Improving the Quality of Software Requirements Specifications with Semantic Web Technologies

Verónica Castañeda, Luciana Ballejos and Ma. Laura Caliusco

CIDISI – UTN –FRSF – Lavaise 610 – Santa Fe – Argentina
{vcastaneda,lballejo,mcaliusc}@frsf.utn.edu.ar

Abstract. A good software requirements specification (SRS) offers a number of benefits which include: being the contract between customers and suppliers, reducing the development effort, being a good basis for estimating costs and planning, a reference for verification and validation, and a basis for the identification of possible improvements in the analyzed processes. Mainly due to the lack of information and differences in interpretation between all those involved in the development of an information system, a good SRS is very difficult to define. In this paper, an approach based on Semantic Web technologies for defining a good SRS is presented. The main component of this approach is an ontology called OntoSRS that preserves the consistency, correctness, traceability, organization and unambiguity of a SRS. In addition, an application example is discussed.

Keywords. Software Requirements Specification, Semantic Web Technologies, quality, consistency, correctness, traceability, organization, unambiguity.

1 Introduction

With the main purpose of documenting all results obtained during Requirements Engineering (RE) process, diverse requirements artefacts (e.g. requirements documents, diagrams, models, among others) and deliverables are generated. Depending on the purpose of the documentation, the resulting information from the different RE activities are documented using different representation formats and at different levels of detail [1].

The most important deliverable, built and depurated in this iterative process, is the Software Requirements Specification (SRS), since it must be validated by the customer in order to continue the advancement in the development process. More precisely a SRS establishes software functions and capability, as well as its restrictions. It is the basis for any other subsequent planning, design and coding activity, and also for software tests and the generation of user documentation. All significant requirements for the system should be acknowledged and treated in the SRS. The main goal is that developers and customers do not need to make any assumption. If any functional or non-functional requirement is not identified in the SRS, it is not part of the agreement and, thus, nobody has to wait for that requirement to be in the final product.

Mainly due to the lack of information and differences in interpretation between all those involved in the process of developing an information system, an SRS is very difficult to define [2]. Some of the common risks which arise when this document is generated are:

- *Ambiguous Requirements:* which produce waste of time and repeated work. Their origin resides in the diverse stakeholders, who produce diverse interpretations for the same requirement. Moreover, the same stakeholder can interpret in diverse ways the same requirement. Ambiguity leads to erroneous product tests or to the implementation of new tests supposing fails in their construction.
- *Insufficient specifications:* can produce the absence of key requirements. This leads to developers' frustration, since their work is based on incorrect suppositions and, thus, the required product is not developed, which displeases customers.
- *Requirements not completely defined:* which complicate planning tasks and project monitoring. Requirements poor understanding leads to optimistic estimations, which turns against when agreed limits are surpassed.

The incomplete and ambiguous SRS definition directly affects software development projects [3]. Then, a tool for reducing the differences in interpretation between all those involved in the RE process with the aim of mitigating the aforementioned risks is needed.

With the emergence of the Semantic Web and the technologies for its realization, the possibility of applying ontologies as a means to define the semantics of information and knowledge are growing in different domains [4]. An ontology, from the point of view of software engineering, can be seen as a representational artefact that specifies a vocabulary of a given domain [5]. Ontologies are responsible for defining the terms used to describe and represent a given area of knowledge. An ontology serves to facilitate the communication between different actors in a domain by providing a common language and common understanding or comprehension of the conceptualization of the domain, reducing ambiguity and, terminological and semantic conflicts in a given area of knowledge [6].

There is an increasing amount of research devoted to utilizing ontologies in Software Engineering (SE) in general and RE in particular. A review of the latter can be found in [7]. In RE, ontologies were used separately for describing the semantics of the requirements specification documents, for the representation of requirements knowledge and for the representation of the requirements domain without taking into account quality criteria in an integrated way.

The contribution of this paper is an ontology-based approach for defining a good SRS. The main component of this approach is an ontology for conceptualizing the SRS called OntoSRS whose main objective is to guide stakeholders in the definition of a SRS maintaining the following SRS qualities: consistency, unambiguity, correctness, organization and traceability.

This paper is organized as follows. Section 2 introduces the main concepts around the proposed approach. Section 3 shows the generation of the OntoSRS. Section 4 presents an extension of the OntoSRS. Section 5 presents an application of the extended OntoSRS in a real case study. Finally, Section 6 discusses the results of the research and states future research directions.

2 Background

2.1 Semantic Web Technologies

2.b.1. Ontology Definition

In computer science, an ontology is understood as a representational artifact for specifying the semantics or meaning about the information or knowledge in a certain domain in a structured form [8]. From a pragmatic viewpoint, an ontology is a representational artefact whose representational units are: *terms*, *relations*, *instances*, *axioms* and *rules*. A *term* is a word or group of words representing an entity from the domain of discourse. An *instance* is a certain individual of a corresponding entity in the domain of discourse. A term representing an entity and its instances are related by the association *instance-of*. *Relations* are elements that glue together other ontology elements. A classification of the relations can be found in Caliusco et al. [9]. *Axioms* serve to represent sentences that are always true in the considered domain properties. *Rules* are logical sentences used to express the features of a domain, i.e. business rules [6].

An ontology is used to describe the meaning of the entities belonging to a certain domain, and its main characteristic is that it allows reasoning about the properties of that domain.

2.b.2. Ontology Development Methodologies

Several methodologies for developing ontologies have been described during the last two decades [10-11]. The objective of these methodologies is to define a strategy for identifying key concepts in a given domain, their properties and the relationships between them; identifying natural language terms to refer to such concepts, relations and attributes; and structuring domain knowledge into explicit conceptual models.

In literature, two groups of methodologies can be figured out. The first one is the group of experience-based methodologies represented by the Grüninger and Fox methodology defined in the TOVE project [12] and by the Uschold and King methodology based on the experience of developing the Enterprise Ontology [13]. The second one is the group of methodologies that propose a set of activities to develop ontologies based on their life cycle and the prototype refinement, such as the METHONTOLOGY methodology [6], the Ontology Development 101 Method [14] and the methodology defined by Brusa et al. [15].

2.b.3. Ontology Representation Languages

Different languages exist for ontology representation in a machine-interpretable way. Ontology languages are usually declarative languages commonly based on either first-order logic or description logic. The ones based on first-order logic have higher expressive power, but computational properties such as decidability are not always achieved due to the complexity of reasoning [16]. The most popular language based on description logic is OWL DL, which have attractive and well-understood computational properties [17].

The main ontology component is a set of rules. Considering that the OWL language is the standard for implementing an ontology and that it is not always enough to do certain deductions, then it is needed to combine OWL with other representation formalism as rules. One of the integration approaches is the Semantic Web Rule Language (SWRL), which aims to be the standard rule language of the Semantic Web. It provides the ability to express Horn-like rules. SWRL allows users to write rules that can be expressed in terms of OWL concepts and can reason about OWL individuals [18].

In order to extract information from OWL ontologies a query language is needed. The most powerful language is SQWRL, which is based on the SWRL rule language and uses SWRL's strong semantic foundation as its formal underpinning. The resulting language provides a small but powerful array of operators that allows users to construct queries on OWL ontologies. SQWRL also contains a novel set operators that can be used to perform closure operations to allow limited forms of negation as failure, counting, and aggregation [19].

2.2 SRS Quality Criteria

In RE, quality criteria can be defined for each individual requirements artifact or requirements document, as well as for an entire requirements document (SRS) or for specific sections in the document. Thus, the quality criteria defined for the SRS can be used during the RE process in a constructive or analytic manner.

Different works defining criteria for evaluating the quality of a SRS can be found in literature [1], [20-22]. Many of them describe needed attributes for the SRS as a whole. Nevertheless, other authors describe characteristics to be demanded for each specified requirement in order to affirm the SRS complies with them. In this work, the following criteria are considered:

2.2.a. Consistency

A SRS is consistent if and only if no subset of individual requirements stated therein is in conflict.
Some authors consider this attribute as "Internal Consistency". Meanwhile, they consider "External Consistency" when no individual requirement states in conflict with no other project documentation [1].

2.2.b. Correctness

A SRS is correct if and only if every requirement represents something required of the system to be built, i.e., every requirement in the SRS contributes to the satisfaction of some need, and all needs are considered in the specification.

2.2.c. Traceability

An SRS is traceable if and only if it is written in a manner that facilitates the referencing of each individual requirement. The origin of each of its requirements must be clear. This implies that every requirement that has a basis is cross-referenced to that basis.

2.2.d. Unambiguity

An SRS is unambiguous if and only if every requirement stated therein has only one possible interpretation. As a minimum, this requires that each characteristic of the final product is described using a single unique term.

2.2.e. Organization

An SRS is organized if and only if its content is arranged, so that information and logical relationships among adjacent sections is apparent and can be easily located by readers. One way is to follow any of the many SRS standards.

3 Building OntoSRS

Considering that an ontology is an artefact intended to avoid the ambiguity of a set of concepts in a given domain, using an ontology of the SRS minimize the ambiguity in the definition of the requirements of a software system. In this work is proposed an ontology called OntoSRS.

With the aim of defining an organized SRS, the OntoSRS is based on the 830 standard defined by the IEEE [22]. The IEEE standard [22] recommends a set of practices for the SRS generation, proposing a structure for it, plus an explanation of the features or qualities that must be considered in the generation of this document is given.

In order to build the OntoSRS an adaptation of the methodology defined by Brusa et al. [15] was followed since this methodology gather the best practices for ontology building. In the next subsections, the activities that were carried out are described.

3.1 Specification subprocess

Activity 1: Describing the domain

According to the 830 standard defined by the IEEE [22], a SRS establishes the functions and capacity of the software, as well as its restrictions. The SRS is a specification for a particular software product, program, or set of programs that performs certain functions in a specific environment. The SRS may be written by one or more representatives of the supplier, one or more representatives of the customer, or by both.

The basic issues that the SRS writer(s) shall address are the following:

a) Functionality. What is the software supposed to do?

b) External interfaces. How does the software interact with people, the systems hardware, other hardware, and other software?

c) Performance. What is the speed, availability, response time, recovery time of various software functions, etc.?

d) Attributes. What are the portability, correctness, maintainability, security, etc. considerations?

e) Design constraints imposed on an implementation. Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.?

Activity 2: Elaborating motivating scenarios and competency questions

A motivating scenario describing the main features of the elaboration of a good SRS was defined according to the IEEE 830 standard [22], which is presented with an activity diagram in Figure 1. The template proposed by Brusa et al. [15] is used to describe it, on Table 1. Based on this scenario, simple and complex competency questions arise, some of them are shown in Table 2. Competency questions (CQs) are questions at a conceptual level, informally written in natural language, which the ontology should be able to answer [12].

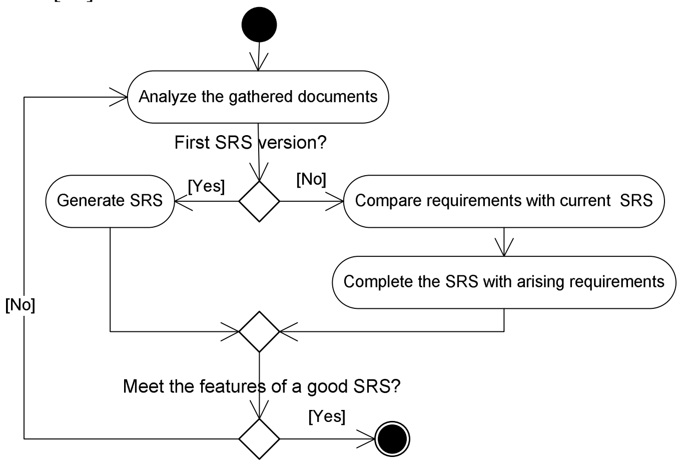

Analyze the gathered documents
First SRS version?
[Yes]
[No]
Generate SRS
Compare requirements with current SRS
Complete the SRS with arising requirements
[No]
Meet the features of a good SRS?
[Yes]

Fig. 1. Activity diagram of the SRS generation process.

Activity 3: Determining the ontology goal and scope

The ontology goal is to give support to the stakeholders to accomplish a complete, consistent, organized, traceable, and unambiguous SRS.

This ontology only assure an improvement in the definition of a SRS with the characteristics mentioned above, it doesn't guarantee the correctness, verifiability, modifiability, and traceability of the document.

3.2 Concretization subprocess

Activity 1: Defining classes and class hierarchy

Using the middle out strategy a list of terms that represents the most important entities in the SRS generation was created. It does not include partial or total overlapping of terms, synonyms, properties and relations. From that list of terms a UML diagram was elaborated with the hierarchy relations among those terms.

From the class hierarchy diagram, disjoint classes, exhaustive decompositions and partitions were identified. In Figure 2 an excerpt of the UML class hierarchy is shown.

Table 1. Scenario of the elaboration of a good SRS.

Scenario number	1	
Name	Iteration of a good SRS	
Description	Activities involved in each iteration of a good SRS.	
Site	Customer's organization	
Actors	Customer – Supplier – User – Stakeholders	
Pre-requirements	The customer needs a software product and he/she is willing to collaborate in the elaboration process of the SRS.	
Associated requirements	The customer is acquainted of the software needs and the operations and interaction of the user with the software product.	
Normal sequence	1	Analyze the gathered documents during the RE activities.
	2	Generate the SRS
	3	Customers and suppliers must check if the SRS meet the characteristics of a good SRS according to the standard 830 of the IEEE.
	4	Customers and suppliers approve the new version of SRS.
Alternative sequence	2.1	Complete the SRS with arising requirements
	2.2	Compare it with the previous version of the SRS.
Post-condition	A resulting SRS complete, consistent and unambiguous.	
Exceptions	3	The SRS doesn't meet the characteristics of a good SRS according to the standard 830 of the IEEE. Go to 1.
	4	Customers and suppliers don't approve the new version of SRS.
Main problems	Managing the great volume of information and knowledge generated by the RE activities. Ambiguous Requirements. Different stakeholders might produce different interpretations for the same requirement. Different stakeholders' backgrounds, perspectives and individual goals. Requirements are not completely and consistently defined. Insufficient Specifications, which produce absence of key requirements. Dynamic, changing requirements: which require constant requirements revision in order to help to understand new clients' needs and to identify how they can be satisfied.	
Main terms	Stakeholder, customer, supplier, software requirements specification, requirements.	

Table 2. Competency questions

Given a domain entity, what are the requirements related to it?	Are the dynamic requirements contradictory with other requirements?
Is the SRS unambiguous?	Does a certain term a unique meaning?
Is a certain non functional requirement specified in the SRS?	Do different words have the same meaning?
Is a certain functional requirement specified in the SRS?	How do different requirements relate to each other?

Activity 2: Identifying class relations, attributes and properties

The next step is the elaboration of a table that reflects the relations, the domain and range, cardinality and inverse relations. The relation direction depends on competency questions to be solved and the possible conflicts with other represented class restrictions. An excerpt of the table built during this activity is shown in Table 3. In addition, from this table a UML diagram was developed. An excerpt of it is shown in Figure 3.

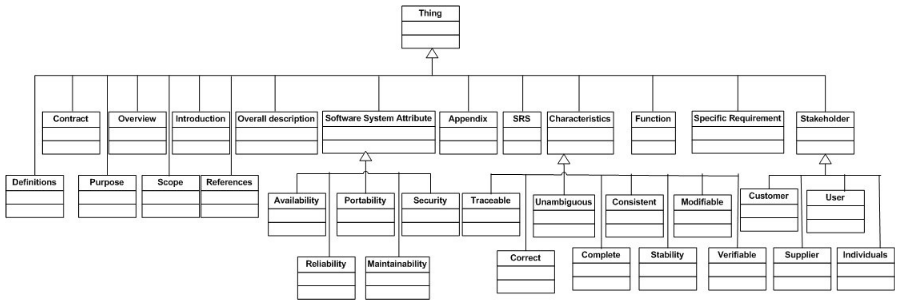


Fig. 2. An excerpt of the class hierarchy in UML.

Table 3. An excerpt of the relation table of the SRS ontology

Class name	Relation	Cardi-nality	Class name	Inverse relation
SRS	SRSHas	1	Introduction	isPartOf
SRS	SRSHas	1	Overall description	isPartOf
SRS	SRSHas	1	Specific requirements	isPartOf
SRS	SRSHas	1	Index	isPartOf
SRS	SRSHas	0..1	Apendix	isPartOf
Introduction	introductionHas	1	Definitions, acronyms and abbreviations	isPartOf
Introduction	introductionHas	1	Overview	isPartOf
Introduction	introductionHas	1	Purpose	isPartOf
Overall description	overallDescriptionHas	1	Product Perspective	isPartOf
Overall description	overallDescriptionHas	1	Product Functions	isPartOf
Overall description	overallDescriptionHas	1	User characteristic	isPartOf
Overall description	overallDescriptionHas	1	Constraint	isPartOf

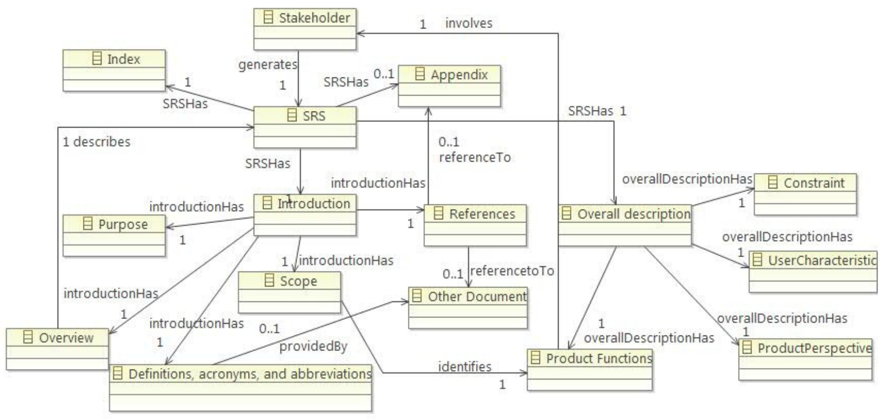


Fig. 3. An excerpt of the UML class diagram.

Activity 3: Representing axioms and restrictions

Once class relations, attributes and properties are identified, along with the scenario template, the next step is to analyze the restrictions among them.

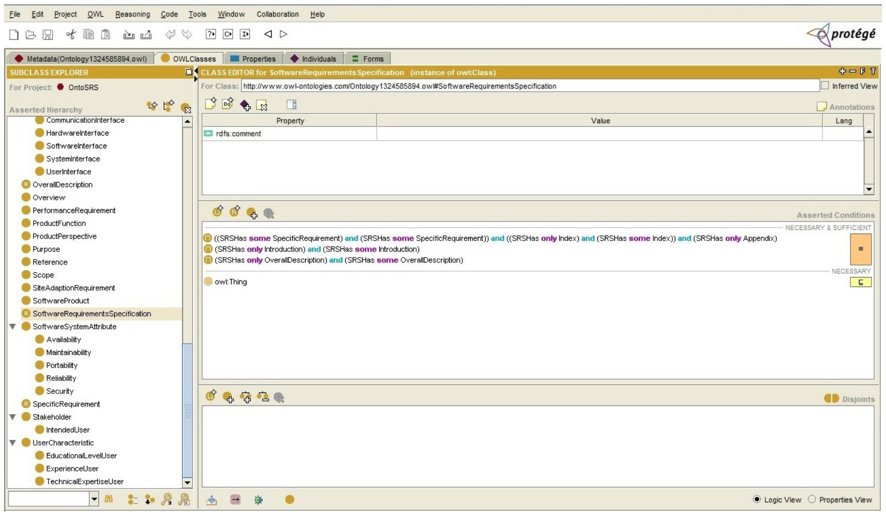
The rules are formally represented in Description Logic. For example, the following axiom states that a Software Requirements Specification is composed of an Introduction, an Overall Description, Specific Requirements, an Index and it could contain an Appendix.

$$\begin{aligned}
&\mathit{SoftwareRequirementSpecification} \equiv \\
&\quad (\forall \text{SRSHas.Introduction} \wedge \exists \text{SRSHas.Introduction}) \wedge \\
&\quad (\forall \text{SRSHas.OverallDescription} \wedge \exists \text{SRSHas.OverallDescription}) \wedge \\
&\quad (\forall \text{SRSHas.SpecificRequirement} \wedge \exists \text{SRSHas.SpecificRequirement}) \wedge \\
&\quad (\forall \text{SRSHas.Index} \wedge \exists \text{SRSHas. Index}) \wedge (\forall \text{SRSHas.Appendix})
\end{aligned}$$

3.3 The implementation subprocess

Activity 1: Creating a computational ontology

Protégé was chosen to implement the ontology because it is extensible and provides a plug-and-play environment that makes it a flexible base for rapid prototyping and application development. The OntoSRS ontology was implemented in OWL by using the Protégé[1], which is an open source editor. Figure 4 shows the logical view window.


File Edit Project OWL Reasoning Code Tools Window Collaboration Help
protégé
Metadata(Ontology1324585894.owl)
OWLClasses
Properties
Individuals
Forms
SUBCLASS EXPLORER
For Project: OntoSRS
Asserted Hierarchy
CLASS EDITOR for SoftwareRequirementsSpecification (instance of owl:Class)
For Class: http://www.owl-ontologies.com/Ontology1324585894.owl#SoftwareRequirementsSpecification
Inferred View
Annotations
Asserted Conditions
NECESSARY & SUFFICIENT
NECESSARY
Disjoints
Logic View
Properties View

CommunicationInterface
HardwareInterface
SoftwareInterface
SystemInterface
UserInterface
OverallDescription
Overview
PerformanceRequirement
ProductFunction
ProductPerspective
Purpose
Reference
Scope
SiteAdaptionRequirement
SoftwareProduct
SoftwareRequirementsSpecification
SoftwareSystemAttribute
Availability
Maintainability
Portability
Reliability
Security
SpecificRequirement
Stakeholder
IntendedUser
UserCharacteristic
EducationalLevelUser
ExperienceUser
TechnicalExpertiseUser

Property	Value	Lang
rdfs:comment		

((SRSHas some SpecificRequirement) and (SRSHas some SpecificRequirement)) and ((SRSHas only Index) and (SRSHas some Index)) and (SRSHas only Appendix)
(SRSHas only Introduction) and (SRSHas some Introduction)
(SRSHas only OverallDescription) and (SRSHas some OverallDescription)
owl:Thing

Fig. 4. Protégé Logical view window for the OntoSRS ontology.

Activity 2: Verifying the ontology

With the aim of verifying the ontology, the correctness of the ontology, consistency, completeness and conciseness have to be proved [6].
•*Consistency*. A given definition is consistent if the individual definition is consistent and no contradictory sentences can be inferred using other definitions and axioms.

[1] http://protege.stanford.edu/

•*Completeness*. In fact, cannot be proved either the completeness of an ontology or the completeness of its definitions, but we can prove both the incompleteness of an individual definition, and thus deduce the incompleteness of an ontology, and the incompleteness of an ontology if at least one definition is missing regards to the established reference framework.
•*Conciseness*. An ontology is concise if it does not store any unnecessary or useless definitions, if explicit redundancies do not exist between definitions, and redundancies cannot be inferred using other definitions and axioms.
Two main types of measurement for verification could be identified: structural measures and formal measures. The first one is required to graphically representing the ontology. The second implies to use a reasoner. In this study case, the OWLViz and OntoViz plugins of the Protégé were used to compare the computable ontology with the designed in UML (see Section 3.2 - Activity 2). On the other hand, the reasoner Pellet which is included in the Protégé editor was used[2].

Activity 3: Validating the ontology

In order to validate the computable ontology, the competency questions defined in Process 1 were codified by using the SPARQL query language[3]. Then, they were executed with the aim of checking if the competency questions are being correctly answered by the OntoSRS.

4 Extending the OntoSRS

The SRS should be unambiguous both to those who create it and to those who use it. However, these groups often do not have the same background and therefore do not tend to describe software requirements in the same way. Within the requirements engineering, one artefact that describes the terminology of a given domain from the users and designers perspectives is the Lexicon Extended Language (LEL) [23]. LEL is a representation of the terminology in the application language, which is classified in four categories: object, subject, verb, and state. Each term is described in two ways. The former, notion, is the denotation of the term indicating its meaning, who is, when it occurs, which process involves. The latter, behavioural response, describes the connotation of the term indicating the effects that such term generates on others terms and the effects that other terms generate on it.

In order to fulfill the previous objective, an ontology that conceptualizes the LEL, called OntoLEL, was built. It was done following the same steps as those for creating OntoSRS which were previously explained. This ontology is shown in the Figure 5.

In addition, the use of LEL contributes to maximize the correctness since some requirements can be associated with at least one impact defined in the LEL, assuring that the requirement represents something demanded to the system to be built. Another requirement criterion for a good SRS is the traceability which is achieved when the origin of each individual requirement is clearly defined. In Figure 6 is shown how

[2] Support about the integration of Protégé and Pellet can be found in http://protege.stanford.edu/
[3] Specifications can be found in http://www.w3.org/TR/rdf-sparql-query/

some of the elements defined in the LEL ontology are related with some of the elements of the OntoSRS.

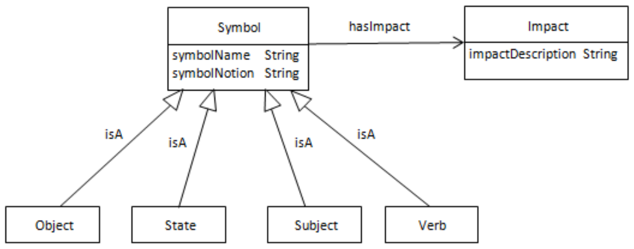


Fig. 5. OntoLEL ontology.

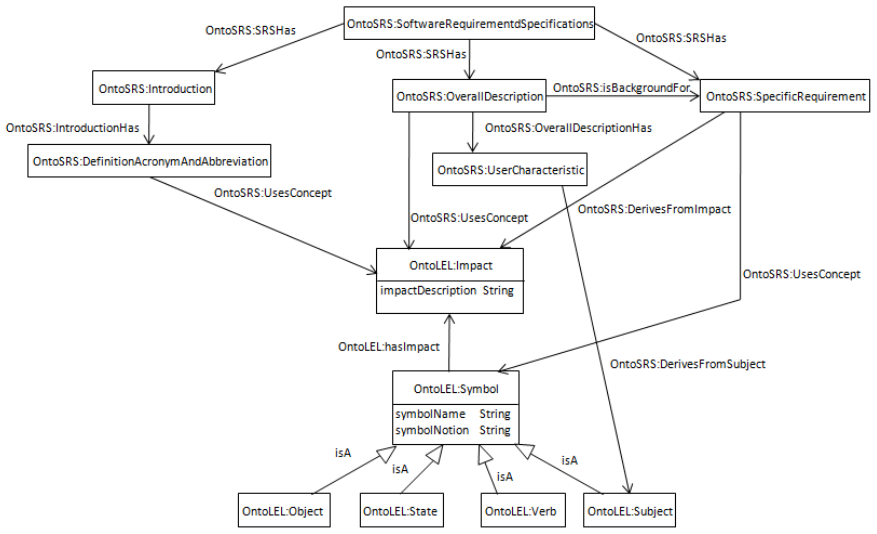


Fig. 6. Some relationships between OntoSRS and OntoLEL.

5 Study Case

In order to exemplify the proposal, an excerpt of the well-known Meeting Scheduler System also considered by Breitman and Sampaio do Prado Leite [24] will be used.

In general, RE methodologies and techniques describe concepts, scenarios and situations of the business process, either the current observable process or the future one, when considering the application implementation [25]. Thus, these techniques can be applied in two ways. In the last case, when considering future situations, the generated documentation contains the most of the requirements, which must be considered in the SRS creation, thus, the evaluation of diverse quality criteria for SRS such as traceability or correctness can be improved with the consideration of LEL documentation, and much more, of its ontology.

LEL is one of those techniques, capable to be applied in the "as is" situation, or, even, in the "to be" situation. Figure 7 shows the LEL documentation for Initiator, one of the actors for the Meeting Scheduler System, considering the "to be" situation.

	LEL
Initiator	**type: subject**
Notion: Person that invites other meeting participants. He or she may be a participant.	
Impacts: ▪ Defines meeting goal. ▪ Defines meeting date. ▪ Defines meeting agenda. ▪ Is responsible for meetings cancellations. ▪ Invites participants. ▪ Asks participants for assistance confirmation.	

Fig. 7. Initiator LEL for Meeting Scheduler System.

In general, when LEL technique is applied for the "to be" situation (considering the application under analysis), and the symbol being described is a "subject", functional requirements can be derived from the impacts defined for the LEL. Thus, with the aim to exemplify how LEL and SRS elements are related, Figure 8 shows the ontology instantiation for the example.

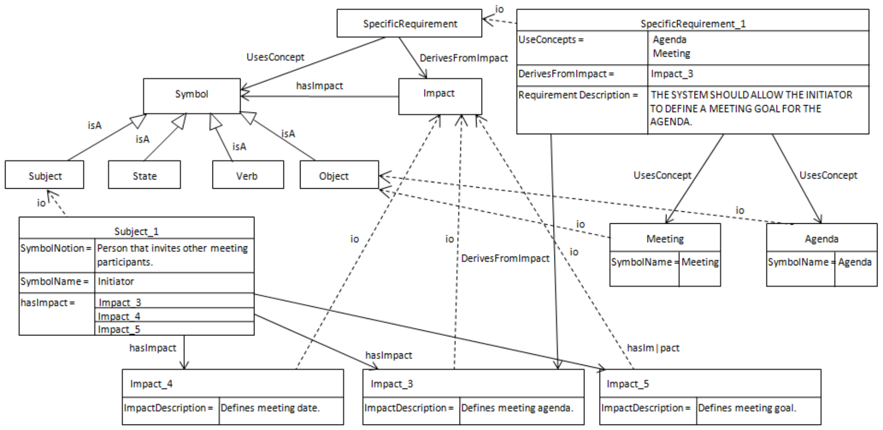


Fig. 8. Ontology instantiation.

6 Discussion and Future Research Directions

Diverse authors describe the benefits of using ontologies along software development projects [5], [26]. Specifically in RE phases, the major benefit can be obtained when using an ontology in the SRS creation, since it guides the specification process for obtaining a quality SRS. This goal can be reached through the application of OntoSRS, based on the IEEE 830 Standard [21], in order to ensure the organization quality criterion. In addition, in this paper this ontology was greatly enhanced by relating it with an ontology for LEL specification, with the aim of describing a given domain by using a common vocabulary. This improves the unambiguity, correctness and traceability quality criteria.

Another advantage of using a computable ontology for supporting the definition of a SRS is the possibility to automatically validate its quality. The OWL language allows consistency checking of an ontology, which means that the ontology does not contain any contradictory axioms. Some axioms were added to the OntoSRS that restrict the definition of the SRS (see Section 3.2 - Activity 3). Then, the consistency of the OntoSRS can be checked, ensuring in some way the consistency of the SRS.

On the other hand, OWL does not reason about indivuals of the ontology. Then, in order to validate the quality criteria that involve instances, SWRL rules and SQWRL can be used. As an example, the following one implements traceability validation for the SRS:

SoftwareRequirementSpecification(?srs) ^ SpecificRequirement (?r) ^
SRSHas(?srs, ?r) ^ Impact(?i) ^ derivedFromImpact(?r, ?i) ^
sqwrl:makeSet(?s, ?r) ^sqwrl:isEmpty(s) → notTraceable(?srs)

In case all rule components are true, the *?srs* instance is classified as *notTraceable*. This rule infers that an *SRS* is not traceable if the set of requirements specified in the SRS that has associated an impact defined in the LEL is empty. This is a preliminary rule that demonstrates the usefulness of rules applied to the extended OntoSRS for deducing the SRS quality criteria. The current work is devoted to analyze the approaches for measuring the quality criteria considered in this paper, in order to implement the most suitable ones.

In the definition of a SRS, the knowledge related with the specific requirements is also important. In the literature, different classifications of requirements and ontologies for conceptualizing them can be found. An overview of them is presented in Castañeda et al. [7]. In this sense, future work will be conducted in order to incorporate this kind of ontology in the proposed approach and, thus, study how quality criterion is affected.

References

1. Pohl, K.: Requirements Engineering: Fundamentals, Principles, and Techniques. Springer (2010).
2. Noppen, J., van den Broek, P., Aksit, M.: Imperfect requirements in software development. In P. Sawyer, B.P., (Eds.): Requirements Engineering for Software Quality - REFSQ 2007, LNCS 4542, Springer-Verlag Berlin Heidelberg, pp. 247-261 (2007).
3. Alexander, I.F., Stevens, R.: Writing Better Requirements. Pearson Education (2002).
4. Shadbolt, W.H., Berners-Lee, T. The semantic web revisited. IEEE Intelligent Systems, 21(3), pp. 96-101 (2006).
5. Happel, H.J., Seedorf, S.: Applications of ontologies in software engineering. In: International Workshop on Semantic Web Enabled Software Engineering (SWESE'06) (2006).
6. Gómez-Pérez, A., Fernández-López, M., Corcho, O.: Ontological Engineering, Springer, Heidelberg (2004).

7. Castañeda, V., Ballejos, L., Caliusco, M.L., Galli, M.R.: The Use of Ontologies in Requirements Engineering. Global Journal of Computer Science and Technology, 10(6), pp. 2-8 (2010).
8. Allemang, D., Hendler, J.A.: Semantic web for the working ontologist: Modeling in RDF, RDFS and OWL. Elsevier, Amsterdam (2008).
9. Caliusco, Ma. L., Galli, Ma. R., Chiotti O. "Technologies for Data Semantic Modeling". Intl. Journal of Metadata Semantics and Ontology. Special Issue. Inderscience Publishers. ISSN 1744-2621. 1(4), pp. 320-331 (2006).
10. Corcho, O., Fernández-López, M., Gómez Pérez, A. Methodologies, tools and languages for building ontologies: where is the meeting point? Data and Knowledge Engineering, 46(1), pp. 41-64 (2003).
11. Wouters, B., Deridder, D., Van Paesschen, E. The use of ontologies as a backbone for use case management. In: Proceedings European Conference on Object-Oriented Programming (2000).
12. Grüninger, M., Fox, M.S.: Methodology for the design and evaluation of ontologies, In: Proceedings International Joint Conference on Artificial Intelligence, Workshop on Basic Ontological Issues in Knowledge Sharing (1995).
13. Uschold, M., Grüninger, M.: Ontologies principles, methods and applications, Journal Knowledge Engineering Review 11, pp. 93–155 (1996).
14. Noy, N., McGuinness, D.: Ontology Development 101: A Guide to Creating Your First Ontology (2001).
15. Brusa, G., Caliusco, M.L., Chiotti, O.: Towards ontological engineering: a process for building a domain ontology from scratch in public administration. Expert Systems, 25(5), pp. 484-503 (2008).
16. Brachman, R., Levesque, H.: Knowledge Representation and Reasoning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2004).
17. Smith, M., Welty, C., McGuiness, D.: Owl web ontology language guide. Recommendation W3C 2(1) (2004).
18. O'Connor, M., Knublauch, H., Tu, S., Musen M.: Writing Rules for the Semantic Web Using SWRL and Jess. In: Proceedings 8th International Protege Conference, Protégé with Rules Workshop (2005).
19. O'Connor, M.J., Das, A.K.: SQWRL: a Query Language for OWL. In: Proceedings OWL: Experiences and Directions, 6th International Workshop, Chantilly, VA (2009)
20. Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynols, P. Sitaran, P., Ta, A., Theofanos, M.: Identifying and measuring quality in software requirements specification. In: Proceedings 1st Int'l Software Metrics Symposium (1993).
21. Fabbrini, F., Fusani, M., Gnesi, S., Lami, G.: An Automatic Quality Evaluation for Natural Language Requirements. In: Proceedings 7th Int'l Workshop on Requirements Engineering: Foundation for Software Quality, Interlaken, Switzerland (2001).
22. IEEE 830 Standard: IEEE Guide to Software Requirement Specification, Standard 830-1998, New York: IEEE Computer Society Press (1998).
23. Sampaio do Prado Leite, J.C., Franco, A.P.M.: A Strategy for Conceptual Model Acquisition. In: Proceedings of IEEE International Symposium on Requirements Engineering. IEEE Computer Society Press, pp. 243-246 (1993).
24. Breitman, K.K., Sampaio do Prado Leite, J.: Ontology as a requirements engineering product. In: 11th IEEE International Requirements Engineering Conference (2003).
25. Hadad, G.; Doorn, J., Kaplan, G.: Elicitar Requisitos del Software usando Escenarios. In: Proceedings Workshop on Requirements Engineering (WER'09) (2009).
26. Calero, C.; Ruiz, F.; Piattini, M.: Ontologies for Software Engineering and Software Technology, Springer (2006).