Enabling Compliance Checking against Safety Standards from SPEM 2.0 Process Models

Julieth Patricia Castellanos Ardila, Barbara Gallina, and Faiz UL Muram
School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden
{julieth.castellanos, barbara.gallina, faiz.ul.muram}@mdh.se

***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 fulfilment 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.**

***Index Terms*—Compliance checking, SPEM 2.0, Regorous.**

I. 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 II, we provide background. In Section III, we present our compliance checking vision and the mechanism to annotate software process. In Section IV, we examine an ISO 26262-based small example. In Section V, we present related work. Finally, in Section VI, we provide conclusions and future work.

II. Background

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

A. 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 I).

TABLE I: 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	

B. 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]. This approach 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.

C. 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.

D. 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 II. The interested reader may refer to our previous work [12] 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 II). 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 II: 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

$$
\begin{aligned}
r_1 :\Rightarrow\ & [OM]addressSwUnitDesignProcess \\
r_2 :\ & addressSwUnitDesignProcess \\
\Rightarrow\ & [OANPNP] - performSpecifySwUnit \\
r_2' :\ & performProvideSwArchitecturalDesign, \\
& performProvideSwSafetyRequirements \\
\Rightarrow\ & [P]performSpecifySwUnit \\
r_3 :\ & performSpecifySwUnit \\
\Rightarrow\ & [OANPNP]selectMandatoryNotationsforSwDesign \\
r_3' :\ & provideRationaleForNotSelectMandatoryNotationsforSwDesign \\
\Rightarrow\ & [P] - selectMandatoryNotationsforSwDesign \\
& r_2' > r_2, r_3' > r_3
\end{aligned}
\tag{1}
$$

III. 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 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.

The rest of the content of this paper focuses on the region delimited by the dotted line depicted in Figure 1, which has a twofold function. First, we identify which modeling capabilities should be used for capturing standard's information,

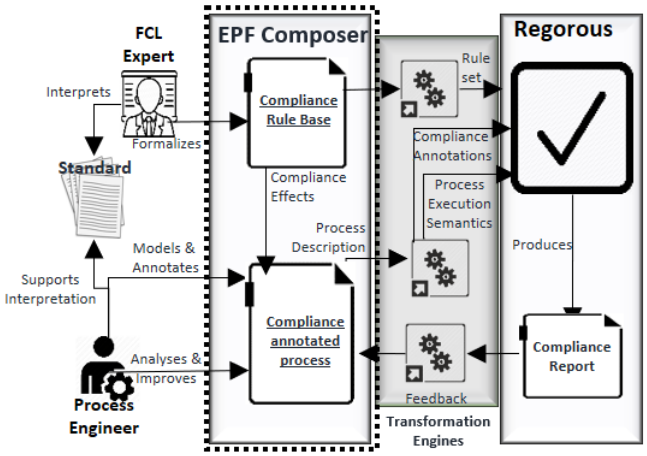

Fig. 1: Automated Compliance Checking Vision.

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 III. 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 II-B. The first plugin captures *standard's requirements* by using the customized SPEM 2.0 elements presented in Table III. 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.

TABLE III: SPEM 2.0 Customization

SPEM 2.0	Compliance Information	Suggested Icons
Reusable Asset	Rule Set	
Concept	Compliance Effect	
Custom category	Standard requirement	

IV. 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 III) 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 II, 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 2.

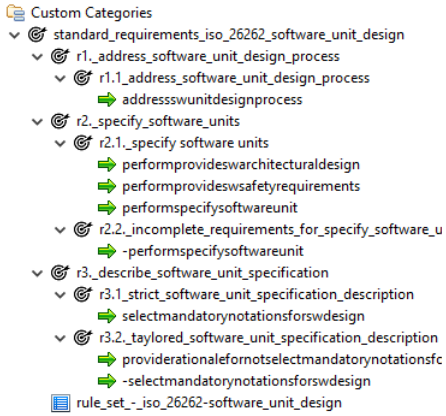

Fig. 2: Requirements and their Associated Elements.

The actual rule is written in the *main description* field of the compliance effect (See Figure 3).

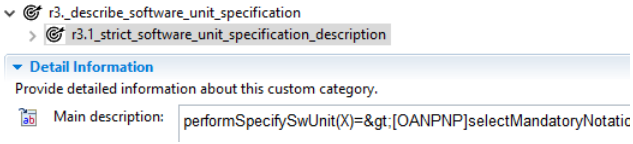

Fig. 3: Specification of Rule r3.1

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 4).

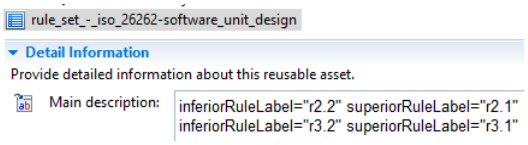

Fig. 4: Rule Set Specification

Then, we create the plugin for capturing process elements. In this example (See Figure 5), the definition of the process elements is based on our interpretation of the requirements provided in Table II. 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*.

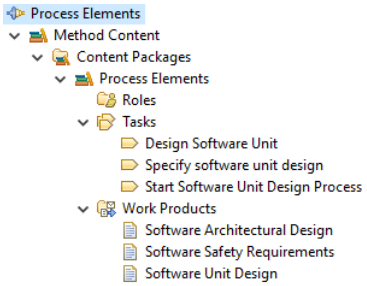

Fig. 5: Process Elements Plugin.

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 6).

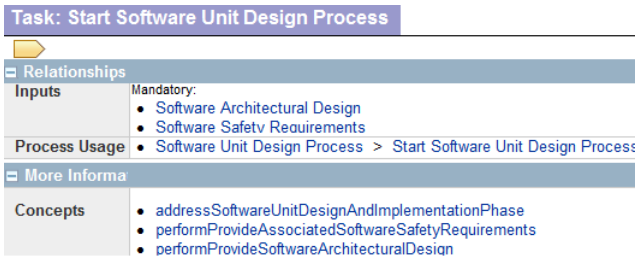

Fig. 6: Start Software Unit Design Process Task

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).

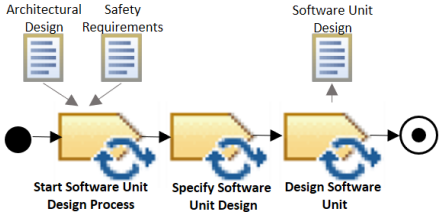

Fig. 7: Activity Diagram of the Software Unit Design Process.

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 IV). 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 IV: 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 V. As expected, Regorous reports the compliance violation that occurs in the process model. Specifically, it says that the

TABLE V: Regorous Report

Compliance Check Results: Process is non-compliant. **Description:** Unfulfilled obligation to 'selectMandatoryNotationsForSwDesign' (Achievement, non-pre-emptive, non-persistent). **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 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.

V. 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 II-B, works related to **mapping regulations** have been proposed to facilitate process engineers work. In [13], the authors examine techniques to map a single taxonomy to multiple regulations. In [14], 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 [15], in which the authors propose a metamodel (SafetyMet) that includes concepts and relationships for safety targeted for facilitating safety compliance. In [16], 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 [17], in which the concepts involved in the Capability Maturity Model Integration (CMMI) standard are mapped to SPEM 2.0. As in [17], we have used *Category* to classify standard's requirements. Approaches for **verifying software process models** are presented in [18] and [19]. In [18], 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 [19], 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.

VI. 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].

References

[1] J. Castellanos Ardila and B. Gallina, "Towards Increased Efficiency and Confidence in Process Compliance," in *24th European Conference EuroSPI.*, pp. 162–174, 2017.

[2] B. Gallina, F. Ul Muram, and J. Castellanos Ardila, "Compliance of Agilized (Software) Development Processes with Safety Standards: a Vision," in *4th international workshop on Agile Development of Safety-Critical Software*, p. 5, 2018.

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

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

[5] B. McIsaac, "IBM Rational Method Composer: Standards Mapping.," tech. rep., IBM Developer Works, 2015.

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

[7] G. Koliadis and A. Ghose, "Verifying Semantic Business Process Models in Verifying Semantic Business Process Models in Inter-operation," in *IEEE International Conference on Service-Oriented Computing*, pp. 731–738, 2007.

[8] Internationl Standards Organization, "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. http://www.amass-ecsel.eu/."

[11] ECSEL Research and Innovation actions (RIA) - AMASS, "D6.5 Prototype for Cross/Intra-Domain Reuse (b). https://www.amass-ecsel.eu/content/deliverables," 2017.

[12] J. Castellanos Ardila and B. Gallina, "Formal Contract Logic Based Patterns for Facilitating Compliance Checking against ISO 26262," in *1st Workshop on Technologies for Regulatory Compliance*, pp. 65–72, 2017.

[13] C. Cheng, G. Lau, and K. Law, "Mapping regulations to industry-specific taxonomies," in *11th international conference on Artificial intelligence and law* ., pp. 59–63, 2007.

[14] S. Sunkle, D. Kholkar, and V. Kulkarni, "Toward better mapping between regulations and operational details of enterprises using vocabularies and semantic similarity," *Complex Systems Informatics and Modeling Quarterly*, no. 5, pp. 39–60, 2015.

[15] J. L. De La Vara and R. Panesar-Walawege, "SafetyMet: A metamodel for safety standards," in *International Conference on Model Driven Engineering Languages and Systems*, pp. 69–86, 2013.

[16] S. Fenz, T. Pruckner, and A. Manutscheri, "Ontological mapping of information security best-practice guidelines," *International Conference on Business Information Systems*, pp. 49–60, 2009.

[17] C. Portela, A. Vasconcelos, A. Silva, A. Sinimbú, E. Silva, M. Ronny, W. Lira, and S. Oliveira, "A Comparative Analysis between BPMN and SPEM Modeling Standards in the Software Processes Context," *Journal of Software Engineering and Applications*, vol. 5, no. 5, pp. 330–339, 2012.

[18] P. Kingsbury and A. Windisch, "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.

[19] R. Bendraou, B. Combemale, X. Crégut, and M. Gervais, "Definition of an executable SPEM 2.0," in *14th Asia-Pacific Software Engineering Conference.*, pp. 390–397, 2007.