Towards Efficiently Checking Compliance Against Automotive Security and Safety Standards

Julieth Patricia Castellanos Ardila
IDT, Mälardalen University,
Västerås, Sweden
Email: julieth.castellanos@mdh.se

Barbara Gallina
IDT, Mälardalen University,
Västerås, Sweden
Email: barbara.gallina@mdh.se

***Abstract*—The growing connectivity of the systems that we rely on e.g. transportation vehicles is pushing towards the introduction of new standards aimed at providing a baseline to address cybersecurity besides safety. If the interplay of the two normative spaces is not mastered, compliance management might become more time consuming and costly, preventing engineers from dedicating their energies to system engineering. In this paper, we build on top of previous work aimed at increasing efficiency and confidence in compliance management. More specifically, we contribute to building a terminological framework needed to enable the systematization of commonalities and variabilities within ISO 26262 and SAE J3061. Then, we focus our attention on the requirements for software design and implementation and we use defeasible logic to prove compliance. Based on the compliance checking results, we reveal reuse opportunities. Finally, we draw our conclusions and sketch future research directions.**

***Keywords*—ISO 26262, SAE J3061, compliance management, compliance proofs, reuse, defeasible logic**

I. Introduction

The growing connectivity of the systems that we rely on e.g. transportation vehicles is pushing towards the introduction of new standards aimed at providing a baseline to address cybersecurity besides safety. If the semantic interplay of the two normative spaces is not mastered, a twofold negative consequence needs to be faced: system's safety might be compromised and compliance management might become more time consuming and costly, preventing engineers from dedicating their energies to system engineering. To enable the mastery of the semantic interplay between the two normative spaces, various contributions have been provided. Bloomfield et al [1] coined the expression Security-informed Safety (shortened SiS) to highlight the truth that "if you are not secure, you are not safe". Bloomfied et al. also pointed out the necessity for a *lingua franca*. Gallina et al [2] built on top of Bloomfield et al 's work and elaborated on a possible SiS terminological framework aimed at revealing syntactical as well as semantic commonalities between safety and security. A SiS terminological framework opens an opportunity for synergy between the two normative spaces.

To increase efficiency in compliance management, Gallina et al [2] also introduced the notion of Security-informed Safety-oriented Process Line (SiSoPL) as an extension of Safety-oriented Process Lines (SoPLs) [3], [4]. Via SiSoPLs, the two normative spaces are treated as part of a single family, and their commonalities and variabilities systematized. Methodological guidelines to engineer (SiS)oPLE were also investigated [5]. More recently, to further increase efficiency and at the same time increase confidence, we proposed an approach, called SoPLE&Logic-basedCM [6], which combines Safety-oriented Process Line Engineering (SoPLE), Defeasible Logic [7], and the approach for compliance management (CM) proposed for business processes [8].

In this paper, we build on top of our previous work aimed at increasing efficiency and confidence in compliance management [2], [5], [6]. More specifically, first we contribute to building a terminological framework needed to enable the systematization of commonalities and variabilities within ISO 26262 and SAE J3061, i.e., a contribution towards an automotive SiS terminological framework and a possible alignment of safety and cybersecurity life-cycles highlighting the key role of the architecture. Second, we focus our attention on the requirements for software design and implementation and we create an automotive SiSoPLE. Then, we use defeasible logic combined with the approach for compliance by design to prove compliance. Based on the compliance checking results, we reveal reuse opportunities. Finally, we draw our conclusions and sketch future research directions.

The rest of the paper is organized as follows. In Section II, we provide background information related to our work. In Section III, we present SiSoPLE&Logic-basedCM, the adaptation of SoPLE&Logic-basedCM [6] in the context of SiS normative space. In Section IV, we apply our approach for identifying reusable compliance proofs between the automotive standards. In Section V, we discuss our findings. In Section VI, we discuss related work. Finally, in Section VII, we present conclusions and future work.

II. Background

This section presents the background required in this paper. Section II-A, recalls key role of the (software) architecture in the lifecycle process. In Sections II-B and II-C, we recall the two automotive standards currently in use for addressing functional safety and cybersecurity. Finally, in Section II-D, we present SoPLE&Logic-basedCM, our previously introduced approach for increasing efficiency and confidence.

A. Software Architecture in the Lifecycle

Software architecture [9] is: *"a collection of software and system components, connections and constrains, a collection of system stakeholders' needs, and a rationale which* *demonstrates that the components, connections, and constraints define a system that, if implemented, would satisfy the collection of system stakeholders' needs statements."* In contrast with classical definitions, this definition highlights the relevance of the architecture's rationale. The reasoning behind the software structure provides a connection between the components description and the stakeholder's needs. This connection is required since the various stakeholders involved in the creation of the software have different expectations from the software architecture. For instance, the customer may expect at the architecture stage an estimate of cost. However, users need software architecture to clarify their requirements, while architects and system engineers are concerned with translating requirements into high-level design. For developers, the architecture is a reference for developing and assembling components. Safety and cybersecurity engineers, jointly with architects, use the architecture to identify and design the trade-offs due to the interplay of safety and security. Architecture is also useful when pre-existing component are evaluated for reuse. In this sense, the architecture can serve as the key milestone during the entire lifecycle process, evolving in every stage according to the stakeholder's perspectives. Moreover, software architecture concerns affect the gathering of requirements, the design decisions, the validation and capturing of the design and the transformation of the design into implementation [10]. Therefore, there is a reciprocal relationship between the software architecture and the process lifecycle.

B. ISO 26262

ISO 26262, Road Vehicles-Functional Safety [11], is a standard that focuses on electric/electronic systems located in vehicles with maximum gross mass up to 3500 kg. ISO 26262 uses *ASIL* (Automotive Safety Integrity Levels) to specify applicable requirements of ISO 26262 and safety measures to apply (activities or technical solutions to avoid or control systematic failures or random hardware failures, or mitigate their harmful effects). Functional safety is influenced by the development lifecycle process. Therefore, ISO 26262 specifies a *safety lifecycle* that comprises the entirety of phases from concept through decommissioning of the system. ISO 26262 safety lifecycle is based upon a V-model [12]. Planning, coordinating and documenting the *safety activities* of all phases of the safety lifecycle are key management tasks during the implementation of ISO 26262. Based on the item definition, a description of the system with regard to its functionality, interfaces, environment, etc., the safety lifecycle is initiated. An *item* is a system or array of systems to implement a function at the vehicle level, to which ISO 26262 is applied. After the initiation of the safety lifecycle, *HARA* (Hazard Analysis and Risk Assessment) is performed to categorize *hazardous events*, the combination of *hazard* (potential source of harm) and *operational situations* (scenarios that can occur during vehicle's life). *Harm* is defined as the physical injury or damage to the health of persons. The results of HARA are the *safety goals* (top-level safety requirements). *Functional safety requirements* (implementation-independent safety requirements) are derived from the safety goals and refined into *technical safety requirements* (implementation-specific safety requirements), at the system level. Further, the technical safety requirements are refined into *software safety requirements*. The *safety architecture* (set of elements and their interaction to fulfil the safety requirements) is created at the *software level*. The design of the software units is based on the previously defined safety architecture.

We focus on the activities and requirements for the sub-phase *Software Unit Design and Implementation*, specified in ISO 26262 part 6, clause 8, which defines three objectives. First, to specify software units in accordance with the software architectural design and the associated software safety requirements. Second, to implement the software units as specified. Third, to perform static verification of the design of the software units and their implementation. These three objectives characterize four ISO 26262-related activities (IA), namely: *IA1 Software unit design*, *IA2 Software unit design verification*, *IA3 Software unit implementation*, and *IA4 Software unit implementation verification*. The previous activities are associated to the ISO 26262-related requirements (IR) described in clause 8 of the standard (see Table I).

TABLE I. REQUIREMENTS FOR ISO26262.

Ref[1]	ID	Requirements description
8.2	IR1	Based on the software architectural design, the detailed design of the software units is developed.
	IR2	The detailed design will be implemented as a model or directly as source code, in accordance with the modelling or coding guidelines respectively.
	IR3	The implementation-related properties are achievable at the source code level if manual code development is used. If model-based development with automatic code generation is used, these properties apply to the model and need not apply to the source code.
	IR4	In order to develop a single software unit design both software safety requirements as well as all non-safety-related requirements are implemented. Hence in this sub-phase safety-related and non-safety-related requirements are handled within one development process.
8.4.1	IR5	The requirements of this subclause shall be complied with if the software unit is safety-related. Note: "Safety-related" means that the unit implements safety requirements
8.4.2	IR6	Software units are designed by using a notation that depends on the ASIL and the recommendation level.
8.4.3	IR7	The specification of the software units shall describe functional behaviour and internal design.
8.4.4	IR8	Design principles for software unit design and implementation shall be applied depending on the ASIL and the recommendation levels.
8.4.5	IR9	Software unit design and implementation are verified by applying methods according to the ASIL and the recommendation levels
4.2[2]	IR10	When ASIL and recommendation levels are not applied, a rationale must be provided.

C. SAE J3061

SAE J3061 [14] consist 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[3], but the definition of Automotive Cybersecurity Integrity Level (ACsIL)[4] 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

[1]Reference to ISO 26262 clauses

[2]The interpretation of clause 4.2, a general clause that applies to all the parts of ISO 26262, is provided in [13].

[3]http://standards.sae.org/j3061_201601/

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

unauthorized access or attacks. A *threat* is a circumstance or event with the potential to cause harm, where *harm* may be respect to 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. The cybersecurity lifecycle initiates at the concept phase with the *feature definition* in which the scope of the *feature* (a system or an array of systems to implement a function at the vehicle level to which a cybersecurity process is applied) is specified with respect the physical boundaries, cybersecurity perimeter, and trust boundaries of the feature. After the initiation of the lifecycle, *TARA* (Threat Analysis and Risk Assessment) is performed. TARA is an analysis technique applied to help identify potential threats to a feature and to assess the risk associated with the identified threats. *Cybersecurity goals*, the highest level cybersecurity requirements for achieving cybersecurity for the feature, are identified for each of the highest risk potential threats resulting in the TARA. Once the cybersecurity goals are established, the *functional cybersecurity requirements* are determined (during cybersecurity concept). A vulnerability analysis, which is designed to find areas where an attack is likely to occur, is performed at the system level. The result of the vulnerability analysis in conjunction with the functional cybersecurity requirements will be used to create a *technical cybersecurity concept*, which is the base for defining the *technical cybersecurity requirements* (implementation-specific requirements) at system level. The technical Cybersecurity requirements are used to understand the software support to the overall system purpose including the cybersecurity functions at software level. As a result *software cybersecurity requirements* are defined and allocated to the software architecture. Software unit design and implementation are then realized based on the software architecture. 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.

We focus our attention on SAE J3061, section 8.6.5, which describes the *Software Unit Design and Implementation* phase, and section 6.3, which describes the *review* activities that apply to the mentioned phase. These two sections define four SAE J3061-related activities (JA), namely *JA1 Software unit design*, *JA2 Software unit implementation*, *JA3 Design review* and *JA4 Implementation review*. The previous activities are associated to the SAE J3061-related requirements (JR) which are described in several parts of the guidebook (see Table II).

TABLE II. REQUIREMENTS FOR SAE J3016.

Ref[5]	ID	Requirements description
6.2.3.3	JR1	Software unit design implementation is based on the Cybersecurity requirements allocated in the software architectural design
6.3	JR2	Design and implementation reviews comprises activities analysis, review and refine Cybersecurity assessment.
8.2	JR3	Include the Cybersecurity activities described in this document[6] for each lifecycle phase, with the corresponding activities for each lifecycle phase described in the safety process.
8.6.1[7]	JR4	Use the extensive tables and methods provided by ISO 26262 Part 6 for design and implementation. The selection of methods and the rationale are recorded in the documented planning of software development.
8.6.5		During software design and implementation, good coding practices should be followed. Many of the methods are described in ISO 26262 and are called design principles .

D. SoPLE&Logic-basedCM

SoPLE&Logic-basedCM [6] is an approach for providing reusable compliance proofs. Three are the main components of this approach. **SoPLE** [3], Safety-oriented Process Line, is a methodological framework for modelling commonalities (equal process elements) and variabilities (process elements that vary) that are present in safety processes. SoPLE is constituted of two phases: one aimed at engineering reusable safety process-related commonalities and variabilities, and the second aimed at engineering single safety processes via selection and composition of previously engineered reusable process elements. Units of work for the processes provided by the standards are called different (e.g. activities/tasks). Therefore, the approach first proposes to align the work breaking down structures and use the same terms by adopting SPEM2.0 as common process modelling language. Common aspects constitute the commonality points (CP) while variability points (VP) are the process elements that are replaced with particular instances of process elements. **SiSoPLE**, Security-informed Safety-oriented Process line is an extension of SoPLE, in which it is expected to enable the alignment of security with safety. Therefore, for creating a SiSoPL it is required to overcome irrelevant terminological differences, through the definition of a common terminological framework between safety and security. **Defeasible Logic** [7], is a non-monotonic logic to formalize defeasible reasoning, reasoning with incomplete and inconsistent information. Defeasible theories are the knowledge base in defeasible logic. A defeasible theory 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 superiority relation is symbolized by $>$. **Compliance Management (CM) by design** is an approach in which the compliance of a process with a norm is verified before the process is deployed. The abstract formal framework for modeling *compliance by design* defined in [8], [15] is recalled in this section. This approach is based on deontic logic of violations [16], in which deontic notions are modelled using defeasible logics. One deontic notion that is present in normative systems is the *obligation*. An obligation constraint the bearer to a specific situation. Obligations are attached to process elements (i.e., activities) by semantic annotations (functions that describe the environment in which a process operates). Two semantic annotations are used in this framework: *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* $\{p\}$.

[5]Reference to SAE J3061 numerals

[6]Document in this case refers to SAE J3061

[7]The part selected is called *Selection of Methods*

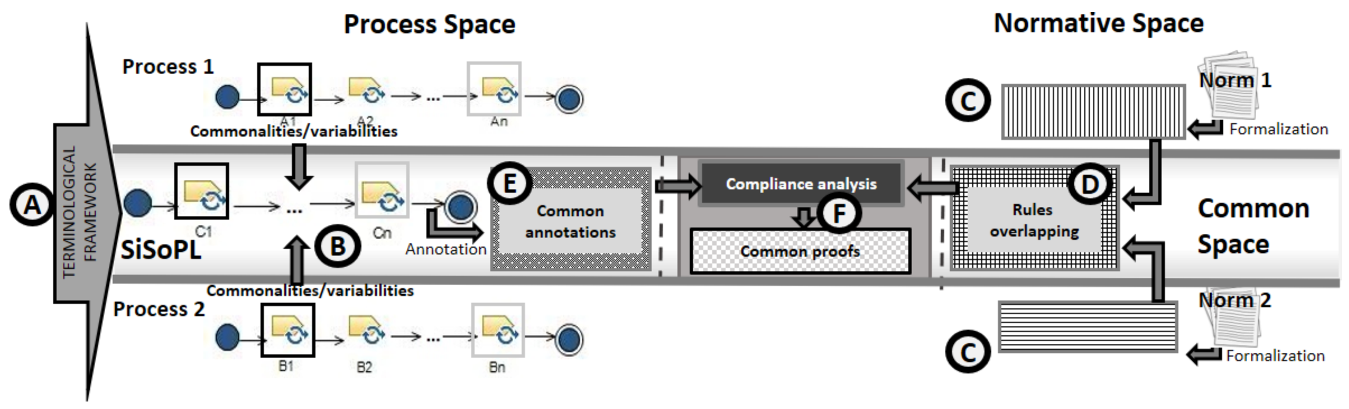

Fig. 1. SiSoPLE&Logic-basedCM framework.

III. SiSoPLE&Logic-basedCM

SiSople&Logic-basedCM, the approach presented in this paper, is a variation of Sople&Logic-basedCM, recalled in Section II-D. This variation adds the benefits of Security-informed Safety-oriented Process Line [2] by replacing SoPLs with SiSoPLs, a step that involves the definition of a common terminological framework. An overview of the approach is presented in Figure 1. To apply the methodology, a process engineer is expected to:

A define a common terminological framework by studying the similarities between the concepts presented in the standards,

B model a SiSoPL, by comparing the process elements that are present in the process lifecycle provided by the standards (called also Process Reference Model), and extracting the commonalities and variability points,

C formalize the rules by defining defeasible theories (studying the requirements provided by the standards and transform them into defeasible rules),

D compare defeasible theories and define the overlapping set.

E annotate the SiSoPL, using *Ann* to describe the sequencing of activities in the process, and *Force* to add the effects of the defeasible rules, and

F analyze the compliance of the SiSoPL with the common set of rules and define common proofs.

IV. Applying SiSoPLE&Logic-basedCM

We apply our approach to an overlapping portion of the process reference models provided for ISO 26262 and SAE J3061. This portion is small, but it provides enough information to illustrate our approach. It is structured as follows. In Section IV-A, we present an automotive-SiS terminological framework. In Section IV-B, we model a SiSoPL. In Section IV-C, we define formalize the standards rules. In Section IV-D, we define the overlapping set of rules between the two standards. In Section IV-E, we annotate the SiSoPL with the common rules. Finally, in Section IV-F, we analize the compliance of the SiSoPL.

A. Automotive-SiS Terminological Framework

We aim at offering an initial brainstorming on automotive-SiS by looking at two aspects. First, the underlying similarities between the concepts presented in Sections II-B and II-C. Second, the evident commonalities among process elements in both standards, taking into account that SAE J3061 bases its process reference model in the one provided by ISO 26262.

To represent a system in a vehicle, ISO 26262 uses the term *item* while SAE J3061 uses the term *feature*. Therefore, ISO 26262 and SAE J3061 are meant to be applied to a similar computational structure. An *attack* to a cybersecurity-critical system has the potential to produce a system failure. A similar potential is found for an *external fault* in a safety-critical system. The concept of *harm* is explicitly described in both standards, referring to a kind of loss. A potential source of harm is known in safety as a *hazard*, while in Cybersecurity it is knows as *threat*. *HARA* and *TARA* are phases in the lifecycle that are also similar, in the sense that they provide common techniques to mitigate a potential source of harm. HARA is the base for the definition of *safety goals*, and the provision of *safety measures*, while TARA is used to define *cybersecurity goals* and provide *cybersecurity measures*. Goals, in both standards, are the top-level requirements which are decomposed in more refined requirements during the lifecycle stages. When both standards are applied, the architecture evolves in an similar way. Initially, both processes require a high level system description (*item/feature definition*), which is used to generate *preliminary architectural assumptions*. During system level, technical requirements are allocated into the *system architecture* which is then refined into *software architectural design* at the software level. Process phases and their relationship with the requirements decomposition and the architectural evolution is presented in Figure 2.

B. SiSoPL Modeling

To define a SiSoPL, we compare the activities that are present in the process reference model of ISO 26262 and SAE J3061. This comparison is based on the descriptions given in natural language in Sections II-B and II-C, and the automotive-SiS terminological framework described in Section IV-A. The phase selected is the *software unit design and specification*, highlighted in Figure 2. The selected phase is populated in both processes by common steps, defining

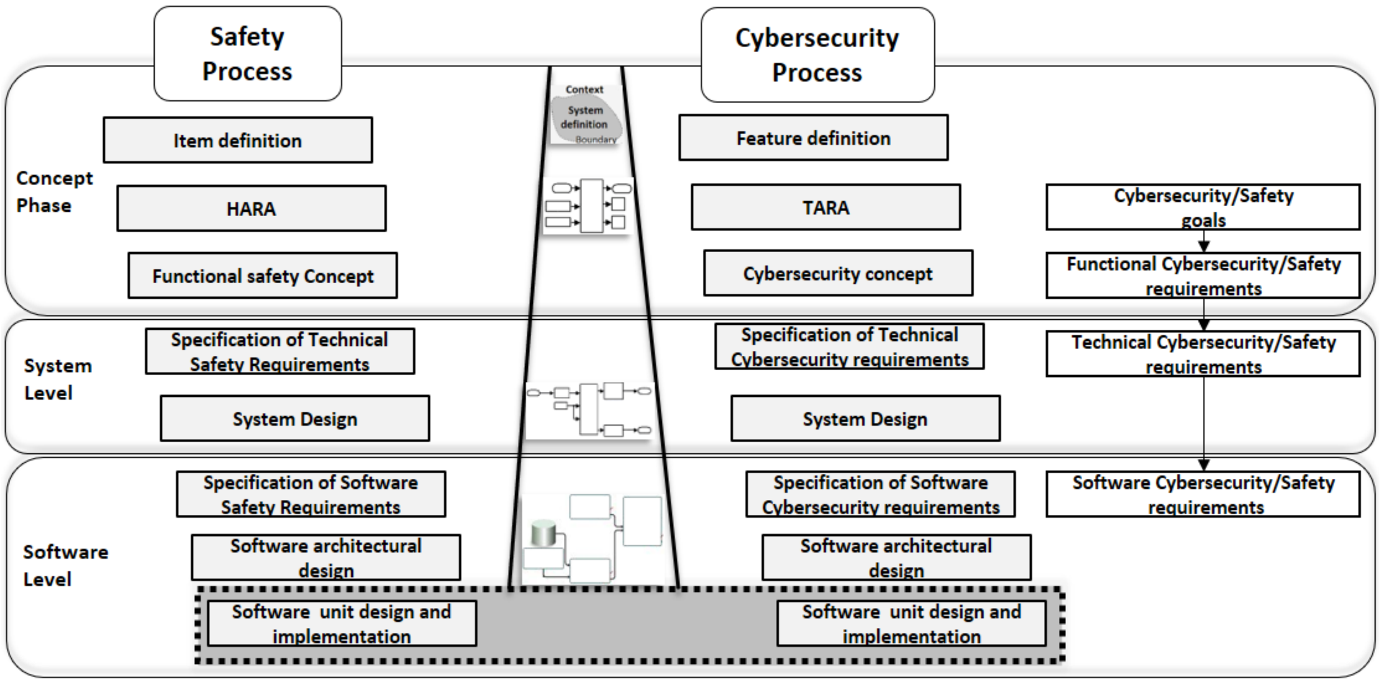
Safety Process
Cybersecurity Process
Concept Phase
Item definition
HARA
Functional safety Concept
Context
System definition
Boundary
Feature definition
TARA
Cybersecurity concept
Cybersecurity/Safety goals
Functional Cybersecurity/Safety requirements
System Level
Specification of Technical Safety Requirements
System Design
Specification of Technical Cybersecurity requirements
System Design
Technical Cybersecurity/Safety requirements
Software Level
Specification of Software Safety Requirements
Software architectural design
Software unit design and implementation
Specification of Software Cybersecurity requirements
Software architectural design
Software unit design and implementation
Software Cybersecurity/Safety requirements

Fig. 2. Process elements comparison between SAE J3061 and ISO 26262.

partial commonality-tasks, characterized by a variation point that takes into consideration the variability. For example, the activity *software unit design* is a partial commonality point (CP) in both processes. However, for safety, safety-related units are designed, while cybersecurity-related units are designed for cybersecurity. This kind of specificities are referred as the variability points (VP) in the model. The commonalities identification is presented in Table 3 and the resulting SiSoPL is presented in Figure 3.

TABLE III. 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

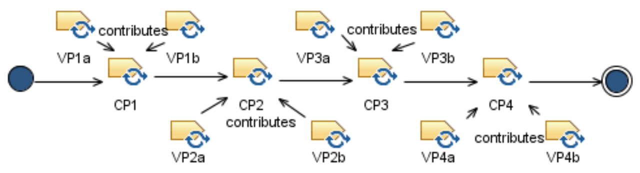

Fig. 3. SiSoPL model.

C. Formalising the rules

To formalise the rules, we take the set of requirements presented in each standard, organize them according to the activity to which they are applied[8], and decompose them in atomic expressions[9]. For example requirement *IR1* (see Table I) can be decomposed in two atomic expressions, namely, *1) There is a software design and implementation phase (sdip)*[10], and *2) (sdip) have available the software architectural design (sad)*. The first atomic expression corresponds to a fact, and the second atomic expression corresponds to a strict rule. Facts and strict rules are indisputable statements, and both are represented as strict rules, using the symbol $\rightarrow$ (see R1 and R2 in Table IV). The difference in the representation of strict rules and facts is that facts do not have antecedent. When the atomic expression refers to a rule that is weakened by other rule, it is represented as a defeasible rule. For example, requirement *IR2* (see Table I) can be decomposed in the atomic expressions: *1) A software unit (sui) is usually implemented as a model (im)* and *2) A software unit (sui) is usually implemented as source code (isc)*. As seen the adverb *"usually"* is added to the description of the rule. The defeasible rule is defined using the symbol $\Rightarrow$ (see R27 and R30 in Table IV). Rules that prevent conclusions (defeaters) are defined using the symbol $\rightsquigarrow$, and the verb *"prevents"* is added to the description of the rule. An example of this type or rules can be found with the two defeasible rules described before: *Implementing as a model (im) prevents the direct implementation as a source code (-isc)* (see R33 in Table IV). Priority relations are created when there are two rules that are in conflict. An example of a priority relations is created when modelling requirements IR6 and IR10 for ISO 26262 (see Table I). The formalization of these two

[8]This step makes the rules derived from the standards requirements appear in a different order.

[9]An atomic expression, in the context of this paper, is an expression that is equivalent to a proposition in logic (statement which is either true or false, but not both [17]).

[10]For every atom (or variable) presented in the rule, we define acronyms, so the visualization of the defeasible rules is easier.

rules results in the rules R10, R11 and R12 in Table IV and the Superiority relation SR1 in Table V. R10 implies that the design notations for safety-related units have to be selected according to ASIL and recommendations levels while R12 implies that if there is a rationale (R11)[11], the design notations for safety-related units does not need to be done according to ASIL and recommendation levels. It means that R10 and R12 are in conflict if R11 is fulfilled. However the superiority relation SR1, resolves this conflict, giving priority to R10. Rules for both ISO 26262 and SAE J3061 are presented in Table V, and the superiority relations for ISO 26262[12] are presented in Table V.

D. Definition of the overlapping set of rules

Requirement JR3 in SAE J3061 (see Table II) prescribes the inclusion of Cybersecurity activities for each lifecycle phase described in the safety process. Moreover, requirement JR4 prescribes also the use of ISO 26262 methods and rationale. For this reason, the rules derived for ISO 26262, that are not specifically safety-related, are adopted for modelling SAE J3061 rules. Therefore, rules that apply to process elements in ISO 26262 are inherited by the process elements in SAE J3061. Safety-related rules are only applied to ISO 26262 process lifecycle and cybersecurity-related rules are only applied to SAE J3061 process lifecylce. The defeasible theory that applies to both standards is presented in Table IV, which includes the common rules that apply to both standards, as well as the standard-specific rules (dark-gray for ISO 26262 and light-gray for SAE J3061).

E. Annotating the SiSoPL

Formalized rules are annotated in the corresponding process activities defined in the SiSoPL. For instance, R1 is a rule common rule (presented in both standards) related to design. Therefore, R1 is annotated to the activity *Unit design* that corresponds to the commonality CP1. In the same way, standard-specific rules are annotated to standard-specific tasks, e.g., R3, which is an ISO 26262-related rule is annotated to the task VP1a, which is a ISO 26262-related task. An abstraction of the annotation process is presented in the Figure 4.

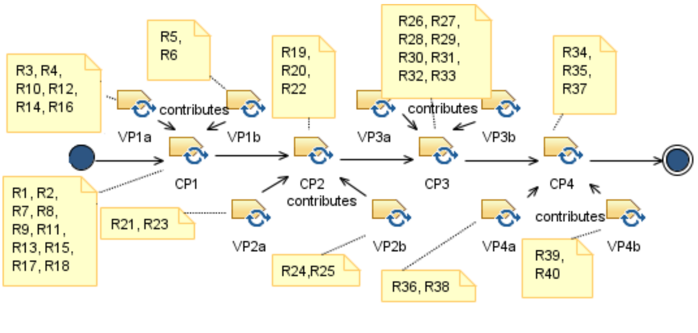

Fig. 4. Annotated SiSoPL.

The initial part of the annotation process requires the determination of the states of the tasks and their effects on

[11]A valid rationale that explains why this requirement does not apply, or a rationale that explains how tailoring activities make that the requirement is not required

[12]There were not superiority relations for SAE J3061

TABLE IV. DEFEASIBLE THEORY FOR ISO 26262 AND SAE J3061.

ID	Req.	Rule	Rule description
R1	IR1/JR1	$\rightarrow$ sdip	There is a software design and implementation phase (sdip).
R2		sdip $\rightarrow$ sad	(sdip) have available the software architectural design (sad).
R3	IR4-5	sdip $\rightarrow$ ssr	(sdip) have available the software safety requirements (ssr).
R4		sad, ssr $\rightarrow$ sru	(sad) and (ssr) are used to design safety-related software units (su).
R5	JR1	sdip $\rightarrow$ scr	(sdip) have available software cybersecurity requirements (scr).
R6		sad,scr $\rightarrow$ cru	(sad) and (scr) are used to design cybersecurity-related units (cru)
R7	IR4-5/JR1	sad,-ssr,-scr $\rightarrow$ nscru	If (sad) is provided but (ssr) and (scr) are not provided, non safety/non cybersecurity-related (nscru) are designed.
R8	IR6/JR3	sdip $\rightarrow$ dsu	(sdip) has an activity called design of software unit (dsu).
R9	IR6/JR4	dsu $\rightarrow$ dn	(dsu) is done using design notations (dn).
R10	IR6	dn,sru $\Rightarrow$ dnArl	(dn) for (sru) are usually selected according to ASIL and recommendation levels (dnArl).
R11	IR6-10/JR4	$\Rightarrow$ rdn	There is usually a rationale about design notations (rdn).
R12	IR6/10	dn,sru,rdn $\Rightarrow$ -dnArl	Rationale (rdn) about (dn) for (sru) is provided and (dnrl) is usually not required.
R13	IR8/JR4	dsu $\rightarrow$ dp	(dsu) is done using design principles (dp).
R14	IR8	dp,sru $\Rightarrow$ dpArl	(dp) for (sru) are usually selected according to ASIL and recommendation levels (dpArl).
R15	IR8-10/JR4	$\Rightarrow$ rdP	There is usually a rationale about design principles (rdp).
R16	IR8-10	dp,sru,rdp $\rightarrow$ -dpArl	Rationale (rdp) about (dp) for (sru) is provided and (dpArl) is usually not required.
R17	IR7/JR3	dsu $\rightarrow$ sufb	(dsu) describes software unit functional behavior (sufb).
R18		dsu $\rightarrow$ suid	(dsu) describes software unit internal design (suid).
R19	IR9/JR3	dsu $\rightarrow$ sudv	(dsu) have a software unit design verification (sudv).
R20	IR9/JR4	sudv $\rightarrow$ vm	(sudv) is done using verification methods (vm).
R21	IR9	vm, sru $\Rightarrow$ vmArl	(vm) for (sru) are usually selected according to ASIL and recommendation levels (vmArl).
R22	IR9-10/JR4	$\Rightarrow$ rvm	There is usually a rationale about verification methods (rvm).
R23	IR9/10	vm,sru,rvm $\Rightarrow$ -vmArl	Rationale (rvm) about (vm) for (sru) is provided and (vmArl) is usually not required.
R24	JR2	sudv, scr $\rightarrow$ cdaa	(sudv) for (scr) have available design activities analysis (cdaa).
R25		sudv, scr $\rightarrow$ cdar	(sudv) for (scr) have cybersecurity design assessment refinement (cdar).
R26	IR2/JR3	dsu, sudv $\rightarrow$ sui	after design (dsu) and design verification (sudv), software unit implementation (sui) is done.
R27		sui $\Rightarrow$ im	(sui) is usually implemented as a model (im).
R28		im $\rightarrow$ mg	(im) have available modelling guidelines (mg).
R29	IR3/JR3	im $\rightarrow$ irp	(im) achieve implementation-related properties (irp).
R30	IR2/JR3	sui $\Rightarrow$ isc	(sui) is usually implemented as source code(isc).
R31		isc $\rightarrow$ scg	(sc) have available source code guidelines (scg).
R32	IR3/JR3	isc $\rightarrow$ irp	(isc) achieve (irp).
R33	IR2/JR3	im $\rightsquigarrow$ -isc	implementing as a model (im) prevents the direct implementation as a source code (-isc).
R34	IR9/JR4	sui $\rightarrow$ suiv	(sui) has a software unit implementation verification (suiv).
R35		suiv $\Rightarrow$ ivm.	(suiv) is done using implementation verification methods (ivm)
R36	IR9	ivm, sru $\Rightarrow$ ivmArl	(ivm) for (sru) are usually selected according to ASIL and recommendation levels (ivmArl).
R37	IR9-10/JR4	$\Rightarrow$ rivm	There is usually a rationale about implementation verification methods (rivm).
R38	IR9-10	ivm,sru,rivm $\Rightarrow$ -ivmArl	Rationale (rivm) about (ivm) for (sru) is provided and (ivmArl) is usually not required.
R39	JR2	ivm, scr $\rightarrow$ ciaa	(ivm) for (scr) have a cybersecurity implementation activities analysis (ciaa).
R40		ivm, scr $\rightarrow$ ciar	(ivm) for (scr) have a cybersecurity implementation assessment refinement (ciar).

other tasks. We, first, want to find the compliance of the commonality points (CPs), in order to understand the reuse possibilities. For this, we annotate the SiSoPL model, which

TABLE V. SUPERIORITY RELATIONS FOR ISO 26262 RULES.

ID	Req.	Rule	Rule description
SR1	IR6	r10>r12	Selecting (dn) according to ASIL and reccomendation levels have priority over not doing it an include rationale.
SR2	IR8	r14>r16	Selecting (dp) according to ASIL and reccomendation levels have priority over not doing it an include rationale.
SR3	IR9	r21>r23	Selecting (vm) according to ASIL and reccomendation levels have priority over not doing it an include rationale.
SR4		r36>r38	Selecting (ivm) according to ASIL and reccomendation levels have priority over not doing it an include rationale.

is constituted by one trace i.e. *tSiSoPL*, using the function *Ann* (See Listing 1).

```
Ann(tSiSoPL,CP1,1)={CP1}
Ann(tSiSoPL,CP2,2)=Ann(tSiSoPL,CP1,1)U{CP2}
Ann(tSiSoPL,CP3,3)=Ann(tSiSoPL,CP2,4)U{CP3}
Ann(tSiSoPL,VP3,6)=Ann(tSiSoPL,CP3,5)U{CP4}
```

Listing 1. *tSiSoPL* states.

A task determines its state taking its effect and inheriting the previous effects. For example the state of CP2 depends on the state of CP1, i.e., if CP1 is not executed, CP2 is also not executed. Once the states are determined, the obligations in force (rules that apply to the tasks) are assigned, using the function *Force*. Since the *tSiSoPL* is composed by CPs, the defeasible rules that can be annotated are those that are common to both processes, i.e., rules that are not highlighted in Table IV (See Listing 2).

```
Force(tSiSoPL,CP1,1)={R1}U{R2}U{R7}U{R9}{R11}U{R13}U{R15}U{
    R17}U{R18}
Force(tSiSoPL,CP2,2)=Force(tSiSoPL,CP1,1)U{R19}U{R20}U{R22}
Force(tSiSoPL,CP3,3)=Force(tSiSoPL,CP2,2)U{R26}U{R27}U{R28}
    U{R29}U{R30}U{R31}U{R32}U{R33}
Force(tSiSoPL,CP4,4)=Force(tSiSoPL,CP3,3)U{R34}U{R35}U{R37}
```

Listing 2. Obligations in force for the trace *tSiSoPL*.

F. *Analizing the compliance of the SiSoPL*

To be compliant, every task in the trace must be compliant, meaning that every rule that applies to the task must be true. Rules are true if there is an element in the task that fulfill the rule, e.g. specific tasks steps. For instance, nine rules applied to the task CP1, which means that nine task elements are defined in CP1, so that the rules are fulfilled in CP1 and, therefore, it is compliant. Particularly, for fulfilling R1 a step called *"Obtain a software architectural design"* should exist. Once the SiSoPL is determined as compliant, the standard-specific tasks are deployed and the standard-specific rules are annotated. The corresponding analysis of compliance of the standard-specific process just required the compliance analysis of the new elements added e.g. standard-specific elements. For instance, in Listing 3, ISO 26262-specific tasks are deployed and annotated with ISO 26262-specific rules. As seen, the first part of the annotation includes the obligations in force for the SiSoPL. Since the SiSoPL was previously analysed and defined as compliant, only the specific rules need to be analyzed.

```
Force(ISO26262,IA1,1)=Force(tSiSoPL,CP1,1)U{R3}U{R4}U{R10}U
    {R12}U{R14}U{R16}
Force(ISO26262,IA2,2)=Force(tSiSoPL,CP2,2)UForce(ISO26262,
    IA1,1)U{R21}U{R23}
Force(ISO26262,IA3,3)=Force(tSiSoPL,CP3,3)UForce(ISO26262,
    IA2,2)
Force(ISO26262,IA4,4)=Force(tSiSoPL,CP4,4)UForce(ISO26262,
    IA3,3)U{R26}U{R38}
```

Listing 3. Obligations in force for ISO 26262.

V. DISCUSSION

From the application of SiSople&Logic-basedCM, despite the simplicity of our example, we have identified some points that are worth to discuss. Our first observation is related to the selection of the logic. Defeasible logic provides the elements and the flexibility required for working with normative systems, which are often difficult to interpret, inconsistent, or incomplete. The existence of the different kind of rules (fact, strict rule, defeasible rule and defeater) as well as the superiority relations have facilitated the formalization of safety and security standards, as well as the compliance analysis of the processes with the formalized rules. Our second observation is connected with the interpretation of standards requirements. As it is well known, standards requirements are difficult to interpret due to their presentation in natural language. The definition of atomic expressions, as a prerequisite for creating the formal rules, can be considered a step towards better interpretation of the standards requirements. If rules are correctly formalized, hidden rules, or inconsistencies between rules can be discovered. Even a different sequencing of the rules, that facilitate their application, can be spotted. In addition, the definition of atomic expressions can reveal the granularity level of the steps in a task, i.e., every rule make mandatory the existence of specific elements that fulfill the rule. Moreover, the analysis of the similarity/variability between two atomic expressions extracted from two different standards is easier than performing the same analysis in two requirements extracted from the same two standards. This observation leads to the conclusion that our approach has general soundness, since commonalities and variabilities present in the process reference models elements, i.e., tasks, as well as common and variable standards rules can be identified and modeled. Our last observation is related to reusability. Process elements commonalities aligned with common formalized rules enable compliance proof reuse. From the example, 34 rules apply to ISO 26262 and 30 apply to SAE J3061. However, 24 of the rules presented in Table IV are common rules, which presupposes, in this specific case (and assuming that the rules are correctly formalized), a high level of proofs reuse. Proof reuse can make the compliance management task more efficient since time and effort are saved.

VI. RELATED WORK

Related work regarding automated compliance checking was already discussed in [6], and works related to combine safety and security, as well as harmonization and cross fertilization are discussed in [2]. Therefore, in this paper we limit our attention to works that are focused in enabling the mastery of the semantic interplay between safety and security standards. Several efforts have been done in the past. For example, a technical note presenting the foundations for safety, security and survivability engineering [18] was carried out to clarify the similarities and differences between those concepts. A comparison between security and safety engineering process based on ISO 26262 and a generic standard for IT (ISO 15408) was carried out in [19]. A more recent and limited comparison between terms and process elements of the concept phase in ISO 26262 and SAE J3061 was done in [20]. In their work, the authors discuss initial terminology and process elements found in the concept phase of the two standards, e.g., item/feature, HARA/TARA. In our work, we have done a similar comparison, but we include more terms and more process elements. In addition, we take into consideration the requirements decomposition and the evolution of the architecture in the lifecycle, as well as the systematic reuse of process elements and standards rules.

VII. Conclusions and Future Work

In this paper, we have presented a variation of our previously introduced approach Sople&Logic-basedCM. Our variation, called SiSople&Logic-basedCM includes the elaboration of an automotive-SiS terminological framework, by initiating a brainstorming around common terminological aspects present in the automotive standards, as well as the commonalities identified in their process reference models. The identified automotive-SiS terminological framework is then used to enable the systematization of commonalities and variabilities within ISO 26262 and SAE J3061. To perform our approach, we have selected a restricted but generic portion of the automotive safety and cybersecurity standards, namely, the *software unit design and specification* phase. This portion was selected due to the expertise gained in their analysis in previous projects [6], [13]. From the application of our approach in this restricted portion, we have concluded that the interpretation of the standards can be benefited by the creation of defeasible theories, since there is an step in which atomic expressions, the base of the defeasible theories, are created. We also have found that our approach is sound, since commonalities and variabilities in both, the process space and the normative space can be identified and modelled. Additionally, the alignment of the mentioned spaces enables compliance proofs reuse.

As future work, as presented by [6], we aim at further develop SiSople&Logic-basedCM, as well as Sople&Logic-basedCM in various directions. First, to be able to generalize the approach, we plan to apply it in more significant portions of the automotive standards ISO 26262 and SAE J3061. Second, we have applied standards rules to process activities. However, there are other process elements e.g. work products, guidelines, roles, required to be compliant, that have not being addressed yet. Therefore, we plan to address these elements in future extensions of our approach. Third, we will further explore methodologies and/or strategies to be make the formalization of the rules more accurate. Four, we have use the deontic notion *Obligation*, but other deontic notions (permission, prohibition) and their effects in safety/cybersecurity standards are planned to be studied. Fifth, we have applied our approach manually, but tools support is required. Therefore, we plan to explore tools that addressed defeasible logic reasoning and their adaptability to the current tool used for modelling SoPLE and SiSople, namely SPEM2.0/EPF composer[13].

Acknowledgments.: This work is supported by the EU and VINNOVA via the ECSEL JU project AMASS (No. 692474) [21].

References

[1] R. Bloomfield, K. Netkachova, and R. Stroud, "Security-Informed Safety : If It's Not Secure , It's Not Safe," in *International Workshop on Software Engineering for Resilient Systems*, 2013, pp. 17–32.

[2] B. Gallina and L. Fabre, "Benefits of security-informed safety-oriented process line engineering," in *IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, 2015, pp. 8C11–8C19.

[3] B. Gallina, I. Sljivo, and O. Jaradat, "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, pp. 148–157.

[4] B. Gallina, S. Kashiyarandi, H. Martin, and R. Bramberger, "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, pp. 504–509.

[5] I. Ayala and B. Gallina, "Towards tool-based security-informed safety oriented process line engineering," in *Proccedings of the 10th European Conference on Software Architecture Workshops (ECSAW)*, 2016, pp. 1–7.

[6] J. P. Castellanos Ardila and B. Gallina, "Towards Increased Efficiency and Confidence in Process Compliance," in *24th European & Asian Systems, Software &Service Process Improvement & Innovation*, 2017, p. 12.

[7] G. Antoniou, D. Billington, G. Governatori, and M. J. Maher, "Representation Results for Defeasible Logic," *ACM Transactions on Computational Logic*, no. 2, pp. 255–287, 2000.

[8] G. Governatori and S. Sadiq, "The Journey to Business Process Compliance," *Public Law*, pp. 1–32, 2008.

[9] C. Gacek, A. Abd-Allah, B. Clark, and B. Boehm, "On the Definition of Software System Architecture," in *The First International Workshop on Architectures for Software Systems*, 1995, pp. 85–95.

[10] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, third edit ed. Addison Wesley, 2013.

[11] ISO 26262, "Road Vehicles-Functional Safety. International Standard," 2011.

[12] N. B. Ruparelia, "Software development lifecycle models," *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 3, pp. 8–13, 2010.

[13] B. Gallina, J. P. Castellanos Ardila, and M. Nyberg, "Towards Shaping ISO 26262-compliant Resources for OSLC-based Safety Case Creation," in *4th International Workshop on Critical Automotive Applications: Robustness & Safety*, Göteborg, Sweden, 2016, p. 4.

[14] SAE, "Surface Vehicle Recommended Practice," Tech. Rep., 2016.

[15] M. Hashmi, G. Governatori, and M. T. Wynn, "Normative Requirements for Regulatory Compliance: An Abstract Formal Framework," *Information Systems Frontiers*, pp. 429–455, 2016.

[16] G. Governatori, A. Rotolo, and G. Sartor, "Temporalised Normative Positions in Defeasible Logic," in *10th International Conference on Artificial Intelligence and Law (ICAIL)*, 2005, pp. 25–34.

[17] K. H. Rosen, "The foundations: Logic and Proofs," in *Discrete Mathematics and Its Applications*, 7th ed. New York: Mc Graw Hill, 2012, ch. 1, p. 1071.

[18] D. Firesmith, "Common concepts undelying safety, security and survivability engineering," DTIC Document, Tech. Rep., 2003.

[19] C. Robinson-Mallett, "Coordinating security and safety engineering processes in automotive electronics development," *9th Annual Cyber and Information Security Research Conference (CISR)*, pp. 45–48, 2014.

[20] C. Schmittner, Z. Ma, C. Reyes, O. Dillinger, and P. Puschner, "Using SAE J3061 for Automotive Security Requirement Engineering," in *International Conference on Computer Safety, Reliability, and Security*, 2016, pp. 157–170.

[21] AMASS, "Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems. http://www.amass-ecsel.eu/," Accessed: 2017-05-30.

[13] http://www.eclipse.org/epf/