Towards a Requirements Engineering Framework based on Semantics

KALLIOPI KRAVARI
School of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece, kkravari@csd.auth.gr

CHRISTINA ANTONIOU
School of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece, antoniouc@csd.auth.gr

NICK BASSILIADES
School of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece, nbassili@csd.auth.gr

Requirements engineering is one of the most important issues in systems development. Whether it is software or hardware systems or embedded systems, the need for well-defined requirements remains the same. The ultimate success or failure of developing a system stems largely from the initial definition and management of its requirements. However, despite the efforts that have been made, a coherent and easily understood process that leads from the requirements to correct implementations is still an open research issue, which seeks alternative promising approaches. To this end, in this paper, we propose a requirements engineering approach based on Semantics. It provides a novel mechanism that combines semantics, ontologies, and appropriate NLP techniques. The ultimate goal is to propose a framework that will include the minimum consistent set of formalities and languages to determine the requirements and perform the necessary verifications.

CCS CONCEPTS • Software and its engineering • Software creation and management • Designing software

Additional Keywords and Phrases: Requirements Engineering, Semantics, Ontologies, Boilerplates

ACM Reference Format:

Kalliopi Kravari, Christina Antoniou, and Nick Bassiliades. 2020. Towards a Requirements Engineering Framework based on Semantics. In PCI '20: 24th Pan-Hellenic Conference on Informatics, November 20–22, 2020, Athens, Greece. ACM, New York, NY, USA, 6 pages.

PCI 2020, November 20–22, 2020, Athens, Greece

ACM ISBN 978-1-4503-8897-9/20/11…$15.00

https://doi.org/10.1145/3437120.3437278

1 Introduction

The ultimate success or failure of developing a system stems largely from the initial definition and management of its requirements. Indeed, the most difficult part of developing a system is deciding what exactly it will develop and what its structure and behavior will be. No other part of the project concept is as difficult to determine and fix later. However,

despite the efforts that have been made, an efficient recording and management solution regarding the requirements of an evolving system remains an open research issue [9]. Actually, the vast majority of requirements are still written in natural language and even with loose guidelines. It is, in essence, an expression of the acceptable behavior of the system rather than a rigorous recording of its required features or functions. This tactic raises important issues since it does not ensure that the requirements, and therefore the specifications, are understood clearly and in the same way by all parties. Hence misunderstandings and errors can lead to failure or high cost of remediation at a later stage, as the set of requirements ultimately did not lead to proper design and implementation. Therefore, consistent and clear recording of requirements is a prerequisite for the development of strong systems at the lowest possible cost. Thus, the research community gradually aimed to offer some kind of structure or standardization, focusing on translation and analysis automation.

Some research efforts are mainly focused on translating from a document written in one natural language to another formally defined language [6]. However, the ambiguity of natural languages makes translation difficult and the results in many cases are doubtful. Hence, it is important a shift from translation approaches to approaches that use standard languages and methods from the beginning, minimizing the use of natural language. In addition, various machine learning techniques continue to play a central role in the relevant research field. Despite their undeniable importance and offer, they present difficulties in grasping the semantics, namely the meaning of the requirements. Thus, there is still a need for mechanisms capable of capturing the semantics and using it efficiently in order to enhance the accuracy and completeness of the requirements recorded [7].

Such an alternative approach is proposed in this article. The approach uses semantics, ontologies, and appropriate NLP (Natural Language Processing) techniques in order to provide a complete (flow-down) process of production and standardization of requirements; a new framework that will include the minimum consistent set of formalities and languages to determine the requirements and perform the necessary verifications. In this context, it proposes the use of integrated boilerplates, namely standardized natural language expressions with well-defined semantics along with systematic checks to verify the validity and completeness of the requirements, based on the boilerplate used, the type of requirement, the type of system, and the framework included.

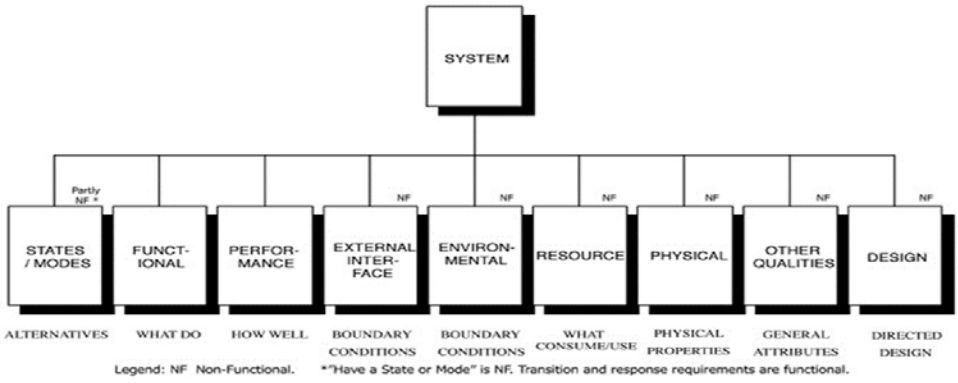

Figure 1: Requirement categories

2 Requirements

Although requirements of a system can vary, according to the IEEE [1] requirements engineering involve specific stages, four of which are considered common. These are the elicitation, the analysis, the specification, and the validation. These stages follow a specific chronological order, although they often alternate with each other. A requirement is a single documented operational demand that a system must be able to perform or a specific constraint that it must have. In other words, it is a statement that identifies a necessary property, capability, characteristic, quality or constraint that it must be present. In practice, the requirements are used as inputs to the design stage while they make a significant contribution to the verification process, since checks should refer to specific requirements. Requirements can be divided

into two main categories [8], functional and non-functional, with each of them having its own subcategories (Figure 1). Functional requirements describe the system functionality while non-functional requirements describe its quality and constraints.

3 The Semantics - based Framework

The aim of the proposed framework is to provide a semantics-based systematized specification of requirements. It is based on a knowledge base while it comprises boilerplates, semi-complete requirements used as an input mechanism for capturing the underlying system-related semantics behind requirements. Boilerplates are instantiated into complete requirements by replacing placeholders with entities representing concepts from the particular system under design. The entities referenced in the boilerplate placeholders are mapped to a concrete semantic model of the system's domain, namely an Ontology that is also the means to capture implicit knowledge that has to be made explicit, in order to ensure design correctness and consistency.

3.1 Software Requirements Specification and Ontologies

Although, the proposed approach, is generic and can be used in a variety of cases, a proper ontology should be used in order to grasp the semantics and provide the necessary guidelines given specific requirements. Officially, an ontology is a formal naming and definition of the concepts, their characteristics and relationships between them that exist in a particular domain. Common components of ontologies include individuals, instances, classes, attributes, relations, restrictions, rules, axioms as well as events. Therefore, in this context, for the purposes of this study we have developed the isU Ontology which will serve as core example. Of course other existing ontologies could be adopted. This ontology covers the requirements of an inventory software system for a University. The inventory system will be able to handle all the faculties of the University as well as Individuals, Places, Resources, and Permissions. It allows users to add, edit, or delete items in the database (DB), depending on their permissions (Figure 2).

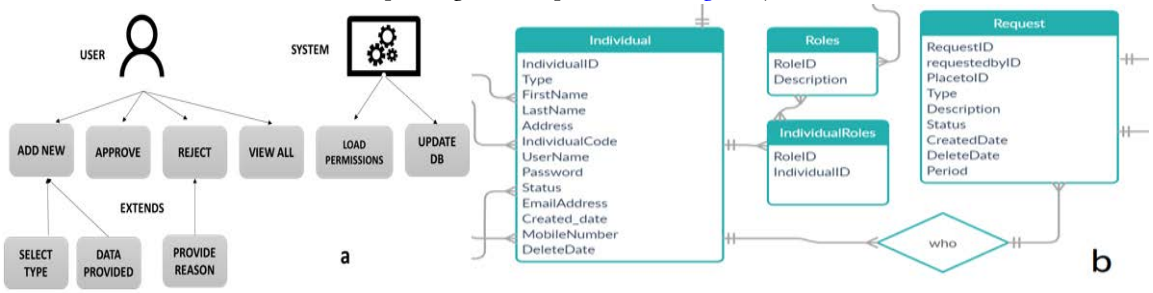

Figure 2: (a) Functional requirements of isU ontology system regarding new request – (b) Part of the ER diagram

3.2 Boilerplates

The approach copes with the ambiguity problem in natural-language requirements by systematizing the specification of requirements through a natural-like artificial language that allows combining boilerplates [3] through a well-defined syntax and a precisely defined semantics using reasoning procedures. Boilerplates allow requirements to be recorded consistently, reducing, among others, misspellings, poor grammar and ambiguity, making it easier to understand, categorize and identify requirements. Boilerplates are semi-complete and customizable. The mechanism of boilerplates is a fixed dictionary of concepts from a knowledge base, relevant to the field of the system being developed and a set of predefined structures. The basic building block of a boilerplate is a clause, which expresses some aspect of the requirement. Each such clause has a type, e.g. capacity, function, etc., which indicates the type of the requirement. In addition, each clause may have a target type, indicating the general objective pursued, e.g. minimizing or maximizing

something, exceeding a certain value. The clauses can be combined and put together to produce complete boilerplates, which will express multiple aspects of a specific requirement. As mentioned, the semantics of the boilerplates language corresponds to an appropriate ontology. Practically, each boilerplate is defined as a sequence of attributes and fixed syntax elements, which facilitate the construction of a requirement. An example boilerplate could be: IF [redacted]. Obviously, the terms IF and shall are fixed components while the rest, enclosed in parentheses, are properties that can take any value. This example consists of a prefix, IF <state>, which defines a condition, and a main clause, <subject> shall <action>, which defines a capability. For instance, requirement "If user add new request, the system loads permissions" can be expressed using the aforementioned boilerplate as follows: [redacted]. Figure 3 depicts part of the proposed GUI.

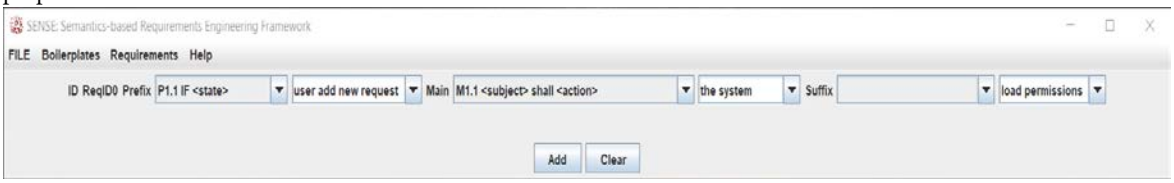

Figure 3: Framework GUI – Example of a boilerplate

The grammar (Table 1) is context-free, containing both necessary and optional clauses. Boilerplates consists of up to three different types of clauses, a main clause, which is mandatory, and two optional clauses, the prefix and the suffix clauses (Table 2). Some requirements may be complex and require more than one concept to describe, hence the boilerplate may include multiple prefixes or suffixes separated by logic connectives (e.g. and, or). However, each requirement should have a single main clause for reasons of explicit reference. Prefixes correlate the main clause with preconditions referring to actions, states, or events. Suffixes are used to configure or to calibrate the main clause, by specifying additional information for involved actions and entities.

Table 1: Boilerplate grammar – (a) notation meaning, (b) language syntax

Symbol	Notion Meaning (a)	Boilerplate language syntax (b)
[...]	optional	<prefix> ::= <simple prefix> <logic connective> <prefix> | <simple prefix>
... | ...	or	< suffix> ::= <simple suffix> <logic connective> <suffix> | <simple suffix>
*	>= 0 occurrences	<logic connective> ::= or | and | xor
+	>= 1 occurrences	<simple prefix> ::= <P_i>
<...>	boilerplate attribute	<simple suffix> ::= <S_j>
		<main> ::= <M_k>

Table 2: Boilerplate grammar clauses– Part of (a) prefix, (b) main, (c) suffix clauses

prefix (P_i) clauses (a)	main (M_k) clauses (b)	suffix (S_j) clauses (c)
P_1: if | unless | while <state>	M_1: <subject> shall [not] <action>	S_1: after | before <event>
P_2: if | unless <event>	M_2: <subject> shall [not] be <state>	S_2: other than (<action>|<entity>)
P_3: if | unless | while <action>	M_3: <subject> shall [not] be able to <action>	S_3: using <entity>
	M_4: <subject> shall [not] support <action>	S_4: without (<action>|<entity>)

3.3 Boilerplates Recommendations

The framework provides non-binding recommendations, taking advantage of the information and relationships recorded in the ontology as well as those given by the user to the boilerplate. The approach combines two methods, namely

controlled natural language (boilerplates) and phrasal semantic parsing (Apache Jena [5], ontology). Mention that Jena is a Java framework for building Semantic Web applications. It provides extensive Java libraries for helping developers develop code that handles RDF, RDFS, RDFa, OWL and SPARQL in line with published W3C recommendations. Jena includes a rule-based inference engine to perform reasoning based on OWL and RDFS ontologies, and a variety of storage strategies to store RDF triples in memory or on disk. When the engineer writes a boilerplate, the system performs partial semantic analysis using Jena upon the knowledge base (ontology) proposing the closest semantics. Recommendations refer either to similar terms (general recommendations), e.g. "course lab" of isU ontology is connected to "teaching lab" via symmetric object property (isSynonymOf), or to requirement categorization (specialized recommendations). At run-time, via Jena reasoning mechanism, the requirement that is entered is dynamically categorized. For instance, the basic types of requirements are usually related to functional, performance, interface, design and construction issues. An example is the storage requirement at the isU inventory. In this context, as soon as the user starts writing a functional clause, the framework extracts through reasoning the knowledge that storage space is required and informs the user about it with a message. The approach uses also a semantic distance metric that assesses the similarity between a given pair of terms by calculating the (shortest) distance between the nodes corresponding to these terms in the ontology hierarchy:

$$Dist\left(C_1, C_2\right) = \frac{2SPR}{D_1 + D_2 + 2SPR} \qquad (1)$$

where D_1 and D_2 are, respectively, the shortest paths from C1 and C2 to C (their nearest common ancestor on the ontology hierarchy), and SPR is the shortest path from C to the root.

3.4 Requirements Validation

Finally, the framework provides checks to ensure the validity and completeness of the requirements, ending up with a set of requirements that are all covered by properties while the system's behavior is not under-specified [6]. For the first issue, it is necessary to identify any inconsistencies that could lead to semantic, structural, or behavioral contradictions while for the second it is necessary to identify possible requirements that have been omitted while they should not. The framework deals with that by running appropriate SPARQL (ontology) queries. The advantage of the questions is that they need to be made once, although manually, and then they can be reused many times. For instance, at the isU inventory each Individual that has the role of Administrator should have permission to insert resources. The following query checks if such permission is not completely covered by the requirements. The query checks if there are requirements related to Administrator that should have the aforementioned permission (refers to main clause of the M_3 type (Table 2): <subject> shall [not] be able to <action>). Hence, there should exist a prefix clause of the type P3 (Table 2), for the same action, namely insertResourse. Yet, we define for each query, a number of SPIN (SPARQL Inferencing Notation), a de-facto industry standard, constraints, which are SPARQL queries that are placed at appropriate classes of the ontology and if evaluated positively they indicate a constraint violation from instances of the class. Thus, finally with the use of SPIN we are able to carry out ontology validation checks related to (a) incompleteness of requirements, (b) inconsistency of requirements and (c) deficiencies of the system model based on requirements Table 3:.

Table 3: (a) SPARQL Query, (b) SPIN Constraint

SPARQL Query (a)	SPIN Constraint (b)
Select * WHERE {	
?r a sense:Individual .	ASK WHERE {
?r sense:hasRole ?x .	? this sense:isRelatedToPermission ?Permission.
?x a sense:Admistrator.	NOT EXISTS {
?x sense:isRelatedToPermission ?Permission .	?pr a sense: insertResourse.
NOT EXISTS { ?pr a sense:insertResourse .	?pr sense: isRelatedToPermission ?Permission .} .}
?pr sense: isRelatedToPermission ?Permission.} .	

4 Related work

In [2] researchers study the use of boilerplates for requirements specification in an approach for transforming informal requirements to formally specified properties. The verifiability criteria and the needs for properties preservation using the principles of correctness are not addressed. In [4] is discussed a tool-supported methodology for transforming stakeholder requirements to formal specifications. The framework consists of an ontology of requirements, a modeling language for functional and non-functional requirements, and a set of refinement operators. Yet, it does not support recommendations and appropriate verification background.

5 Conclusions

Requirements engineering is a core issue in systems development, determining the ultimate success or failure. Despite the efforts, there is space for flow-down solutions regarding requirements recording and management. The vast majority of requirements are still written in natural language demanding translation and analysis. Yet, the ambiguity of natural languages makes translation difficult. On the other hand, machine learning techniques present difficulties in grasping the semantics. In this context, this paper reported on a shift from translation to standard languages and methods, proposing a framework that minimizing the use of natural language, using formal methods from the beginning [6]. The approach uses semantics, ontologies, and appropriate NLP techniques in order to provide a complete process of production and standardization of requirements. The framework uses boilerplates, standardized natural language expressions with well-defined semantics to determine the requirements, including the minimum consistent set of formalities, while it performs necessary verifications using SPARQL (SPIN) queries. As for future directions, we are going to extent and upgrade the recommendation component of the system while more analysis will be conducted regarding the requirement verifications. Our plan is to provide more SPIN constraints, updating the ontology, in order to collect all constraint violations.

ACKNOWLEDGMENTS

This research is funded in the context of the project «Requirements Engineering based on Semantics» (MIS 5047826) under the call for proposals «Support for researchers with an emphasis on young researchers-cycle B'» (EDULLL 103). The project is co-financed by Greece and the European Union (European Social Fund- ESF) by the Operational Programme Human Resources Development, Education and Lifelong Learning 2014-2020.

REFERENCES

[1] 29148-2018 - ISO/IEC/IEEE Int. Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering.

[2] Ajitha Rajan and Thomas Wahl. 2013. CESAR - Cost-efficient Methods and Processes for Safety-relevant Embedded Systems. Springer.

[3] Elizabeth Hull, Ken Jackson, and Jeremy Dick. 2010. Requirements engineering. Springer Science & Business Media.

[4] Feng-Lin Li, Jennifer Horko, Alexander Borgida, Giancarlo Guizzardi, Lin Liu, and John Mylopoulos. 2015. From stakeholder requirements to formal specifications through refinement. Requirements Engineering: Foundation for Software Quality, 9013 LNCS, Springer, 164-180.

[5] Jena. 2020. The Apache Software Foundation. Retrieved at 06 June 2020.

[6] Kon. Mokos, and Panagiotis Katsaros. 2020. A survey on the formalisation of system requirements and their validation. Array 7: 100030

[7] Mavridou A., Stachtiari E., Bliudze S., Ivanov A., Katsaros P., Sifakis J. (2017) Architecture-Based Design: A Satellite On-Board Software Case Study. In Formal Aspects of Component Software. FACS 2016. Lecture Notes in Computer Science, vol 10231. Springer, Cham.

[8] Muhammad Azeem, Akbar Jun Sang, Arif Ali Khan, and Shahid Hussain. 2019. Investigation of the requirements change management challenges in the domain of global software development. J Softw Evol Proc. 2019; 31:e2207. https://doi.org/10.1002/smr.2207

[9] Muhammad Suhaib. 2019. Conflicts Identification among Stakeholders in Goal Oriented Requirements Engineering Process (October 18, 2019). International Journal of Innovative Technology and Exploring Engineering (IJITEE).