Received 15 June 2022, accepted 15 July 2022, date of publication 8 August 2022, date of current version 15 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3197281

RESEARCH ARTICLE

Semantic Modeling and Analysis of Natural Language System Requirements

KONSTANTINOS MOKOS, THEODOROS NESTORIDIS, PANAGIOTIS KATSAROS, AND NICK BASSILIADES, (Member, IEEE)
School of Informatics, Aristotle University of Thessaloniki, 541 24 Thessaloniki, Greece

Corresponding author: Panagiotis Katsaros (katsaros@csd.auth.gr)

This work was supported by the European Union's Horizon 2020 Research and Innovation Program under Grant 956123 (FOCETA).

ABSTRACT System requirements specify how a system meets stakeholder needs. They are a partial definition of the system under design in natural language that may be restricted in syntax terms. Any natural language specification inevitably lacks a unique interpretation and includes underspecified terms and inconsistencies. If the requirements are not validated early in the system development cycle and refined, as needed, specification flaws may cause costly cycles of corrections in design, implementation and testing. However, validation should be based on a consistent interpretation with respect to a rigorously defined semantic context of the domain of the system. We propose a specification approach that, while sufficiently expressive, it restricts the requirements definition to terms from an ontology with precisely defined concepts and semantic relationships in the domain of the system under design. This enables a series of semantic analyses, which guide the engineer towards improving the requirement specification as well as eliciting tacit knowledge. The problems addressed are prerequisites to enable the derivation of verifiable specifications, which is of fundamental importance for the design of critical embedded systems. We present the results from a case study of modest size from the space system domain, as well as an evaluation of our approach from the user's point of view. The requirement types that have been covered demonstrate the applicability of the approach in an industrial context, although the effectiveness of the analysis depends on pre-existing domain ontologies.

INDEX TERMS Embedded systems, ontologies, requirements validation, semantic reasoning.

I. INTRODUCTION

Validation of system requirements, which are usually written in natural language, is a prerequisite to guarantee the validity of a system specification, for an acceptable (correct) and attainable solution. According to this perspective, we should aim to the earliest possible discovery and resolution of specification issues; the goal is to replace part of the effort needed in the verification testing phase with significantly less effort, earlier in the system development cycle. Requirements validation is associated with the problem of transforming them into a formal specification amenable to verification (formalization). We have addressed this problem in [1] (Fig. 1), within the context of a correctness-by-construction design approach, which allows to discover *behavioral* (not semantic) inconsistencies in requirements. However, formal reasoning is possible only if a well-defined interpretation semantics exists, for the requirements specification.

In this paper, we introduce an approach which allows formulating requirements in natural language (Fig. 1) through relying on a semantic modeling and analysis framework that complements our previous work in [1], for the requirements formalization and system design. Natural language requirements are informal specifications that incorporate personal views, tacit knowledge and/or may assume facts not explicitly specified, but expected to be applicable. Any such specification, in general, is not characterized by a clear, unambiguous, complete, and consistent semantics.

To this end, an *ontology-driven* approach is presented, in which requirement specifications are restricted to terms

The associate editor coordinating the review of this manuscript and approving it for publication was Hui Liu.

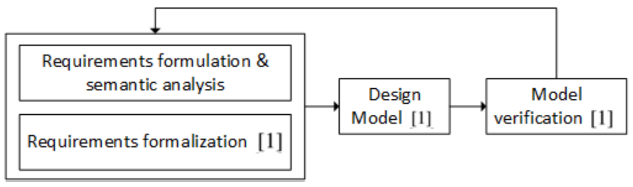

Requirements formulation & semantic analysis
Requirements formalization [1]
Design Model [1]
Model verification [1]

FIGURE 1. **Model-based approach for requirements validation and system design.**

from an ontology with precisely defined concepts and semantic relationships in the domain of the system under design. To tackle the lack of a unique interpretation for the natural language syntax, we employ *requirement boilerplates*, i.e. textual templates with placeholders to be filled with ontology elements. These elements are semantically interrelated, as in the conceptual model that supports the process in [1] and they are part of a well-defined *ontology architecture*.

The ontology architecture relies on an *upper ontology* with concepts and common relations that are applicable across a range of *system-specific domain ontologies*. The upper ontology concepts *determine the scope and the expressiveness* of the semantic modeling framework, which allow combining three different perspectives, when specifying requirements using boilerplates, namely [2]: (i) the data, (ii) the functional and (iii) the behavioral perspective. The domain ontologies integrate general knowledge for the problem domain, including knowledge that may be omitted from the requirements but complements them. All domain-specific concepts are "typed" as concepts of the upper ontology.

This modular and extensible ontology architecture supports, through integrating different domain ontologies, the main motivation behind our work, i.e. *to deliver a semantic modeling and analysis framework that can be adapted to multiple domains, while not being dependent on a specific design modeling language* [3]. Instead, the ontology architecture opens the possibility to adapt it to different industrial standards (e.g. AUTOSAR) through introducing in the domain ontologies constructs of particular language metamodels. When comparing our approach with domain-specific languages (DSLs) for the specification of system requirements in particular domains, our approach is more general: it allows to adapt the domain ontologies in order to expand the scope of specification capabilities, instead of relying on fixed language semantics.

The whole semantic framework, together with the ontology terms mentioned in requirements, enable a series of automated *semantic analyses* that guide the engineer towards improving the requirements specification. These analyses detect specification flaws, such as requirements that are incomplete or missing (incompleteness), requirements with terms undefined in the domain ontologies (noise), inconsistencies and so on. We also introduce a technique for detecting semantically related terms within different requirements (semantic relatedness analysis), with the aim to *elicit possible tacit knowledge*. This allows to derive and explicitly define additional domain knowledge. Using concrete examples from industrial experience, specifically on requirements for space systems, we show how the different analyses can be combined with the knowledge stored in domain ontologies, for eliminating the specification issues detected.

Our work is not the first ontology-based specification approach using requirement boilerplates. A recent work was reported in [4], [5], whereas the CESAR platform [6] has also introduced a similar approach. In [4], [5], the requirements formalization/validation takes place by semantically mapping the boilerplates to the design abstraction levels of the EAST-ADL modeling language. In our case, the ontology concepts used in the boilerplates are "typed" according to the upper ontology, which results to not having been locked in a specific modeling language. Another difference is that, for the requirements validation, we support more analyses, apart from consistency and completeness, as is the case in [4], [5]. Regarding [6], our approach differs in that the semantic analysis extends even beyond the boundaries of existing system-specific domain ontologies, because all specified terms are interrelated according to the concepts of the upper ontology. At the behavioral level, our correct-by-construction design process in [1], avoids verification exclusively by model checking that does not easily scale to larger models.

The evaluation of the proposed approach is two-fold, i.e. with respect to the adequacy of the semantic framework to *express real industrial requirements of different types*, and with respect to its *efficiency and effectiveness from the users point of view*. Regarding the first priority, a case study is presented with 65 requirements of different types (functional, performance, interface, operational) from a virtual earth-observation reference satellite, called Eagle Eye [7] that was developed by the European Space Agency (ESA). As for the second priority, we ran an experiment to assess the required effort for engineers to accomplish a specification task and their effectiveness. The engineers who participated in the experiment did not know the details of the ontology architecture, but they could only take advantage of it using our web-based tool,[1] for specifying requirements. More concretely, we measured the man-effort for formulating requirements from the natural language description of a space system, into our boilerplate language. We claim that *the additional cost for applying the approach is affordable and certainly justified for critical embedded systems*, such as the space systems considered. These findings substantiate the *applicability of our proposal* and provide valuable feedback, for a number of aspects that have to be addressed, when adopting it into an industrial context.

The concrete research contributions of this article are:

1) The upper ontology and the ontologies of the proposed architecture[2] that facilitate integration of domain

[1]Ontology-based Tool for Requirements Specification and Analysis: https://iotlab.csd.auth.gr:8081.

[2]Ontology provided online: https://depend.csd.auth.gr/software/requirements-ontology.

TABLE 1. Ontology-based requirements analysis.

Contribution	Related works	Our approach
Requirements qualities		
(In-)completeness analysis	[8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [6], [4]	✓
(In-)consistency analysis	[24], [25], [9], [26], [27], [28], [12], [13], [14], [15], [16], [29], [30], [31], [17], [32], [33], [19], [34], [35], [36], [37], [38], [20], [39], [22], [23], [40], [6], [4], [41]	✓
Ambiguity analysis	[25], [42], [26], [27], [12], [14], [15], [43], [17], [44], [45], [23], [46], [6]	✓
Noise analysis	[12], [6]	✓
Requirements management/evolution	[47], [8], [48], [24], [49], [9], [11], [28], [50], [14], [15], [51], [52], [53], [54], [55], [18], [34], [35], [56], [57], [37], [58], [59], [60], [61], [62], [63], [6]	✗
Domain knowledge representation	[13], [14], [64], [53], [65], [17], [32], [66], [38], [20], [67], [68], [69], [70], [71], [62], [23], [6]	✓
Tool	[29], [65], [54], [37], [68], [60]	✓
Tacit knowledge extraction		
Relatedness analysis	[72], [73], [74], [6]	✓
Similarity analysis	[40], [63], [46]	✗
Implicit assumptions	[24], [27], [28], [12], [13], [15], [51], [31], [43], [75], [65], [17], [32], [33], [19], [35], [36], [37], [20], [59], [71], [39], [22], [6], [4], [41]	✓
Requirements formalization/validation		
Properties derivation	[23], [4], [41]	✓ [1]
Formal verification	[22], [23], [4], [41]	✓ [1]
Correctness by construction		✓ [1]
Validation wrt. industrial requirements	[49], [28], [14], [18], [20], [67], [69], [62], [21], [6], [4]	✓

ontologies, without needing to modify the semantic analyses implemented.
2) A methodology for building domain ontologies, which was applied to capture the domain knowledge of a satellite Attitude & Orbit Control System (AOCS).
3) The syntax and semantics of the boilerplate language and the automated inference of relationships between terms in the boilerplate placeholders.
4) The semantic analyses, including the semantic relatedness analysis for eliciting possible tacit knowledge.
5) The Eagle Eye case study and the users' experiment results, as well as a critical discussion of issues related to the adoption of the approach in an industrial context.

The rest of the article is structured as follows. The next section discusses the related work. Section III presents the overall semantic modeling framework, including the upper ontology and the proposed ontology architecture. Section IV introduces a method for building domain ontologies, the syntax and semantics of the boilerplate language and how the various ontology relationships are used in the semantic representation of requirements. The implementation of the proposed semantic analyses is described in Section V, whereas the semantic relatedness analysis for eliciting tacit knowledge is presented in Section VI. In Section VII, we discuss the results from the Eagle Eye case study. Section VIII presents the experimental evaluation of the approach from the users point of view and Section IX summarizes the issues related to its industrial adoption. Finally, the last section concludes the paper and refers to future research prospects.

II. RELATED WORK

Table 1 summarizes the research contributions of related works on ontology-driven requirements engineering. This classification was based on the systematic literature review in [76], from which all references until 2015 were examined with respect to the scope of our work, whereas some additional works beyond 2015 were also included. Only the tools that were found to be publicly available are reported.

It is worth to note that a number of studies referenced in Table 1 are part of a larger project or continuation work covering various aspects of requirement analysis. However, only a small number of them, including ours, provide a detailed formalization of the ontology relationships and rules. Moreover, according to Fig. 2, it seems that this work provides a more complete coverage of capabilities. An interesting approach, in terms of the provided support, is the CESAR project [6] with 9 contributions, whereas [4], [23] and [14] follow with 6 contributions (the latter article is also related to the CESAR project). The closely related works to our research contributions are described henceforward.

a: PATTERN-BASED SPECIFICATION

The CESAR reference technology platform [6], [14], [74], [77] addresses the standardization of requirements across the development stages of embedded systems. CESAR introduces a Requirements Specification Language (RSL) with boilerplates that combine three clauses, like in our language (prefix, main part and suffix). RSL's boilerplate placeholders (attributes) are defined in an ontology. These placeholders must be filled in with terms from a domain-specific ontology. A language of *property patterns* with formal semantics

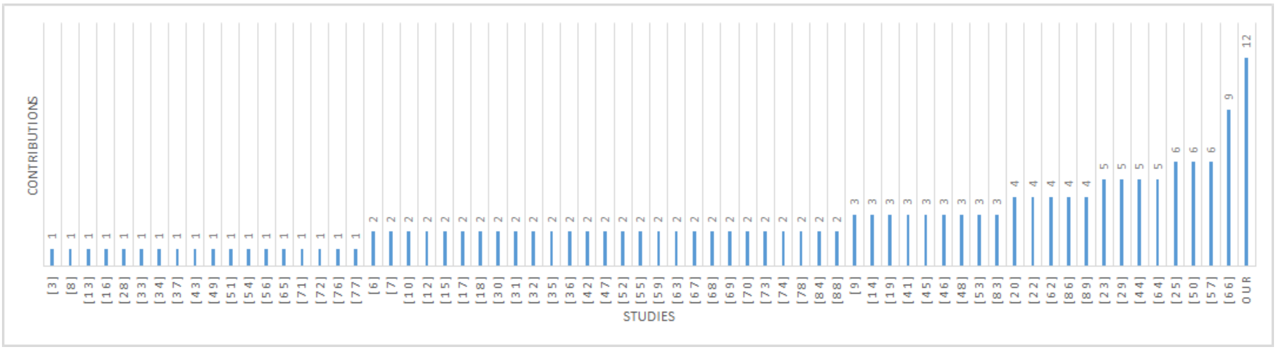


FIGURE 2. Contributions per related work.

is used, but no exact association of boilerplates with the specific patterns is provided, as we do in [1]. CESAR can also perform relatedness analysis to guide the engineers in writing requirements using semantically related concepts. Elicitation of tacit knowledge, one of the contributions of this work, is not supported. Finally, semantic analysis in CESAR seems to be scoped only in comparing the information specified in requirements with the domain ontology. Our semantic analyses differ in that they are based on the upper (domain-independent) ontology, which makes them applicable to multiple system domains, while the existence of domain knowledge is not necessary. CESAR users rely on the DODT (Domain Ontology Design Tool) tool, which allows editing requirements, based on the available boilerplates, and supports the analysis of requirements, like our web-based tool.

In [4], [5], the authors introduced the Requirement Specification language for Automotive systems (RESA). RESA supports the specification of embedded system requirements, their semantic representation using events and their semantic analysis through identifying thematic roles. When compared to that work, our semantic analysis is not limited only to consistency and completeness checking, whereas we additionally propose a way to uncover tacit knowledge in a set of requirements. Finally, as it was already noted, an additional difference is that for the requirements validation at the level of system's behavior, we follow a correctness-by-construction design methodology [1].

b: FORMAL VERIFICATION

A noteworthy consistency checking methodology for requirements is the one in [41]. This approach introduces a grammar for a natural-like language to parse time-related functional requirements and translate them into Linear Temporal Logic (LTL) formulas, amenable to semantic analysis for consistency. The proposed solution is evaluated over a rich set of examples, which is however limited only to a small subset of functional requirements. Furthermore, when interpreting sentences without using domain ontologies, the matching with words is inevitably a manual procedure and this is an important limitation in the applicability of the method to real-world problems.

c: DOMAIN ONTOLOGIES FOR SPACE SYSTEMS

The Systems and Software Division of the Jet Propulsion Laboratory (JPL) has performed related work towards improving the systems engineering practices, in an attempt to master the increasing complexity of space flight missions. More concretely, JPL has developed several ontologies in Web Ontology Language 2 (OWL 2), for the domain of space flight missions [78], [79], [80], [81]. Their model-based approach is focused on the Systems Modeling Language (SysML). The ontology is used to generate a consistent model by semantic reasoning and analysis techniques. However, JPL's ontologies address only top-level application domain knowledge. The inner structure of requirement specifications is addressed briefly and mainly outsourced to SysML. According to [82], the adaptation of the SysML requirements concept is considered as a weakness, therefore they propose to combine the JPL mission concepts with a more detailed requirement semantics model.

d: SEMANTIC RELATEDNESS

The problem of generating knowledge recommendations through semantic relatedness has been previously investigated in [83]. The analysis is applied to a learning organizer ontology, in order to measure the semantic relevance between a learning resource and the learning context of a learner. According to the authors, their measurement approach is simple and achieves adequate results. Semantic relatedness analysis has also been applied to enhance traceability of requirements [72], [73], or to guide engineers in using semantically related concepts within boilerplate requirements [74]. The latter studies are outside the scope of our work. Similarity analysis, a special case of relatedness analysis, has been previously used in several studies [40], [46], [63] to process and analyze natural language requirements. However, similarity analysis is more limited in scope than semantic relatedness analysis, which takes all kinds of relations into account.

III. SEMANTIC MODELING FRAMEWORK FOR REQUIREMENTS SPECIFICATION

A. PRELIMINARIES ON ONTOLOGY-BASED MODELING

An ontology is an explicit definition of concepts in a well-bounded domain of knowledge. The main components

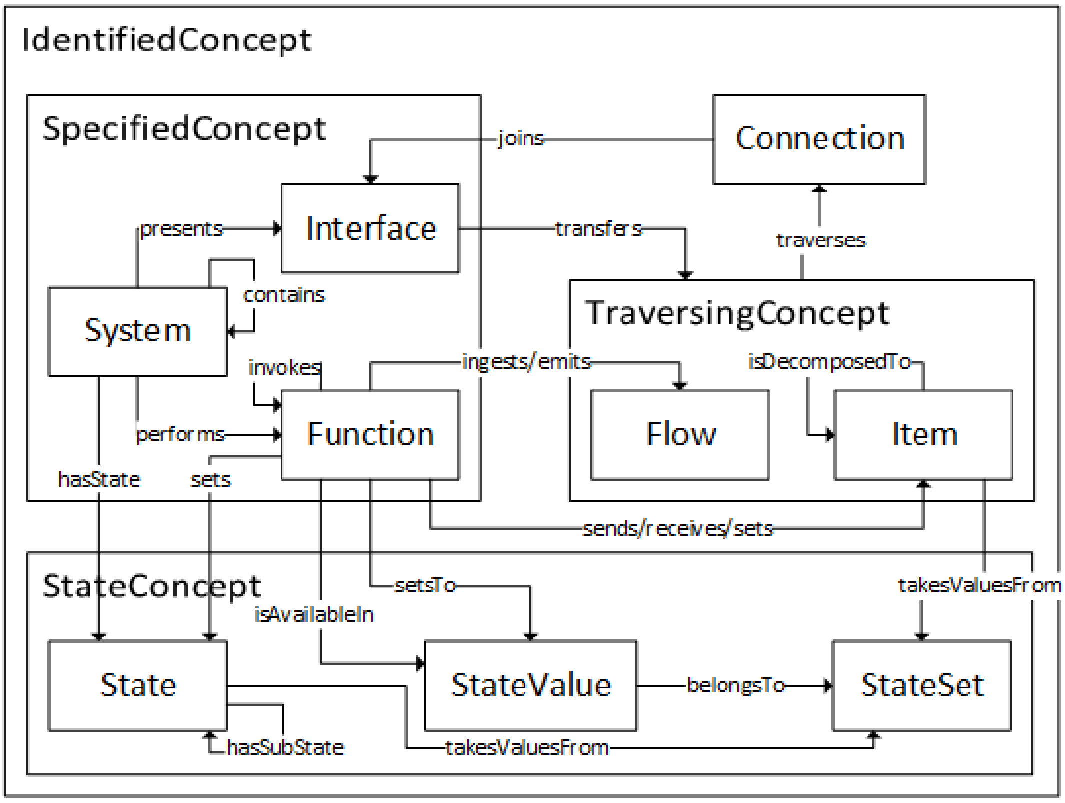


FIGURE 3. System attributes ontology (SAO).

of an ontology are the classes, their individuals, the attributes, the relationships and the axioms.

Classes (also termed *concepts*) are abstract groups or collections of entities/"things" within a domain. *Individuals* (also termed *instances*) are the "things" represented by a concept. Classes have *attributes* (or *properties*), which can be another class or individual, to store information that is specific to the class. *Relationships* (or *relations*) describe the way classes and individuals can be related to one another (e.g. the subsumption relation, according to which a class - called subclass - is subsumed by another class). Finally, axioms are used to constrain values for classes or individuals. They are *assertions* (and *rules*) in a logical form that all together represent the overall theory that the ontology describes in its domain of application.

B. SYSTEM AND ATTRIBUTES ONTOLOGY (SAO)

The upper ontology called *System Attributes Ontology (SAO)* is depicted in Fig. 3 and provides the context for encoding all semantic relationships between the terms mentioned in the boilerplate placeholders (the boilerplate syntax is given in Tables 6, 7 and 8). SAO allows expressing requirements by combining three different perspectives, namely [2]: (i) the data, (ii) the functional and (iii) the behavioral perspective. Moreover, it integrates the conceptual model in [1], which supports the formalization of system requirements.

1) DATA PERSPECTIVE

The data perspective concerns with the input-output data relationships and the structural dependencies in a system. Within the SAO ontology, all these relationships are defined using structures for input and output data (e.g. decomposition of `Items`), and how the system is composed (e.g. `contains` relation, `Interfaces` and `Connections` to external systems).

For example, in the following boilerplate requirement, a `Communication` system contains an `S-Band` system, i.e. the verb "contain" expresses a composition relationship between two systems.

M16: `<System:Communication>` shall contain `<System:S-band>`

Also, in the boilerplate requirements below, the `S-band` system receives the uplink signal that is sent by the `Ground`, i.e. verbs "receive" and "send" express an input-output data relationship between the two systems.

M4: `<System:S-Band>` shall receive `<Item:uplink signal>`

M3: `<System:Ground>` shall send `<Item:uplink signal>`

2) FUNCTIONAL PERSPECTIVE

The functional perspective allows expressing how the system's functions handle data (e.g. through the `performs Function` relation): it is possible to express that some data are received, sent or manipulated by executing specific functions.

For example, in the following boilerplate requirement, the `S-Band` system performs a function that `demodulates the uplink signal`.

M7: `<System:S-band>` shall perform `<Function:demodulate uplink signal>`

The boilerplate requirement below is a specification that combines the data perspective with the functional perspective:

M10: `<System:S-band>` shall transfer `<Item:telemetry data>`

This requirement implies an `Interface` definition (i.e. structural dependency), which can be automatically inferred through semantic reasoning, to enable the `transfer` of `telemetry data` (a functional specification). More concretely, the inference yields the relations implied by the following semantically equivalent specifications:

M9: `<System:S-band>` shall present `<Interface:S-band-interface>`

M10: `<Interface:S-band-interface>` shall transfer `<Item:telemetry data>`

3) BEHAVIORAL PERSPECTIVE

The behavioral perspective allows expressing a system's behavior in a state-based specification style and how it interacts within the overall system context.

This is possible through defining `State` information for the system, given as `StateValues` that `belong to` a `StateSet`. A `StateValue` is `set` by a `Function` and can be used (through the `isAvailableIn` relation) as a condition that warrants a state-transition.

Finally, a system can affect another system through traversing events (i.e. `Flows`), states or data (i.e. `Items`). For example, for specifying that the `S-Band` system shall receive `uplink signal`, as long as the `S-Band Mode` is `Active`, we use the following prefix to extend the example requirement in Section III-B1.

P3: As long as `<System:S-band>` `<State:Mode>` is `<StateValue:Active>`

M4: `<System:S-Band>` shall receive `<Item: uplink signal>`

4) SAO DESIGN PRINCIPLES AND INFERENCE OF IMPLICIT ASSUMPTIONS

SAO has been defined independently of the type of system under design and we therefore refer to it as the *domain-independent ontology*. This implies that the SAO concepts and their relationships (formally defined in first-order logic in the online Appendix A.1[3]), should be specialized for the type of system and abstraction level of the requirements specification. The highest level of abstraction is the whole system under design, while the lower the level of abstraction, the more detail is taken into account (cf. system decomposition example in Section III-B1).

SAO adopts some (but not all) elementary concepts and properties of the JPL ontologies (cf. Section II); it may be considered as a simplification and at the same time as an extension of the JPL Mission Ontology.

The main intend behind its design is to enable capturing the asymmetric *producer/consumer* relationships that characterize the functional perspective in requirements specification: any `System` is *producing* `Flow` and `Item` entities, and when two systems are connected by a `Connection`, another `System` *consumes* the produced flows and items (Fig. 3).

System requirements are, by definition, a partial specification. Their semantic representation does not allow automated reasoning, unless having *made explicit any implicit information*, which result in spots of incompleteness of the semantic model. To identify assumptions that should become explicit, we take into account the producer/consumer relationships imposed by SAO. All these assumptions result in creating `Connection` instances (and `Interfaces` when necessary) and property relations, using Shapes Constraint Language (SHACL) rules [84]. The derived elements that complete the partial specification remain in the ontology as long as the set of requirements, where they come from, persist. The following two cases are taken into account:

- A `Function` performed by a `System` sends (or emits) a `Traversing Concept` that a `Function` of a different `System` receives (or ingests). Then, (i) `Interface` individuals will be created to transfer the `TraversingConcept`, and (ii) a `Connection` individual will be created to join the `Interfaces` and traverse the `TraversingConcept`.
- A `TraversingConcept` is produced (resp. consumed) by two `Systems` through their `Interfaces` using a `Connection`. In this case, `transfer` and `traverse` relations are created, for the `Interfaces` and `Connection`.

Example 1: Let us consider the following requirements, representing the fact that a `System` sends an `Item`, and another `System` receives it:

M3: `<System:Rate Control Guidance>` shall send `<Item:spacecraft attitude and body rates>`

M4: `<System:Attitude Error Generator>` shall receive `<Item: spacecraft attitude and body rates>`

According to the first implicit assumption, the new instances `Interface1` and `Interface2` are created that are presented respectively by `System:Rate Control Guidance` and `System:Attitude Error Generator`, whereas `Connection1` is also created to join the two interfaces and traverse the referred `Item:spacecraft attitude and body rates`. The resulting representation of these two requirements is shown in Fig. 4.

[3]Appendix: https://depend.csd.auth.gr/software/requirements-ontology.

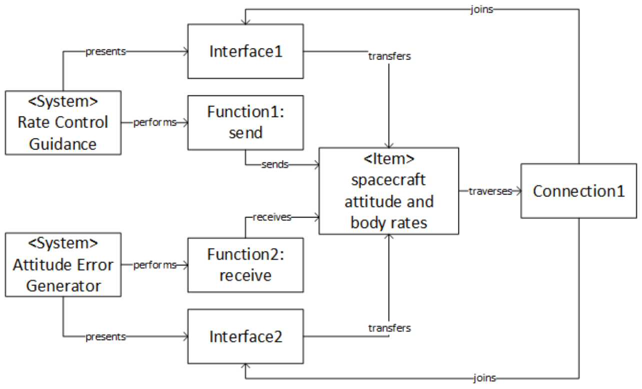
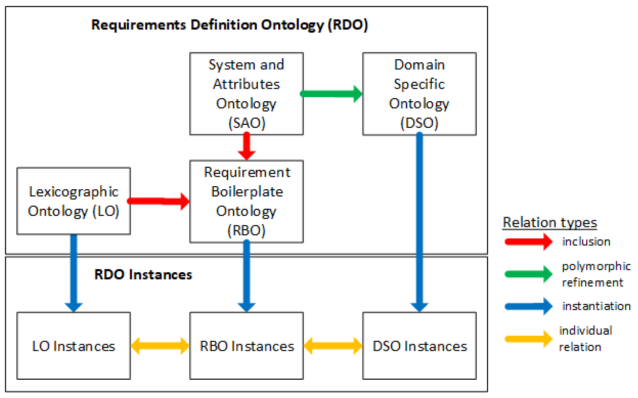


FIGURE 4. Connection representation.



Requirements Definition Ontology (RDO)
System and Attributes Ontology (SAO)
Domain Specific Ontology (DSO)
Lexicographic Ontology (LO)
Requirement Boilerplate Ontology (RBO)
RDO Instances
LO Instances
RBO Instances
DSO Instances
Relation types
inclusion
polymorphic refinement
instantiation
individual relation

FIGURE 5. Ontology architecture.

C. ONTOLOGY ARCHITECTURE

The overall ontology architecture relies on the SAO upper ontology, which provides the core glossary to support various system-specific domain ontologies. This allows *covering multiple heterogeneous system domains*. The overall architecture has been designed in order to enable:

- Design of *domain-specific ontologies with self-contained semantics*, i.e. without need to refer to terms outside of their context (maintainability).
- Integration of additional domain-specific ontologies, *without needing to modify the overall ontology architecture* (extensibility).
- *Resolution of semantic conflicts at the top-level ontology*, when defining mappings between the different lower-level ontologies.
- A *clear separation between an ontology and a knowledge base* (i.e. ontology combined with individuals), such that ontologies only contain classes, properties, relationships and axioms. Semantic analysis takes place at the instance level, but the overall ontology architecture can be easily reused in multiple different projects.

The ontology architecture in Fig. 5 is given as a modular collection of five ontology categories including SAO.

The *Requirement Boilerplate Ontology (RBO)* provides semantic definitions for the boilerplates (e.g. main, prefix, suffix) and their placeholders. It imposes structural constraints in assembling a requirement, which provide a formal syntax. RBO classes are related to classes of the SAO.

The *Domain Specific Ontologies (DSOs)* contain domain-specific classes for the system under design. DSOs import all SAO classes and specialize them.

The *Lexicographic Ontology (LO)* contains synonym and antonym semantic definitions for the boilerplates. It allows using paraphrases inside a boilerplate and contributes to the consistency analysis of requirements.

The *Requirements Definition Ontology (RDO)* serves as a repository of requirements and uses elements from the LO and DSO, following the semantic definitions of RBO. Therefore, RDO imports the RBO, LO and DSO ontologies, and transitively imports (through the DSO) the SAO ontology.

The types of relations in Fig. 5 are explained after showing the semantic representation of the following example requirement, using the ontology architecture.

Example 2: Let us consider a natural language requirement, for the reaction wheel (RW) unit telecommand processing:

> If RW processor receives a TC(8,1) manual commanding command, the RW actuator processor shall set the RW torque.

The example requirement is broken into a prefix and a main clause, as follows:

P1: if `<System:RW processor>` receives `<Flow: TC(8,1) manual commanding command>`
M1: `<System:RW processor>` shall set `<Item:RW torque>`

The details of the representation of each clause within the overall ontology architecture is shown in Fig. 6. The prefix and main clauses are respectively represented by the `RBO:P1` and `RBO:M1` classes of RBO. Both clauses are related (`isRelatedToPlaceholder` property) to the `DSO:RW processor`, individual of the `DSO:Actuator processor unit`. In the main clause, `DSO:RW processor`, `DSO:set` and `DSO:RW torque` are respectively instances of (refinements of) the `SAO:System`, `SAO:Function` and `SAO:Item` classes. In the prefix, `DSO:RW processor`, `DSO:receives` and `DSO: TC(8,1) manual commanding command` are instances of (refinements of) `SAO:System`, `SAO: Function` and `SAO:Flow` respectively. In this representation, every element in the clauses is indirectly related to a SAO class, which enables the execution of semantic analysis queries.

The four types of relations that connect the classes and individuals of the described representation are:

- *inclusion* (red arrow): if ontology *A* includes ontology *B*, then all classes defined in *B* are also found in *A* together with the restrictions, properties and other axiomatic relations of these classes from *B*. Thus, in RBO there are classes from the LO and DSOs that are equipped with structural constraints from those ontologies for assembling a requirement.

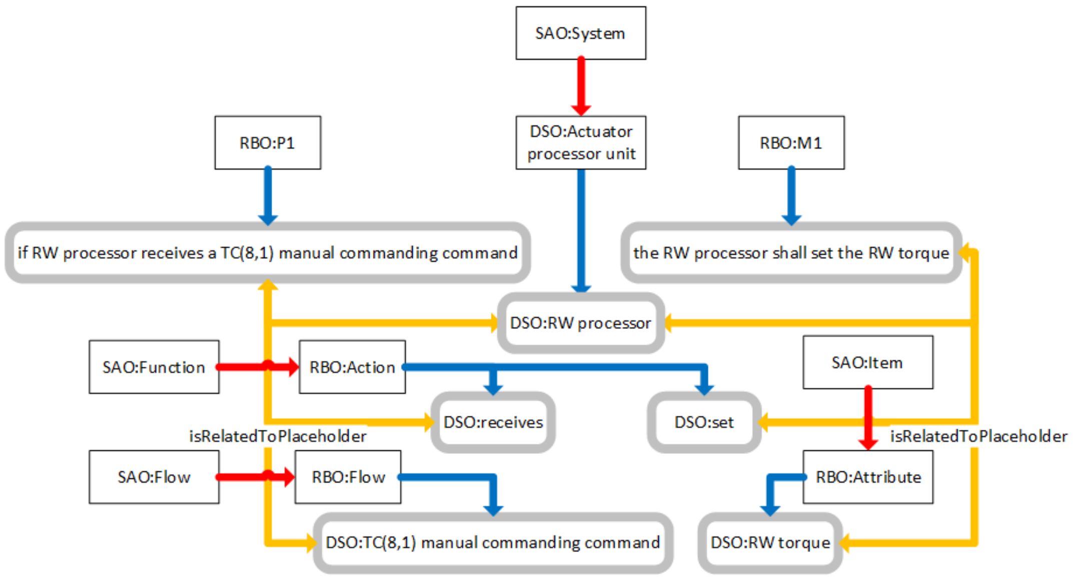


FIGURE 6. Ontology relationships for the requirement in Example 2.

- *polymorphic refinement* (green arrow): a definition from an ontology is included and refined. For example, the `System` class defined in SAO, can be included in a DSO and extended in order to apply to AOCS systems, and included in another DSO and further extended to be applied to electric power supply systems.
- *instantiation* (blue arrow): ontology *A* imports ontology *B*, where *B* is decomposed only into individuals. For example, the RDO Instances import RDO and is composed solely of RDO individuals. The instantiation relation type separates individuals from the repository of ontology classes that can be read or physically shared, for improved reusability, as well as for sharing and confidentiality purposes.
- *individual relation* (orange arrow): denotes an "is related to" relationship between RBO and DSO instance ontologies via a property. For instance, in the Example 2, the RBO individual is related to the `RW processor` of the DSO via the `isRelatedToPlaceholder` (Fig. 6).

Through this representation of requirements within the ontology architecture, we move from the textual form of requirements to a precisely defined semantic structure that consists of uniquely identified ontology elements (i.e. classes, individuals, properties and relationships). The role of these elements is to explicitly state and encode knowledge, which is taken into account in order to perform semantic analysis and possibly extract tacit knowledge, if any.

IV. SEMANTIC REPRESENTATION OF REQUIREMENTS

A. DOMAIN SPECIFIC ONTOLOGY (DSO)

To express requirements from different system domains, we need DSOs that will encompass these domains. Any such ontology will import all SAO classes and will further specialize them. The DSO concepts, together with their interrelationships will therefore provide a semantic model of the system's domain at each level of specification, from the system as a whole to each hardware and software subsystem. The example in Fig. 7 shows the different levels of a DSO for the space system domain.

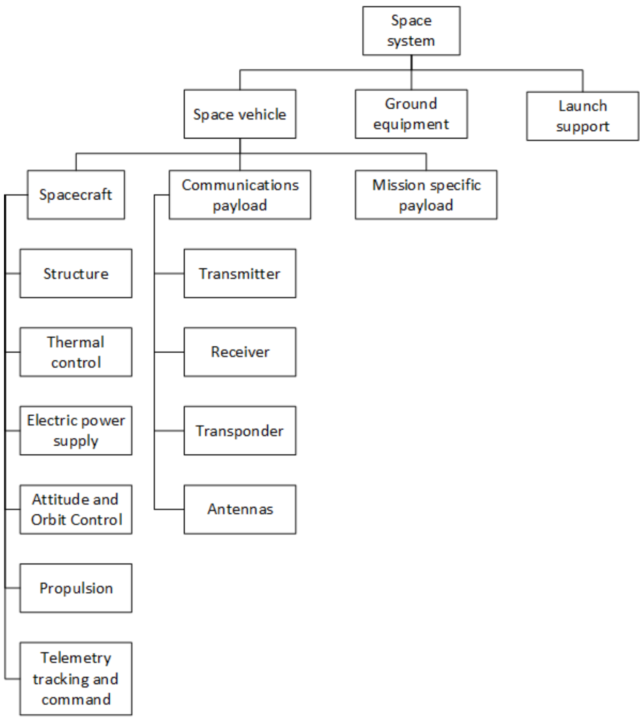


FIGURE 7. Space system DSO.

Fig. 8 introduces a process for the elicitation of a DSO. First, an upper ontology serves as a top-level structure for the

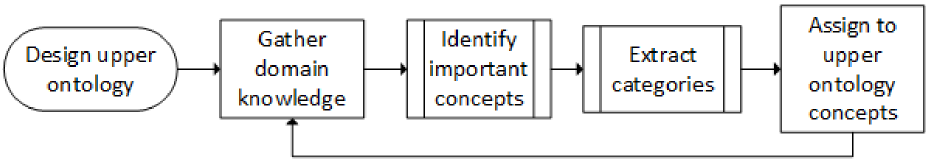


FIGURE 8. Domain-specific ontology elicitation process.

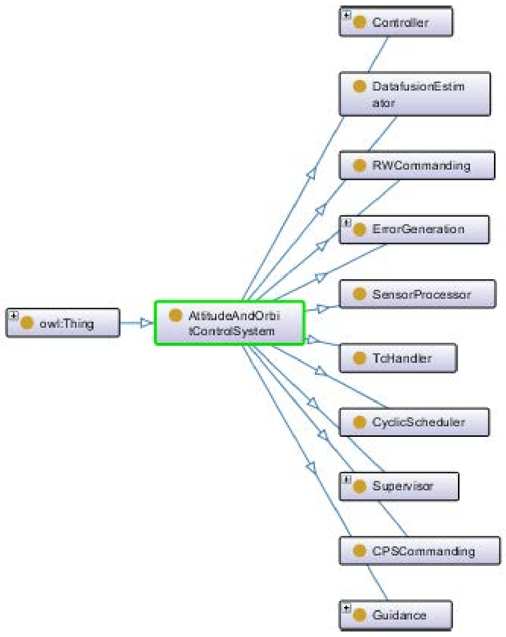


FIGURE 9. Attitude and orbit control abstract concept categories.

TABLE 2. OWL constraints with examples used in DSO.

OWL	DL Symbol	Keyword	Example
someValuesFrom	∃	some	contains some Guidance
allValuesFrom	∀	only	performs only acquires_strobing_rate
hasValue	∋	has	hasState has AOCS_mode
minCardinality	≥	min	contains min 1
Cardinality	=	exactly	contains exactly 3
maxCardinality	≤	max	contains max 2
intersectionOf	∩	and	Guidance and Controller
unionOf	∪	or	CPSCommanding or RWCommanding
complementOf	¬	not	not acquires_telemetry

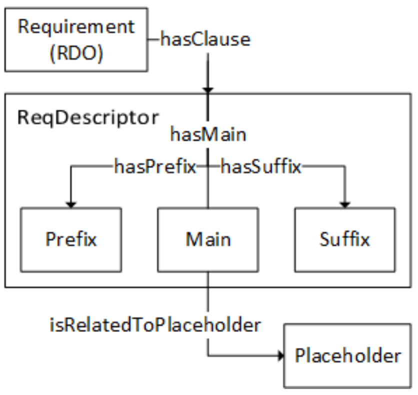


FIGURE 10. Requirement boilerplate ontology.

definition of domain ontologies; in our case, this is the SAO. Then, we gather domain knowledge from various sources, including domain experts, requirement specification documents and architecture designs from previous projects. In this step, we try to identify and extract abstract categories, i.e. concepts that characterize all systems in the domain, which may include collections of objects, distinct functionality, exchanged information, types of objects or kinds of things.

Fig. 9 presents a decomposition of our AOCS domain ontology, in which several categories are shown for different functionalities of the AOCS system. For example, since any AOCS system shall process inputs from a set of sensors a `SensorProcessor` subsystem will exist and therefore it has been added as an abstract category for the `AttitudeAndOrbitalControlSystem` class.

The next step concerns with the assignment of identified categories to the upper ontology concepts and the representation of their interrelations by appropriate OWL constraints (Table 2). This is crucial for incrementally building the DSO, while unfolding the knowledge extraction process. For example, an `AttitudeAndOrbitalControlSystem` shall `contain` *some* `SensorProcessor` instances, and the latter will have *exactly* two functions: (i) `receive` *only* `sensor input` and (ii) `perform` *only* `sensor input processing`, where `sensor input` is a `SAO:Item` subclass and `sensor input processing` is a `SAO:Function` subclass of the `SensorProcessor` class.

Finally, for each of the already modeled categories the process is iterated from the second step following a top-down or bottom-up approach to further assign subcategories. By focusing on a set of clearly defined concept categories, we avoid to restrict knowledge extraction only to specific categorical concepts, while the effort to develop an initial knowledge base can be limited.

B. REQUIREMENT BOILERPLATE ONTOLOGY (RBO)

RBO in Fig. 10 encodes the elements of requirement specifications in a boilerplate form, like in the Examples 1 and 2.

Our boilerplates combine at most three clauses:

- `Prefix`, which specifies a stimulation or a condition,
- `Main` that specifies an expected action or state and
- `Suffix`, which is used to specify additional constraints to the expected action.

TABLE 3. Boilerplate grammar.

<boilerplate>	::=	[<prefix>] <main> [<suffix>]
<main>	::=	M1 | M2 | M3 | M4 | M5 | M6 | M7 | M8 | M9 | M10 | M11 | M12 | M13 | M14 | M15 | M16
<prefix>	::=	<prefix> <logic connective> <simple prefix> | <simple prefix>
<simple prefix>	::=	P1 | P2 | P3
<suffix>	::=	<suffix> <logic connective> <simple suffix> | <simple suffix>
<simple suffix>	::=	S1 | S2 | S3 | S4 | S5 | S6
<logical expression>	::=	<logical expression> <logic connective> <logical expression> | <state value constraint> | <occurring functionality>
<state value constraint>	::=	**item** is **stateValue** | **system state** is **stateValue**
<occurring functionality>	::=	**system/function** sets [<quantifier>] **item** | **system/function** sends [<quantifier>] **item** | **system/function** receives [<quantifier>] **item** | **system/function** ingests **flow** | **system/function** emits **flow** | **system** performs **function**
<logic connective>	::=	or | and | xor

TABLE 4. Auxiliary grammar.

<quantifier>	::=	<affirmative> | <negative> | <closed-interval>
<affirmative>	::=	<non-numerical-affirmative> | <numerical-affirmative>
<non-numerical-affirmative>	::=	all | only
<numerical-affirmative>	::=	more than <numerical> | less than <numerical> | exactly <numerical>
<negative>	::=	none | no
<closed-interval>	::=	at least <numerical> | at most <numerical>
<numerical>	::=	<number> [<number-unit>]
<number>	::=	1 | 2 | 3 | 4 | 5 | etc.
<number-unit>	::=	<time-unit> | meters | kilometers | volt | etc
<time-unit>	::=	seconds | minutes | milliseconds | etc

Definition of boilerplates as a sequence of clauses offers *modularity*, simplifies the problem of boilerplate composition (*expressiveness*), allows for a *unique interpretation* of requirements and facilitates automated reasoning for semantic analysis using the relevant ontology-based technologies. A requirement shall always have a main clause.

The main RBO classes are:

- the `Placeholder` class representing:
  - the SAO ontology concepts that are used in boilerplate placeholders,
  - the fixed elements of the boilerplate syntax (e.g. if, asSoonAs, asLongAs, or, and and xor), and
  - classes/instances of synonyms and antonyms (e.g. `send`, `receive`, `transmit` etc.) defined in the LO.
- the `ReqDescriptor` class that encodes the used boilerplate clauses.

Relations between RBO's classes have been defined in first-order logic (online Appendix A.2).

1) BOILERPLATES SYNTAX AND SEMANTICS

Tables 3 and 4 show the boilerplate language syntax, in a context-free grammar (the notation for the grammar rules is defined in Table 5). In general, the grammar is left-associative; for the rules with ambiguous associativity (e.g. the <*logical expression*> rule), left-associativity is imposed by the parser.

TABLE 5. Notation used in the boilerplate grammar.

Symbol	Meaning
[...]	optional
... | ...	or
<...>	non-terminal symbol (or boilerplate placeholder)
bold string	token with ontologically defined semantics
string	token with string ontological representation

TABLE 6. Prefix templates syntax.

ID	Template	Explanation
P1	If/Unless <logical expression>	Expresses a logical condition. Paraphrases: in case, provided that, on condition that
P2	As soon as <occurring functionality>	Expresses a temporal stimulation concerned with the point in time of an completed occurring functionality. Paraphrases: in the moment, immediately, once
P3	As long as <occurring functionality>	Expresses a temporal condition concerned with a period. The prefix clause and the main clause take place simultaneously. Paraphrases: meanwhile

TABLE 7. Suffix templates syntax.

ID	Template
S1	(<numerical-affirmative> | <closed-interval>) [per <time-unit>]
S2	(after/before) **flow**
S3	[every/for a period of/within/for at least] <number> <time-unit> [from **flow**]
S4	at the beginning/at the end
S5	at even intervals
S6	(concurrently with / sequentially to) **function**

Table 8 introduces the syntax for the main clause, which may be used for system requirements referring to a `System/Function` (templates M1-M7), for specifying implementation details for a `System/Interface` (templates M8-M10) or for subordinate design specifications (templates M11-M16). The optional existence of **not** in main constrains a requirement's semantics, i.e. it affects semantic reasoning, but doesn't have any impact to the resulting semantic model.

As shown in the grammar rule for <*prefix*> in Table 3, a requirement may have a simple prefix or possibly multiple prefixes in sequence, separated with a logic connective (e.g. and, or). The same is true for a requirement suffix. The syntax of prefixes/suffixes is defined in Tables 6 and 7 respectively. When specifying constraints using suffix(es) we need a flexible syntax for the <*quantifier*>s (Table 4) that promotes expressiveness, while admitting a clear semantics. To this end, following the guidelines of [85], the quantifier definitions are grouped into three distinct syntax categories, namely <*affirmative*>, <*negative*> and <*closed-interval*>. The current set of templates does not use all possible quantifiers, but the syntax structure clearly foresees the need for more templates to address evolving specification needs.

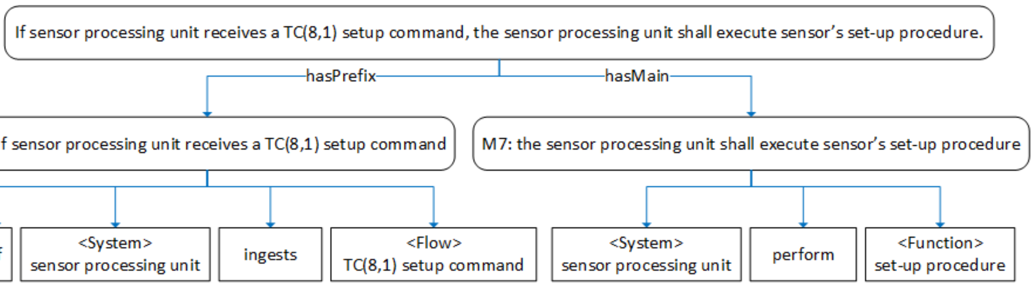

If sensor processing unit receives a TC(8,1) setup command, the sensor processing unit shall execute sensor's set-up procedure.
hasPrefix
hasMain
P1: If sensor processing unit receives a TC(8,1) setup command
M7: the sensor processing unit shall execute sensor's set-up procedure
If
<System> sensor processing unit
ingests
<Flow> TC(8,1) setup command
<System> sensor processing unit
perform
<Function> set-up procedure

FIGURE 11. **Example requirement and its decomposition.**

TABLE 8. **Main templates syntax.**

ID	Template	Explanation
M1	**system/function** shall [not] set [<quantifier>] **item** [to **stateValue**]	Sets item Paraphrases: initiate, reset, update
M2	**system/function** shall [not] set **state** to **stateValue**	Sets state to value Paraphrases: initiate, reset, update
M3	**system/function** shall [not] send [<quantifier>] **item**	Sends item Paraphrases: forward
M4	**system/function** shall [not] receive [<quantifier>] **item**	Receives item Paraphrases: accept, acquire
M5	**system/function** shall [not] ingest **flow**	Ingests flow Paraphrases: get
M6	**system/function** shall [not] emit **flow**	Emits flow Paraphrases: produce
M7	**system** shall [not] perform **function**	System requirement Paraphrases: execute
M8	**function** shall [not] invoke **function**	Function invokes another function Paraphrases: call, request
M9	**system** shall [not] present **interface** [to **system**]	Presents interface Paraphrases: provide
M10	**system/interface** shall [not] transfer **flow/item**	Transfers TraversingConcept Paraphrases: transmit, broadcast
M11	**system** shall [not] interact with **system** [using **<connection>**]	System requirement expressing interface interaction
M12	**system** shall [not] have state **state** [with values **stateSet**]	System requirement expressing system modes
M13	**system state stateValue** shall [not] have substate **state** [with values **stateSet**]	System requirement expressing system sub modes per mode
M14	**system item** shall [not] take values from **stateSet**	System requirement expressing item stateSet
M15	**system item** shall [not] be composed from **item**	System requirement expressing item decomposition stateSet
M16	**system** shall [not] contain **system**	System decomposition

To further explain the prefixes introduced in Table 6, P2 is appropriate for specifying a temporal relation for an *<occurring functionality>*, while P1 for specifying a logical condition. At the semantic level, there is no difference in representing a temporal or a logical condition (or stimulation).

Not all possible combinations of prefix, main and suffix templates are meaningful. Templates M1-M7 refer to the system's expected behavior, and they can be combined with any prefix and suffix, as opposed to other main templates, which introduce the system's architectural details (e.g. state or system decomposition, connections, interfaces etc.).

The requirement in Fig. 11 is expressed by combining M7 with the P1 prefix. The figure shows the terminal symbols of the boilerplate grammar and how each `ReqDescriptor` clause is decomposed into instances of SAO concepts. Specifically, this requirement expresses that a `System` receives a `Flow` and then performs a `System`'s `Function`.

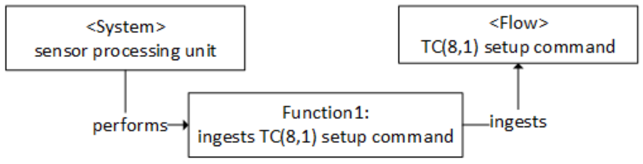

<System> sensor processing unit
<Flow> TC(8,1) setup command
performs
Function1: ingests TC(8,1) setup command
ingests

FIGURE 12. **Prefix representation for the requirement in Fig. 11.**

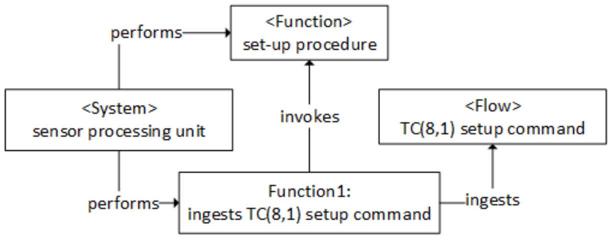

performs
<Function> set-up procedure
<System> sensor processing unit
invokes
<Flow> TC(8,1) setup command
performs
Function1: ingests TC(8,1) setup command
ingests

FIGURE 13. **Representation of the requirement in Fig. 11.**

From Section III-B, it is evident that the semantics of the boilerplate language is closely related to the SAO concept relationships. In many cases, the boilerplate syntax entails the creation of additional relationships between the placeholder terms, by running appropriate SHACL rules. These cases concern with the following three specification patterns:

- Perform relationship
  This pattern refers to main/prefix templates of the form:

  system *verb* (**item**/**flow**/**state**)

  where *verb* is represented by one of the relevant properties (i.e. `ingest`, `emits`, `send`, `receive` and `set`) defined in SAO. Such a pattern entails the creation of two SAO relations:
  - a `System` is related to a `Function` through the `performs` property,
  - a `Function` is related to a `Traversing Concept` or a `StateConcept` through a SAO property relevant to the used *<verb>*.

  For the example requirement in Fig. 11, the additional relations for the prefix are shown in Fig. 12: `System:sensor processing unit` is related to `Function1:ingests TC(8,1) setup command` through the `performs` property, while the latter is related to `Flow:TC(8,1) setup command`

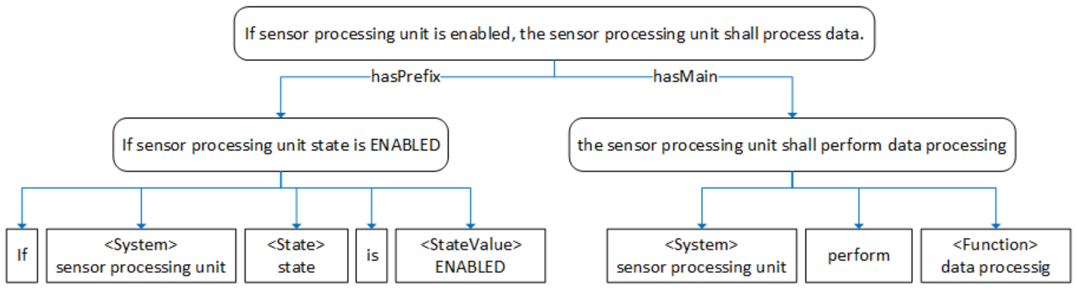

If sensor processing unit is enabled, the sensor processing unit shall process data.
hasPrefix
hasMain
If sensor processing unit state is ENABLED
the sensor processing unit shall perform data processing
If
<System> sensor processing unit
<State> state
is
<StateValue> ENABLED
<System> sensor processing unit
perform
<Function> data processig

FIGURE 14. Example requirement with state value constraint.

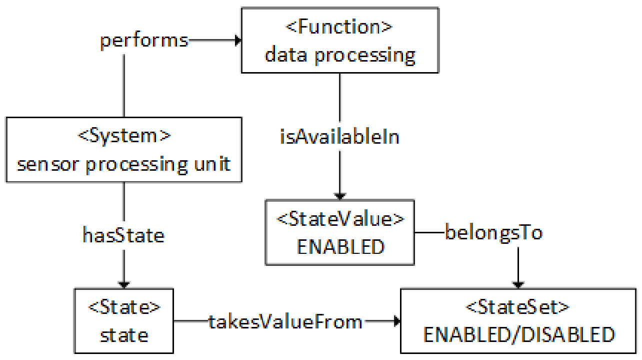


FIGURE 15. Representation of the requirement in Fig. 14.

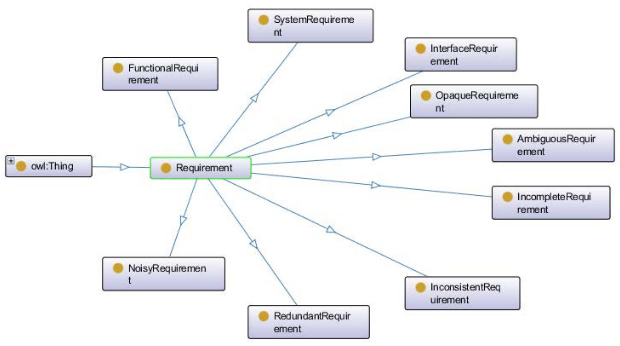


FIGURE 16. Requirement classification.

through the `ingests` property. `Function1` is not explicitly mentioned, but it is an inferred individual of the `Function:ingests` class.

- Invocation relationship
  When a prefix defines an <*occurring functionality*>, a relation through the SAO property `invokes` will be created, to reflect the fact that a `Function` is invoked as a response to a `flow/item` stimuli. Fig. 13 shows how this relation is created, for the example in Fig. 11: the new relationship represents the invocation of the functionality of the main template (`Function:set-up procedure`) by the <*occurring functionality*> in the prefix (`Function1`).
- Availability relationship
  A <*state value constraint*> in a boilerplate prefix implies an `isAvailableIn` relation. In this way, the **state** mentioned in the prefix is related with the functionality specified in the main clause. An example is the requirement in Fig. 14 that is decomposed into a prefix and a main clause. This requirement expresses the fact that a system `State` is in a `StateValue`, and performs a system's `Function` (Fig. 15).

C. LEXICOGRAPHIC ONTOLOGY (LO)

LO is defined independently from the system to be specified and therefore, it is a domain-independent ontology. It encodes in the RDF/OWL language synonymous and antonymous lexical units, which can be used in boilerplate placeholders. For example, as shown in Table 8, the lexical unit "receive" is synonymous to "accept" and "acquire"; when the latter are used in boilerplate placeholders, they are considered as semantically similar to "receive" and dissimilar to "send" (and its synonym "forward"). Semantic similarity/dissimilarity is used in consistency analysis (Section V-B).

The LO consists of lexical units taken from two public ontologies, namely the Gellish English Dictionary [86] and the WordNet Lexical Database [87].

D. REQUIREMENTS DEFINITION ONTOLOGY (RDO)

RDO defines the `Requirement` class and provides the framework, in which the classification of requirements and their semantic analysis takes place. It imports the SAO, RBO, LO and DSO ontologies and defines the mappings and `subclassof` relationships between them, while preserving their semantics.

The relations of RDO concepts have been defined in first-order logic (online Appendix A.3). For the purpose of classifying the erroneous requirements, when running the semantic analyses of Section V, the following subclasses of the `Requirement` class have been defined (Fig. 16): `IncompleteRequirement`, `InconsistentRequirement`, `AmbiguousRequirement`, `NoisyRequirement`, `OpaqueRequirement` and `RedundantRequirement`.

V. SEMANTIC ANALYSIS

The ontology architecture and the semantic representation of boilerplate-based requirements, make it possible to detect flaws in specifications by running appropriate SHACL inference rules [84] based on SPARQL [88]. The semantic analyses that have been implemented are:

- *(In-)Completeness*. Detect requirements that are incomplete or missing.
- *(In-)Consistency*. Detect requirements with similar boilerplates that specify in corresponding placeholders different instances or use contradicting words or different quantifiers (e.g. numbers or number units).
- *Ambiguity*. Detect requirements that have in their boilerplate placeholders values, which can be replaced with instances of more concrete subclasses.
- *Noise*. Detect requirements that refer to concepts/ instances that are undefined in existing domain ontologies.
- *Opacity*. Detect requirements that specify irrelevant instances of concepts in their boilerplate placeholders for the subject.
- *Redundancy*. Detect requirements with the same boilerplates that specify semantically equivalent values to their placeholders.

A. (IN-)COMPLETENESS

The completeness of a set of requirements is assessed from two different perspectives. From an *internal* point of view, incomplete requirements are those that omit to refer to specific instances, in some of their boilerplate placeholders. From an *external completeness* perspective, incompleteness occurs when there are requirements that refer to instances that have not yet been specified in any other requirement (missing requirements). The SHACL rules that implement this analysis are presented in the online Appendix C.1. An aggregate indicator of incompleteness is also computed:

$$\frac{ReqCount_{incomplete}}{ReqCount_{Total}} * 100$$

where $ReqCount_{incomplete}$ denotes the number of incomplete requirements and $ReqCount_{Total}$ is the total number of requirements. If the set of requirements is empty, then the reported incompleteness is 0 %.

B. (IN-)CONSISTENCY

A set of requirements is consistent when there are no conflicts between requirements, for every possible pair of them. Conflicts occur, when the same boilerplate placeholders in a pair have been assigned diverse values. The SHACL rules that implement this analysis are presented in the online Appendix C.2. A metric that summarizes the findings is also computed:

$$\frac{ReqCount_{inconsistent}}{ReqCount_{Total}} * 100$$

where $ReqCount_{inconsistent}$ denotes the number of all inconsistent requirements and $ReqCount_{Total}$ is the total number of requirements. If the set of requirements is empty, the reported inconsistency is 0 %.

C. AMBIGUITY

Ambiguity refers to requirements, which have boilerplate placeholders with values that can be replaced with instances of more concrete subclasses. The SHACL rules that implement this analysis are presented in the online Appendix C.3. A metric that quantifies existing ambiguity is computed as follows:

$$\frac{ReqCount_{ambiguous}}{ReqCount_{Total}} * 100$$

where $ReqCount_{ambiguous}$ denotes the number of ambiguous requirements and $ReqCount_{Total}$ is the total number of requirements. If the set of requirements is empty, then the reported ambiguity is 0 %.

D. NOISE

Noise refers to requirements that specify a boilerplate placeholder value, which is not an instance of a domain ontology class (i.e. a term that is not defined in existing domain ontologies). The SHACL rules that implement this analysis are presented in the online Appendix C.4.

A metric of existing noise is computed as follows:

$$\frac{ReqCount_{noisy}}{ReqCount_{Total}} * 100$$

with $ReqCount_{noisy}$ is the number of noisy requirements and $ReqCount_{Total}$ the total number of requirements. If the set of requirements is empty, then the reported noise is 0 %.

E. OPACITY

Opacity is concerned with requirements specifying terms that are irrelevant (i.e. there is no SAO dependency) to the requirement subject. The SHACL rules that implement this analysis are presented in the online Appendix C.5

A metric of existing opacity is computed:

$$\frac{ReqCount_{opaque}}{ReqCount_{Total}} * 100$$

where $ReqCount_{opaque}$ is the number of opaque requirements and $ReqCount_{Total}$ is the total number of requirements. If the set of requirements is empty, the reported opacity measurement is 0 %.

F. REDUNDANCY

Redundant specifications in a set of requirements occur, when for the same subject, two corresponding boilerplate placeholders are assigned semantically equivalent instances or the same quantifiers (e.g. numbers and number units). The SHACL rules for this analysis are presented in the online Appendix C.6. A metric of redundancy is computed as follows:

$$\frac{ReqCount_{redundant}}{ReqCount_{Total}} * 100$$

where $ReqCount_{redundant}$ denotes the number of redundant requirements and $ReqCount_{Total}$ is the total number of requirements. If the set of requirements is empty, then the reported redundancy is 0 %.

VI. TACIT KNOWLEDGE EXTRACTION

The discovery of abstract categories in a system domain (cf. Section IV-A) and the identification of their relationships are complex tasks, due to the need to apply techniques of generalization and inference. We introduce here a practical approach, called *relatedness analysis*, to aid the retrieval of latent semantic connections from knowledge that is defined either in the domain ontologies or in the requirements. The findings of this analysis are likely to uncover tacit knowledge, which should be taken into account, in order to improve the overall semantic coverage of the domain or the completeness of the requirements specification.

In essence, for any two terms we estimate the "closeness" of their semantic relationships using weights that depend on the type of their SAO properties. The semantic relatedness of any two terms is defined as follows:

$$Rel_{c_i,c_j} = \begin{cases} 0 & \text{if no SAO property relates } c_i \text{ with} \\ & c_j \text{ transitively} \\ W_{c_k/c_1} & \text{otherwise} \end{cases}$$

where W_{c_k/c_1} is the weight for term $c_k = c_i$ relative to $c_1 = c_j$ with k being the index of c_i in the minimal path from c_j. The weight is computed using the recursive definition:

$$W_{c_1/c_1} = 1$$
$$W_{c_2/c_1} = 1/a$$
$$W_{c_n/c_1} = (1/a)W_{c_{n-1}/c_1}$$

where α is assigned values that are prime numbers to ensure a unique product that quantifies the semantic relatedness of any two in a group of terms.

The highest weight ($\alpha = 2$) is assigned to all properties with meaning that relates a `Function` with `TraversingConcepts` (`send`/`receive`/`ingests`/`emits`), since they result in the inference of `Interfaces`, `Connections` and associated properties (cf. Section III-B4). The immediately lower weight ($\alpha = 3$) is assigned to properties with decomposition meaning (`isDecomposedTo`, `invokes`, `contains` or the inverse `isComposedOf`/`isInvokedBy`/`isContainedIn`), because any such information enriches our semantic knowledge for a system's design. The next lower weight ($\alpha = 5$) is assigned to the only remaining property that relates a `Function` with a `TraversingConcept` (`set` or the inverse `isSetBy`). The remaining weights are:

- $\alpha = 7$ if the property relating c_n and c_{n-1} is `perform`/`present` or their inverse (`isPerformedBy`/`isPresentedBy` respectively)
- $\alpha = 11$ if the property relating c_n with c_{n-1} is `transfer` or its inverse `isTransferedBy`
- $\alpha = 13$ if the property relating c_n with c_{n-1} is `joins`
- $\alpha = 17$ if the property relating c_n with c_{n-1} is `isAvailableIn`
- $\alpha = 19$ if the property relating c_n with c_{n-1} is `hasState` or its inverse `isStateOf`
- $\alpha = 23$ if the property relating c_n with c_{n-1} denotes any other relationship

We consider as less important all properties related with `StateConcept`, due to the need to precede the information extraction for all asymmetric producer/consumer relationships defined in SAO, from any state-related information.

Example 3: Fig. 17 depicts the graph of the SAO properties, for the following set of requirements:

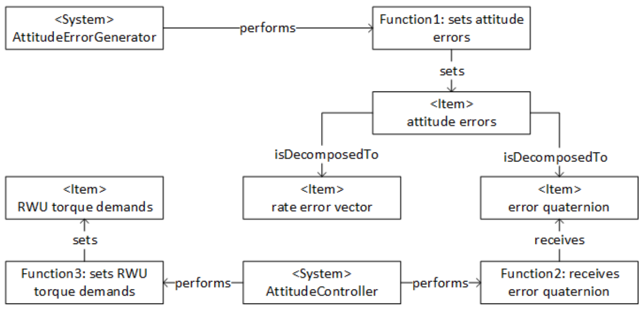

<System> AttitudeErrorGenerator
performs
Function1: sets attitude errors
sets
<Item> attitude errors
isDecomposedTo
isDecomposedTo
<Item> RWU torque demands
<Item> rate error vector
<Item> error quaternion
sets
receives
Function3: sets RWU torque demands
performs
<System> AttitudeController
performs
Function2: receives error quaternion

FIGURE 17. SAO representation of requirements in Example 3.

M1: `<System:Attitude Error Generator>` shall set `<Item:Attitude errors>`
M15: `<Item:Attitude errors>` shall be composed from `<Item:attitude error quaternion>` and `<Item:rate error vector>`
M4: `<System:Attitude Controller>` shall receive `<Item:attitude error quaternion>`
M1: `<System:Attitude Controller>` shall set `<Item:reaction wheel unit torque demands>`

The semantic relatedness values for all pairs of terms found in boilerplate placeholders are shown in Table 9.

Algorithm 1 introduces a knowledge extraction process to spot pairs of terms that an expert could potentially decide to relate through a missing SAO property. Discovery of such pairs takes place by traversing for each term the next closely related term through an existing property, until no other semantic assumptions can be accessed. This results in discovering tacit knowledge, which should either be included in a domain ontology or appended as additional requirement(s). The process depends on weighted semantic relatedness properties that can be adapted or used with other weights, if needed, in order to better suit a particular system context. Application of Algorithm 1 is illustrated in Example 4, for a set of requirements from the Eagle-Eye case study.

VII. THE EAGLE-EYE CASE STUDY

Eagle-Eye [7] is a virtual satellite for earth observation, which carries a payload (called "GoldenEye)" consisting of a high-resolution imaging camera. The Eagle Eye system is decomposed into the subsystems shown in Fig. 18: (i) Data

TABLE 9. Semantic relatedness of all terms in the requirements of Example 3.

RWU torque demands	0	0	0	0	0	0	0	$\frac{1}{5}$	1
Function3 (set)	0	0	0	0	0	0	$\frac{1}{7}$	1	$\frac{1}{5}$
Attitude Controller	0	0	0	$\frac{1}{7}*\frac{1}{2}$	0	$\frac{1}{7}$	1	$\frac{1}{7}$	$\frac{1}{7}*\frac{1}{5}$
Function2 (receive)	0	0	0	$\frac{1}{2}$	0	1	$\frac{1}{7}$	0	0
rate error vector	$\frac{1}{7}*\frac{1}{5}*\frac{1}{3}$	$\frac{1}{5}*\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}*\frac{1}{3}$	1	0	0	0	0
error quaternion	$\frac{1}{7}*\frac{1}{5}*\frac{1}{3}$	$\frac{1}{5}*\frac{1}{3}$	$\frac{1}{3}$	1	$\frac{1}{3}*\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{7}*\frac{1}{2}$	0	0
attitude errors	$\frac{1}{7}*\frac{1}{5}$	$\frac{1}{5}$	1	$\frac{1}{3}$	$\frac{1}{3}$	0	0	0	0
Function1 (set)	$\frac{1}{7}$	1	$\frac{1}{5}$	$\frac{1}{5}*\frac{1}{3}$	$\frac{1}{5}*\frac{1}{3}$	0	0	0	0
Attitude Error Generator	1	$\frac{1}{7}$	$\frac{1}{7}*\frac{1}{5}$	$\frac{1}{7}*\frac{1}{5}*\frac{1}{3}$	$\frac{1}{7}*\frac{1}{5}*\frac{1}{3}$	0	0	0	0
	Attitude Error Generator	Function1 (set)	attitude errors	error quaternion	rate error vector	Function2 (receive)	Attitude Controller	Function3 (set)	RWU torque demands

Algorithm 1 Discovery of Latent Semantic Relations

```
Input: Semantic relatedness ($sr$) matrix $M$
Output: Potentially missing SAO property relating two existing terms
1: $SR_h \leftarrow$ highest $sr$ of matrix, found in $M[c_i, c_j]$, such that $sr \neq 1$
2: $SR \leftarrow SR_h$
3: $c' \leftarrow c_j$
4: $k \leftarrow$ highest $sr$ in $c'$, with $sr \leq SR_h$, found in $M[c', c_j]$
5: while (($SR_h \neq$ minimum $sr$ in row $c'$) and (SAO property exists that relates $c'$ with $c_j$)) do
6:     $c' \leftarrow c_j$
7:     $SR_h \leftarrow$ highest $sr$ in row $c'$ found in $M[c', c_j]$, not previously selected, with $sr \leq SR_h$
8: end while
9: $P_{FOUND} \leftarrow false$
10: if $SR_h \neq 0$ then
11:     Potentially missing property (if not contradicting with existing property) for any SAO valid combination of $c_i$, $c_j$ of Step
        1 and $c'$, $c_j$ of Step 8.
12:     $P_{FOUND} \leftarrow$ choose $\begin{cases} true & \text{if property is approved by domain expert} \\ false & \text{otherwise} \end{cases}$
13:     if ($P_{FOUND} = true$) then
14:         return true
15:     end if
16: end if
17: if (($SR_h = 0$) or ($P_{FOUND} = false$)) then
18:     $SR_h \leftarrow$ highest $sr \leq k$ in row $c'$ of Step 4, not previously selected
19:     if ($SR_h \neq 0$) then
20:         goto 5
21:     else
22:         $SR_h \leftarrow$ highest $sr$ of matrix, found in $M[c_i, c_j]$, such that $sr \leq SR$
23:         if ($SR_h = 0$) then
24:             return false
25:         else
26:             goto 2
27:         end if
28:     end if
29: end if
```

Management System (DMS), (ii) RF communication, (iii) thermal subsystem, (iv) power subsystem, (v) AOCS, (vi) sensors/actuators and (vii) payload.

The DMS controls and processes commands and data originated from the equipment, the ground and the Golden Eye payload. DMS also handles all the data flow, data storage

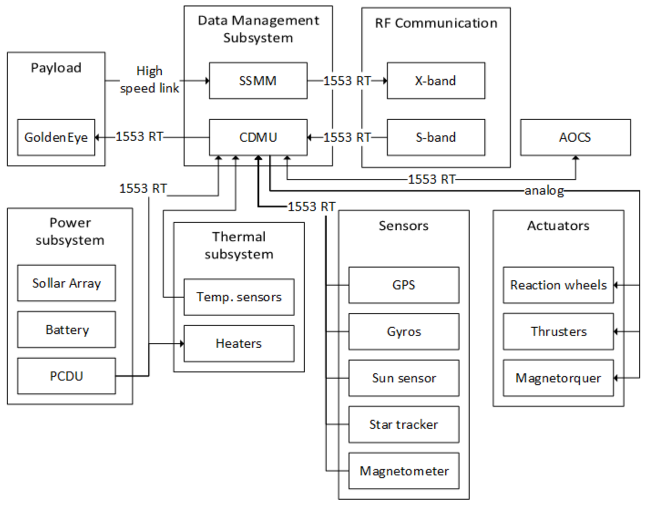


FIGURE 18. EagleEye virtual spacecraft architecture.

and data encryption/decryption between the subsystems. The RF Communication receives commands from the ground and transmits the satellite status (S-Band) and the payload data (X-Band) to the ground. The Power Control and Distribution Unit (PCDU) of the power subsystem controls the onboard power generated by the solar array cells. For energy storage, Eagle Eye uses batteries. The AOCS controls the attitude and position of the satellite, and interacts with: (i) sensors (Global Positioning System - GPS, Gyroscopes, Sun sensors, Star tracker and Magnetometer), and (ii) actuators (Reaction wheels, Thrusters and Magnetorquer).

The thermal subsystem is passive during nominal operations. Heaters are only activated in case of contingencies. The GoldenEye imaging unit transfers science data to the Solid State Mass Memory (SSMM), where they are buffered. The SSMM transfers the data to the X-band unit upon a command received from the Central Data Management Unit (CDMU). All on-board I/O for EagleEye are conducted across a single MIL-STD-1553 bus.

Eagle-Eye has been developed in a number of ESA studies as a workbench for evaluating new tools and techniques. Our case study includes a list of 65 natural language requirements of the following types (Table 10):

- functional requirements (F)
- performance requirements (P)
- interface requirements (I)
- operational requirements (O)
- design and implementation requirements (D)

To apply our boilerplate syntax we had to decompose the natural language requirements into 234 requirements. For example, the design requirement shown in Table 11 had to be decomposed into six distinct boilerplates.

It is important to emphasize the role of the AOCS DSO, described in Section IV-A, which facilitated the specification and the semantic representation of the AOCS-related requirements. The non-AOCS requirements were not the same easy to decompose and represent them using appropriate

TABLE 10. Eagle-Eye system requirements per subsystem.

System/subsystem	Natural language requirements	Boilerplate requirements
EagleEye	3	8
RF Communcation	6	30
Power	3	8
AOCS	20	97
Sensors	2	15
Actuators	1	7
Thermal	2	4
Payload	5	9
Data Management	21	47
Failure Detection Isolation Recovery (FDIR)	2	9

ontology concepts. Since the relevant expertise did not exist in the respective domains, the DSOs had to be updated, while expressing the requirements using the boilerplate placeholders. Still, at the end it was not possible to specify all relationships between the additional concepts included in the DSOs.

The semantic analysis of Section V yielded the results shown in Table 12. No conflicts were found, which is explained by the fact that these requirements are specifications at a relatively high level of abstraction. As expected, no internal incompleteness was found, since we had to update the DSOs with additional concepts, relevant to the values assigned in the boilerplate placeholders. On the other hand, the external incompleteness is relatively high, due to several concepts that are referred only once, while our DSOs cover sufficiently only the AOCS. Noise analysis is also adversely affected, for the same reason. Ambiguity is relatively low, but it reflects only the AOCS requirements, which include concepts that can be replaced by subclasses of them. Finally, the opacity score demonstrates the partial coverage of domain-specific knowledge (i.e. for the AOCS), in relation with the knowledge needed to cover all satellite subsystems.

The abstraction level of requirements has the following impact. If the requirements are at a relatively high level of abstraction, i.e. they are not detailed to the lowest level of subsystem decomposition, as is the case of our AOCS requirements, external incompleteness may seem to be high, without necessarily implying a real case of missing information. In this case, we expect that the requirements may be refined in subsequent iterations of allocating them to subsystem components.

Finally, it is worth to highlight the important role of the ambiguity analysis in guiding the refinement of requirements. Table 13 shows a requirement that refers to Eagle-eye's AOCS system. In this case, `<Item:absolute pointing error>` and `<Item:absolute measurement error>` were already defined within the `Error Generation` and `Controller` subsystems of the AOCS domain ontology (Fig. 9). Therefore, the ambiguity analysis reported that `<System:AOCS>` can be replaced by the respective subsystems, thus showing how these requirements can be refined.

The semantic relatedness analysis of Section VI was also applied, in an attempt to discover additional knowledge, in particular for the system domains in which we had only

TABLE 11. Eagle Eye subsystems requirement.

Requirement	Type	Boilerplate requirements
The Eagle Eye shall provide the following subsystems: - Communication - Power - AOCS - Thermal - DMS - Golden Eye payload	D	M16: `<System:Eagle Eye>` shall contain `<System:Communication>` M16: `<System:Eagle Eye>` shall contain `<System:Power>` M16: `<System:Eagle Eye>` shall contain `<System:AOCS>` M16: `<System:Eagle Eye>` shall contain `<System:Thermal>` M16: `<System:Eagle Eye>` shall contain `<System:DMS>` M16: `<System:Eagle Eye>` shall contain `<System:Golden Eye payload>`

TABLE 12. Eagle-Eye semantic analysis results.

Analysis	Percentage	Ideal
Incompleteness (internal)	0%	0%
Incompleteness (external)	34%	0%
Inconsistencies	0%	0%
Ambiguity	8%	0%
Noise	80%	0%
Opacity	3%	0%
Redundancy	0%	0%

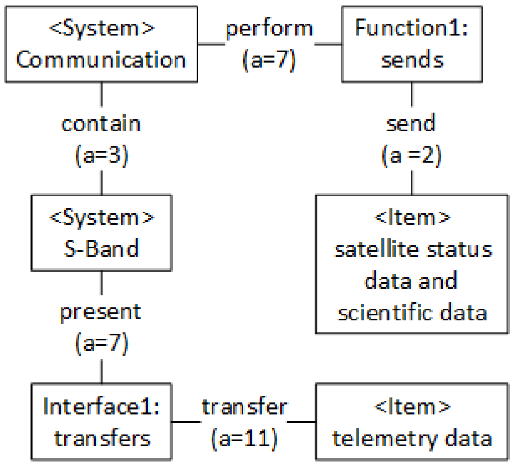


FIGURE 19. SAO representation of requirements in Example 4.

limited expertise. Example 4 refers to a small subset of requirements for satellite communication, which is not supported by a DSO and points out a common problem in requirements specification: functionality defined at different levels of abstraction.

Example 4: We consider the following set of requirements and the graph in Fig. 19, which shows the SAO property assignments for the terms in the placeholders, with their corresponding weights:

(EE-MR-0160_7) M3: `<System:Communication>` shall send `<Item:satellite status data and scientific data>`

(EE-MR-0170_1) M16:`<System:Communication>` shall contain `<System:S- band>`

(EE-MR-0170_3) M10:`<System:S-band>` shall transfer `<Item: telemetry data>`

The semantic relatedness of the mentioned terms is shown in Table 14.

The analysis of Algorithm 1 starts from the highest semantic relatedness (step 1), which is given in cell `[satellite status data and scientific data, Function1]`. In the row for `Function1`, the highest semantic relatedness that is selected next (step 4) is found in the `Communication` column. Finally, in the `Communication` row, the semantic relatedness value corresponding to `satellite status data and scientific data` cannot be selected and the next highest value that is selected is found in column `Interface1`. Table 15 shows the selected semantic relatedness values.

It is evident that no SAO property exists that relates `Communication` with `Interface1`, therefore the while loop in step 5 exits and the algorithm yields the following candidate properties:

a. `Communication performs Function1`
b. `Communication presents Interface1`
c. `Interface1 transfers satellite status data and scientific data`

The pair `Communication` and `satellite status data and scientific data` is excluded from the list, since no SAO property can relate a `System` with an `Item`. The first candidate property already exists, whereas the second property is not valid, since the `Communication` system cannot present `Interface1`, which is an interface of its `S-Band` subsystem (a system cannot present the same interface with its subsystem). The last candidate property, according to which `Interface1 transfers` the `satellite status data and scientific data`, can be a valid missing property if the `telemetry data` contains or is contained in `satellite status data and scientific data`.

If none of the candidate properties is valid, based on the domain expert knowledge, then according to step 17 of Algorithm 1, the next lower semantic relatedness value is selected, in order to retrieve additional candidate properties.

As shown in Table 16, this value is given in `[Function1,S-Band]`, the analysis diverts to step 5 and yields (without entering in the while loop) the following candidate property:

`S-Band performs Function1`

This property denotes that the functionality specified in EE-MR-0160_7 could be performed at a lower system level, such that the transmission of telemetry data is done by the `Communication` system through the `S-Band`.

TABLE 13. **Eagle-Eye AOCS performance requirement.**

Requirement	Type	Decomposition
The AOCS subsystem shall provide performances such as: - Absolute measurement error in the range of 100 μrad - Absolute pointing error in the range of 1 mrad	P	M3: `<System:AOCS>` shall send `<numerical-affirmative:`less than `<numerical:<number:100>` `<number_unit:`μ`rad>>>` `<Item:absolute measurement error>` M3: `<System:AOCS>` shall send `<numerical-affirmative:`less than `<numerical:<number:1>` `<number_unit:mrad>>>` `<Item:absolute pointing error>`

TABLE 14. **Semantic relatedness of all terms in the requirements of Example 4.**

Communication	1	$\frac{1}{7}$	$\frac{1}{7}*\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{3}*\frac{1}{7}$	$\frac{1}{3}*\frac{1}{7}*\frac{1}{11}$
Function1 (send)	$\frac{1}{7}$	1	$\frac{1}{2}$	$\frac{1}{7}*\frac{1}{3}$	$\frac{1}{7}*\frac{1}{3}*\frac{1}{7}$	$\frac{1}{7}*\frac{1}{3}*\frac{1}{7}*\frac{1}{11}$
satellite status data and scientific data	$\frac{1}{2}*\frac{1}{7}$	$\frac{1}{2}$	1	$\frac{1}{2}*\frac{1}{7}*\frac{1}{3}$	$\frac{1}{2}*\frac{1}{7}*\frac{1}{3}*\frac{1}{7}$	$\frac{1}{2}*\frac{1}{7}*\frac{1}{3}*\frac{1}{7}*\frac{1}{11}$
S-Band	$\frac{1}{3}$	$\frac{1}{3}*\frac{1}{7}$	$\frac{1}{3}*\frac{1}{7}*\frac{1}{2}$	1	$\frac{1}{7}$	$\frac{1}{7}*\frac{1}{11}$
Interface1 (transfer)	$\frac{1}{7}*\frac{1}{3}$	$\frac{1}{7}*\frac{1}{3}*\frac{1}{7}$	$\frac{1}{7}*\frac{1}{3}*\frac{1}{7}*\frac{1}{2}$	$\frac{1}{7}$	1	$\frac{1}{11}$
telemetry data	$\frac{1}{11}*\frac{1}{7}*\frac{1}{3}$	$\frac{1}{11}*\frac{1}{7}*\frac{1}{3}*\frac{1}{7}$	$\frac{1}{11}*\frac{1}{7}*\frac{1}{3}*\frac{1}{7}*\frac{1}{2}$	$\frac{1}{11}*\frac{1}{7}$	$\frac{1}{11}$	1
	Communication	Function1 (send)	satellite status data and scientific data	S-Band	Interface1 (transfer)	telemetry data

TABLE 15. **Discovery of latent semantic relations for Example 4 (1).**

Row	Column	Property	SR_h
satellite status data and scientific data	Function1	send	$\frac{1}{2}$
Function1	Communication	perform	$\frac{1}{7}$
Communication	Interface1	-	$\frac{1}{3}*\frac{1}{7}$

TABLE 16. **Discovery of latent semantic relations for Example 4 (2).**

Row	Column	Property	SR_h
satellite status data and scientific data	Function1	send	$\frac{1}{2}$
Function1	S-Band	-	$\frac{1}{7}*\frac{1}{3}$

VIII. EXPERIMENTAL EVALUATION

A. WEB-BASED TOOL FOR REQUIREMENTS SPECIFICATION AND ANALYSIS

A web-based tool was developed to automate the specification, the semantic analysis and editing of system requirements based on the methodology proposed and the underlying ontologies. No ontology engineering skills are required for the tool user; it is not even necessary to have a basic understanding of the ontology technology.

The tool provides essential functionality for adding, editing and deleting boilerplate requirements based on the overall ontology architecture and the existing DSOs. The user does not need to be knowledgeable for the available templates in the boilerplate language (cf. Section IV-B1). Instead, the right boilerplate template is automatically chosen based on the placeholder terms that the user types or selects. If there is no matching boilerplate, a warning is raised. A browsing window is also provided, while editing a requirement, for exploring the available classes in the existing DSOs. The user can create new instances for any of the browsed classes, in order to use them in requirement specifications.

Once the user has specified a set of requirements, it is possible to run the semantic analysis that flags those requirements, which are found to contribute to any of the metrics for the semantic issues that are checked (cf. Section V). The semantic analysis runs through invoking the Apache Jena API.[4] According to the feedback provided (e.g. an inconsistency case or incompleteness), the user can edit a requirement or create additional requirement(s) towards improving the quality of the requirements specified.

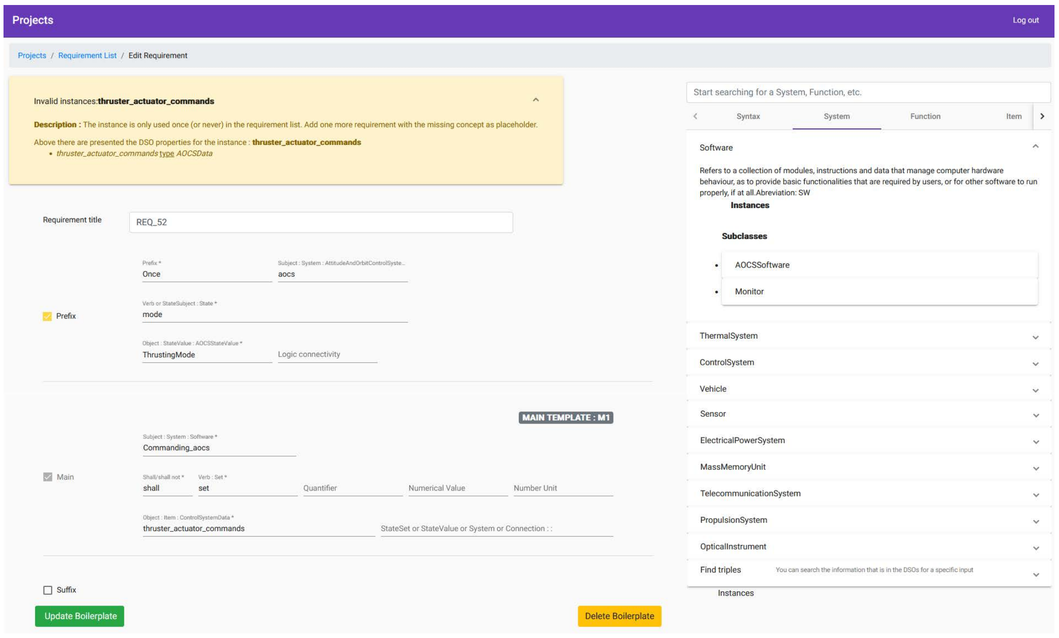
The tool interface consists of two windows:

- The requirement entry/editing window (Fig. 20), through which it is possible to add, edit or delete a boilerplate-based requirement. Editing takes place by selecting terms (instances) from the DSOs shown in the right-hand side of the window; it is also possible to use instances of the SAO classes, if no relevant DSO class is found.
- The semantic analysis window (Fig. 21) that shows all requirements specified in a project, along with the results of the last semantic analysis run.

B. EXPERIMENT DESCRIPTION

The experiment aimed to retrieve empirical evidence on the effectiveness and efficiency of the proposed method into a real industrial context. The experiment took the form of an *observation case study*, whose design is described here according to the relevant template by Runeson *et al.* [89].

[4]Appache Jena: https://github.com/TopQuadrant/shacl.



Invalid instances:**thruster_actuator_commands**

Description : The instance is only used once (or never) in the requirement list. Add one more requirement with the missing concept as placeholder.

Above there are presented the DSO properties for the instance : **thruster_actuator_commands**
- *thruster_actuator_commands type AOCSData*

Requirement title	REQ_52

[✓ Prefix]

Prefix *	Once
Subject : System : AttitudeAndOrbitControlSyste...	aocs
Verb or StateSubject : State *	mode
Object : StateValue : AOCSStateValue *	ThrustingMode
Logic connectivity	

MAIN TEMPLATE : M1

[✓ Main]

Subject : System : Software *	Commanding_aocs
Shall/shall not *	shall
Verb : Set *	set
Quantifier	
Numerical Value	
Number Unit	
Object : Item : ControlSystemData *	thruster_actuator_commands
StateSet or StateValue or System or Connection : :	

☐ Suffix

Software

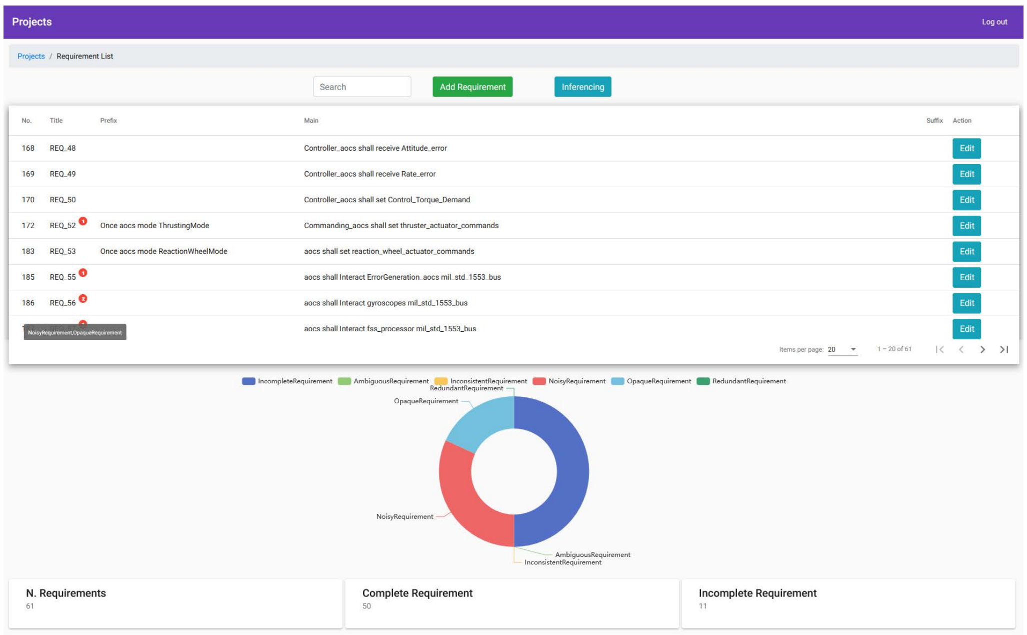
Refers to a collection of modules, instructions and data that manage computer hardware behaviour, as to provide basic functionalities that are required by users, or for other software to run properly, if at all.Abreviation: SW

Instances

Subclasses
- AOCSSoftware
- Monitor

ThermalSystem
ControlSystem
Vehicle
Sensor
ElectricalPowerSystem
MassMemoryUnit
TelecommunicationSystem
PropulsionSystem
OpticalInstrument
Find triples You can search the information that is in the DSOs for a specific input
Instances

FIGURE 20. Requirement entry window of the web-based tool for requirements specification and analysis.



No.	Title	Prefix	Main	Suffix	Action
168	REQ_48		Controller_aocs shall receive Attitude_error		Edit
169	REQ_49		Controller_aocs shall receive Rate_error		Edit
170	REQ_50		Controller_aocs shall set Control_Torque_Demand		Edit
172	REQ_52 1	Once aocs mode ThrustingMode	Commanding_aocs shall set thruster_actuator_commands		Edit
183	REQ_53	Once aocs mode ReactionWheelMode	aocs shall set reaction_wheel_actuator_commands		Edit
185	REQ_55 1		aocs shall Interact ErrorGeneration_aocs mil_std_1553_bus		Edit
186	REQ_56 2		aocs shall Interact gyroscopes mil_std_1553_bus		Edit
[illegible]	[illegible]		aocs shall Interact fss_processor mil_std_1553_bus		Edit

NoisyRequirement,OpaqueRequirement

Items per page: 20 1 – 20 of 61

N. Requirements	Complete Requirement	Incomplete Requirement
61	50	11

FIGURE 21. Semantic analysis window of the web-based tool for requirements specification and analysis.

a: RESEARCH OBJECTIVES & RESEARCH QUESTIONS

According to the Goal - Question - Metric template [90], the goal is to ''*analyse* the methodology proposed *for the purpose of* assessing it *with respect to* its effectiveness and efficiency *from the point of view of* the requirement engineers, *in the context of* the requirements formulation and semantic analysis problem''.

To this end, three research questions (RQ) were posed:

RQ1. *Are there important differences in the time needed by the engineers, to complete a given requirements specification task using our methodology?*

To answer this question, the time (in hours) taken by the subjects to complete the requirements specification is recorded. This is an indication for the efficiency of the methodology.

RQ2. *How effective is the methodology for assuring that everyone has the same expectations, for the system under design?*

Based on a natural language description of the system under design, we assess the overall understanding of the methodology by focusing on the difficulty to identify, which boilerplates to use and which values to fill

TABLE 17. **Subjects know-how and experience.**

Engineer	Space domain expert	Requirement specification experience
E1	No	2 years
E2	No	< 1 year
E3	Yes	10 years
E4	Yes	35 years
E5	Yes	40 years

in their placeholders. Also, we evaluate the extent of semantic similarity between the requirement specifications delivered by the different subjects.

RQ3. *How effective is the methodology towards improving a requirements specification, i.e. to find cases of incompleteness, inconsistency etc. and revise the corresponding requirements?*
We explore the effectivenes of the semantic analysis in guiding the subjects to improve their requirements specification. First, we count the number of specification issues that are initially identified using the semantic analysis. Afterwards, we measure the degree of difficulty, for the subjects, to understand these issues (if any) and the degree of difficulty to remove them.

The research questions focus on the evaluation of the method itself, which depends on the web-based tool used by the subjects. No questions were posed for the tool evaluation, because this was left as future work, as the tool may be further improved.

b: CASE STUDY AND SUBJECTS SELECTION

To answer questions RQ1 - RQ3, the selected subjects were assigned the task of specifying requirements for the Orbit Control System that is described in online Appendix B. The subjects were prompted to use the AOCS DSO (cf. Section IV-A) and the web-based tool, in order to specify requirements for the Orbit Control System, to analyze them semantically and subsequently fix as many problems as possible.

The subjects were five engineers with diverse domain expertise and industrial experience in requirements specification (Table 17). The reason for having selected subjects with diverse domain knowledge was to assess the degree to which the DSO is used and the impact of this to the effectiveness metrics. The subjects did not know each other, so that they could not discuss, while working on the specification task.

c: DATA COLLECTION

Each subject studies a brief introduction (4 pages) to the SAO ontology and the boilerplate language, along with concrete examples on the use of specific boilerplates, for the data, functional and behavioral perspectives when specifying requirements (cf. Section III-B). Then, the case study description was given to them, as well as private access to the web-based tool. No additional help was provided to the subjects, while working on the case study.

After having completed their work, the questionnaire in Fig. 22 was given to the subjects, in a personal interview, to collect qualitative data on the research questions RQ1 - RQ3. The subjects were asked not only to answer the linkert scale questionnaire, but also to convincingly justify their answers. Moreover, the evaluator (designer of the SAO, the boilerplate language and the DSO) asked questions on the final specification outcome, in order to interpret the answers of the subjects. The interviews were not recorded, but the evaluator kept notes.

- Method efficiency:
  - Time in hours to finish assignments (initial specification & improved specification based on the semantic analysis results)
  - Estimate the overall understanding of the methodology (0 no understanding - 5 fully understood)
  - Estimate the difficulty to identify which boilerplates to use (0 not difficult - 5 very difficult)
  - Estimate the difficulty to identify the values for the boilerplate placeholders (0 not difficult - 5 very difficult)
- Method effectiveness:
  - Estimate the number of specification issues identified through semantic analysis (number of issues identified)
  - Estimate the difficulty to understand specification issues through semantic analysis (0 not difficult - 5 very difficult)
  - Estimate the difficulty to remove specification issues, if any, to improve the initial specification (number of eliminated specification issues)

FIGURE 22. **Questionnaire for research questions.**

d: DATA ANALYSIS

Given the data collected from the interviews, the evaluator performed simple quantitative and qualitative analyses. In the quantitative analysis, the linkert scale values from the questionnaires were compared. For the qualitative analysis, the evaluator studied the requirement specifications provided by the subjects, in order to measure the number of common boilerplate requirements, as well as to find errors in the specifications, due to a possible misuse of the DSO. Common boilerplate requirements are considered those using the same boilerplate templates, with all their placeholders filled in with instances of the same ontology classes.

C. EXPERIMENTAL RESULTS

Table 18 reports the quantitative results on the metrics associated with the answers to the questions of Fig. 22.

All subjects found the methodology easy to understand, well documented and not particularly difficult to apply. Since the boilerplate language is based on the SAO concepts and relationships, we expected that this would affect its ease of use. Actually, it was found that once the subjects became

TABLE 18. Quantitative experimental results.

Question	E1	E2	E3	E4	E5
Time for initial specification (in hours)	4:00	7:30	2:45	2:30:	2:00
Time for improved specification (in hours)	1:00	0:30	0:10	0:10	0:10
Overall understanding	4	5	5	4	4
Boilerplate identification difficulty	1	1	1	1	1
Placeholder identification difficulty	2	2	1	1	1
Number of issues identified:					
- Incompleteness	12/64	7/70	5/80	5/75	4/73
- Redundancy	16/64	4/70	0/80	0/75	0/73
- Noise	0/64	0/70	0/80	0/75	0/73
- Opacity	3/64	8/70	0/80	0/75	0/73
- Ambiguity	0/64	0/70	0/80	0/75	0/73
Difficulty to understand specification issues	2	3	2	2	2
Number of eliminated specification issues	17/31	10/20	5/5	5/5	4/4

familiarized with the concept relationships in SAO, it came more natural to them to express requirements using the language, without consulting the grammar notes provided. In general, the subjects found "boilerplates to be self-explanatory" and several times during their interviews characterized the language as highly "expressive". None of the subjects mentioned any difficulty in identifying the placeholder values to use in the boilerplates, but during data analysis, it was found that the two subjects with limited or no domain expertise utilized less instances from the AOCS DSO compared to the number of instances used by the domain experts.

The semantic analysis results showed some opacity issues in the specifications performed by the subjects with limited or no domain expertise. These issues were attributed to incorrect use of the instances in the AOCS DSO, which was confirmed during the interview. Concretely, it was noted that "the most difficult part, while specifying the boilerplate requirements, was to select the right placeholder values from the AOCS DSO" and that they "would like to have more explanations for the terms found in the domain", which is also related to the lack of domain knowledge, for these subjects. Also, it was found that due to the existence of multiple classes and subclasses in the AOCS DSO ontology and the abundance of instances, the task to find the right instances was very time-consuming and they did not manage to avoid making mistakes.

For the subjects that are domain experts, no issues were found in selecting the right instances of the AOCS DSO. During their interview, these subjects commented that the domain ontology "is highly extensive and extensible".

Regarding the other semantic analysis results, all subjects stated that "incomplete and redundant requirements were fully understood" and they had no difficulty to eliminate these issues. On the other hand, for the noisy and opaque requirements, the subjects would like to have more explanations on how to handle them.

Table 19 reports the results of the qualitative analysis conducted in the requirement specifications by the different subjects. Similarity, i.e. the percentage of common boilerplate requirements, lies from 60% to 65,63% among the non experts in the domain, from 83,75% to 94,52% among the experts, and from 29,33% to 39,1% when comparing the non experts with the domain experts. Specification problems stemming from erroneous use of the DSO were found only in the requirements written by the non experts.

The conclusions drawn from the quantitative and qualitative analysis regarding the research questions posed are summarized as follows:

RQ1. There are noteworthy differences in the time efficiency to complete the given requirements specification task, between the subjects with no domain expertise and the domain experts.

RQ2. All subjects reported that they had almost no difficulty to understand the methodology from the point of view of identifying the right boilerplate, but the subjects with no domain expertise had some more difficulties to find the right values for the boilerplate placeholders. High similarity was observed between the requirement specifications of the domain experts, which means that, for them, the methodology assures the same expectations for the system under design.

RQ3. All issues identified in the specifications of the domain experts have been eventually eliminated, whereas the subjects with limited or no experience managed to eliminate about 55% of the issues detected. A significant number of the remained issues was due to incorrect use of the DSO.

Some enlightening feedback was also provided by the subjects, which is worth to mention:

- In the description of the Orbit Control System that was given to the subjects, it was not clearly stated whether the MIL-STD-1553 bus protocol is used for both external and internal system communication. The domain experts, based on the AOCS DSO, specified the MIL-STD-1553 bus connection for external communication only, which is correct. On the other hand, the subjects with limited expertise in the domain, misunderstood the AOCS DSO and used the MIL-STD-1553

TABLE 19. Qualitative experimental results.

Common boilerplates	E1	E2	E3	E4	E5	Specification errors
E1	-	65,63%	46,88%	39,1%	39,1%	18,75%
E2	60%	-	38,57%	31,43%	31,43%	14,29%
E3	37,5%	33,75%	-	83,75%	83,75%	0%
E4	33,33%	29,33%	89,33%	-	92%	0%
E5	34,25%	30,14%	91,78%	94,52%	-	0%

bus connection in the interconnect with the orbit control subsystems, which is not correct.
This clearly shows how the DSO can affect the end-result, if there are ambiguities in the description of the system under design.

- The domain experts pointed out the need to interfere with the design of the DSO and allow extending it according to their project-specific needs, which is not currently supported by the web-based tool. This need stemmed from the fact that the AOCS DSO differentiates control systems from the software, whereas for the domain experts the control systems would not be expected as decoupled from the software. Moreover, it was also mentioned that the current AOCS DSO does not cover domain knowledge that is mission-specific (e.g. differentiate between Lagrangian or Geostationary, where in the former, no GPS sensors can be used in attitude determination).
- The domain experts emphasized the need to allow for differentiating the naming of terms in the DSO, when specifying requirements, due to possible discrepancies in terms, between the different companies. This can be addressed by the ontology architecture proposed, since it supports the definition of synonyms (through the LO ontology) for the terms given as instances of the classes in the DSO.

D. THREATS TO VALIDITY

The potential threats to validity of the experiment's results are distinguished in threats against the internal and external validity. Internal validity refers to possible influences that may alter the experiment's outcome, whereas external validity concerns with the degree to which the results can be generalized.

While the scope of the conducted experiment is the method's efficiency and effectiveness, the performance of the subjects inevitably depends on the available tool for applying the proposed methodology. In particular, even though the subjects could easily access the semantic analysis results, including feedback for each specified requirement, a potential internal validity threat is whether the subjects cannot fully comprehend and eliminate the reported issues in their requirement specifications. This threat concerns with a potentially inadequate design of our web-based tool, which could affect the internal validity of the experimental results. However, from the results presented in Section VIII-C,, we note that the subjects were eventually able to select the right boilerplates, whereas the cause for any specification issue that was not eliminated (i.e. when the subject did not manage to identify the right values for boilerplate placeholders) was related exclusively to the lack of domain expertise. Consequently, even if the semantic analysis results would be presented in a different way, this could not affect the experimental results for the method's efficiency and effectiveness.

On the side of external validity, a potential threat is the dependence of the experimental results on how the domain knowledge is presented to the prospective users of our methodology. In practice, for the DSO ontologies to be effectively utilized, users with limited or no domain expertise need a thorough presentation and an adequate description of their classes and instances. The degree to which the current interface of the web-based tool affects the external validity of the experimental results needs to be further examined, and this will happen when being able to consider and compare the present implementation with alternative user interface designs.

IX. LESSONS LEARNED

Among the lessons learned from the case study of Section VII and the experiment in Section VIII, we noted that the set of requirements affects some analysis results more than some other. Incompleteness and inconsistency analyses, in principle, depend on the requirement specifications, for which they check the boilerplate placeholders for missing references (e.g. producer/consumer relationships, states etc.) and conflicts (e.g. contradicting functions and placeholder values). The rest of the semantic analyses mainly depend on the availability of concepts and their relationships in domain specific ontologies, i.e. on the degree to which these ontologies have been developed.

Furthermore, the requirement specification approach and boilerplate language did not present any difficulties for the engineers to understand and use them, irrespective of whether they are domain experts or not. It is also noteworthy that the size of a specification task matters in what is concerned with the detail reflected in the DSO. Thus, for a small specification task, a large DSO may not be easily utilized, whereas for a relatively large specification task, the DSO may be limited in scope and need to be extended.

The domain ontology was characterized as highly extensive. The engineers without experience in the particular domain still can perform their requirement specification task, despite that they will likely not use the domain ontology as effectively as the domain experts. This affects the time needed to specify requirements, as well as the degree to which DSO instances are used.

During the experiment analysis we have recorded the engineers expectations: (i) to provide an overview of what exactly is supported in the domain and (ii) to provide the means to enhance the existing domain knowledge on a per project use. For the first requirement, the current version of the web-based tool provides only a limited view of the domain ontology contents, i.e. it shows only the existing classes and instances, while their relations are limited only to the SAO relationships. For the second requirement, the domain ontology architecture is open to extensions, but we currently assume that DSO extensions will not be allowed for the engineer(s) involved in the requirements specification, therefore this functionality is not supported by the tool. Possible extensions to the DSO are currently implemented through external ontology editing tools (e.g. TopBraid Composer), a task that is usually entrusted to an ontology engineer, who knows how to properly extend the domain knowledge.

Last but not least, our experience on extending the AOCS DSO has shown the need to perform alternating cycles of requirements specification and DSO updates, a practice that contributes towards building a highly expressive and beneficial domain ontology.

X. CONCLUDING REMARKS

We presented an ontology-driven requirements formulation and semantic analysis approach, for system requirements. This work complements a previous work in [1] that focuses on the requirements formalization and their model-based validation. To address the lack of a universal interpretation of the natural language syntax, we employ requirement boilerplates with ontology-based semantics. This also serves as a means to generate semantic relationships between the values in the boilerplate placeholders, as well as to derive all implicit assumptions by utilizing the asymmetric producer/consumer relationships defined in an upper ontology.

The overall semantic modeling framework makes it easy to develop and integrate domain specific ontologies, which provide an essential aid for specifying requirements. We showed how to accomplish this goal and then we presented a series of analyses that allow detecting inconsistencies, missing information, ambiguity and other semantic omissions. Last but not least, a semantic relatedness analysis was introduced that facilitates the extraction of tacit knowledge from a set of requirements, in order to improve the semantic coverage of the system domain. This particular contribution leads to explicitly represent any implicit knowledge that will have to be taken into account.

The expressiveness of the requirements specification approach and the scope of the semantic analyses were evaluated based on a set of system requirements of diverse types, for a virtual earth-observation reference satellite by ESA, called Eagle Eye. A user experiment was also conducted, to assess the efficiency and the effectiveness of the proposed solution. The subjects of this experiment were requirement engineers with varied degrees of experience in the space industry. The results confirmed that the additional cost for applying the approach is affordable and justified for the kind of systems they are working on (critical embedded systems), but provided also valuable feedback towards introducing it into the industrial practice.

The system of ontologies of the overall framework is available online, along with the detailed formalization of the ontology relationships and rules that are provided in an online appendix of the present article. Finally, our web-based tool makes the approach accessible to engineers, who do not have any ontology engineering experience.

The main ingredients of the overall approach, i.e. the boilerplate syntax (Section IV-B1), the semantic analysis (Section V) and the tacit knowledge extraction process (Section VI), are founded on the SAO upper ontology (Section III-B). As a future prospect, we are interested to assess the suitability of this framework for other industrial contexts, beyond space systems. We expect that the SAO upper ontology and the expressiveness of requirement boilerplates, will have to be extended in order to cover an expanding range of extra-functional requirements.

A tempting prospect is also the validation of the ontology with respect to the overall coverage of the domain knowledge. There are several ontology analysis tools, which allow to evaluate qualitatively or quantitatively an ontology (e.g. average or maximum depth of class inheritance tree or class ancestors etc.) and they can help to identify potentially problematic parts. However, none of the existing evaluation methods, neither alone nor in combination, can guarantee a ‘‘sound’’ ontology [91].

Finally, a related direction of research is the derivation of formal properties from semantically validated requirements and their verification on a formal model of system design. In [1], we proposed a correctness-by-construction design approach to limit the need for a posteriori model checking, which does not scale to solve realistic problems. However, further work is required to incorporate fault handling and diagnosability analysis (e.g. failure mode and effects analysis, fault tree generation, failure recovery analysis etc).

REFERENCES

[1] E. Stachtiari, A. Mavridou, P. Katsaros, S. Bliudze, and J. Sifakis, ‘‘Early validation of system requirements and design through correctness-by-construction,’’ *J. Syst. Softw.*, vol. 145, pp. 52–78, Nov. 2018.

[2] K. Pohl and C. Rupp, *Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam-Foundation Level-IREB Compliant*, 1st ed. San Rafael, CA, USA: Rocky Nook, 2011.

[3] P. Feiler, J. Delange, and L. Wrage, ‘‘A requirement specification language for AADL,’’ Softw. Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2016-TR-008, 2016. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=464370

[4] N. Mahmud, C. Seceleanu, and O. Ljungkrantz, ‘‘Specification and semantic analysis of embedded systems requirements: From description logic to temporal logic,’’ in *Proc. Int. Conf. Softw. Eng. Formal Methods*, Trento, Italy, 2017, pp. 332–348.

[5] N. Mahmud, C. Seceleanu, and O. Ljungkrantz, ‘‘ReSA: An ontology-based requirement specification language tailored to automotive systems,’’ in *Proc. 10th IEEE Int. Symp. Ind. Embedded Syst. (SIES)*, Jun. 2015, pp. 1–10.

[6] M. Oertel and B. Josko, *CESAR-Cost-Efficient Methods and Processes for Safety-Relevant Embedded Systems*. Berlin, Germany: Springer, 2013.

[7] M. Schön and G. Caspersen, "EagleEye virtual spacecraft system architecture," ESA Model. Simul. Sect., Tech. Rep. TOS-EMS-VSRF-TN-0002, Mar. 2004.

[8] S. Anwer and N. Ikram, "A process for goal oriented requirement engineering," in *Proc. IASTED Int. Conf. Softw. Eng.* Calgary, AB, Canada: ACTA Press, 2008, pp. 255–261.

[9] I. Cardei, M. Fonoage, and R. Shankar, "Model based requirements specification and validation for component architectures," in *Proc. 2nd Annu. IEEE Syst. Conf.*, Apr. 2008, pp. 1–8.

[10] I. Castillo, F. Losavio, A. Matteo, and J. Bøegh, "Requirements, aspects and software quality: The REASQ model," *J. Object Technol.*, vol. 9, no. 4, pp. 69–91, 2010.

[11] J. Chicaiza, J. López-Vargas, N. Piedra, O. Bonastre, and E. Caro, "Usage of social and semantic web technologies to design a searching architecture for software requirement artefacts," *IET Softw.*, vol. 4, pp. 407–417, Dec. 2010.

[12] D. V. Dzung and A. Ohnishi, "A verification method of elicited software requirements using requirements ontology," in *Proc. 19th Asia–Pacific Softw. Eng. Conf.*, vol. 1, Dec. 2012, pp. 553–558.

[13] D. V. Dzung and A. Ohnishi, "Improvement of quality of software requirements with requirements ontology," in *Proc. 9th Int. Conf. Quality Softw.*, Aug. 2009, pp. 284–289.

[14] S. Farfeleder, T. Moser, A. Krall, T. Stålhane, I. Omoronyia, and H. Zojer, "Ontology-driven guidance for requirements elicitation," in *The Semanic Web: Research and Applications* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2011, pp. 212–226.

[15] S. Ghaisas and N. Ajmeri, "Knowledge-assisted ontology-based requirements evolution," in *Managing Requirements Knowledge*. Berlin, Germany: Springer, 2013, pp. 143–167.

[16] R. Guizzardi, X. Franch, and G. Guizzardi, "Applying a foundational ontology to analyze means-end links in the *i** framework," in *Proc. 6th Int. Conf. Res. Challenges Inf. Sci. (RCIS)*, 2012, pp. 1–11.

[17] M. Kitamura, R. Hasegawa, H. Kaiya, and M. Saeki, "A supporting tool for requirements elicitation using a domain ontology," in *Software and Data Technologies* (Communications in Computer and Information Science). Berlin, Germany: Springer, 2009, pp. 128–140.

[18] M. Kossmann and M. Odeh, "Ontology-driven requirements engineering—A case study of ontorem in the aerospace context," in *Proc. INCOSE Int. Symp.*, 2010, vol. 20, no. 1, pp. 1000–1012.

[19] J. Lasheras, R. Valencia-García, J. Fernández-breis, and J. Álvarez, "Modelling reusable security requirements based on an ontology framework," *J. Res. Pract. Inf. Technol.*, vol. 41, no. 2, pp. 119–133, 2009.

[20] T. H. Nguyen, B. Q. Vo, M. Lumpe, and J. Grundy, "KBRE: A framework for knowledge-based requirements engineering," *Softw. Quality J.*, vol. 22, no. 1, pp. 87–119, 2014.

[21] E. Yu, L. Liu, and J. Mylopoulos, "A social ontology for integrating security and software engineering," in *Social and Human Elements of Information Security: Emerging Trends and Countermeasures*. Hershey, PA, USA: IGI Global, Jan. 2008, pp. 148–177.

[22] Y. Zhang and W. Zhang, "Description logic representation for requirement specification," in *Computational Science—(ICCS)* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2007, pp. 1147–1154.

[23] Z.-Y. Li, W. Zhi, X. Zhang, and X. Yong, "The domain ontology and domain rules based requirements model checking," *Int. J. Softw. Eng. Appl.*, vol. 1, no. 1, pp. 89–100, 2007.

[24] N. Assawamekin, T. Sunetnanta, and C. Pluempitiwiriyawej, "Ontology-based multiperspective requirements traceability framework," *Knowl. Inf. Syst.*, vol. 25, no. 3, pp. 493–522, Dec. 2010.

[25] E. Bagheri, M. Asadi, F. Ensan, D. Gašević, and B. Mohabbati, "Bringing semantics to feature models with SAFMDL," in *Proc. Conf. Center Adv. Stud. Collaborative Res.* Armonk, NY, USA: IBM, 2011, pp. 287–300.

[26] V. Castañeda, L. C. Ballejos, and M. L. Caliusco, "Improving the quality of software requirements specifications with semantic web technologies," in *Workshop Em Engenharia De Requisitos (WER)*. Buenos Aires, Argentina, 2012.

[27] O. Daramola, G. Sindre, and T. Moser, "Ontology-based support for security requirements specification process," in *On the Move to Meaningful Internet Systems: OTM Workshops*. Berlin, Germany: Springer, 2012, pp. 194–206.

[28] O. Daramola, T. Stålhane, I. Omoronyia, and G. Sindre, "Using ontologies and machine learning for hazard identification and safety analysis," in *Managing Requirements Knowledge*. Berlin, Germany: Springer, 2013.

[29] J. Guo, Y. Wang, P. Trinidad, and D. Benavides, "Consistency maintenance for evolving feature models," *Exp. Syst. Appl.*, vol. 39, no. 5, pp. 4987–4998, Apr. 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417411014990

[30] H. He, Z. Wang, Q. Dong, W. Zhang, and W. Zhu, "Ontology-based semantic verification for UML behavioral models," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 23, no. 2, pp. 117–146, 2013.

[31] H. Hu, D. Yang, C. Ye, C. Fu, and R. Li, "Detecting interactions between behavioral requirements with OWL and SWRL," *Int. J. Comput. Inf. Eng.*, vol. 4, no. 12, pp. 1833–1839, 2010.

[32] L. Kof, R. Gacitua, M. Rouncefield, and P. Sawyer, "Ontology and model alignment as a means for requirements validation," in *Proc. IEEE 4th Int. Conf. Semantic Comput.*, Sep. 2010, pp. 46–51.

[33] P. Kroha, R. Janetzko, and J. E. Labra, "Ontologies in checking for inconsistency of requirements specification," in *Proc. 3rd Int. Conf. Adv. Semantic Process.*, Oct. 2009, pp. 32–37.

[34] J. F. Lima, B. P. Garcia, C. M. G. Amaral, and G. M. Caran, "Building an ontological model for software requirements engineering," in *ENTERprise Information Systems* (Communications in Computer and Information Science). Berlin, Germany: Springer, 2011, pp. 228–237.

[35] C.-L. Liu, "CDADE: Conflict detector in activity diagram evolution based on speech act and ontology," *Knowl.-Based Syst.*, vol. 23, no. 6, pp. 536–546, Aug. 2010.

[36] C. López, H. Astudillo, and L. M. Cysneiros, "Semantic-aided interactive identification of reusable NFR knowledge fragments," in *On the Move to Meaningful Internet Systems: OTM Workshop* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2008, pp. 324–333.

[37] T. Moser, D. Winkler, M. Heindl, and S. Biffl, "Requirements management with semantic technology: An empirical study on automated requirements categorization and conflict analysis," in *Advanced Information Systems Engineering* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2011, pp. 3–17.

[38] A. Mtibaa and F. Gargouri, "A multi-representation ontology for the specification of multi-context requirements," in *Advanced Internet Based Systems and Applications* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2009, pp. 259–269.

[39] H. H. Wang, Y. F. Li, J. Sun, H. Zhang, and J. Pan, "Verifying feature models using OWL," *J. Web Semantics*, vol. 5, no. 2, pp. 117–129, 2007. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1570826807000042

[40] S. Park, H. Kim, Y. Ko, and J. Seo, "Implementation of an efficient requirements-analysis supporting system using similarity measure techniques," *Inf. Softw. Technol.*, vol. 42, no. 6, pp. 429–438, Apr. 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584999001020

[41] R. Yan, C.-H. Cheng, and Y. Chai, "Formal consistency checking over specifications in natural languages," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2015, pp. 1677–1682. [Online]. Available: http://dl.acm.org/citation.cfm?id=2757012.2757200

[42] I. Boukhari, L. Bellatreche, and S. Jean, "An ontological pivot model to interoperate heterogeneous user requirements," in *Leveraging Applications of Formal Methods, Verification and Validation Applications and Case Studies* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2012, pp. 344–358.

[43] A. A. AlSanad, A. Chikh, and A. Mirza, "A domain ontology for software requirements change management in global software development environment," *IEEE Access*, vol. 7, pp. 49352–49361, 2019.

[44] J. Polpinij, "An ontology-based text processing approach for simplifying ambiguity of requirement specifications," in *Proc. IEEE Asia–Pacific Services Comput. Conf. (APSCC)*, Dec. 2009, pp. 219–226.

[45] A. Rashwan, O. Ormandjieva, and R. Witte, "Ontology-based classification of non-functional requirements in software specifications: A new corpus and SVM-based classifier," in *Proc. IEEE 37th Annu. Comput. Softw. Appl. Conf.*, Jul. 2013, pp. 381–386.

[46] J. Matsuoka and Y. Lepage, "Ambiguity spotting using wordnet semantic similarity in support to recommended practice for software requirements specifications," in *Proc. 7th Int. Conf. Natural Lang. Process. Knowl. Eng.*, Nov. 2011, pp. 479–484.

[47] T. H. Al Balushi, P. R. F. Sampaio, and P. Loucopoulos, "Eliciting and prioritizing quality requirements supported by ontologies: A case study using the ElicitO framework and tool," *Exp. Syst.*, vol. 30, no. 2, pp. 129–151, May 2013. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1468-0394.2012.00625.x

[48] G. N. Aranda, A. Vizcaino, and M. Piattini, "Analyzing ontology as a facilitator during global requirements elicitation," in *Proc. 4th IEEE Int. Conf. Global Softw. Eng.*, Jul. 2009, pp. 309–314.

[49] I. Bicchierai, G. Bucci, C. Nocentini, and E. Vicario, "An ontological approach to systematization of SW-FMEA," in *Computer Safety, Reliability, and Security* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2012, pp. 173–184.

[50] E. J. de Lima, J. A. R. Nt, G. B. Xexéo, and J. M. de Souza, "ARARA—A collaborative tool to requirement change awarenes," in *Proc. 14th Int. Conf. Comput. Supported Cooperat. Work Design*, 2010, pp. 134–139.

[51] S. Hayashi, T. Yoshikawa, and M. Saeki, "Sentence-to-code traceability recovery with domain ontologies," in *Proc. Asia Pacific Softw. Eng. Conf.*, Nov. 2010, pp. 385–394.

[52] A. M. Hickey and A. M. Davis, "An ontological approach to requirements elicitation technique selection," in *Ontologies: A Handbook of Principles, Concepts and Applications in Information System*. Boston, MA, USA: Springer, 2007, pp. 403–431.

[53] A. M. Hoss and D. L. Carver, "Towards combining ontologies and model weaving for the evolution of requirements models," in *Innovations for Requirement Analysis From Stakeholders Needs to Formal Designs* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2008, pp. 85–102.

[54] K. Karwowski, W. Wysota, and J. Wytrebowicz, "Computer aided requirements management," in *Computational Collective Intelligence Semantic Web, Social Networks and Multiagent Systems* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2009, pp. 389–400.

[55] M. Kassab, O. Ormandjieva, and M. Daneva, "Relational-model based change management for non-functional requirements: Approach and experiment," in *Proc. 5th Int. Conf. Res. Challenges Inf. Sci.*, May 2011, pp. 1–9.

[56] C.-L. Liu and H.-L. Yang, "Applying ontology-based BLOG to detect information system post-development change requests conflicts," *Inf. Syst. Frontiers*, vol. 14, no. 5, pp. 1019–1032, Dec. 2012.

[57] B. N. Machado, L. O. Arantes, and R. Falbo, "Using semantic annotations for supporting requirements evolution," in *Proc. SEKE*, 2011, pp. 185–190.

[58] E. Niemelä, A. Evesti, and P. Savolainen, "Modeling quality attribute variability," in *Proc. ENASE*, 2008, pp. 169–176.

[59] P. F. Pires, F. C. Delicato, R. Cóbe, T. Batista, J. G. Davis, and J. H. Song, "Integrating ontologies, model driven, and CNL in a multi-viewed approach for requirements engineering," *RequirementS Eng.*, vol. 16, no. 2, pp. 133–160, Jun. 2011.

[60] T. Riechert and T. Berger, "Leveraging semantic data Wikis for distributed requirements elicitation," in *Proc. ICSE Workshop Wikis Softw. Eng.*, May 2009, pp. 7–13.

[61] P. Schugerl, J. Rilling, R. Witte, and P. Charland, "A quality perspective of software evolvability using semantic analysis," in *Proc. IEEE Int. Conf. Semantic Comput.*, Sep. 2009, pp. 420–427.

[62] H. H. Wang, D. Damljanovic, and J. Sun, "Enhanced semantic access to formal software models," in *Formal Methods and Software Engineering* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2010, pp. 237–252.

[63] J. N. O. Dag, B. Regnell, P. Carlshamre, M. Andersson, and J. Karlsson, "A feasibility study of automated natural language requirements analysis in market-driven development," *Requirements Eng.*, vol. 7, no. 1, pp. 20–33, Apr. 2002.

[64] F. Gailly, S. España, G. Poels, and O. Pastor, "Integrating business domain ontologies with early requirements modelling," in *Advances in Conceptual Modeling—Challenges and Opportunities*. Berlin, Germany: Springer, 2008, pp. 282–291.

[65] H. Kaiya, Y. Shimizu, H. Yasui, K. Kaijiri, and M. Saeki, "Enhancing domain knowledge for requirements elicitation with web mining," in *Proc. Asia Pacific Softw. Eng. Conf.*, Nov. 2010, pp. 3–12.

[66] G. Li, Z. Jin, Y. Xu, and Y. Lu, "An engineerable ontology based approach for requirements elicitation in process centered problem domain," in *Knowledge Science, Engineering and Management* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2011, pp. 208–220.

[67] I. Omoronyia, G. Sindre, T. Stålhane, S. Biffl, T. Moser, and W. Sunindyo, "A domain ontology building process for guiding requirements elicitation," in *Requirements Engineering: Foundation for Software Quality* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2010, pp. 188–202.

[68] J. Osis, A. Slihte, and A. Jansone, "Using use cases for domain modeling," in *Proc. ENASE*, 2012, pp. 224–231.

[69] M. Saeki, S. Hayashi, and H. Kaiya, "Enhancing goal-oriented security requirements analysis using common criteria-based knowledge," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 23, no. 5, pp. 695–720, Jun. 2013.

[70] M. Shibaoka, H. Kaiya, and M. Saeki, "GOORE: Goal-oriented and ontology driven requirements elicitation method," in *Advances in Conceptual Modeling—Foundations and Applications* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2007, pp. 225–234.

[71] A. Souag, C. Salinesi, I. Wattiau, and H. Mouratidis, "Using security and domain ontologies for security requirements analysis," in *Proc. IEEE 37th Annu. Comput. Softw. Appl. Conf. Workshops*, Jul. 2013, pp. 101–107.

[72] W. Alhoshan, R. Batista-Navarro, and L. Zhao, "Using frame embeddings to identify semantically related software requirements," in *Proc. REFSQ Workshops*, 2019, pp. 1–9.

[73] W. Alhoshan, R. Batista-Navarro, and L. Zhao, "Semantic frame embeddings for detecting relations between software requirements," in *Proc. 13th Int. Conf. Comput. Semantics-Student Papers*, 2019, pp. 44–51. [Online]. Available: https://www.aclweb.org/anthology/W19-0606

[74] S. Farfeleder, T. Moser, and A. Krall, "Using semantic relatedness and locality for requirements elicitation guidance," in *Proc. 24th Int. Conf. Softw. Eng. Knowl. Eng.*, 2012, pp. 19–24.

[75] Z. Jin, X. Chen, and D. Zowghi, "Performing projection in problem frames using scenarios," in *Proc. 16th Asia–Pacific Softw. Eng. Conf.*, Dec. 2009, pp. 249–256.

[76] D. Dermeval, J. Vilela, I. B. Bittencourt, J. Castro, S. Isotani, P. Brito, and A. Silva, "Applications of ontologies in requirements engineering: A systematic review of the literature," *Requirements Eng.*, vol. 21, no. 4, pp. 405–437, 2016.

[77] S. Farfeleder, T. Moser, A. Krall, T. Stalhane, H. Zojer, and C. Panis, "DODT: Increasing requirements formalism using domain ontologies for improved embedded systems development," in *Proc. 14th IEEE Int. Symp. Design Diag. Electron. Circuits Syst.*, Cottbus, Germany, Apr. 2011, pp. 271–274.

[78] "Analysis ontology—Integrated model-centric engineering," NASA, JPL Syst. Softw. Division, Jet Propuls. Lab., California Inst. Technol., Pasadena, CA, USA, Tech. Rep. D-68444, 2012.

[79] "Mission ontology—Integrated model-centric engineering," NASA, JPL Syst. Softw. Division, Jet Propuls. Lab., California Inst. Technol., Pasadena, CA, USA, Tech. Rep. D-68443, 2012.

[80] "Project ontology—Integrated model-centric engineering," NASA, JPL Syst. Softw. Division, Jet Propuls. Lab., California Inst. Technol., Pasadena, CA, USA, Tech. Rep. D-68445, 2012.

[81] D. A. Wagner, M. B. Bennett, R. Karban, N. Rouquette, S. Jenkins, and M. Ingham, "An ontology for state analysis: Formalizing the mapping to SysML," in *Proc. IEEE Aerosp. Conf.*, Mar. 2012, pp. 1–16.

[82] T. Fancott, P. Kamthan, and N. Shahmir, "Towards next generation requirements engineering," in *Proc. Int. Conf. Social Informat.*, vol. 22, Dec. 2012, pp. 328–331.

[83] A. Yessad, C. Faron-Zucker, R. Dieng-Kuntz, and M. T. Laskri, "Ontology-based semantic relatedness for detecting the relevance of learning resources," *Interact. Learn. Environ.*, vol. 19, no. 1, pp. 63–80, Jan. 2011.

[84] (2022). *Shapes Constraint Language (SHACL)*. Accessed: Feb. 11, 2022. [Online]. Available: https://www.w3.org/TR/shacl/

[85] K. Winter, H. Femmer, and A. Vogelsang, "How do quantifiers affect the quality of requirements?" in *Requirements Engineering: Foundation for Software Quality* (Lecture Notes in Computer Science). Cham, Switzerland: Springer, 2020, pp. 3–18.

[86] A. van Renssen, *Gellish: A Generic Extensible Ontological Language—Design and Application of a Universal Data Structure*. Amsterdam, Netherlands: IOS Press, Jan. 2005.

[87] G. A. Miller, "WordNet: A lexical database for English," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[88] (2008). *SPARQL Query Language for RDF—W3C Recommendation*. Accessed: Dec. 31, 2019. [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/

[89] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering: Guidelines and Examples*, 1st ed. Hoboken, NJ, USA: Wiley, 2012.

[90] R. Van Solingen, V. Basili, G. Caldiera, and H. D. Rombach, *Goal Question Metric (GQM) Approach*. Hoboken, NJ, USA: Wiley, 2002. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/0471028959.sof142

[91] D. Vrandečić, "Ontology evaluation," in *Handbook on Ontologies* (International Handbooks on Information Systems). Berlin, Germany: Springer, 2009, pp. 293–313.

KONSTANTINOS MOKOS was born in Thessaloniki, Greece, in 1980. He received the B.S. degree in computer science from the Aristotle University of Thessaloniki, Greece, in 2005, and the M.S. degree (Hons.) in software engineering from York University, York, U.K., in 2006. He is currently pursuing the Ph.D. degree in software engineering with the Aristotle University of Thessaloniki.

From 2011 to 2013, he was a National Trainee with the European Space Agency; ESTEC; and TEC-SWE, Noordwijk, The Netherlands. Since 2013, he has participated in various programs including ESA's in-flight Solar Orbiter and Sentinel-6 missions, in which except from developing AOCS systems, he was also responsible for the on-board software verification and validation. His research interests include space systems formal modeling and validation, requirements engineering, dependability assessment of software intensive systems, and knowledge-based design and development.

THEODOROS NESTORIDIS received the undergraduate and postgraduate degrees in informatics from the Department of Informatics, Aristotle University of Thessaloniki, Greece, where he is currently pursuing the Ph.D. degree in software engineering. His research interests include software and networks systems engineering.

PANAGIOTIS KATSAROS received the bachelor's degree in mathematics from the Aristotle University of Thessaloniki (AUTh), Greece, the Master of Science degree in software engineering from Aston University, Birmingham, and the Ph.D. degree in computer science from AUTh. He is an Associate Professor with the School of Informatics, AUTh. He has published over 100 research papers in international journals and conference proceedings on software engineering. His research interests include the formal verification of software/systems, the model-based design, the analysis of dependability and security, and the simulation-based performance analysis and optimization. He is a coordinator (or participates) in national and European research and development projects focusing on engineering of software for the Internet of Things systems, space systems, and more recently autonomous systems. Regular updates on his recent research achievements can be accessed online at https://depend.csd.auth.gr

NICK BASSILIADES (Member, IEEE) received the M.Sc. degree in applied artificial intelligence from the Computing Science Department, University of Aberdeen, in 1992, and the Ph.D. degree in parallel knowledge-based systems from the School of Informatics, Aristotle University of Thessaloniki (AUTh), Greece, in 1998. He is currently a Professor with AUTh, where he is also currently serving as the Head for the IT Center. He is the Director of RuleML, Inc. He has published more than 230 papers at journals, conferences, and books; and has coauthored five books and co-edited 11 volumes. His published research has received over 5000 citations (H-index 33), while seven of his papers have received awards. He has been involved in 40 research and development projects leading 11 of them. His research interests include knowledge-based and rule systems, multiagent systems, ontologies/linked data/semantic web, electric vehicles charging scheduling, and eXplainable AI. He is a member of the Greek Computer Society and ACM. He has been a member of the program committee of more than 150 and on the organizational committee of nine conferences/workshops and was the program chair of 11 conferences/workshops. He has been the General Secretary of the Board of the Greek Artificial Intelligence Society. Regular updates on his recent research achievements can be accessed online at: http://tinyurl.com/bassiliades