standards. In [18], a validation of the process model is carried out with formal tools, specifically model-checkers available in the area of Petri nets. The validation consists of evaluating

process properties such as termination of the process, and process planning fulfillment (process constraints). This work is only conceptual, and no tool support is provided. In our case, we have provided tool support for our methodology in EPF Composer and determined compliance with Regorous.

## 7.6 Conclusion and Future Work

In this paper, we explained our compliance checking vision which consists of the combination of process modeling capabilities via SPEM 2.0 reference implementation, and compliance checking capabilities via Regorous. Then, we focus on the identification and exploitation of the appropriate (minimal set of) SPEM 2.0-like elements available in the selected reference implementation. We illustrated our vision by applying it to a simple example from ISO 26262. Also, we manually map the obtained model into the input model of Regorous to check compliance and show how a compliance report can help the process engineer to trace the unfulfilled requirements.

In future, we plan to add a rule editor to support the modeling of the FCL rules. Moreover, we plan to address the transformation required to convert the information provided by EPF Composer into the input format required by Regorous. Additionally, we plan to back-propagate the compliance report information produced by Regorous into the EPF Composer to facilitate the analysis work that the process engineer has to perform. From a validation perspective, we are aware that we are using a small academic example. For this reason, we plan to study complex use cases to further validate our approach.

## Acknowledgment

This work is supported by the EU and VINNOVA via the ECSEL JU project AMASS (No. 692474) [10]. We thank I. Ayala for her contribution on requirements modeling using customized elements in EPF Composer [11].

# Bibliography


[1] Castellanos Ardila, J.P., Gallina, B.: Towards Increased Efficiency and Confidence in Process Compliance. In: Systems, Software and Services Process Improvement (EuroAsiaSPI). Volume 748., Springer International Publishing (2017) 162–174

[2] Gallina, B., Ul Muram, F., Castellanos Ardila, J.: Compliance of Agilized (Software) Development Processes with Safety Standards: a Vision. In: 4th international workshop on Agile Development of Safety-Critical Software (ASCS). (2018)

[3] Object Management Group Inc.: Software & Systems Process Engineering Meta-Model Specification. Version 2.0. OMG Std., Rev (2008) 236

[4] Koudri, A., Champeau, J.: MODAL: A SPEM extension to improve co-design process models. International Conference on Software Process (2010) 248–259

[5] McIsaac, B.: IBM Rational Method Composer: Standards Mapping. Technical report, IBM Developer Works (2015)

[6] Governatori, G.: The Regorous approach to process compliance. In: IEEE 19th International Enterprise Distributed Object Computing Workshop. (2015) 33–40

[7] Koliadis, G., Ghose, A.: Verifying Semantic Business Process Models in Inter-operation. In: IEEE International Conference on Service-Oriented Computing. (2007) 731–738

[8] ISO 26262: Road vehicles Functional safety. (2011)

[9] The Eclipse Foundation.: Eclipse Process Framework (EPF) Composer 1.0 Architecture Overview. http://www.eclipse.org/epf/composer_architecture/ (2013)

[10] AMASS.: Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems

[11] ECSEL Research and Innovation actions (RIA) - AMASS: D6.5 Prototype for Cross/Intra-Domain Reuse (b) (2017)

[12] Cheng, C., Lau, G., Law, K.: Mapping regulations to industry-specific taxonomies. In: 11th international conference on Artificial intelligence and law . (2007) 59–63

[13] Sunkle, S., Kholkar, D., Kulkarni, V.: Toward better mapping between regulations and operational details of enterprises using vocabularies and semantic similarity. Complex Systems Informatics and Modeling Quarterly (5) (2015) 39–60

[14] De La Vara, J.L., Panesar-Walawege, R.: SafetyMet: A metamodel for safety standards. In: International Conference on Model Driven Engineering Languages and Systems. (2013) 69–86

[15] Fenz, S., Pruckner, T., Manutscheri, A.: Ontological mapping of information security best-practice guidelines. International Conference on Business Information Systems (2009) 49–60

[16] Portela, C., Vasconcelos, A., Silva, A., Sinimbú, A., Silva, E., Ronny, M., Lira, W., Oliveira, S.: A Comparative Analysis between BPMN and SPEM Modeling Standards in the Software Processes Context. Journal of Software Engineering and Applications **5**(5) (2012) 330–339

[17] Kingsbury, P., Windisch, A.: Modeling of Agile Avionics Software Development Processes through the Application of an Executable Process Framework. In: International Conference on Design and Modeling in Science, Education, and Technology. (2011)

[18] Bendraou, R., Combemale, B., Crégut, X., Gervais, M.: Definition of an executable SPEM 2.0. In: 14th Asia-Pacific Software Engineering Conference. (2007) 390–397

# Chapter 8

# Paper B: Transforming SPEM 2.0-compatible Process Models into Models Checkable for Compliance

Julieth Patricia Castellanos Ardila, Barbara Gallina and Faiz UL Muram.
In Proceedings of the 18th International SPICE Conference (SPICE-2018), Thessaloniki, Greece, October 2018.

**Abstract**

Manual compliance with process-based standards is time-consuming and prone-to-error. No ready-to-use solution is currently available for increasing efficiency and confidence. In our previous work, we have presented our automated compliance checking vision to support the process engineers work. This vision includes the creation of a process model, given by using a SPEM 2.0 (Systems & Software Process Engineering Metamodel)-reference implementation, to be checked by Regorous, a compliance checker used in the business context. In this paper, we move a step further for the concretization of our vision by defining the transformation, necessary to automatically generate the models required by Regorous. Then, we apply our transformation to a small portion of the design phase recommended in the rail sector. Finally, we discuss our findings, and present conclusions and future work.

## 8.1 Introduction

Claiming compliance with process-based standards requires that companies show, via the provision of a justification which is expected to be scrutinized by an auditor, the fulfillment of its requirements [1]. The manual production of this justification is time-consuming and prone-to-error since it requires that the process engineer checks hundreds of requirements [2]. A process-based requirement is checkable for compliance if there is information in the process that corroborate that the requirement is fulfilled [3]. This checking can be facilitated by using FCL (Formal Contract Logic) [4], a rule-based language that can be used to generate automatic support to reason from requirements and the description of the process they regulate. In our previous work [5], we have presented our automatic compliance checking vision (See Fig. 8.1). It consists of the combination of process modeling capabilities via SPEM 2.0 [6]-reference implementation, specifically by using EPF (Eclipse Process Framework) Composer [7], and compliance checking capabilities via Regorous [8], an FCL-based reasoning methodology, and tool.

In our vision, EPF Composer contributes with the appropriate (minimal set of) SPEM 2.0-compatible elements required by Regorous, which, in turn, produces a report that can be used to analyze and improve compliance.

In this paper, we define the transformation necessary (dotted line region shown in Fig. 8.1) to automatically generate the models required by Regorous, i.e., the FCL rule set, the structural representation of the process and the compliance effects annotations (cumulative interactions between process tasks that produce the desired global properties mandated by the standards [9]). Then, we apply our transformation to a small portion of the design phase recommended in the rail sector and discuss our findings.

The rest of the paper is organized as follows. In Section 11.2, we recall essential background information. In Section 8.3, we present the transformations specification for generating Regorous inputs. In Section 8.4, we illustrate the transformation with a small example from the rail sector. In Section 11.4.3, we discuss our findings. In Section 11.5, we discuss related work. Finally, in Section 11.6, we derive conclusions and future work.

## 8.2 Background

In this section, we provide basic information on which we base our work.

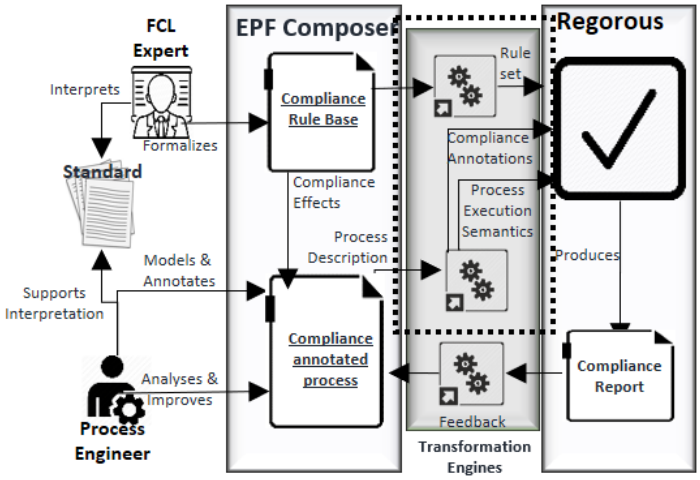


Figure 8.1: Automated Compliance Checking Vision [5].

### 8.2.1 EPF Composer

EPF Composer [7] is an open-source tool aiming at supporting the modeling of customizable software processes. We recall two open source standards used by EPF Composer and also required in this paper. **UMA** (Unified Method Architecture) Metamodel [10], a subset of SPEM 2.0 [6], is used to model and manage reusable method content and processes. Method Content defines the core elements used in a process, i.e., *tasks*, *work products* and *roles*. Managed Content defines textual descriptions, such as *Concept* and *Reusable Asset*. *Custom Category* defines a hierarchical indexing to manage method content. A *delivery process* describes a complete and integrated approach for performing a specific project and it contains a *Breakdown Structure*, which allows nesting of tasks. **UML 2.0 Diagram Interchange Specification** [11] supports diagram interchange among modeling tools by providing an UML activity diagram representation. An *Activity* corresponds to a process, while a *Node* represents a point in the process, and an *Edge* is used to connects points. Nodes can be of different types. An *Activity Parameter Node* represents a task. *Initial and Final Nodes* represent the start and the end of the process. *Fork and Join Nodes* represent the parallel flows and *Decision and Merge Nodes* represent conditional behavior.

### 8.2.2 Regorous

Regorous [8] is a tool-supported methodology for compliance checking in which the compliance status of a process is provided with the causes of existing violations. To check compliance, Regorous requires a **rule set**, which is the formal representation of the standard's requirements in Formal Contract Logic (FCL) [4]. An FCL rule has the form $r : a_1, ..., a_n \Rightarrow c$, where $r$ is the unique identifier, $a_1, ..., a_n$, are the conditions of the applicability of a norm and $c$ is the normative effect. The different kind of normative effects can be found in [4]. A rule set is represented in the schema called *Combined Rule Set* from which we recall some elements. *Vocabulary* contains an element called *term*, which attribute *atom* is used to describe rule statements. The second element, called *Rule*, is used to define every rule of the logic. A rule is specified with the unique identifier called *label*, the description of the rule called *control objective*, and the actual rule called *formal representation*. Regorous current implemented tool uses the **Canonical Process Format (CPF)** [12], a modeling language agnostic representation that only describes the structural characteristics of the process. A *Canonical Process* is the container of a set of *Nets* which represent graphs made up of *Nodes* and *Edges*. Nodes types can be *(OR, XOR, AND) Splits/Joint*, which capture elements that have more than one incoming/outgoing edge. Nodes can also represent *Tasks* and *Events*, which are nodes that have at most one incoming/outgoing edge. The **compliance effect annotations**, which represents the fulfillment of a rule on a process element, are captured in Regorous by using a schema called *Compliance Check Annotations*. A *ruleSetList* contains the *ruleSets uri* which is the identification of the rule set. The *conditions* and the *taskEffects* represent the process sequence flow and the tasks respectively and have an associated *effects name* which corresponds to its actual compliance effects annotation.

### 8.2.3 Automatic Compliance Checking Vision: The Modeling Part

In this section, we recall the methodology used for modeling the SPEM 2.0-compatible models in EPF Composer required by Regorous. The methodology is explained with an example from ISO 26262 presented in [5]. The modeled requirement is obtained from part 6 clause 8, number 8.1, which states: *"Specify software units in accordance with the architectural design*

*and the associated safety requirements"*. The formal representation of this requirement is presented in Equation 10.6.

$$\begin{aligned} r2.1 : addressSwUnitDesignProcess \Rightarrow [OANPNP] - performSpecifySwUnit \\ r2.2 : performProvideSwArchitecturalDesign, \\ performProvideSwSafetyRequirements \\ \Rightarrow [P]performSpecifySwUnit \\ r2.2 > r2.1 \end{aligned} \tag{8.1}$$

The modeling in EFP Composer required the creation of three plugins. Initially, we create a plugin for capturing standard's requirements (See Fig. 8.2), which contains not only their description in natural language e.g., *R2*, but also its atomization e.g., *r2.1* and *r2.2*. The requirement atomization is used to assign the rule representation (See Equation 10.6). A second plugin is used to capture process elements, as depicted in Fig. 8.3.

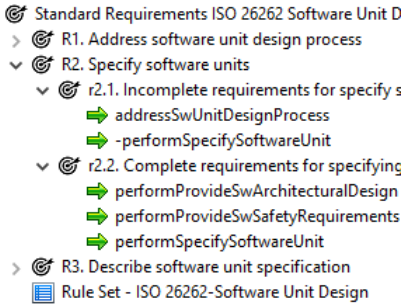


Figure 8.2: Requirements Plugin.

Finally, a third plugin is used to capture the compliance annotated tasks, in which we also create the delivery process and its corresponding activity diagram (See Fig. 8.4). To annotate the tasks, the concept that represents the compliance effects is added to the task. The reader can discover more details about the previous modeling in [5].

### 8.2.4 CENELEC EN 50128

CENELEC EN 50128 [13] is a standard that prescribes requirements for the development, deployment, and maintenance of safety-related software for railway control and protection application. The software component design

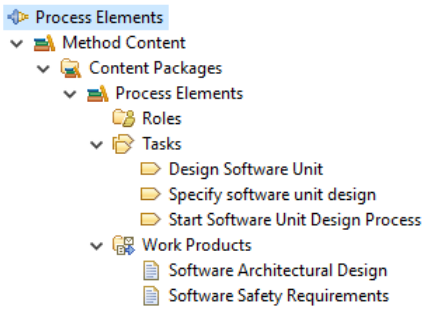


Figure 8.3: Process Elements Plugin.

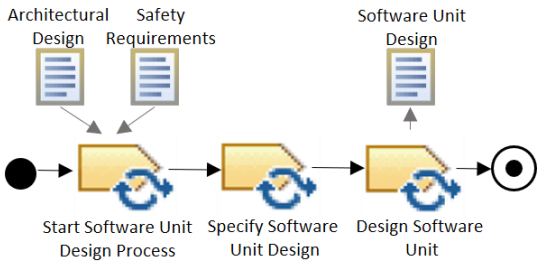


Figure 8.4: Process Activity Diagram.

phase is part of the lifecycle required by the software quality assurance, which states that the quality concerning the lifecycle shall address activities and tasks consistent with the plans (e.g., the safety plan). We recall some requirements corresponding to the software component design phase in Table 8.1.

Table 8.1: Requirements from the Rail Standard.

| ID | Description |
|---|---|
| R1 | Initiate component design phase. |
| R2 | Input documents: Software Design Specification. |
| R3 | A software component design shall be written under the responsibility of the designer. |

## 8.3 Generating Regorous Inputs

In this section, we present the two steps required to generate Regorous inputs, namely the **mapping** between the elements provided by EPF Composer and required by Regorous, and their **algorithmic solution**. We start with the mapping of the elements required for creating the **rule set**. As presented in Table 8.2, the information related to the rules is obtained from the *Delivery Process* provided by EPF Composer (described with UMA elements), and should conform to the Regorous schema called *Combined Rule Set*. Then, in Table 8.3, we present the mapping required for the process structure, which is provided in a UML activity diagram and required to be transformed to the canonical process (CPF). Finally, the *compliance effects annotations* require a structure that complies to the Regorous schema called *Compliance Check Annotations*. This information can be retrieved from EPF Composer taking into account that the process elements can be extracted from the process structure (described with UML elements) and the compliance effects annotations can be extracted from the delivery process (described with UMA elements). The mapping is presented in Table 8.4.

Table 8.2: Mapping Elements from UMA to the Rule Set

| **UMA** | **Rule Set** | **Mapping Description** |
|---|---|---|
| Reusable Asset | Rule Set | Reusable Asset is used in EPF composer to storage the information related to the rule set. Therefore, its information is transformed into the rule set required by Regorus. The attributes transferred are *name*, *presentationName* and *briefDescription*. |
| Concept | Term | Concept is used in EPF Composer to storage the information related to the vocabulary used in the creation of the rules. Therefore its content is transformed into the vocabulary required in the rule set, specifically, each Concept is a Term. The attribute transferred is *name*. |
| Content Category | Rule | Content categories contain rules. Therefore, their content is transformed into the body of the rule. The attributes transferred are *name*, *presentationName*, and *briefDescription*. |

Table 8.3: Mapping Elements from UML Diagram to the Canonical Process

| **UML** | **CPF** | **Mapping description** |
|---|---|---|
| Activity | Canonical Process | The UML activity diagram is used in EPF Composer to describe the dynamics of the software process. Therefore, its information is transformed into a canonical process in CPF. The attribute transferred is *id*. |
| Initial Node | Start Event | The initial node of the activity diagram becomes a node with type start event in CPF. |
| Parameter Node | Task Type | Each parameter node in the activity diagram becomes a task type in the CPF. Attributes transferred are *id* and *name*. |
| Control Flow | Edge | Each control flow in the activity diagram becomes an edge in CPF. Attributes transferred are *id*, *name*, *source* and *target*. |
| Final Node | End Event | The final node in the activity diagram becomes an end event type in CPF. |
| Decision /Merge Node | XOR Split /Join | The decision/merge nodes in the activity diagram becomes an XORSplit/XORJoin Type in CPF. |
| Fork /Join Node | AND Split/Join | The fork/join nodes in the activity diagram becomes an ANDSplit/ANDJoin Type in CPF. |

The algorithmic solution for obtaining the rule set, which mapping is described in Table 8.2, is presented in Algorithm 1. The algorithm initiates with the description of its required input (DeliveryProcess), and the expected output (RuleSet). Then, the input is parsed with the function *getElemementsByTagName*, which searches the elements to be mapped, with the function *Map* to the output. The first element searched is the *uma:ReusableAsset*, which attribute name is mapped to the rules URI. Then, the algorithm searches for the elements *uma:ContentCategory*, which provides the attributes *id*, *controlObjective* and *formalRepresentation* of each rule. Algorithm 2, which maps the elements described in Table 8.3, takes as input the UML Activity Diagram and provide the Canonical Format. The function *getElementsByTagName* searches for every elements that describes process structure and maps it to their counterpart in CPF. The mapping of the

Table 8.4: Mapping from UMA/UML Metamodel to the Compliance Annotations

| **UML /UMA** | **Compliance Annotations** | **Mapping Description** |
|---|---|---|
| Reusable Asset | ruleSet | A reusable asset becomes a ruleSetList. The attribute transferred is the *name*. |
| edge | conditions | Each edge becomes a special element in the compliance annotations file called condition. The attribute transferred is the id. |
| node | Task Effects | Each node becomes a Task Effect. The attribute transferred is the id. Then, the id is also used to search for the concepts that should be converted into the compliance effects in the delivery process file. |
| Concept | Effect | Every concepts associated to the task is transferred to the Effect. The attribute transferred is the name. |

process structural elements requires a unique identifier that is generated internally each time the function *Map* is used. Algorithm 3 describes the solution for mapping the elements presented in Table 8.4. The required inputs are the UML Activity Diagram and the DeliveryProcess. The expected output is the ComplianceEffectsAnnotations. The algorithm searches in the delivery process the element tagged as *uma:ReusableAsset* and mapped it to the rule set. Similarly, the algorithm searches for the elements tagged as *uml:edge* and *uml:node* in the UML Activity Diagram and mapped them to the conditions and taskEffects respectively. The node id is used to search for the elements tagged as *uma:concept* in the DeliveryProcess, which is mapped to the effects.

## 8.4 Models Checkable for Compliance from the Rail Sector

The purpose of this section is to provide evidence that the models provided by EPF Composer, and transformed with our algorithm, are checkable for compliance with Regorous. The software process model to be checked for compliance is the one modeled in Fig. 8.4 (originally created for compliance

```
input  : DeliveryProcess
output: RulseSet
LoadFile (DeliveryProcess);
NodeReusableAsset←getElementsByTagName (uma:ReusableAsset);
Map (ruleSet←ReusableAsset);
conceptsList←getElementsByTagName (uma:Concept);
for i ← 0to getLength (ConceptsList) do
  | Map (Term.atom ←Concept.name)
end
contentCategoryList ← getElementsByTagName (uma:ContentCategory);
for j ← 0to getLength (contentCategoryList) do
  | ruleControlObjective←getAttribute (briefDescription);
  | if ruleControlObjective is not empty then
  |   | Map (rule←contentCategory)
  | end
end
```

**Algorithm 1:** Algorithm for Obtaining the Rule Set.

```
input  : UMLActivityDiagram
output: CanonicalFormat
LoadFile (UMLActivityDiagram);
NodeActivity←getElementsByTagName (uml:Activity) ;
Map (CanonicalProcess← NodeActivity);
nodesList←getElementsByTagName (uml:node);
for i ← 0 to getLength (nodesList) do
  | if nodeType=uml:ActivityParameterNode then
  |   | Map (TaskType←node)
  | end
  | if nodeType=uml:InitialNode then
  |   | Map (StatEvent←node)
  | end
  | if nodeType=uml:ActivityFinalNode then
  |   | Map (EndEvent←node)
  | end
  | if nodeType=uml:ForkNode then
  |   | Map (ANDSplitType←node)
  | end
  | if nodeType=uml:JoinNode then
  |   | Map (ANDJoinType←node)
  | end
  | if nodeType=uml:DecisionNode then
  |   | Map (XORSplitType←node)
  | end
  | if nodeType=uml:MergeNode then
  |   | Map (XORJoinType←node)
  | end
end
edgesList←getElementsByTagName (uml:edge);
for j ← 0to getLength (edgeList) do
  | Map (Edge←edge)
end
```

**Algorithm 2:** Algorithm for Obtaining the Process Structure.

```
input  : UMLActivityDiagram,DeliveryProcess
output: ComplianceEffectsAnnotations
LoadFile (UMLActivityDiagram, DeliveryProcess) ;
NodeReusableAsset (from DeliveryProcess)←getElementsByTagName (uma:ReusableAsset) ;
Map ((ruleSet←ReusableAsset) ;
edgesList(from UMLProcess)← getElementsByTagName (uml:edge);
for i ← 0 to getLength (edgeList) do
  | Map (conditions←edge)
end
nodeList(from UMLProcess)← getElementsByTagName (uml:node);
for j ← 0 to getLength (nodeList) do
  | Map (taskEffects←node) TaskId←ObtainUMAValue(nodeList);
  | ContentElementList(from DeliveryProcess)← getElementsByTagName(ContentElement);
  | for k ← 0 to getLength (ContentElementList) do
  |   | if ContentElementList.id = TaskId then
  |   |   | ConceptsList(from DeliveryProcess)← getElementsByTagName(Concept);
  |   |   | for l ← 0 to getLength (ConceptsList) do
  |   |   |   | Map (effects←Concept);
  |   |   | end
  |   | end
  | end
end
```

**Algorithm 3:** Algorithm for Obtaining the Compliance Effects Annotations.

with an automotive standard). In this evaluation, three steps are required. Initially, we generate the compliance annotated software process in EPF Composer, following the methodology described in Section 8.2.3. Second, we apply the transformation described in Section 8.3. Finally, we verify that the models generated have enough information to be processed by Regorous. This verification is done manually, namely, we highlight the mapping of the elements required for checking compliance. We also check compliance with Regorous and describe the type of analysis that can be carried out after compliance checking.

We start by annotation a small portion of the design phase (modeled in Fig. 8.4) with the recommended requirements provided in the rail sector (see CENELEC requirements in Section 8.2.4). First, we formalize the standard's requirements applying the definitions for creating the rules presented in Section 8.2.2. As the formula 8.2) shows, the rule r1.1, which is the formalization of the requirement R1, defines an obligation of addressing the phase. Rules r.2.1 and r.2.2 are related to the requirement R2 in the following way: r.2.1 prohibits the specification of the design, but r.2.2 permits the specification of software units if the software design specification is obtained. Similarly to r.3.1 and r.3.2, which are related to requirement R3. Rule r.3.1 prohibits the production of software units, but r.3.2 permits them if not only

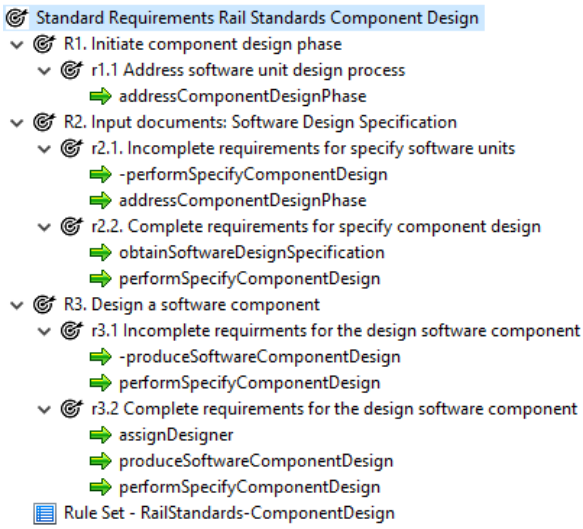


Figure 8.5: Requirements Plugin.

the specification is performed but also is a designer has been assigned. In the previous rules, priority relations are defined to give higher priority to the permits over the obligations.

$$\begin{aligned} r1.1 &: [OM] addressComponentDesignPhase \\ r2.1 &: addressComponentDesignPhase \\ \Rightarrow [OANPNP] &- performSpecifyComponentDesign \\ r2.2 &: obtainSoftwareDesignSpecification \\ \Rightarrow [P] &performSpecifyComponentDesign \\ r3.1 &: performSpecifyComponentDesign \\ \Rightarrow [OANPNP] &- produceSoftwareComponentDesign \\ r3.2 &: performSpecifyComponentDesign, assignDesigner \\ \Rightarrow [P] &produceSoftwareComponentDesign \\ & r2.2 > r2.1 \\ & r3.2 > r3.1 \end{aligned} \tag{8.2}$$

Standards requirements and the respective rules are modeled in EPF Composer in a plugin as depicted in Fig. 8.5.

Then, we import the plugin that contains the process elements (See Fig. 8.3). Finally, we create the plugin for annotating the process tasks. In this

plugin, we copy the tasks from the plugin that contains the process elements and make them *contribute* to the original ones, which allows to extend them in an additional way. The tasks are annotated according to the compliance effects they represent. For this, we check the process model depicted in Fig. 8.4. As we see, the task *Start Software Unit Design Process* represents the initiation of the software component design and therefore it produces the compliance annotation *addressComponentDesignPhase*. This task also responds to the compliance effect *obtainSoftwareDesignSpecification* since it has a work product with a similar name. Task *Specify Software Units* responds to the compliance effect *performSpecifyComponentDesign*. Finally, the task *Design Software Unit* has a work product *Software Unit Design*, which makes the task respond to the compliance effect *produceSoftwareComponentDesign*. Once the tasks are annotated, we create the delivery process and the activity diagram, export the plugins and apply the transformations to obtain the Regorous inputs to check compliance.

In what follows, we provide essential code snippets, in which we highlight the mapping of the elements required for checking compliance. We start showing the generated *Rule Set*. As presented in Listing 8.1, the generated *Rule Set* has the elements *Vocabulary*, which contains the rules, described in EPF Composer with an *uma:concept*. It also contains the *rules*, which were described in the content category elements that correspond to the rules.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RuleSet xmlns="http://www.nicta.com.au/bpc/CombinedRuleSetDefinition/0.1"
      uri="RuleSetRailStandards" >
   <Vocabulary>
      <Term atom="addressComponentDesignPhase"/>
      ...<!--other Term atoms -->
   </Vocabulary>
   <Rules>
      <Rule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:type="DflRuleType" ruleLabel="r1.1">
         <ControlObjective>r1.1 Address software unit design process</
               ControlObjective>
         <FormalRepresentation>=&gt;[OANPP]addressComponentDesignPhase</
               FormalRepresentation>
      </Rule>
      ...<!--other rules -->
   <SuperiorityRelations>
      ...
   </SuperiorityRelations>
</RuleSet>
```

Program 8.1: Rule set generated

In Listing 8.2, we present the generated process structure. We highlighted one *Node* that represents the start point of the process and one node that

represents a task Type. An *Edge* represents a connection between the nodes.

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
<ns4:CanonicalProcess name="Software Unit Design Process" ...>
   <Net id="1529072497607">
      <Node id="1529072497608" xsi:type="ns4:EventType" xmlns:xsi="http://
           www.w3.org/2001/XMLSchema-instance">
         <name>Start</name>
         <attribute value="starteventl" typeRef="Id"/>
      </Node>
      <Node id="1529072497609" xsi:type="ns4:TaskType" xmlns:xsi="http://
           www.w3.org/2001/XMLSchema-instance">
         <name>Start Software Design Process</name>
         <attribute value="StartSoftwareDesignProcessID" typeRef="Id"/>
      </Node>
      ...<!--other nodes -->
      <Edge id="1529072497612" targetId="1529072497609" sourceId="
           1529072497608" default="false">
      ...<!--other edges -->
   </Net>
</ns4:CanonicalProcess>
```

Program 8.2: Process structure generated

In Listing 8.3, we present the compliance annotations. For example, the *rule set uri* is the rule set identification, *conditions element id* represent control flows identification, and the *taskEffects* represent the tasks, which *effects name* corresponds to the effects.

```
<?xml version="1.0" encoding="ASCII"?>
<cca:ComplianceAnnotations xmi:version="2.0" xmlns:xmi="http://www.omg.org
     /XMI" xmlns:cca="http://www.nicta.com.au/bpc/eclipse/
     ComplianceCheckAnnotations">
<ruleSetList>
   <ruleSets uri="RuleSetRailStandards"/>
</ruleSetList>
   <conditions elementId="_jNj1AExVEeiW4M4duzOA6Q"/>
   <conditions elementId="_jukQUExVEeiW4M4duzOA6Q"/>
   ...<!--other conditions-->
   <taskEffects elementId="_hCKUcExVEeiW4M4duzOA6Q">
      <effects name="addressComponentDesignPhase"negation="false">
      <effects name="obtainSoftwareDesignSpecification"negation="false">
   ...<!--other taskEffects -->
   <localVocabulary/>
</cca:ComplianceAnnotations>
```

Program 8.3: Compliance annotations generated

Then, we checked compliance with Regorous. The report results (See Fig. 8.6) not only shows that the *process in non-compliant*, but also the description of the uncompliant situation, the element that may be the source of the violation, the rule that has been violated and the possible resolution. With this information, it may be easier for the process engineer to make a focused

analysis to improve the compliance status. In the example, the rule 3.1 (highlighted in Fig. 8.5), refers to *Incomplete requirements for the design of software Components*, which means that we do not have the requirements in place to address the task called *Specify Software Unit Design*. To solve the uncompliant situation, we refer to the counterpart rule, which is the one marked as *r.3.2*, in which the compliance effects *assign designer* and *produceSoftwareComponentDesign* and *performSpecifyComponentDesign* are included. To be able to complete the assignment of these effects, we need to include a role called *designer* to the task *Specify Software Unit Design* as presented in Fig. 8.7. The improved process is again checked, resulting in a report with no violations of the rules.

**Compliance Check Results:** Process is non-compliant.

**Description:** Unfulfilled obligation to 'Not produceSoftwareComponentDesign'.

**Element name:** Specify Software Unit.

**Rule label:** r3.1.

**Possible resolutions:** Prevent violation by performing 'Not produceSoftwareComponent-Design' after 'Specify Software Unit Design'.

Figure 8.6: Compliance Report.

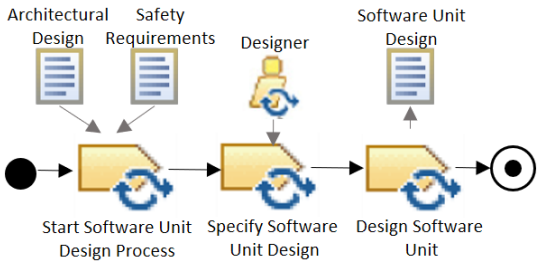


Figure 8.7: Activity Diagram.

## 8.5 Discussion

Automated compliance checking of software processes with Regorous generates a compliance report that not only communicate the compliance status of the software process, i.e., whether the process is compliant or not, but also the sources of violations, i.e, the rules that have being violated and the target of the uncompliant situations (specific tasks), and possible resolutions. This information may increase **efficiency** in the process compliance since it permits the process engineer to focus on specific process elements and the reparation policies they may require. In the example presented in Section 8.6, it was clear, from the compliance report, that the task affected was *Specify Software Unit* and we focus on it to understand the missing process elements. If rules are correctly formalized, and their formalization covers the standards requirements entirely, also **confidence** can be increased since uncompliant situations, in all the levels, would be spotted. Since we have modeled in detail the requirements provided in Table 8.1, we can consider the checking of the small process reliable. A software process can be checked for compliance with different standards. This specific aspect could potentially be beneficial since it promotes **process reusability**, i.e., a process engineer can take processes designed in previous projects, check their compliance status with the normative requirements of the new project and improve it, based on the violations reported. In the example, we saw that the software process model created for automotive could be used as a base for model a small portion of the design phase recommended in the rail sector.

As we see in Fig. 8.1, the adopted methodological approach for our automatic compliance checking vision, is tool supported. While the **maturity** of the methodology is high, its tool support still requires additional work. EPF Composer and Regorous have been tested separately and the bridge between them, namely, the transformations between the EPF Composer and Regorous, have been designed and implemented. The transformations, applied to the portion of the design phase recommended in the rail sector (See Section 8.4), are *correct* since they have generated a complete set of inputs that are compatible with Regorous schemas, making possible to check compliance. The transformation implementation, which is still in a prototyping stage, could be improved if techniques, such as Model Driven Engineering (MDE) are applied. We consider essential to further exploit the process modeling language agnosticism underlying Regorous methodology to be able to perform a future seamless integration of the tools required for our compliance checking vision.

## 8.6 Related Work

Automatic compliance checking of processes is one of the mechanisms that can provide benefits, as we have discussed above, to compliance management. In particular, researchers in the business and legal compliance context have explored potential formalisms to create compliance checking frameworks, such as the ones presented in [14] and in [15]. However, they are based on temporal logics, in which the modeling of normative requirements is still considered difficult. To model the rules more naturally, we have chosen Regorous, which underlying formalism called FCL, permits the modeling of deontic notions (i.e., obligations, prohibitions and permissions) which are the actual notions that describe normative requirements. Automatic compliance checking of safety-critical software processes has not been as explored as in business management. However, in [16], the authors presented initial steps of an approach for process reasoning and verification, which is based on the combination of Composition Tree Notations (CTN), a high-level modeling notation used for modeling process structure, and Description Logics (DL). DL is used to reason about the compliance of the process structure. Instead, our approach includes the accumulation of compliance effects that trigger new effects, focusing on the process behavior. Another difference we have included in our approach is the use of SPEM 2.0-compatible software process models, which may be preferred over other process modeling languages since it allows the creation of process method contents that can be reused in different kind of processes. SPEM 2.0-related community, to the best of our knowledge, has not addressed compliance checking. However, based on SPEM 2.0, some solutions for compliance management exists. In [3], compliance tables are generated. Compliance tables require the modeling of the standard's requirements, which should be mapped to the process elements that fulfill them. The modeling of compliance elements is also exploited in [17], in which the modeling of standards requirements is required to detect whether the process model contains sufficient evidence for supporting the requirements. The approach provides feedback to the safety engineers regarding detected fallacies and recommendations to solve them. In our case, we have also exploited not only the modeling of standard requirements, but also we have provided a mechanism to include rules within the standard's requirements, which facilitate the resolution of uncompliant situations after the automatic compliance checking is performed.

## 8.7 Conclusions and Future Work

In this paper, we defined the transformation necessary to automatically generate the models checkable for compliance in Regorous from SPEM 2.0-compatible process models. We also applied our transformation to a small portion of the software component design phase recommended in the rail sector and discussed aspects related to our findings.

To increase the maturity of the results shown in this paper, a proper plugin is going to be implemented to enable the push-button solution for the entire generation of the inputs required by Regorous. Also, as presented in [5], we need to further validate our approach and complete some tasks, i.e., the addition of the rule editor to facilitate the modeling of FCL rules, which currently is done manually, and the mechanism to back-propagate compliance results into EPF Composer. This work is expected to be partly delivered within the final release of the AMASS platform [18].

## Acknowledgment

This work is supported by the EU and VINNOVA via the ECSEL JU project AMASS (No. 692474) [19].

# Bibliography


[1] Gallina, B., Ul Muram, F., Castellanos Ardila, J.: Compliance of Agilized (Software) Development Processes with Safety Standards: a Vision. In: 4th international workshop on Agile Development of Safety-Critical Software (ASCS). (2018)

[2] Castellanos Ardila, J.P., Gallina, B.: Towards Increased Efficiency and Confidence in Process Compliance. In: Systems, Software and Services Process Improvement (EuroAsiaSPI). Volume 748., Springer International Publishing (2017) 162–174

[3] McIsaac, B.: IBM Rational Method Composer: Standards Mapping. Technical report, IBM Developer Works (2015)

[4] Governatori, G.: Representing business contracts in RuleML. International Journal of Cooperative Information Systems. (2005) 181–216

[5] Castellanos Ardila, J.P., Gallina, B., Ul Muram, F.: Enabling Compliance Checking against Safety Standards from SPEM 2.0 Process Models. In: Euromicro Conference on Software Engineering and Advanced Applications (SEAA). (2018)

[6] Object Management Group Inc.: Software & Systems Process Engineering Meta-Model Specification. Version 2.0. OMG Std., Rev (2008) 236

[7] The Eclipse Foundation.: Eclipse Process Framework (EPF) Composer 1.0 Architecture Overview. http://www.eclipse.org/epf/composer_architecture/ (2013)

[8] Governatori, G.: The Regorous approach to process compliance. In: IEEE 19th International Enterprise Distributed Object Computing Workshop. (2015) 33–40

[9] Koliadis, G., Ghose, A.: Verifying Semantic Business Process Models in Inter-operation. In: IEEE International Conference on Service-Oriented Computing. (2007) 731–738

[10] IBM Corporation: Key Capabilities of the Unified Method Architecture (UMA)

[11] Object Management Group: UML 2 . 0 Diagram Interchange Specification. (2003)

[12] La Rosa, M., Reijers, H., van der Aalst, W., Dijkman, R., Mendling, J., Dumas, M., García-bañuelos, L.: APROMORE: An advanced process model repository. Expert Systems With Applications (2011) 7029–7040

[13] EN50128, B.: Railway applications-Communication, signaling and processing systems Software for railway control and protection systems. British Standards Institution (2011)

[14] Elgammal, A., Turetken, O., van den Heuvel, W., Papazoglou, M.: Formalizing and applying compliance patterns for business process compliance. Software and Systems Modeling. (2016) 119–146

[15] El Kharbili, M.: Business Process Regulatory Compliance Management Solution Frameworks: A Comparative Evaluation. In: 8th Asia-Pacific Conference on Conceptual Modelling. (2012) 23–32

[16] Kabaale, E., Wen, L., Wang, Z., Rout, T.: Representing Software Process in Description Logics: An Ontology Approach for Software Process Reasoning and Verification. In: Software Process Improvement and Capability Determination., Springer (2016) 362–376

[17] Ul Muram, F., Gallina, B., Gomez Rodriguez, L.: Preventing Omission of Key Evidence Fallacy in Process-based Argumentations. In: 11th International Conference on the Quality of Information and Communications Technology (QUATIC). (2018)

[18] AMASS Platform: https://www.polarsys.org/opencert/

[19] AMASS.: Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems

# Chapter 9

# Paper C: Formal Contract Logic Based Patterns for Facilitating Compliance Checking against ISO 26262

Julieth Patricia Castellanos Ardila and Barbara Gallina.
In Proceedings of the 1st Workshop on Technologies for Regulatory Compliance (TeReCom-2017), Luxembourg, Luxemburg, December 2017.

**Abstract**

ISO 26262 demands a confirmation review of the safety plan, which includes the compliance checking of planned processes against safety requirements. Formal Contract Logic (FCL), a logic-based language stemming from business compliance, provides means to formalize normative requirements enabling automatic compliance checking. However, formalizing safety requirements in FCL requires skills, which cannot be taken for granted. In this paper, we provide a set of ISO 26262-specific FCL compliance patterns to facilitate rules formalization. First, we identify and define the patterns, based on Dwyer' et al.'s specification patterns style. Then, we instantiate the patterns to illustrate their applicability. Finally, we sketch conclusions and future work.

## 9.1 Introduction

Safety critical systems designers rely on safety standards, which embody the public consensus of acceptable risk [1]. Particularly, the automotive industry has adopted the functional safety standard ISO 26262 [2], which guides the development of the safety-related systems included in a specific class of road vehicles. To claim compliance with ISO 26262 from a process perspective, necessary pieces of evidence are: *the safety plan*, which is used to manage the execution of safety activities, as well as the corresponding *confirmation review*, which includes the compliance checking of planned processes against safety requirements. In [3, 4], we have identified that automatic compliance checking of safety processes involves the definition of a finite state model of the process, where normative safety requirements provides the permissible states of the process elements. This task can be supported with available on the shelf tools. In particular, Formal Contract Logic (FCL) [5], a logic-based language stemming from business compliance, provides means to formalize normative requirements. However, formalizing safety requirements in FCL requires skills, which cannot be taken for granted. Patterns, which are *"abstractions from concrete forms which keeps recurring in specific non-arbitrary context"* [6], could represent a solution. In this paper, we follow Dwyer et al.'s specification patterns style [7], created to ease the formalization of systems requirements for finite state system model verification, to draw a general definition of safety compliance pattern. We also use specification patterns to identify and define a set of ISO 26262-specific FCL compliance patterns. The defined patterns are instantiated to illustrate their applicability. This work will contribute to AMASS project [8], in particular, it aims at support the compliance management vision proposed, in which automotive is one of the 11 domains involved [9].

The rest of the paper is organized as follows. In Section 11.2, we provide essential background. In Section 9.3, we provide our definition of safety compliance pattern as well as our identified and defined ISO 26262-related patterns. In Section 9.4, we instantiate the defined safety compliance patterns. In section 11.5, we present related work. Finally, in Section 9.6, we present conclusions and future work.

## 9.2 Background

This section recalls essential information required in our work.

### 9.2.1 ISO 26262

ISO 26262 [2] addresses functional safety in automotive. The safety process influences functional safety. Thus, a confirmation review of the safety plan, which includes the compliance checking of the planned process against safety requirements is mandatory. The safety process can be either strictly planned, i.e., including all the activities provided by the reference process, or flexibly planned, i.e., by tailoring activities (omitting/performing them differently) [10]. According to ISO 26262:2, if a safety activity is tailored, *a) the tailoring shall be defined in the safety plan* and *b) a rationale as to why the tailoring is adequate and sufficient to achieve functional safety shall be available*.

From a structure perspective, ISO 26262 is divided into parts, which are subdivided into clauses. Some clauses represent phases of the safety process, which also describe activities and tasks. ISO 26262 uses Automotive Safety Integrity Levels (ASIL), which are levels to specify item's necessary safety requirements. Alternative methods to use in the planning of safety activities (e.g., tables) have to be chosen according to the higher recommendation for the ASIL assigned, but if not, a rationale shall be given that the selected methods comply with the corresponding requirement. Disjoint alternatives are also included in the text of the normative requirements. Frequently recurring expressions, which can guide the reading of the standard, can also be found, e.g., *in accordance with*. In Table 11.2, we recall a subset of requirements from ISO 26262:6 clause 8, which specifies the software unit design and implementation phase.

Table 9.1: Requirements for ISO 26262:6 clause 8.

| ID | Ref | Description |
|---|---|---|
| R1 | 8 | The Software unit design and implementation phase start. |
| R2 | 8.1 | Specify software units in accordance with the architectural design and the associated safety requirements. |
| R3 | 8.2 | The detailed design will be implemented as a model or directly as source code. |
| R4 | 8.4.2 | The software unit design shall be described using specific notations, which are listed as alternative methods. |

### 9.2.2 Specification Patterns

The specification patterns, formulated by Dwyer et al.'s [7], are *"generalized descriptions of commonly occurring requirements on the permissible state sequence of a finite state model of a system."* A selected set of Dwyer et al.'s patterns is presented in Table 9.2. The reader may refer to [11] to see the complete set of patterns with their entire descriptions. Each pattern has a *scope*, which is the extent of the program execution over which the pattern must hold. The types of scope that we consider in this paper are: *global*, which represent the entire program execution, *before*, which includes the execution up to a given state, and *after* which includes the execution after a given state.

Table 9.2: Dwyer's specification patterns [7]

| Name | Description |
|---|---|
| **Absence** | A given state P does not occur within a scope |
| **Existence** | A given state P must occur within a scope |
| **Universality** | A given state P must occur throughout a scope |
| **Precedence** | A state P must always be preceded by a state Q within a scope |
| **Response** | A state P must always be followed by a state Q within a scope |

### 9.2.3 Formal Contract Logic

Formal Contract Logic (FCL) [5] is a language designed to formalize normative requirements. FCL is implemented in Regorous, a tool developed by Data61/CSIRO in Australia[1]. An FCL rule is represented as follows:

$r : a_1, ..., a_n \Rightarrow c$, where:

$a_1, ..., a_n =$ Conditions of the applicability of the norm.

$c =$ Normative effect.

If a rule has an empty antecedent, it represents the definition of a new term. Otherwise, it represents the triggering of deontic notions, i.e., *obligations*, situations to which the bearer is legally bounded, or that should avoid, and *permissions*. If something is permitted the obligation to the contrary does not

[1] `https://research.csiro.au/data61/regorous/`.

hold [12]. In the modeling of the rules, the normative effect requires a notation that clarifies the applicability of the norm (presented in Table 9.3). Thus, if an obligation has to be obeyed during all instants of the process

Table 9.3: FCL rule notations [12]

| Notation | Description |
|---|---|
| [P]P | P is permitted |
| [OM]P | There is a maintenance obligation for P |
| [OAPP]P | There is an achievement, preemptive, and non-perdurant obligation for P |
| [OANPP]P | There is an achievement, non-preemptive and perdurant obligation for P |
| [OAPNP]P | There is an achievement, preemptive and non-perdurant obligation for P |
| [OANPNP]P | There is an achievement, non-preemptive and non-perdurant obligation for P |

interval, it is called *maintenance obligation*. If achieving the content of the obligation at least once is enough to fulfill it, it is called *achievement obligation*. An achievement obligation is *preemptive* if it could be fulfilled even before the obligation is actually in force. Otherwise, it is *non-preemptive*. If the obligation persists after being violated, it is a *perdurant obligation*, otherwise is a *non-perdurant*. A binary relation between rules ($>$) allows handling rules with conflicting conclusions.

## 9.3 Safety Compliance Patterns Identification and Definition

This section introduces our definition of safety compliance pattern as well as our identified and defined ISO 26262-related compliance patterns.

### 9.3.1 Our definition of Safety Compliance Pattern

As recalled in the introduction, automatic compliance checking of safety process involves the definition of a finite state model of the process, where normative safety requirements provide the permissible states of the process elements. This statement allows us to think of a process as a kind of system that can be verified. Thus, we can translate the specification pattern definition (see Section 9.2.2) into our context as follows: ***safety compliance patterns*** *are patterns that describe commonly occurring normative safety requirements on the permissible state sequence of a finite state process model.* With this

definition, we can develop a mapping between specification patterns and safety compliance patterns, as follows: the presence of a state in a system can be interpreted as the state of the obligation imposed to an element in the process, and the scope corresponds to the interval in a process when the obligations formulated by the pattern are in force. In Section 9.3.2, we identify the safety compliance patterns extracted from ISO 26262.

### 9.3.2 ISO 26262-related Compliance Patterns Identification

For identifying a safety compliance pattern in ISO 26262, we have delineated five methodological steps. The first step consists of the selection of a **recurring structure** in the standard since, as recalled in Section 11.2.2, safety requirements in ISO 26262 have implicit and explicit structures. The second step is the description of the **obligation for compliance**, namely, the reasons why the structure is required for safety compliance. The third step is the **pattern description**, based on similar (or a combination of) behaviors of the patterns described by Dwyer et al.'s (see Table 9.2). This description is contextualized to safety compliance, based on the mapping presented in Section 9.3.1. In this step, we also assign a name for the safety pattern, which reflects the related obligation for compliance. The fourth step is the definition of the **scope** of the pattern, which we also base on Dwyer et al.'s work. The fifth step is the **formalization in FCL**. To formalize the pattern, the scope defined for the pattern requires being mapped into the rule notations provided by FCL. Therefore, a *global* scope, which represents the entire process model execution, can be mapped to *maintenance obligation*, which represents that an obligation has to be obeyed during all instants of the process interval. A *before* scope, which includes the execution of the process model up to a given state, can be mapped to the concept of *preemptive obligation*, which represents that an obligation could be fulfilled even before it is in force. An *after* scope, which includes the execution of the process model until a given state, can be mapped to the concept *non-preemptive obligation*, which represents that an obligation cannot be fulfilled until it is in force. It should be noted that, in safety compliance, it is possible to define exceptions for the rules. Therefore, if the obligation admits an exception, the part of the pattern that corresponds to the exception is described as a permission, since, as recalled in Section 9.2.3, if something is permitted the obligation to the contrary does not hold. The obligation, to which the exception applies, is modeled as non-perdurant, since the permission is not a violation of the obligation, and therefore the obligation does not persist after the permission is

granted. In this case, the obligation and a permission have contradictory conclusions, but the permission is superior since it represent an exception. These methodological steps have helped us to define an initial set of four ISO 26262 - related FCL compliance patterns, presented in Section 9.3.3. The description of our patterns has information related to the steps mentioned above. Therefore, the corresponding expressions in bold represent the elements of the pattern's description.

### 9.3.3 ISO 26262-related Compliance Patterns Definition

In what follows, we define our safety compliance patterns in ISO 26262.

**Pattern:** *Address Phase*. **Recurring structure:** A phase. **Obligation for compliance:** Every phase proposed by the safety model must be addressed. A phase can be omitted if tailoring is performed and a rationale is provided. **Pattern description:** *Universality* + *absence* - A phase must occur. Not addressing the phase requires its tailoring and the provision of a rationale. **Scope:** Global. **FCL mapping:** A maintenance obligation *address*{*Phase*} is triggered by a previous task {*optionalTriggeringObligation*}, which can be empty if the phase is checked for compliance in isolation from the other phases. The permission for not addressing the phase requires two antecedents, *tailor*{*Phase*} and *rationaleForOmiting*{*Phase*} (See Formula 10.1).

$$\frac{\begin{array}{r} r : \{optionalTriggeringObligation\} \Rightarrow [OM]address\{Phase\} \\ r' : tailor\{Phase\}, rationaleForOmiting\{Phase\} \Rightarrow [P] - address\{Phase\} \end{array}}{r' > r} \quad (9.1)$$

**Pattern:** *Perform Preconditions*. **Structure:** The structure implicit in the expression *in accordance with*. **Obligation for compliance:** A task is prohibited until the preconditions are performed. **Pattern description:** *Absence* + *precedence* - A given task cannot occur within a scope. The task is permitted to be performed if the preconditions are performed. **Scope:** After. **FCL mapping:** A rule triggered by a previous rule {*TriggeringObligation*} prohibits the performing of the task *perform*{*Task*}. The permission of performing *perform*{*Task*} is granted after the preconditions are fulfilled *perform*{*Preconditions*} (See Formula 10.2).

$$\frac{\begin{array}{r} r : \{TriggeringObligation\} \Rightarrow [OANPNP] - perform\{Task\} \\ r' : perform\{Precondition1\}, ..., perform\{PreconditionN\} \Rightarrow [P]perform\{Task\} \end{array}}{r' > r} \quad (9.2)$$

**Pattern:** *Select Disjoint Methods*. **Structure:** Structure implicit when the word *or* is used to list two methods. **Obligation for compliance:** Only one method can be selected from a list of two. **Pattern description:** *Existence* + *absence* - A given method can be selected within a scope. The presence of a second method derogates the selection of the first method. **Scope:** After. **FCL mapping:** A rule triggered by previous obligations $\{TriggeringObligation\}$ imposes the obligation of selecting a method $select\{Method1\}$. The selection of a second method $select\{Method2\}$, derogates the previous selection $select\{Method1\}$ (See Formula 10.3).

$$\begin{aligned} r : \{TriggeringObligation\} \Rightarrow [OANPNP]select\{Method1\}] \\ r' : select\{Method2\} \Rightarrow [P] - select\{Method1\} \\ r'>r \end{aligned} \tag{9.3}$$

**Pattern:** *Select alternative methods*. **Structure:** Alternative methods given in tables. **Obligation for compliance:** Methods should be selected according to ASIL/recommendation levels. Alternative methods can be selected if a rationale is provided. **Pattern description:** *Response* + *absence* - A given obligation has to occur. The provision of a rationale grants the permission to derogates the obligation. **Scope:** After. **FCL mapping:** A rule triggered by previous obligations $\{TriggeringObligation\}$ imposes the selection of methods according to the requirements $select\{mandatoryMethods\}$. The provision of the rationale is the permission that derogates the obligation (See Formula 10.4).

$$\begin{aligned} r : \{TriggeringObligation\} \Rightarrow [OANPNP]select\{mandatoryMethods\} \\ r' : provideRationaleForNotSelect\{mandatoryMethods\} \\ \Rightarrow [P] - select\{mandatoryMethods\} \\ r'>r \end{aligned} \tag{9.4}$$

## 9.4 ISO 26262-related Compliance Patterns Instantiation

In this section, we instantiate the patterns defined in Section 9.3.3, using the ISO 26262 requirements presented in Table 11.2.

Requirement R1, which defines the phase *software unit design and specification*, can be specified by using the pattern *Address Phase*. We assume

that the phase is checked in isolation from other phases (See Formula 10.5).

$$\begin{aligned} r_1 :\Rightarrow [OM]addressSwUnitDesignAndImplementation \\ r_1' : tailorSwUnitDesignAndImplementation, \\ rationaleForOmitingSwDesignAndImplementation \\ \Rightarrow [P] - addressSwUnitDesignAndImplementation \\ r_1' > r_1 \end{aligned} \tag{9.5}$$

Requirement R2 have the expression *in accordance with*, which can be represented with the pattern *Perform Preconditions*. Specifically, the software architectural design and the associated safety requirements are preconditions to specify the software units. We assume that the triggering rule is *addressSwUnitDesignAndImplementation* (See Formula 10.6).

$$\begin{aligned} r_2 : addressSwUnitDesignAndImplementation \\ \Rightarrow [OANPNP] - performSpecifySwUnit \\ r_2' : performProvideSwArchitecturalDesign, performProvideSafetyRequirements \\ \Rightarrow [P]performSpecifySwUnit\{Task\} \\ r_2' > r_2 \end{aligned} \tag{9.6}$$

Requirement R3 mentions the use of two disjoint implementation strategies, namely implementation as a model or directly as source code. Therefore, this requirement can be modeled using the pattern *Select Disjoint Methods*. We assume that the triggering rule is *implementingSwUnit* (See Formula 10.7).

$$\begin{aligned} r_3 : implementingSwUnit \Rightarrow [OANPNP]selectImplementingAsSourceCode(X) \\ r_3' : selectImplementingAsModel(X) \Rightarrow [P] - selectImplementingAsSourceCode(X) \\ r_3' > r_3 \end{aligned} \tag{9.7}$$

Requirement R4 refers to a table with alternative entries. This requirement can be represented by using the pattern *Select alternative methods*. We assume that the triggering rule is *performSpecifySwUnit* (See Formula 9.8).

$$\begin{aligned} r_4 : performSpecifySwUnit \Rightarrow [OANPNP]selectMandatoryNotationsforSwDesign \\ r_4' : provideRationaleForNotSelectMandatoryNotationsforSwDesign \\ \Rightarrow [P] - selectMandatoryNotationsforSwDesign \\ r_4' > r_4 \end{aligned} \tag{9.8}$$

## 9.5 Related Work

Patterns for the formal specification of system safety requirements are presented in [13]. These patterns consider the cases and the terminology used

in industrial automation systems to facilitate formal verification. Our patterns, instead, are restricted to process-centered requirements. Some works provide patterns for facilitating the formalization of the normative requirements for compliance checking in areas like business process compliance, e.g., the works presented in [14, 15] which extends Dwyer et al.'s specification patterns, and the work presented in [16], which uses REA (Resources, Events, and Agents) approach. A similar work, in the context of security, is presented in [17], which aims at providing a pattern structure for generating security policies for service-oriented architectures. In our work, as in some of the previously mentioned works, we use Dwyer et al.'s specification patterns as a base for providing our definition for safety compliance pattern. Also, we identified and defined the safety compliance patterns present in some structures of the standard ISO 26262. Moreover, our patterns are formalized in FCL, a formal language that is explicitly created for compliance checking, providing precise structures for modeling, e.g., deontic effects, which facilitate the expression of normative requirements in a more natural way.

## 9.6 Conclusion and Future Work

In this paper, we use Dwyer et al.'s specification patterns style to provide our definition of safety compliance patterns. Also, we identify and define set of ISO 26262-specific FCL compliance patterns, which were extracted from implicit and explicit recurring structures provided by ISO 26262. In the last part of our work, we have instantiated the defined safety compliance patterns, to illustrate their applicability.

In future, we plan to examine other clauses of ISO 26262 to apply the proposed patterns and discover additional ones. Once a complete catalogue of safety compliance patterns embracing ISO 26262 is ready, we plan to facilitate their instantiation by providing more elaborated guidelines. Our work on safety compliance patterns is expected to be combined with previously achieved results [3, 4] regarding the provision of a framework to increase efficiency and confidence in process compliance management.

## Acknowledgment

This work is supported by the EU and VINNOVA via the ECSEL JU project AMASS (No. 692474) [8].

# Bibliography


[1] Dunn, W.: Designing Safety-Critical Computer Systems. Computer **36**(11) (2003) 40–46

[2] ISO 26262: Road Vehicles-Functional Safety. International Standard (2011)

[3] Castellanos Ardila, J.P., Gallina, B.: Towards Increased Efficiency and Confidence in Process Compliance. In: Systems, Software and Services Process Improvement (EuroAsiaSPI). Volume 748., Springer International Publishing (2017) 162–174

[4] Castellanos Ardila, J.P., Gallina, B.: Towards Efficiently Checking Compliance Against Automotive Security and Safety Standards. In: The 7th IEEE International Workshop on Software Certification (WoSoCer). (2017)

[5] Governatori, G.: Representing business contracts in RuleML. International Journal of Cooperative Information Systems. (2005) 181–216

[6] Riehle, D., Züllighoven, H.: Understanding and using patterns in software development. Tapos **2**(1) (1996) 3–13

[7] Dwyer, M., Avrunin, G., Corbett, J.: Property Specification for Finite-State Verification. In: International Conference on Software Engineering. (1998) 411–420

[8] AMASS.: Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems

[9] AMASS: Case studies description and business impact D1.1. https://www.amass-ecsel.eu/content/deliverables. Technical report (2017)

[10] Gallina, B.: How to increase efficiency with the certification of process compliance. In: The 3rd Scandinavian Conference on Systems & Software Safety. (2015)

[11] Santos Laboratory: Specification Patterns. http://patterns.projects.cs.ksu.edu/

[12] Hashmi, M., Governatori, G., Wynn, M.: Normative requirements for regulatory compliance: An abstract formal framework. Information Systems Frontiers. **18**(3) (2016) 429–455

[13] Bitsch, F.: Safety patterns: the key to formal specification of safety requirements. In: International Conference on Computer Safety, Reliability, and Security., Springer (2001) 176–189

[14] Namiri, K., Stojanovic, N.: Pattern-Based Design and Validation of Business Process Compliance. On the Move to Meaningful Internet Systems. (2007) 59–76

[15] Elgammal, A., Turetken, O., van den Heuvel, W., Papazoglou, M.: Formalizing and applying compliance patterns for business process compliance. Software and Systems Modeling. (2016) 119–146

[16] Karimi, V.R.: Formal Analysis of Access Control Policies for Pattern-Based Business Processes. In: World Congress on Privacy, Security, Trust and the Management of e-Business. (2009) 239–242

[17] Menzel, M., Warschofsky, R., Meinel, C.: A pattern-driven generation of security policies for Service-oriented Architectures. In: IEEE International Conference on Web Services. (2010) 243–250

# Chapter 10

# Paper D: Lessons Learned while Formalizing ISO 26262 for Compliance Checking

Julieth Patricia Castellanos Ardila, Barbara Gallina and Guido Governatori.
In Proceedings of the 2st Workshop on Technologies for Regulatory Compliance (TeReCom-2018), Groningen, The Netherlands, December 2018.

**Abstract**

A confirmation review of the safety plan is required during compliance assessment with ISO 26262. Its production could be facilitated by creating a specification of the standard's requirements in FCL (Formal Contract Logic), which is a language that can be used to automatically checking compliance. However, we have learned, via previous experiences, that interpreting ISO 26262 requirements and specifying them in FCL is complex. Thus, we perform a formalization-oriented pre-processing of ISO 26262 to find effective ways to proceed with this task. In this paper, we present the lessons learned from this pre-processing which includes the identification of the essential normative parts to be formalized, the identification of SCP (Safety Compliance Patterns) and its subsequent documentation as templates, and the definition of a methodological guideline to facilitate the formalization of normative clauses. Finally, we illustrate the defined methodology by formalizing ISO 26262 part 3 and discuss our findings.

## 10.1 Introduction

A confirmation review of the safety plan is a piece of evidence required for compliance assessment with ISO 26262 [1] in the automotive industry. Producing this evidence is time-consuming since ISO 26262 contains hundreds of requirements that have to be checked based on the information provided by the specification of the development processes used to engineer the safety-critical systems included in cars. To automate this task, requirements should be encoded in formal notations, which can express not only their contradictory, incomplete and inconsistent nature, but also their normative provisions, which are the notions anchored to the structure of regulations [2]. From the compliance perspective, the normative provisions of importance are those related to the obligations, permissions, and prohibitions [3]. Therefore, a promising approach for formalizing requirements could be based on defeasible logic, in which contrary evidence defeats earlier reasoning, supporting the management of inconsistencies [4]. Also, normative provisions should be encoded as implications in which the antecedent is read as a property of a state of affairs, and the conclusion has a deontic nature [5]. Thus, we argue that deontic defeasible reasoning formalisms, such as Formal Contract Logic (FCL) [6], can be used to generate automatic support to reason from standard's requirements and the description of the process they regulate [7, 8].

In our previous work [9], we explored mechanisms to support the formalization work of the process engineers. From this initial attempt, a definition of SCP (Safety Compliance Patterns), as well as a set of ISO 26262-specific FCL-SCP were formulated. We also performed the formalization of standard's requirements into FCL in [7, 8]. Via these experiences, we learnt that interpreting ISO 26262 requirements and specifying them in FCL is a complex task. Therefore, we perform a formalization-oriented pre-processing of ISO 26262 to gain fundamental knowledge about efficient ways to proceed. In this paper, we present the lessons learned resulting form this pre-processing which includes the identification of the essential normative parts to be formalized, the identification of Safety Compliance Patterns (SCP), and its subsequent documentation as templates, and the definition of a methodological guideline, which can be used to facilitate the formalization of normative clauses. We also illustrate the defined methodology by formalizing ISO 26262 part 3 and discuss our findings.

The rest of the paper is structured as follows. In Section 10.2, we present

essential background. In Section 10.3, we describe the formalization-oriented pre-processing of ISO 26262. In Section 10.4, we illustrate the methodological guideline derived from the pre-processing of ISO 26262. In Section 10.5, we discuss our findings. In Section 10.6, we present related work. Finally, in Section 10.7, we present conclusions and future work.

## 10.2 Background

This section presents the background required in this paper. Section 10.2.1 recalls essential information related to ISO 26262. Section 10.2.2 recalls the basis of Formal Contract Logic. Finally, Section 10.2.3, recalls Safety Compliance Patterns.

### 10.2.1 ISO 26262

ISO 26262 [1] is a functional safety standard that regulates all phases of the production process of road vehicles with a gross mass up to 3500 kg. ISO 26262 uses *ASIL (Automotive Safety Integrity Levels)* to specify the safety requirements needed to be fulfilled during the development process of safety-critical systems included in cars, e.g., automobile brake system. ISO 26262 is composed of ten parts. Part 1 specifies the terms, definitions and abbreviated terms for application in all parts of ISO 26262. The remaining nine parts, which are normative, are structured in a similar way, containing, a *foreword*, *introduction*, *bibliography*, *annexes*, and *clauses*. *Clause 1* recalls the general scope of the standard and situates the particular part in this scope. *Clause 2* and *Clause 3*, recalls the normative references indispensable for the adoption of the specific part. *Clause 4*, which is repeated in all parts, contains two general compliance conditions. Item 4.1 relates to the tailoring of the safety activities, which is valid if *"an assessed rationale is available that the non-compliance is acceptable"*. Item 4.2 relates to the interpretation of tables, as follows:

- **Tables with consecutive entries:** All methods shall be applied as recommended in accordance with the ASIL. If methods other than those listed are to be applied, a rationale shall be given.
- **Tables with alternative entries:** An appropriate combination of methods shall be applied in accordance with the ASIL indicated. Methods with the higher recommendation for the ASIL should be preferred. A rationale shall be given that the selected combination of methods complies with the corresponding requirement.

The description of the phases of the safety process is distributed in clauses included in the nine normative parts starting from Clause 5. Each of these

clauses states its *Objectives*, which describe the generic goals of the clause, *General information*, which gives an overall explanation of the clause, *Inputs of the clause*, i.e., prerequisites, *Requirements and Recommendations (R&R)*, which describe the specific conditions that the process should fulfill, and the *Work products*, which are the mandatory deliverables. *Notes* and *Examples* are expected to help the applicant in interpreting the requirements. We focus on a subset of requirements from ISO 26262 part 3 presented in Table 10.1, which specifies the requirements for the concept phase for automotive applications.

Table 10.1: ISO 26262:2011 part 3 (Adapted from [1])

**5 Item definition** - *5.3 Inputs of this clause:* None.

*5.4 Requirements and recommendations*

5.4.1 Functional and non-functional requirements shall be made available, including: a) functional concept and b) operational constraints.

*5.5 Work products*: Item definition resulting from the requirements of 5.4.

**6 Initiation of the safety lifecycle** - *6.3 Inputs of this clause:* item definition in accordance with 5.5.

**7 Hazard analysis and risk assessment**

7.4.3 The severity shall be assigned to one of the severity classes S0, S1, S2 or S3 in accordance with:

| **Severity** | S0 | S1 | S2 | S3 |
|---|---|---|---|---|
| **Category** | No Injury | Moderate | Severe | Life-threatining |

7.4.4 Determination of ASIL and safety goals - The safety requirements shall be specified by an appropriate combination of the methods as presented in the table (H means *Highly* and R means *Recommended*).

| **Notation** | | **A** | **B** | **C** | **D** |
|---|---|---|---|---|---|
| 1a | Informal | HR | HR | R | R |
| 1b | Semi-formal | R | R | HR | HR |
| 1c | Formal | R | R | R | R |

## 10.2.2 Formal Contract Logic

Formal Contract Logic (FCL) [6] is a defeasible deontic logic created for checking the compliance of business contracts, and modelling normative requirements. An FCL rule has the form:

$r : a_1, ..., a_n \Rightarrow c$, where:

$a_1, ..., a_n =$ Conditions of the applicability of the norm.

$c =$ Normative effect.

Normative effects trigger deontic notions, namely, *Obligations*, *Prohibitions*, and *Permissions*. An *Obligation* is a statement describing a mandatory situation. In FCL, an obligation is represented by the operator $[O]$ plus a proposition, which corresponds to the content of the obligation. FCL is equipped with different kind of obligations, which depend on the timing of the application of the normative provision and their persistence after a violation

(see [6]). A *Prohibition* is a forbidden situation. In FCL a prohibition is represented as the negation of the content of an obligation. A *Permission* is an allowed situation. Exceptions for the rules can be formalized by using permissions, taking into account the premise *"if something is permitted the obligation to the contrary does not hold"* [3]. Permissions in FLC are represented with the operator $[P]$. The formalization of normative systems sometimes may contain conflicting normative effects, such as the obligation of performing an action but also its prohibition. In FCL, these conflicts can be solved by defining a superiority relation between rules ($>$).

### 10.2.3 Safety Compliance Patterns

Safety compliance patterns (SCP) [9] describe commonly occurring normative safety requirements on the permissible state sequence of a finite state model of a process. An SCP has a general formalization structure (which in our case is defined in FCL), which is derived from the interpretation of a recurring structure described in the text of the standard. Currently, a list of SCP is defined in [9].

## 10.3 Formalization-oriented Pre-processing of ISO 26262

Our initial efforts to formalize ISO 26262 into FCL (see our previous work [9, 8]), gave us some insights about the complexity that this task entails. As recalled in Section 10.2.1, ISO 26262 is structured in a specific way, i.e., it is composed of parts, which are subdivided into very structured clauses. We encounter that not all the structures are required to be formalized. We also find that some structures are repetitive, and can be represented as SCP. Therefore, to be able to formalize effectively, we consider that doing a pre-processing of ISO 26262 was necessary. The pre-processing, which is depicted in Figure 10.1, includes three tasks. Initially, we identify the essential normative structures (see Section 10.3.1), namely those structures that define the safety process to be adopted for developing the car's safety-critical systems. Then, we identified the repetitive structures of the standard that can be considered SCP (see Section 10.3.2). With the identified SCP, we create templates to consolidate a reusable knowledge base for future formalization jobs (see Section 10.3.3). Finally, the knowledge gathered in the pre-processing is used to define a methodological guideline for facilitating the formalization of

normative clauses in ISO 26262 (see Section 10.3.4). The pre-processing tasks were performed in the form of intensive group brainstorming sessions (in total ten 5-hour sessions). The group included three participants. The first participant has expertise in formal approaches (particularly FCL) applied to legal informatics. The second participant has expertise in certification (particularly in the safety-critical context). The third participant is a Ph.D. student whose research work is focused on the compliance checking of safety processes against safety standards (particularly ISO 26262).

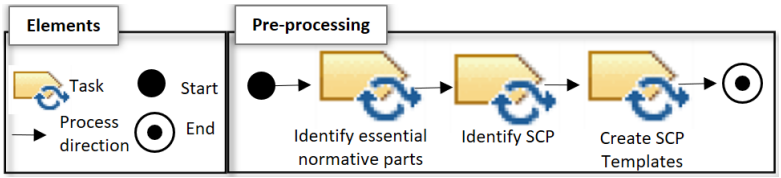


Figure 10.1: Methodological Guidelines to formalize ISO 26262 into FCL.

### 10.3.1 Identify essential normative parts in ISO 26262

As presented in Section 10.2.1, ISO 26262 has nine normative parts. In each of the normative parts, Clauses 1, 2 and 3 are not required to be formalized since the text does not represent a constraint to the development process. Clause 4 is subdivided into two items. Item 4.1, which title is *General requirements*, describes the tailoring of safety activities, namely its application in a way different to the indicated by the standard. Item 4.2, which title is *Interpretation of tables*, illustrate the way in which normative methods listed in tables (with consecutive and alternative entries) should be applied. By themselves, these two requirements are not directly constraining the process. However, they shape the way in which other requirements should be applied. Therefore, Clause 4 is an essential structure that gives important elements for the formalization process. The specific normative clauses that describe the phases of the safety process are documented from Clause 5 in each of the nine normative parts. In those clauses, the title represents the initiation of the phase. Therefore, the title is part of the formalization process. However, the formalization of the title must be preceded by the formalization of the *Prerequisites*, since they represent the preconditions constraining the initiation of the particular phase. *Prerequisites* as well as *Work products* are essential since they are part of the description of the safety process which is expected to be represented via a model embracing input/output elements. The presence of

the verbs *shall* and *must* in the section *R&R* is an explicit indication of a normative provision that constrains the breakdown of the work as well as for guidance on how it should be planned and executed. Information under the titles *Objectives*, *General*, *Notes* and *Examples* are not formalized since they do not prescribe the process to be adopted. However, these elements can be used to provide context for the application of the requirements.

### 10.3.2 Identify SCP

Within the essential normative parts of ISO 26262, seven SCP are identified. In Clause 4, the *provision of a rationale* is done in the same way whenever a safety activity is tailored. Similarly, the applicable methods that are described in *tables with alternative entries* and *tables with consecutive entries*. Therefore, Clause 4 describes three repetitive structures. Within the description of the phases of the safety process, represented from clause 5 in each of the normative parts, other three repetitive structures are easily recognizable. The first one is the *Initiation of a phase*, which is recognized in the title of every clause. The second repetitive structure corresponds to the *Prerequisites*, which describe the preconditions of the phase. Similarly, the *Work products*, which are defined as the result of safety activities, represent a third repetitive structure. Since *R&R* contains many requirements, and each one describes a different structure, we cannot consider this structure as repetitive itself (even though we can find the title *R&R* in all the normative parts). However, inside the *R&R* one repetitive structure, called *Guidance*, is recognized. A requirement, which we call the main normative effect, contains guidance when it is accompanied by a list of descriptive items ( a, b,..., n). Together, those items provide additional normative descriptions about the main normative effect, and therefore, they also become mandatory requirements.

### 10.3.3 Create SPC templates

For each SCP, we provide a general formalization structure in FCL which is derived from the interpretation of the repetitive structure it represents (as described in Section 10.2.3). The rules in the template contains the symbol # between brackets ({ }), which should be replaced with the identification of the requirement that the rule is representing. The space between the brackets in the rule statement ({ }) is a placeholder for the particular instantiation of the template.

**Provision of a rationale:** A rationale implies *compliance with conditions*. For being valid, it should be always verified by an expert. Therefore, when a rationale is attached to a safety process, its verification is obligated (see Template 10.1).

$$\mathbf{r}\{\mathbf{\#:}\}attach\{Rationale\} \Rightarrow [O]verifyByExpert\{Rationale\} \tag{10.1}$$

**Alternative entries:** The normative provision for tables with alternative entries obliges the verification of the ASIL, the provision of a combination of the listed methods and a the provision of a weak rationale. The combination of these methods also obliges the inclusion of those marked with the highest recommendation level. The provision of other methods is also possible if a strong rationale is provided. There is an inconsistency between rules **r**{**#.a:**}and **r**{**#.g:**}. Thus, a superiority relation that gives priority to **r**{**#.g:**} (which describes an exception for the requirement) is provided (see template 10.2).

$$\begin{aligned}
&\mathbf{r}\{\mathbf{\#.a:}\}verify\{ASIL\}\\
&\Rightarrow [O]provideCombinationOfListedMethods\\
&\mathbf{r}\{\mathbf{\#.b:}\}provideCombinationOfListedMethods\\
&\Rightarrow [O]include\{HigherRecommendedNotationsForASIL\}\\
&\mathbf{r}\{\mathbf{\#.c:}\}include\{HigherRecommendedNotationsForASIL\}\\
&\Rightarrow [O]attachWeakRationaleSupportingListedMethods\\
&\mathbf{r}\{\mathbf{\#.d:}\}attachWeakRationaleSupportingMethods\\
&\Rightarrow [O]VerifyWeakRationaleByDomainExpert\\
&\mathbf{r}\{\mathbf{\#.e:}\}includeNotListedMethods\{OtherMethods\}\\
&\Rightarrow [O]atachStrongRationaleSupporting\{OtherMethods\}\\
&\mathbf{r}\{\mathbf{\#.f:}\}atachStrongRationaleSupporting\{OtherMethods\}\\
&\Rightarrow [O]verifyStrongRationaleByDomainExpert\{OtherMethods\}\\
&\mathbf{r}\{\mathbf{\#.g:}\}includeNotListedMethods\{OtherMethods\},\\
&[O]atachStrongRationaleSupporting\{OtherMethods\},\\
&verifyStrongRationaleByDomainExpert\{OtherMethods\}\\
&\Rightarrow [P]-provideCombinationOfTheListedMethods\\
&r\{\#.g\}>r\{\#.a\}
\end{aligned} \tag{10.2}$$

**Consecutive entries:** The normative provision for tables with consecutive entries obliges the verification of the ASIL and the utilization of all listed methods. The combination of methods beyond the ones listed in the table is also possible if a strong rationale is provided (see Template 10.3). There is an inconsistency between rules **r**{**#.a:**}and **r**{**#.d:**}. Thus, a superiority relation that gives priority to **r**{**#.d:**}

(which describes an exception for the requirement) is provided (see template 10.2).

$$\begin{aligned}
&\mathbf{r\{\#.a:\}}verify\{ASIL\}\\
&\Rightarrow [O]provideAllfListedMethods\\
&\mathbf{r\{\#.b:\}}includeNotListedMethods\{otherMethods\}\\
&\Rightarrow [O]atachStrongRationaleSupporting\{otherMethods\}\\
&\mathbf{r\{\#.c:\}}attachStrongRationaleSupporting\{otherMethods\}\\
&\Rightarrow [O]verifyStrongRationaleByDomainExpert\{otherMethods\}\\
&\mathbf{r\{\#.d:\}}includeNotListedMethods\{otherMethods\},\\
&[O]atachStrongRationaleSupporting\{otherMethods\},\\
&verifyStrongRationaleByDomainExpert\{OtherMethods\}\\
&\Rightarrow [P]-provideCombinationOfTheListedMethods\\
&r\{\#.d\}>r\{\#.a\}
\end{aligned} \tag{10.3}$$

**Prerequisites:** The antecedents of the initiation of a phase are the prerequisites. Therefore, they are obliged before the phase is initiated (see Template 10.4).

$$\begin{aligned}
&\mathbf{r\{\#.a\}:} \Rightarrow [O]provide\{prerequisiteA\}\\
&\ldots\\
&\mathbf{r\{\#.n\}:} \Rightarrow [O]provide\{prerequisiteN\}
\end{aligned} \tag{10.4}$$

**Initiation of a phase:** The template considers the prerequisites (formalization presented in Template 10.4) as the conditions of the applicability of the rule which normative conclusion is the initiation of the phase (see Template 10.5).

$$\begin{aligned}
&\mathbf{r\{\#:\}}provide\{prerequisiteA\}...,provide\{prerequisiteN\}\\
&\Rightarrow [O]initiate\{TitleClause\}
\end{aligned} \tag{10.5}$$

**Guidance:** Guidance components are the conditions that obliges the provision of the element guided (see formula 10.6).

$$\begin{aligned}
&\mathbf{r\{\#.a:\}}\{TriggeringObligation\}\\
&\Rightarrow [O]provide\{FirstGuidanceElement\}\\
&\ldots\\
&\mathbf{r\{\#.n:\}}\{TriggeringObligation\}\\
&\Rightarrow [O]provide\{LastGuidanceElement\}\\
&\mathbf{r\{\#:\}}provide\{FirstGuidanceElement\},...,\\
&provide\{LastGuidanceElement\} \Rightarrow [O]provide\{GuidedElement\}
\end{aligned} \tag{10.6}$$

**Work Product:** Work products are the result of certain requirements. Therefore, these requirements are presented as antecedents that obliged the provision of the related work product (see Template 10.7).

$$\textbf{r\#:}provide\{PreviousObligations\} \Rightarrow [O]produce\{WorkProduct\} \quad (10.7)$$

### 10.3.4 Methodological guideline for formalizing ISO 26262

From the pre-processing tasks described above, we got an understanding of what to formalize and how we could proceed in the formalization process. The parts to formalized are those that determine the safety lifecycle, meaning those clauses that start from Clause 5. To formalize these clauses, we have described a methodological guideline, which we depict in Figure 10.2.

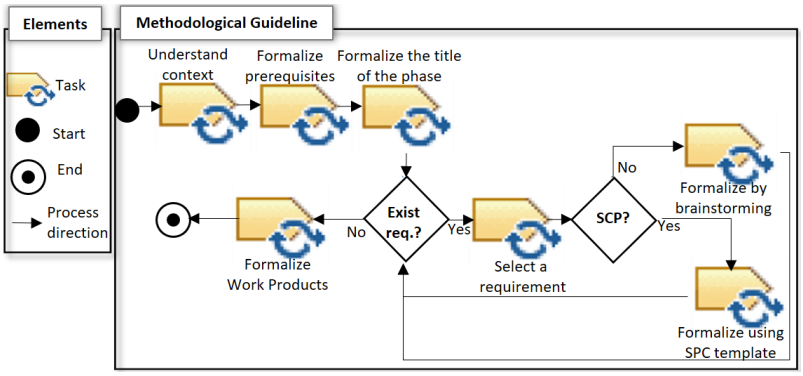


Figure 10.2: Methodological Guidelines to formalize ISO 26262 into FCL.

Initially, the context of the phase should be understood. For this, the reading and analysis of the objectives and the main general information of the clause to be formalized is required. Then, the formalization process initiates with the prerequisites and followed by the title. These two formalizations can be done by following the SCPs called *Prerequisites* and *Initiation of a phase*. After, one requirement is selected from the list of *R&R*. We suggest that the requirements are selected in the order they are presented and that the rules are named following the requirement numeration to ensure consistency and traceability. For instance, if a textual requirement is marked with the label *5.1*, the corresponding rule should be called *r.5.1*. During the formalization of the requirements, SCP templates could be used to facilitate this task. However, if

there are no templates, brainstorming sessions are required. The brainstorming session can be carried out in different ways, but the most relevant is that the group takes one requirement at the time, discuss its importance in the compliance process (e.g., related requirements or permits for tailoring), divide the requirement into smaller sentences that have only one idea, and discuss every sentence. If the requirement has to be divided into several rules, the name of the rule has to be named with the number that accompanies the requirement plus a letter, i.e., *r.5.1.a*, *r.5.1.b*. Finally, when all requirements available in *R&R* are covered, the work products can be formalized by using the SCP template called *Work Product*. The generated rule set should be verified to avoid inconsistencies and typos in the rules since Regorous do not recognize incorrectly formed rule sets.

## 10.4 Formalizing ISO 26262 Part 3

In this section, we illustrate the methodological guideline depicted in Figure 10.2 by formalizing the requirements provided in Table 10.1. The formalization starts from clause 5 (see Table 10.1) providing requirements for the *definition and description of the item*. There are no prerequisites in this clause. Thus, we continue with the formalization of the title of the phase, which provides the obligation to initiate item definition (see rule $r5$). Then, we continue with the formalization of requirement 5.4.1, that defines guidance to provide the functional and non-functional requirements for the item definition. This requirements is formalized by using the SCP *Guidance* in which initially, the components of the guidance should be provided (see rules $r5.4.1.a$ and $r5.4.1.b$), and then, they integrate the main normative provision (see $r5.4.1$). When all the requirements are formalized, we proceed with the work products, which are defined in clause 5.5. To formalize a work product, the requirements (in this case specified by clause 5.4) are presented as antecedents, and the work product itself is the normative provision (see $r.5.5$).

**r5:** $\Rightarrow [O]initiateItemDefinition$

**r5.4.1.a:** $performItemDefinition \Rightarrow [O]provideFunctionalConcept$

**r5.4.1.b:** $performItemDefinition \Rightarrow [O]provideOperationalConstraints$

**r5.4.1:** $provideFunctionalConcept, provideOperationalConstraints \Rightarrow [O]provideFunctionalAndNonFunctionalRequirements$

**r.5.5:** $provideFunctionalAndNonFunctionalRequirements \Rightarrow [O]produceItemDefinition$

Clause 6 in Table 10.1 presents the title of the clause and its inputs. Thus, we only applied the steps related to the formalization of the prerequisites and

the title of the clause. The prerequisites were already formalized in clause 5, (see Table 10.1, *Inputs of this clause: item definition in accordance with 5.5.*). Therefore, it is only needed the formalization of the title which defines the obligation of performing the phase *Initiation of the safety lifecycle* after the normative provision *produceItemDefinition* (see rule $r6$).

**r6:** $produceItemDefinition \Rightarrow [O]initiateInitiationSafetyLifeCycle$

Clause 7, which is related to *Hazard analysis and risk assessment*, does not mention any inputs for the clause and it is formalized as rule $r5$ (see rule $r7$). Two requirements are described in tables. The first requirement refers to a table which has constitutive information. The formalization of this table is done by taking the description of the entries as the antecedent of the rule and its category as the normative provision (see rules $r7.4.3.a$ to $7.4.3.d$).

**r7:** $\Rightarrow [O]performHazardAnalysisAndRiskAssessment$
**r7.4.3.a:** $descriptionSeverityS0 \Rightarrow [O]CatergoryNoInjuries$
**r7.4.3.b:** $descriptionSeverityS1 \Rightarrow [O]CatergoryLightAndModerateInjuries$
**r7.4.3.c:** $descriptionSeverityS2 \Rightarrow [O]CatergorySevereAndlifeThreateningInjuries$
**r7.4.3.d:** $descriptionSeverityS2 \Rightarrow [O]CatergoryFatalInjuries$

Clause 7.4.4 is presented in a table with alternative entries. We take into account the selection of methods for ASIL A. As recalled in Section 10.2.1 for alternative entries the normative provision suggest the obligation to provide a combination of the methods listed in the table (see rule $r7.4.4.a$), which at the same time obliges the selection of those with higher recommendation level for the ASIL, in this case, *Informal Notations* (see rule$r7.4.4.b$). Also, a rationale shall be given that the selected methods comply with the corresponding requirements (see rule $r7.4.4.c$). If the highest recommended is selected, only a weak rationale (i.e., a less stringent explanation of the selection) must be provided. However, if the highest recommended is not selected, a more elaborated rationale (called strong) should be provided (see rule $r7.4.4.e$). A domain expert should verify the rationales (strong and weak) (see rule $r7.4.4.d$ and $r7.4.4.f$). Providing the strong rationale, its verification and the methods selected, grant the permit of not providing the combinations of the

recommended methods (see rule $r7.4.4.g$).

**r7.4.4.a:**$verifyASILA \Rightarrow [O]provideCombinationOfListedMethods$
**r7.4.4.b:**$provideCombinationOfListedMethods \Rightarrow [O]includeInformalNotations$
**r7.4.4.c:**$includeInformalNotations$
$\Rightarrow [O]attachWeakRationaleSupportingListedMethods$
**r7.4.4.d:**$attachWeakRationaleSupportingMethods$
$\Rightarrow [O]VerifyWeakRationaleByDomainExpert$
**r7.4.4.e:**$includeNotListedMethods$
$\Rightarrow [O]atachStrongRationaleSupportingNotListedMethods$
**r7.4.4.f:**$attachStrongRationaleSupportingNotListedMethods$
$\Rightarrow [O]verifyStrongRationaleByDomainExpert$
**r7.4.4.g:**$includeNotListedMethods$,
$atachStrongRationaleSupportingNotListedMethods$,
$verifyStrongRationaleByDomainExpert$
$\Rightarrow [P] - provideCombinationOfTheListedMethods$
$r7.4.4.g > r7.4.4.a$

## 10.5 Discussion

Interpreting and specifying ISO 26262 requirements in FCL can be error-prone and time-consuming. One reason is that ISO 26262 is a large document with hundreds of requirements, which like many other standards and regulations, are difficult to interpret. The other reason is that FCL is not yet enough known and existing examples (mostly from the business and legal domains) are insufficient to guide the formalization of the normative notions that constrain the processes used in safety-critical development projects. However, the effort invested in the production of formal specifications of safety standards is compensated by several advantages, i.e., deep understanding of its requirements, practical application in development projects, and the provision of an essential input for facilitating automated compliance checking. Therefore, we consider that discovering efficient ways to proceed with the formalization work can boost the usage of this formal language in the compliance tasks in the safety-critical context. In the remainder, we discuss some aspects that were observed during the performing of the formalization-oriented pre-processing of ISO 26262 and the formalization of its part 3.

**Useful formalization path:** As recalled in Section 10.2.1, ISO 26262 is structured in a specific way. However, not all the structures should be used to

obtain the formal specification in FCL. Therefore, performing a pre-processing of ISO 26262 provides us an useful formalization path to follow, i.e., a methodological guideline and SCP templates. Process engineers in the automotive context, who are interested in starting their formalization work with FCL, may find useful this formalization path since it permits to focus on specific tasks and skip some others that may be not relevant in the formalization process. Performing a similar pre-processing of safety standards beyond ISO 26262 may also be useful for increasing and spreading the use of FCL and its potential benefits.

**Related skills, competencies and responsibilities:** In our experience, group brainstorming sessions have facilitated the production of the FCL specifications. In particular, the participation of different kind of experts has provided different views that were important in the discussions performed during the formalization process. We highlight the fact that brainstorming sessions were mainly guided by the certification expert, whose knowledge provided specific details that are of importance for the safety auditor during the safety assessment. The opportunity to have an FCL expert speeded up the formalization and the creation of templates for reuse. However, FCL is not a very known language. Thus, there are not many FCL experts available. Therefore, it is necessary to provide training courses for teaching FCL. The target group for the training may be mainly conformed by process engineers who already have expertise in compliance management.

**Tooling:** In our current work, the rules were written manually, introducing the possibility of typos in the syntax of the rules and inconsistencies in the rules statements. Therefore, the production of our initial FCL specifications resulted in incoherent files that were not understood by the compliance checker. To solved this issue, edition and syntactic correctness of the FCL specifications were ensured manually. However, manual corrections are long and tedious activities. For this reason, we consider that the provision of tools for supporting the process of writing and verifying rules, as well as the creation and instantiation of SCP templates should be developed. Part of our work should also be the provision of these tools.

## 10.6 Related Work

The collection of experiences distilled from formalization projects is an advisable way to save time and avoid mistakes in future projects. We can find lessons learned in the formalization of software engineering standards in [10]. In a similar way, we have collected the specific lessons learned in a methodological guideline, aiming at facilitating the formalization of the ISO 26262 clauses. Guidelines are also widely used to spread the used of novel methods in engineering tasks. One example is the Oracle Policy Modeling best practices guidelines [11] whose aim is helping analysts to describe the way in which different types of business rules can be modeled. Similarly, a methodology to guide companies to establish Cyber-Physical Social System data subjects consent and data usage policies (described in OWL) is presented in [12]. Guidelines for supporting the formal representation of safety regulatory requirements (using Z) are introduced in [13]. The use of tabular expressions in [14] can also be seen as a guideline to generate formal models of system requirements. The authors of FCL have also published explicative examples of the modeling of FCL rules within the business context, e.g., [15, 16], which can be used as a guideline for learning the language.

The use of FCL for supporting compliance management tasks in automotive is a novelty. We did not find yet specific examples or guidelines that apply to the domain. Therefore, the results of the formalization-oriented pre-processing of ISO 26262 documented in this paper may be of interest for process engineers involved in the cars manufacturing. Additionally, we consider that this work can be used as a starting point to derive domain-specific guidance applicable to process-based safety standards beyond ISO 26262.

## 10.7 Conclusions and Future Work

In this paper, we presented the lessons learned from performing a formalization-oriented pre-processing of ISO 26262. Initially, we identify the essential normative structures, namely those structures that define the safety process to be adopted for developing the car's safety-critical systems. Then, we identified the repetitive structures of the standard that can be considered SCP. With the identified SCP, we create templates to consolidate a reusable knowledge base for future formalization jobs. The knowledge gathered in the

pre-processing is used to define a methodological guideline for facilitating the formalization of normative clauses in ISO 26262. We also illustrate the defined methodology by formalizing ISO 26262 part 3 and discussed our findings.

From the discussion presented in Section 10.5, we consider that one important part of the future work is the training of process engineers in FCL. Therefore, a course called *Quality assurance - Certification of safety-critical (software) systems*[1], which is under construction, will consider *an overview of compliance checking* and *the formalization of compliance rules with FCL*. We also need to optimize the creation and verification of rule sets. Thus, we are considering the design and development of a pattern-based rule editor to facilitate rules creation, and rule sets verification. We consider that methodological guidelines are also needed in other safety-critical domains. Thus, we aim at studying other safety standards and adapt any required step resulting from their specificities. This work is also expected to be combined with previously achieved results [7, 8] regarding the provision of automated compliance checking vision for the safety-critical context.

# Acknowledgment

This work is supported by the EU and VINNOVA via the ECSEL JU project AMASS (No. 692474) [17].

[1] http://www.promptedu.se/quality-assurance-certification-of-safety-critical-software-systems/

# Bibliography


[1] ISO 26262: Road Vehicles-Functional Safety. International Standard (2011)

[2] Francesconi, E.: Semantic model for legal resources: Annotation and reasoning over normative provisions. Semantic Web **7**(3) (2016) 255–265

[3] Hashmi, M., Governatori, G., Wynn, M.T.: Normative requirements for regulatory compliance: An abstract formal framework. Information Systems Frontiers **18**(3) (2016) 429–455

[4] Nute, D.: Defeasible Logic. In: International Conference on Applications of Prolog, Springer (2001) 151–169

[5] Alberti, M., Gavanelli, M., Lamma, E., Riguzzi, F., Zese, R.: Dischargeable Obligations in Abductive Logic Programming. In Springer, ed.: International Joint Conference on Rules and Reasoning. (2017) 7–21

[6] Governatori, G.: Representing business contracts in RuleML. International Journal of Cooperative Information Systems. (2005) 181–216

[7] Castellanos Ardila, J.P., Gallina, B., Ul Muram, F.: Enabling Compliance Checking against Safety Standards from SPEM 2.0 Process Models. In: Euromicro Conference on Software Engineering and Advanced Applications (SEAA). (2018)

[8] Castellanos Ardila, J.P., Gallina, B., UL Muram, F.: Transforming SPEM 2.0-compatible Process Models into Models Checkable for Compliance. In: 18th International SPICE Conference. (2018)

[9] Castellanos Ardila, J.P., Gallina, B.: Formal Contract Logic Based Patterns for Facilitating Compliance Checking against ISO 26262. In: 1st Workshop on Technologies for Regulatory Compliance (TeReCom). (2017) 65–72

[10] Verlage, M., Munch, J.: Formalizing software engineering standards. In: Third IEEE International Software Engineering Standards Symposium and Forum. (1997) 196–206

[11] Lee, J.: Oracle Policy Automation (OPA). Best Practice Guide for policy Modelers. (2018)

[12] Fernandez, J.: Deliverable 6.1: Privacy policy formalization (v. 1) (2018)

[13] Vilkomir, S., Bowen, J., Ghose, A.: Formalization and assessment of regulatory requirements for safety-critical software. Innovations in Systems and Software Engineering **2**(3-4) (2006) 165–178

[14] Singh, N., Lawford, M., Maibaum, T., Wassyng, A.: Use of Tabular Expressions for Refinement Automation. In: International Conference on Model and Data Engineering. (2017) 167–182

[15] Governatori, G.: Practical normative reasoning with defeasible deontic logic. In: Reasoning Web International Summer School. (2018) 1–25

[16] Governatori, G.: The regorous approach to process compliance. In: IEEE 19th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations. (2015) 33–40

[17] AMASS.: Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems

# Chapter 11

# Paper E: Towards Increased Efficiency and Confidence in Process Compliance

Julieth Patricia Castellanos Ardila and Barbara Gallina.
In Proceedings of the 24th European & Asian Systems, Software & Service Process Improvement & Innovation (EuroAsiaSPI-2017), Ostrava, Czech Republic, September 2017.

**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.

## 11.1 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 11.2, we provide background information related to our work. In Section 11.3, we introduce SoPLE&Logic-basedCM for efficient and confidence process compliance. In Section 11.4, we apply SoPLE&Logic-basedCM to the automotive domain, and based on the application of our approach, we derive our lessons learned. In Section 11.5, we discuss related work. Finally, in Section 11.6, we present conclusions and future work.

## 11.2 Background

This section provides basic information on which we base our work. In Section 11.2.1 and 11.2.2, we present two automotive standards. In Section 11.2.3, we recall SoPLE. In Section 11.2.4, we present defeasible logic, and in Section 11.2.5, we recall an abstract formal framework for regulatory compliance.

### 11.2.1 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 11.1.

### 11.2.2 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

Table 11.1: 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. |

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 11.2.

Table 11.2: 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). |

### 11.2.3 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 11.3). 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 11.3: 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/  |

### 11.2.4 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 $\hookrightarrow = \{\rightarrow\ (strict), \Rightarrow\ (defeasible), or\ \rightsquigarrow\ (defeater)\}$. 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.

### 11.2.5 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 reparation 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)*.

## 11.3 SoPLE&Logic-basedCM

This section provides an overview of SoPLE&Logic-basedCM (see Fig. 11.1), 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 11.2.4, and the use of the framework for modelling compliance, presented in Section 11.2.5. As Fig. 11.1 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.

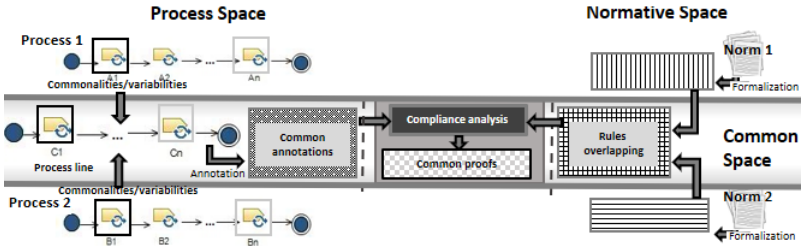


Figure 11.1: SoPLE&Logic-basedCM overview.

## 11.4 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 11.4.1, we model a SoPL. In Section 11.4.2, we define the proofs of compliance. In Section 11.4.3, we present the lessons learnt.

### 11.4.1 SoPL Modeling

In this subsection, we apply SoPLE, recalled in Section 11.2.3. The scope is ASPICE SWE.3, and ISO 26262 SUDI recalled in Sections 11.2.1 and 11.2.2 respectively. In the domain engineering phase, we find the equivalent process

activities by applying the terminology mapping presented in Table 11.3, and by analysing the scope of each activity. Terminological similarity is found between BP1 and A1. However, the scope of A1 (see Table 11.2 - 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 11.2 - 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 11.4). The result of the domain engineering phase is a SoPL, depicted in Fig. 11.2.

Table 11.4: 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 |

## 11.4.2 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

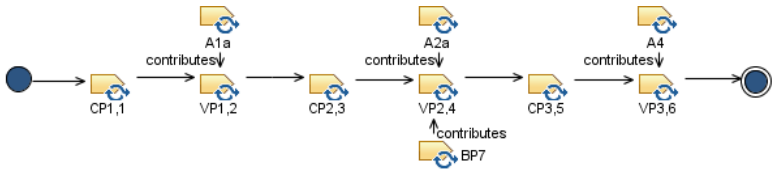


Figure 11.2: SoPL model embracing ASPICE SWE.3 and ISO 26262 SUDI.

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 11.2.4. For ASPICE SWE.3 (see Table 11.5) the rules PO2, PO4, PO5 and PO6 (see Table 11.1) 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 11.5: 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). |
| PO7 | RA8 | sui $\rightarrow$ sud. | software unit implementation (sui) is based on sud. |
| | RA9 | sui $\rightarrow$ i. | sui is implemented (i). |

For ISO 26262 SUDI, a similar analysis is done (see Table 11.6). 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 11.7. 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 11.2 - 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 11.2 - 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 11.6: 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. 11.2) 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 11.2.5) and presented in Listing 11.1.

Table 11.7: Rules comparison and commonality identification.

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

```
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}
```

Program 11.1: 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 11.2.5). In Table 11.8, common defeasible theories (Table 11.7) are assigned to the commonality 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 11.2.5, this means that the commonality points of the trace *tSoPL* are compliant with the set of rules presented in Table 11.7.

Standard-specific processes are generated when variability points are

Table 11.8: 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} |

deployed with specific tasks. ASPICE SWE.3 process is $t_{A:<CP1,\ CP2,\ \mathbf{BP7},\ CP3>}$, where VP2 is contributed with BP7. ISO 26262 SUDI process is $t_{I:<CP1,\ \mathbf{A1a},\ CP2,\ \mathbf{A2a},\ CP3,\ \mathbf{A4}>}$, where VP1 is contributed with A1a, VP2 with A2a and VP3 with A4. Table 11.9 shows the influence of the obligations in force for these tasks.

Table 11.9: 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) |

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 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.

### 11.4.3 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.

## 11.5 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.

## 11.6 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.

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

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

## Acknowledgment

This work is supported by the EU and VINNOVA via the ECSEL JU project AMASS (No. 692474) [20].

# Bibliography


[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/