Towards Shaping ISO 26262-compliant Resources for OSLC-based Safety Case Creation

Barbara Gallina and Julieth Patricia Castellanos Ardila
Mälardalen University,
P.O. Box 883, SE-72123 Västerås, Sweden

Mattias Nyberg
Scania AB,
Södertälje, Sweden

***Abstract*—Traceable documentation management represents a mandatory activity according to ISO 26262. This activity is also essential for the creation of an ISO 26262-compliant safety case, which is defined as a compilation of work products. OSLC represents a promising integration framework for enabling tool interoperability and thus seamless traceability and documentation management, including safety case creation and management. In this paper, we present a step related to our work aimed at offering an OSLC-based infrastructure enabling the automatic generation of safety case fragments. Our step consists of the identification, representation and shaping of resources needed to create the safety case. Finally, conclusion and perspectives for future work are also drawn.**

Keywords*—*ISO 26262, Documentation Management, Safety Cases, Open Services for Lifecycle Collaboration (OSLC), OSLC-compatible Constraint Languages.

I. Introduction

ISO 26262 [1] is a functional safety standard that targets the automotive domain. ISO 26262 defines a safety life-cycle to be adopted during the development of automotive safety-critical systems. As already observed in our previous work [2], ISO 26262 also proposes/imposes guidelines for managing hundreds of hundreds of documents. The documentation process is usually tightly coupled with the development life-cycle. Life-cycle's work products represent immediate as well as direct evidence to be used during the safety assessment process to support the claims about system's safety. More specifically, the left-hand side work products of the V-model (e.g., requirement specification) represent immediate evidence; while the right-hand side work products of the V-model (e.g., verification results) represent direct evidence. Improper documentation/evidence management may indirectly result in certification risk [3]. In the context of ISO 26262, for instance, the goal of the documentation process is to make documentation available: 1) during each phase of the entire safety lifecycle for the effective completion of the phases and verification activities; 2) for the management of functional safety, and 3) as an input to the functional safety assessment. More specifically, in part 10 of the standard it is stated that "the documents should be: a) precise and concise, b) structured in a clear manner, c) easy to understand by the intended users, and d) maintainable". In part 10.4.4, it is stated that the structure of the entire documentation should consider in-house procedures and working practices. It shall be organized to facilitate the search for relevant information. Managing information properly is crucial to enable the creation of safety cases. OSLC represents a promising integration framework for enabling tool interoperability and thus seamless traceability and documentation management, including safety case creation and management. In this paper, we present a step towards the concretization of our vision [2] aimed at offering an OSLC-based infrastructure enabling the automatic generation of safety case fragments. Our step consists of the identification, representation and shaping of resources needed to create the safety case. Our focus is limited to a tiny portion of the ISO 26262 left-hand side of the V-model. Despite the limitation of our investigation, our findings are generalizable to other parts of the standard. Thus, this work may be considered a seed towards the provision of an OSLC-based and ISO 26262-compliant methodological approach for representing and shaping the crucial resources for safety case creation.

The rest of the paper is organized as follows. In Section II, we provide background information. In Section III, we identify the resources that are needed for the safety case compilation and we explain how they should be represented. In Section IV, we shape one resource by using three different constraints languages. In Section V, we discuss our findings and provide a summarizing comparative table. Finally, in Section VI, we present our concluding remarks and future work.

II. Background

In this section, we present the background information related to the problem space. In particular, in Section II-A, we recall ISO 26262 requirements related to software unit design specifications. In Section II-B, we recall essential information on OSLC and the semantic stack. In Section II-C, we present one OSLC compatible way of representing knowledge. Finally, in Section II-D, we present semantic web-compatible languages to properly constrain the knowledge to be represented.

A. ISO 26262-compliant Software Unit Design

In this section, to make the paper self-contained, we recall essential information related to ISO 26262-compliant software unit design specification. The software unit design specification is a mandatory work-product that should be produced according to the requirements stated in Part 6, 8.4.2-8.4.4. In this section we focus on a subset of requirements, those represented in Table 7 of Part 6, 8.4.2. This table is reproduced in Table I. From this table, it can be inferred that the work product Software Unit Design Specification should be characterized by the following properties: ASIL (Automotive Safety Integrity Level), notation, and recommendation level (to indicate for instance highly recommended ++ or simply recommended +). These three properties are closely related:

TABLE I. NOTATIONS FOR SOFTWARE UNIT DESIGN

Notation	A	B	C	D
Natural language	++	++	++	++
Informal notations	++	++	+	+
Semi-formal notations	+	++	++	++
Formal notations	+	+	+	+

the notation to be selected is constrained by the ASIL value and the recommendation level. This constraint is one among many that can be identified by reading the normative clauses of ISO 26262.

B. OSLC

As already recalled in [2], OSLC is a standard that targets tools used during a product's life cycle and enables their integration and interoperability. OSLC 2.0 [4] is the current mostly used version of OSLC. Tools for requirements engineering, design, implementation, verification, etc. are expected to interoperate in a traceable manner i.e. traceability between the respective work products can be easily retrieved and shown. To enable interoperability, different specifications, called domains, need to be provided. Requirements Management domain (RM) and Architecture Management (AM) represent samples of these domains. OSLC builds on top of Linked Data [5], Resource Description Framework (RDF) [6], and HTTP protocol. Each work product is described as an HTTP resource, identified via a Uniform Resource Identifier (URI). Manipulation of work products is performed via GET, PUT, POST and DELETE HTTP methods. To interoperate via a work product, a tool that acts as a provider has to associate an URI to the work product and post it; a tool acting as consumer can get the work product from the URI itself.

C. RDF-based Knowledge Representation

RDF is a framework that allows expressing information about resources on the Web in a machine-understandable format [7]. The RDF model is composed of two key data structures: RDF graph, also called triple (Fig. 1), and RDF datasets, which represent multiple data graphs, maintaining their content separate.

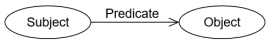

Fig. 1. RDF graphs representation [8].

In RDF terms, the subject is the thing being described (a resource identified by an URI), the predicate is a property type of the resource, and the object is equivalent to the value of the resource property type for the specific subject.

D. OSLC Compatible Constraint Languages

Various constraint languages exist or are being developed to restrict the content of RDF graphs. These languages support more or less the formulation of the numerous constraint types that have been recently identified [9]. Constraints defined together are called *shapes* and they are also used to validate RDF data, document RDF APIs and provide metadata to tools [10]. As OSLC supporting technologies are based on Linked Data principles and RDF representation, constraint languages that shapes RDF data can be explored. In this paper, we briefly recall information related to three popular and declarative constraints languages.

Resource Shape (ReSh) [10] is a constraint language that is part of the OSLC Core 2.0, where grammar rules written using RDF structure and OSLC terms are defined. The shape specification consists of one or more properties enclosed in the construct oslc:property.

Shape Expressions (ShEx) [11] is a constraint language that has it own syntax (called ShExc). It describes RDF graph structures, through a series of constraint rules. The rules can be written as a set of properties on a triple constraint. More specifically, the rules can be formulated as conjunctions of constraints separated by commas (,) and enclosed in brackets ({ }). These rules identify predicates, their associated cardinalities and datatypes, and evaluate the nodes that are referred in the instance data.

Shapes Constraint Language (SHACL) [12] is a language for constraining the content of RDF graphs (called nodes), grouping constraints into shapes, that specify the conditions that a RDF node must follow. SHACL has its own vocabulary (a shape, for example, is specified with the construct sh:shape), but it uses RDF and RDFS vocabulary to define types, classes, subclasses, properties, lists and resources.

III. RESOURCES IDENTIFICATION AND REPRESENTATION

The background has set the stage and introduced the main characters and their context. From the background, we have learnt that to enable tool interoperability and ultimately semi-automatic creation of safety cases, first of all the right resources to be exchanged have to be identified, represented and properly shaped. To identify and represent the right resources, we adopt the same approach that was discussed by Gallina et al. [13]. Briefly, this approach consists of translating into OSLC-resources all the mandatory work products including all their properties, described in the normative parts. In this paper, the result of this adoption, limited to the tiny portion of the ISO 26262 life-cycle, is depicted in Fig. 2. The reader may refer to [14] for a complete resources identification and representation related to ISO 26262, Part 6, clauses 8.4-6. More specifically, resources are offered within a new AM-like OSLC-domain (called iso26262am) that targets ISO 26262.

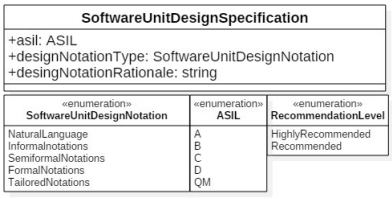

Fig. 2. Work Product Software Unit Design Specification (partial definition).

A. Constraint Definition

As pointed out in Section II-A, the work product Software Unit Design Specification, whose UML-like representation is depicted in Fig. 2, has three normative attributes. The attribute designNotationType shall have exactly one value, and this value shall be one among those listed in the enumeration

SoftwareUnitDesignNotation. In our representation, the enumeration is not limited to the values of Table I but includes an additional value to embrace industry best practices that can be negotiated or simply tailoring decisions. According to Table I, there are highly recommended and recommended notations, depending on the ASIL assigned to the software unit design specification (the attribute asil is exactly one value and it corresponds with one listed in the enumeration ASIL). If the designNotationType selected is not one of the highly recommended, the attribute designNotationRationale becomes mandatory. This attribute is added to ease self-assessment and arguments formulation. This attribute also enables fault tolerance in case of faulty normative tables. This constraint corresponds to the constraint called *Conditional property* in [15]. A conditional property can have several forms, including the following: if specific properties are present, then specific other properties also have to be present.

B. Constraint Formulation

The above information is used to formulate target groups, where nine sub-constraints are identified: two constraints (one that targets highly recommended and a second that targets recommended design notations) for each ASIL (A, B, C and D), and one constraint for ASIL QM (for this ASIL all the design notations are considered Highly Recommended).

IV. Shaping ISO 26262-compliant Resources

In this section, we present our solution. In Section IV-A, we present the shaping performed via Resource Shape. Then, in Section IV-B, we present the shaping performed via Shape Expressions, while the shaping performed via Shapes Constraint Language is presented in Section IV-C.

A. Shaping in Resource Shape (ReSh)

To define enumerations and cardinalities for the properties, Resh provides the constructs *oslc:allowedValues* and *oslc:occurs*. However, constructs for formulating property-values depending on other property-values (for example, the existence of the property-value *designNotationRationale*, depends on the property-value of *designnotationType*) do not exist in ReSh. As a workaround solution, we assign cardinality Zero-or-one the property *designNotationRationale*. The following commented RDF/XML-based fragment shows the ReSh definition for *softwareUnitDesignSpecification*.

```
<oslc:ResourceShape
  rdf:about="http://example.com/iso26262provider/shapes/
        SoftwareUnitDesignSpecificationShape">
<dcterms:title>Software Unit Design Specification Shape</dcterms:title>
<oslc:describes rdf:resource="http://open-services.net/ns/iso26262am#
      SoftwareUnitDesignSpecification"/>
<!--Property 1: asil-->
<oslc:property>
  <oslc:Property>
    <!--property name definition-->
    <oslc:name>asil</oslc:name>
    <!--property cardinality definition-->
    <oslc:occurs rdf:resource="http://open-service.net/ns/core#Exactly-one"/>
    <!--Property identification definition-->
    <oslc:propertyDefinition rdf:resource="http://open-services.net/ns/iso26262am#
        asil"/>}
    <!--Range and associated properties of the resource-->
    <oslc:valueType rdf:resource="http://open-services.net/ns/core#LocalResource "/>
    <oslc:representation rdf:resource="http://open-services.net/ns/core#Inline"/>
    <oslc:range rdf:resource="http://open-services.net/ns/core#Any"/>
    <!--List of values provided by the enumeration ASIL-->
    <oslc:allowedValue rdf:resource="http://open-services.net/ns/iso26262am#A"/>
    ... <!--values for asil B, C, D and QM are defined in the same way-->
  </oslc:Property>
</oslc:property>
<!--Property 2: designNotationType-->
<oslc:property>
   ... <!--similar structure that the given for asil-->
</oslc:property>
<!--Property 3: designNotationType-->
<oslc:property>
  <oslc:Property>
    <!--property cardinality Zero or one, for flexible use-->
    <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one"/>
    ...
</oslc:property>
</oslc:ResourceShape>
```

B. Shaping in Shape Expressions (ShEx)

In ShEx, the conditional property cannot be expressed as such. However, an alternative way to express that type of constraint exists i.e., by using the context-specific Exclusive OR, which outputs true whenever both inputs differ [15]. For exemplification purposes, we present a commented SHEXc-based fragment that constrains software units with Asil A.

```
<SoftwareUnitDesignSpecification> {
   # Assigns ASIL A to the software unit
   (shex_iso26262am:havingAsilA(shex_iso26262am:true),
   # Assigns highly Recommended type to the software unit
   shex_iso26262am:recommendationLevel(shex_iso26262am:highlyRecommended),
   # Restricts the selection of design notation type to those allowed for highly
        recommended
   shex_iso26262am:designNotationType(shex_iso26262am:naturalLanguage
        shex_iso26262am:informalNotations)| Operator Exclusive Or (|)
   # Assigns ASIL A to the software unit
   shex_iso26262am:havingAsilA(shex_iso26262am:true),
   # Assigns Recommended type to the software unit
   shex_iso26262am:recommendationLevel(shex_iso26262am:recommended),
   # Restricts the selection of design notation type to those allowed for
        recommended type
   shex_iso26262am:designNotationType(shex_iso26262am:semiformalNotations
        shex_iso26262am:formalNotations shex_iso26262am:TailoredNotations),
   #Makes the property designNotationRationale mandatory
   shex_iso26262am:designNotationRationale xsd:string|
#(...Same kind of constraints for ASIL B,C and D are defined here, QM restricts only
      to highly recommended level)
}
```

When ASIL A is selected, the value of the property havingAsilA has to be true. When *highlyRecommended* is assigned to *RecommendationLevel*, the *designNotationType* can only take a value within the subset constituted of *natural language* and *informal notations*. In case that the value *recommended* is assigned to *recommendationLevel*, the *designNotationType* can only take a value within the subset constituted of *semiformalNotations*, *formalNotations* and *TailoredNotations*, and the property *designNotationRationale* becomes mandatory.

C. Shaping in SHACL (Shapes Constraint Language)

SHACL provides the construct *sh:filterShape*, which can be used to limit the scope of the nodes that are required to be validated [12]. This characteristic allows the creation of nine filters that meet the requirements mentioned before. The below-given and commented RDF/XML snippet shows the shape for a software unit design specification in SHACL, where the ASIL is A, recommended design notation is selected, and thus the design notation rationale becomes mandatory.

```
shacl_iso26262am:DesignNotationShape_AsilARecommendedLevel
  rdf:type sh:Shape ;
  rdfs:label "DesignNotationShape_AsilARecommendedLevel"^^xsd:string ;
  sh:description "ASIL A, recommended notations and  rationale must be provided" ;
  sh:filterShape [
      rdf:type sh:Shape ;
      sh:property [
          sh:hasValue shacl_iso26262am:A ;
          sh:predicate shacl_iso26262am:asil ; ] ;
      sh:property [
          sh:hasValue shacl_iso26262am:Recommended ;
          sh:predicate shacl_iso26262am:recommendationLevel ; ] ; ] ;
  sh:property [
      sh:datatype xsd:string ;
      sh:minCount 1 ;
      sh:predicate shacl_iso26262am:designNotationRationale ; ] ;
  sh:property [
      sh:in (
          shacl_iso26262am:SemiformalNotations
          shacl_iso26262am:FormalNotations
          shacl_iso26262am:TailoredNotations) ;
      sh:minCount 1 ;
      sh:predicate shacl_iso26262am:designNotationType ; ] ;
  sh:scopeClass shacl_iso26262am:SoftwareUnitDesignSpecification ;
.
```

V. Discussion

Shapes composed of separately-defined properties in ReSh, make this language limited in its expressiveness. Instead, ShEx and SHACL present a structure that allows for defining properties in a more flexible way. When storing the information, the serialization used by ReSh and SHACL is totally compatible with the one used by OSLC Resouces: RDF/XML serialization, while ShEx presents its own serialization called ShExc, that requires a transformation for being able to use as a schema of OSCL resource (or convert the OSCL resources from RDF/XML in ShExc). The testing environment for ShEx and SHACL (Fancy Shex Demo [16] for the first one, and Top Braid Composer [17] for the second one) allows the implementation and assessment of the resources, while the lack of one for ReSh makes this task more difficult. These comparative elements are summarized in Table II.

TABLE II. Comparative study of RDF constraint languages

Comparative element	**ReSh**	**ShEx**	**SHACL**
Expressiveness	Low	High	High
RDF/XML syntax	Yes	No	Yes
Testing environment	No	Yes	Yes
Declarative representation	Yes	Yes	Yes
Tool support	Yes	Yes	Yes

Other comparative elements that are taken into account are the tools support and the declarative representation of the constraints. All the languages can be expressed using declarative constructs, but SHACL, additionally, supports native SPARQL constraints embedded into SHACL constructs like *sh:constraint*. The three languages also are tool-supported.

VI. Conclusion and Future Work

In this paper, we have presented a first step towards the concretization of our vision consisting of enabling the semi-automatic creation of ISO 26262-compliant safety cases thanks to the provision of an OSLC-based infrastructure. Our first step has focused on the identification, representation and shaping of a single resource. More specifically, we have identified an essential resource within ISO 26262-Part 6, we have represented it as an RDF graph and shaped according to three different constraint languages. Given the types of constraints that can be retrieved from the normative parts of the standards, we believe that SHACL should be preferred. The goal of this work was not completeness but moving forward our pioneering and explorative work. Despite the limitations of our investigation, we believe that our work may constitute a seed for the creation of ISO 26262-compliant and OSLC-based domains.

In the future, as presented by Gallina [18], we aim at moving an additional step forward by merging the results of this work (centred on the left-hand side of the V-model) with the work [13] centred on the right-hand side of the V-model. The intention is to exploit query-mechanisms to retrieve the necessary information from the resources in order to formulate a tiny argument-fragment aimed at arguing about a chunk of traceability between a software unit requirement and its corresponding testing results. The argument is expected to not only argue about process-based compliance (by showing that the required immediate evidence is connected to the corresponding and required direct evidence) but also argue about product-safety by showing that the traceable evidence is indeed effective in showing a product behaviour free from intolerable risk.

Acknowledgments: This work has been partially financially supported by the Swedish Foundation for Strategic Research via the SSF Gen&ReuseSafetyCases project [19] project. This work has also benefited from the discussions held in the context of the ECSEL Joint Undertaking project AMASS (No 692474) [20].

References

[1] ISO26262, "Road vehicles Functional safety. International Standard, November," 2011.

[2] B. Gallina and M. Nyberg, "Reconciling the iso 26262-compliant and the agile documentation management in the swedish context," in *Critical Automotive applications: Robustness & Safety (CARS), Matthieu Roy, Paris, France, HAL*, September 2015.

[3] S. Nair, J. de la Vara, A. Melzi, G. Tagliaferri, L. de-la Beaujardiere, and F. Belmonte, "Safety evidence traceability: Problem analysis and model," in *Requirements Engineering: Foundation for Software Quality*, ser. Lecture Notes in Computer Science, C. Salinesi and I. van de Weerd, Eds. Springer International Publishing, 2014, vol. 8396, pp. 309–324.

[4] Open Services for Lifecycle Collaboration Core Specification Version 2.0, "http://open-services.net/bin/view/main/oslccorespecification."

[5] Linked Data, "http://www.w3.org/designissues/linkeddata.html."

[6] RDF Primer, "http://www.w3.org/tr/rdf-primer/."

[7] S. Powers, *Practical RDF*. OReilly Media, 2003. [Online]. Available: https://www.safaribooksonline.com/library/view/practical-rdf/0596002637/index.html

[8] W3C, "RDF 1.1 Concepts and Abstract Syntax," 2014. [Online]. Available: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

[9] T. Bosch and K. Eckert, "Guidance, Please! Towards a Framework for RDF-based Constraint Languages," in *DCMI International Conference on Dublin Core and Metadata Applications, Sao Paulo, Brazil, September*, 2015.

[10] A. Ryman, "Resource Shape 2.0," 2014. [Online]. Available: https://www.w3.org/Submission/shapes/

[11] E. Prud'hommeaux, J. E. Labra Gayo, and H. Solbrig, "Shape expressions: An RDF validation and transformation language," in *the 10th International Conference on Semantic Systems (SEM)*, 2014, pp. 32–40.

[12] W3C, "Shapes Constraint Language (SHACL)," 2016. [Online]. Available: https://www.w3.org/TR/2016/WD-shacl-20160128/

[13] B. Gallina, K. Padira, and M. Nyberg, "Towards an iso 26262-compliant oslc-based tool chain enabling continuous self-assessment," in *10th International Conference on the Quality of Information and Communications Technology- Track: Quality Aspects in Safety Critical Systems (QUATIC), Lisbon, Portugal*, 6-9 September, 2016.

[14] J. P. Castellanos Ardila, "Investigation of an OSLC-domain targeting ISO 26262," Master's thesis, Mälardalen University, School of Innovation, Design and Engineering, Västerås, Sweden, to appear in 2016.

[15] T. Bosch, A. Nolle, E. Acar, and K. Eckert, "RDF validation requirements - evaluation and logical underpinning," *CoRR*, vol. abs/1501.03933, 2015.

[16] Fancy Shex Demo, "https://www.w3.org/2013/shex/fancyshexdemo."

[17] SHACL Tutorial, "http://www.topquadrant.com/technology/shacl/tutorial/."

[18] B. Gallina, "Reconciling the ISO 26262-compliant and the Agile Documentation Management in the Swedish Context. In the 4th Scandinavian Conference on SYSTEM & SOFTWARE SAFETY, Stockholm, March 17," 2016.

[19] Gen&ReuseSafetyCases-SSF, "http://www.es.mdh.se/projects/393-genreusesafetycases."

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