Toward an Ontology-Based Framework for Textual System Requirements Extraction and Analysis

Zakaria Mejdoul[1,2][a], Gaëlle Lortal[1][b] and Myriam Lamolle[2,3][c]
[1]*Thales, cortAIx Labs, France*
[2]*LIASD, Paris 8 University, IUT de Montreuil, France*
[3]*CGI, IMT Mines Albi, Albi, France*
{*zakaria.mejdoul, gaelle.lortal*}*@thalesgroup.com, myriam.lamolle@mines-albi.fr*

Keywords: Ontology Modelling, Knowledge Extraction, Natural Language Processing, Model-Based System Engineering, System Requirements.

Abstract: This paper introduces the context, objectives and expectations of an ontology-driven framework designed to support engineers in the analysis of textual system requirements. The primary goals are twofold: (i) to keep the System Engineering-*(SE)* formal processes that satisfy industrial constraints, and (ii) to provide a semantic representation of textual requirements, enabling consistent semantic analysis through the logical properties of ontologies. Formalization and semantic analysis of system requirements provide early evidence of adequate specification, for reducing the validation tasks and high-cost corrective measures during later system development phases. Integrating ontologies into the SE process enhances system engineers' ability to understand and manage requirements, leading to a smoother design and more accurate operation.

1 INTRODUCTION

The complexity of systems continues to increase depending on their use-case domains (Military, Aerospace, Avionics, etc.). This increasing complexity has brought both new opportunities, and increased challenges for organizations involved in system design and production. These challenges span the entire lifecycle of a system and affect all levels of architectural detail. The ISO/IEC/IEEE 15288:2023 (IEEE, 2023) standard, concerning *Systems and software engineering–System life cycle processes*, provides a standardized framework for processes throughout a system's lifecycle. The technical processes outlined include: (i) Begin from Business or mission analysis process, then (ii) Stakeholder needs and requirements definition process, (iii) System requirements definition process, (iv) System architecture definition process, until (v) Maintenance and Disposal process. This paper specifically focuses on the Requirements Engineering (RE) definition process within the organizational context of Thales company. Textual requirements are essential for defining system functionality, constraints, and expectations. When specifications becomes too large or complex, manually parsing and analysing these requirements risks missing inconsistencies, incomplete requirements, or contradictions, which can result in project errors later on. During the system Requirements Analysis-(RA) phase (Castañeda et al., 2010), system engineers must check that the requirements are unambiguous (clear terminology, common interpretation), consistent (not redundant, no conflicting requirements across system components) and complete (missing information to understand the requirement) (IEEE, 2023). Existing RE tools face limitations, particularly the lack of automated semantic analysis, which is critical when dealing with large-scale specification documents. Consequently, system engineers encounter significant barriers, including the susceptibility to human error, the time-consuming nature of manual analysis, and the need for collaborative efforts among team members. Given these challenges, there is a need for enhanced automation in system requirements management tools. (Dori, 2016) mentions that "systems science and engineering are in need of a well-defined foundational, universal, general, necessary and sufficient ontology that would underpin concepts and terms it uses in order for them to be precise and unambiguous." Therefore, it will be of significant value

[a] https://orcid.org/0000-0003-4341-2139
[b] https://orcid.org/0000-0001-5374-6584
[c] https://orcid.org/0000-0001-9652-7891

Mejdoul, Z., Lortal, G. and Lamolle, M.
Toward an Ontology-Based Framework for Textual System Requirements Extraction and Analysis.
DOI: 10.5220/0013210500003929

In *Proceedings of the 27th International Conference on Enterprise Information Systems (ICEIS 2025) - Volume 2*, pages 136-143
ISBN: 978-989-758-749-8; ISSN: 2184-4992

to follow up the implementation of a framework able to process and ensure a part of system requirements analysis (consistency, completeness, correctness), in order to avoid this tedious task for engineers, which cost time and resources for the product owner organisation.

Currently, semantic technologies, mainly represented by Knowledge Bases-(KB) and Automatic Reasoning (Glimm and Kazakov, 2019), promise new industrial processes development and operation. By exploiting KBs and reasoning tools, it is possible to create semantic applications supported by ontologies and logical reasoners (Fahad et al., 2008). Ontologies are being introduced into SE (Sarder and Ferreira, 2007) to provide a shared understanding among individuals, organizations, and systems, thereby addressing gaps in SE tools, methodologies, and processes. They offer a standardized vocabulary to avoid ambiguity and ensure consistency, serving as a common agreement within a user community (Rousseau et al., 2016; Roussey et al., 2011; Sillitto, 2011). Additionally, ontologies facilitate semantic interoperability between systems, and also between humans and computers (Bittner et al., 2005).

Our SE scenario focuses on supporting engineers in analyzing textual system requirements during the design phase of the intended system. Our framework aims to ensure completeness, consistency, and correctness. When done manually, analyzing requirements, especially in large documents (exceeding 1500 pages for example), can be error-prone and time-consuming, making ensuring optimality quite challenging. This paper intended to answer the following research questions: ***(RQ1.)*** What approach or technique is needed to automatically extract textual requirements (ID, statement, metadata) from a specification document? ***(RQ2.)*** How can ontologies be used to federate our specification document knowledge and requirements? ***(RQ3.)*** How can the resulted ontology be used to perform system requirements analysis: avoid ambiguities, redundancies, or any inconsistencies? The solution must help engineers ensure there are no contradictory, redundant or incomplete elements in the system requirements model. If such issues arise, the engineer (main user) will need to identify and rectify the errors, emphasizing the importance of the model's explainability.

The paper is organized as follows: Section 2 reviews previous work on SE and the use of ontologies for system requirements, Section 3 presents the proposed framework, Section 4 describes an experimental use-case and its outputs, Section 5 discusses the cited research questions, and Section 6 concludes and proposes potential avenues for future research.

2 RELATED WORK

This section summarizes some research contributions and industrial tools related to ontology-driven systems engineering and system requirements analysis. For our research, We consider the general definition: An *ontology* is a formal, explicit specification of a shared conceptualization, in a particular domain of knowledge (Studer et al., 1998). (Yang et al., 2019) reveals the state of the art of the Ontology-Based Systems Engineering (OBSE) and provides a detailed analysis of key SE knowledge areas supported by ontologies. It outlines a clear roadmap of how ontologies support SE and assesses the extent of their application within this domain. The results show that ontologies are applied across various SE knowledge areas, and their benefits to SE are well-recognized. However, despite the low adoption rate of formal ontology engineering techniques in SE, it is gradually increasing. It argues that progressing SE into a model-based discipline will promote further research on the use of ontologies in SE. The frequency of conference papers suggests that the topic is still in its early stages, with researchers who are eager for peer-to-peer discussion.

2.1 System Requirements Analysis

Requirements Engineering (RE) processes need efficient management of the substantial volume of information and knowledge used during RA activity (Castañeda et al., 2010). Ambiguous requirements, for example, must be minimized since they lead to wasted time and repeated work, especially when different stakeholders have different interpretations for the same requirement. Similarly, efforts must be made to minimize redundant, inconsistent, and incomplete requirements. There are other criteria that are not addressed in our research scope, such as feasibility, measurability, etc. The ISO/IEC/IEEE 29148:2018 (IEEE, 2018) Systems and software engineering — Life cycle processes — Requirements engineering standard, provides additional guidance for the application of requirements engineering and management processes for requirements-related activities. (Stachtiari et al., 2018) have addressed RA challenge, within the context of a correctness-by-construction design approach, which identifies behavioral (not semantic) inconsistencies in requirements. However, formal reasoning is possible only if a well-defined interpretation semantics exists for the requirements specification (Glimm and Kazakov, 2019).

Many tools (e.g., DOORS[1], Polarion[2], Jama[3]) support system requirements management. However, automation of the analysis process remains limited. Requirement management tools often rely on manual input or the functionality to import structured documents (e.g., ReqIF), rather than natural language (textual format). In Table 1, we represent a benchmark table of commonly used Requirements Validation Tools. These tools vary in functionalities, such as requirement management, traceability, validation capabilities. We find several gaps exist in both the state of the art and existing tools.

Limited Automation for RA:

Problem. Although many tools (e.g., DOORS, Polarion, Jama) support the management of system requirements, automation of the analysis process remains limited. Most tools provide basic or advanced support for tracking requirements but require manual steps to analyse & validate them through testing, simulation, or formal methods.

Gap. There is a significant opportunity to develop more automated analysis tools, particularly those which align with MBSE viewpoints and processes, to reduce the manual effort involved in analysis phase.

Weak Support for Natural Language Processing (NLP):

Problem. Requirements are often written in natural language, which is prone to ambiguity, inconsistency, and incompleteness. Most tools struggle to process and analyze natural language requirements effectively.

Gap. Existing tools require enhanced NLP capabilities to automatically parse, represent, understand, and analyse natural language requirements, identifying ambiguities and inconsistencies early in the SE process.

Hence, the importance of having a semantically-common vocabulary among engineers, with shared semantics, becomes crucial, especially when they come from heterogeneous fields. We suggest that the use of a formal, shared and explicit conceptualization could fill some of these gaps. This can be achieved by extracting and gathering textual requirements knowledge from specification documents and structuring it. Engineers and stakeholders also need to commonly structure requirements using an *ontology*. An ontology can help ensure that requirements are understood uniformly across different stakeholders, reducing ambiguities and inconsistencies, and thereby improving the overall requirements engineering process.

2.2 Ontologies for System Requirements Analysis

(Mokos et al., 2022) presented an ontology-driven requirements formulation and semantic analysis approach, for system requirements. To address the lack of natural language misinterpretation, the authors employed requirement boilerplates (Pohl and Rupp, 2015) with an ontology. The semantic modeling framework facilitates the development and integration of domain-specific ontologies for specifying requirements. However, automating the knowledge extraction process remains necessary, as engineers currently have to manually insert requirements (quite complex when requirements are too many). Whereas (Castañeda et al., 2010) described several challenges faced during RE activities, such as managing the substantial volume of information, especially when stakeholders come from diverse backgrounds. The author designed a requirements ontology, application domain ontology and requirements specification document ontology. The automatic method for retrieving knowledge and populating these ontologies has not been specified, as the framework had not yet been implemented. In our case, we are particularly interested in ontology framework architecture, especially the specification document ontology, eligible to reuse for other objectives or projects. It is also a formal representation enabling requirements to be linked to their source document (traceable), so that explanations can be generated when inconsistencies occur.

3 METHODOLOGY

In the context of this research, the progression towards solving the identified problem (Section 1) involves methodological steps:

3.1 System Requirements Specification

Document specification used in this research is an internal system specification of COOPANS[4], a system for ATM, developed by Thales and other collaborators. This document contains more than 1500 pages (with 2902 distinct requirements) including system requirements as follows:

[1]Engineering Requirements Management DOORS - visited 22/10/2024

[2]www.polarion.plm.automation.siemens.com - visited 22/10/2024

[3]https://www.jamasoftware.com/ - visited 22/10/2024

[4]https://www.coopans.com/ATM-SYSTEM - visited 22/10/2024; the specification document is confidential, so we cannot publish examples of COOPANS system requirements.

Table 1: Benchmark of the common-used requirements management tools.

Tool Name	Primary Features	Strengths	Weaknesses
IBM Engineering Requirements Management DOORS (DOORS Next)[3]	- Requirements capture and management - Traceability across lifecycle - Change management - Collaboration	- Highly customizable - Strong traceability - Scalable for large projects	- Steep learning curve - Expensive
Jama Connect[3]	- Requirements gathering and validation - Traceability - Impact analysis - Collaboration and reviews	- User-friendly - Strong validation and collaboration features	- Limited customization - Limited support for large-scale projects
Polarion ALM (Siemens)[5]	- Requirements management - Traceability - Versioning and audits - Compliance management	- Strong in compliance and traceability - Integrated with ALM/PLM tools	- Expensive for small teams - User interface can be complex

EU-CONTEXT-001
The system shall ensure a VHF (Very High Frequency) voice communication between pilots and controllers. *EU-CONTEXT-005 VHF-CSCI*

The term "EU-CONTEXT-001" refers to the requirement identifier (ID), and the term "EU-CONTEXT-005 VHF-CSCI" refers to the requirement tags that are often used to mark traceability, linking a requirement to other requirements such as *EU-CONTEXT-005*, or to a system component such as *VHF-CSCI* (Computer Software Configuration Item).

3.2 Framework Workflow

To address the issues of RQ1, RQ2 and RQ3 (see Section 1), we propose a streamlined framework based on text extraction, text analysis, and populating an ontology representing system requirements domain. This framework aim to provide engineers with: (i) A formal, explicit and shared format of requirements representation, allowing the structuring of COOPANS specification document knowledge by principal concepts, properties and individuals (facts) to clarify requirements terms; (ii) Machine-readable representation to automate COOPANS RA support (capturing unambiguous, redundant, inconsistent and incomplete requirements) in a seamless (explainable) way. The solution framework includes: automated text extraction algorithms; exploitation algorithms for analyzing and reasoning about requirements ontology (logical verification, inference). An overview of the textual requirements extraction and analysis automation framework is presented in Figure 1.

3.2.1 Text Extraction & Pre-Processing

The process begins with extraction of textual requirements by parsing the PDF file of the specification document to produce raw text. This text is then cleaned and pre-processed to eliminate unnecessary data (e.g., headers, footers, summaries, etc.). The user must identify the page range of the requirements ID list, in order to parse the IDs list and find all requirements along with their IDs, statements, and tags (see subsection 3.1). We serialize the extracted requirements into a Dictionary (Dict) object, ready to be processed with NLP (see Figure 1).

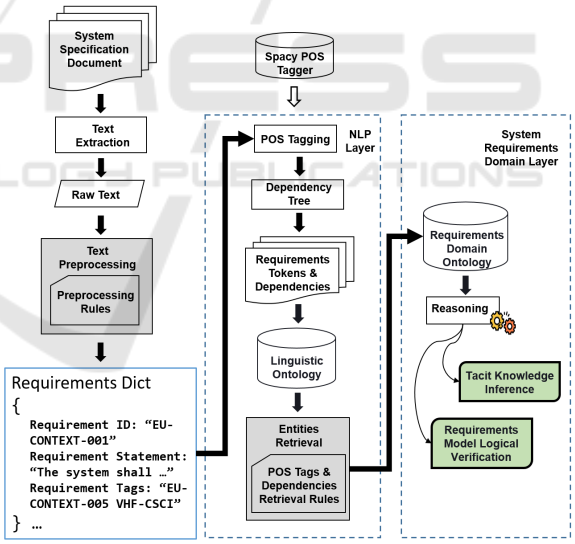

Figure 1: Illustration of the expected Textual Requirements Extraction and Analysis Framework.

3.2.2 NLP Layer: Part-Of-Speech (POS) Tagging

The framework uses NLP techniques, such as Part-Of-Speech (POS) tagging and requirements dependency tree identification, to tag and identify parts of requirements statements and their syntactic depen-

dencies, respectively. We use Spacy[5] NLP library for POS tagging, to identify tokens (individual words) and assign to each token a grammatical class (e.g., *VERB, NOUN, AUX, etc.*). Furthermore, Spacy allows us to explicitly identify syntactic dependencies between tokens (e.g., *nsubj -nominal subject-, dobj -direct object-, etc.*). Key components in the extracted text, such as system entities, actions, and constraints are extracted, while metadata such as requirement IDs and traceability tags are also captured for future use. For example, in the sentence "*The system shall allow user to avoid obstacles.*", the POS tagger identifies "system", "user" and "obstacles" as nouns, and "allow" and"avoid" as verbs. The identified dependencies are: *system* is nominal subject of *allow*, *avoid* is clausal complement of *avoid* and *obstacles* is direct object of *avoid* (see Figure 2).

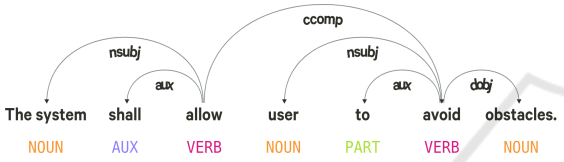

Figure 2: Example of POS Tagging and dependency tree.

Once this extraction is complete, the extracted knowledge is used to structure the requirements into a linguistic ontology (LO). This ontology serves as a semantic representation of the requirements knowledge, and includes POS tagging results, such as tokens and their assigned tags and dependencies. For instance, entities such as "*user*" and "*obstacles*" are modeled as *Tokens*; *VERB, NOUN, AUX* are modeled as *POSTagTypes*; Each requirement has a *TokenList* that groups all its statement tokens by *has_token_list* relationship.

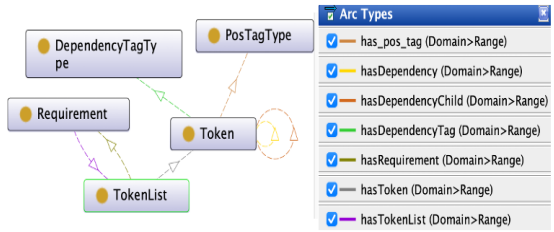

Figure 3: Linguistic ontology resulting from POS tagging.

The generated ontology architecture is detailed in Figure 3. This ontology is populated with the knowledge extracted from the textual requirements and is ready to progress from the syntactic level related to POS tagging to a semantic level specific to the system requirements domain (see Figure 3).

[5]www.spacy.io/usage/linguistic-features#pos-tagging - visited 22/10/2024

3.2.3 System Requirements Domain-Specific Ontology

We use the ontology referenced in subsection 2.2, which represents the system requirements boilerplates (Mokos et al., 2022). This allows us to specify functions, components, actions, conditions and other elements. We define transition rules to parse the linguistic ontology entities (Tokens) and fill the placeholders in the requirements boilerplates within Requirements Definition Ontology (RDO). RDO imposes structural constraints on assembling a requirement, which provides a formal syntax. Here are some examples of RDO boilerplates that define the main textual system requirements templates:

> **system/function** shall [not] set [<quantifier>] **item** [to **stateValue**]
>
> **system/function** shall [not] perform **function**
>
> **system/function** shall [not] transfer **flow/item**

system/function, **item**, **stateValue** and **flow/item** are placeholders of the previous boilerplates. To retrieve these placeholders from LO and fill RDO with them, we define tokens retrieval rules based on their POS tags and dependencies in LO. For instance, in Figure 2, if the token *allow* is tagged as a *VERB* and have dependency *ROOT*, according to RBO boilerplates, the token *system*, which is a *NOUN* and the nominal subject (*nsubj*) of *allow*, is a **system** placeholder. Likewise, the phrase *avoid obstacles*, which constitutes a clausal complement (*ccomp*) of *allow*, is assigned to **function** placeholder. Then we can then represent the referenced requirement as follows: System $<$ system $>$ $-$allow $\rightarrow$ Function $<$ avoid obstacles $>$.

Through this ontology-based representation of requirements, we transition from the textual format to a well-defined semantic structure, consisting of uniquely identified elements such as classes, relationships and individuals. These elements serve to explicitly capture and encode knowledge, enabling logical verification, semantic analysis, and the potential inference of new knowledge within the requirements.

3.2.4 Semantic Analysis

The semantic representation of boilerplate-based requirements makes it possible to detect ambiguities, inconsistencies, redundancies and incompleteness in specifications by reasoning and appropriate SWRL (Semantic Web Rule Language) inference rules (Bossche-Marquette et al., 2024). It is possible to define rules specific to a given application in a language like SWRL. In this case, the reasoner assimilates these new rules and applies them to the ontology

in the same way as pre-established rules. These rules are useful to detect:

- *Ambiguity*: Identifying requirements where boilerplate placeholders are filled with values that could be substituted with similar instances.
- *Redundancy*: Identifying requirements with the same boilerplates that contain semantically equivalent placeholder values. Requirements may not be redundant if they are used in different operational contexts.
- *Inconsistency*: Identifying requirements with similar boilerplates where placeholders are filled with conflicting instances or disjoint terms (e.g., requirements with the same conditions but different actions).
- *Incompleteness*: Identify requirements that are incomplete or have missing elements (e.g., placeholders in the boilerplate that are not filled).

These criteria are essential in the system requirements analysis phase. The reasoner's capability to detect and flag these issues for engineers in a seamless way (axioms verbalization) (Fuchs et al., 2006; Mejdoul. and Lortal., 2022), along with explanations based on ontological axioms, supports this phase effectively.

4 EXPERIMENTS

To enhance our design thinking, we reflect on our requirements semantic analysis approach using a simple use-case: a tricolor traffic light system. In this scenario we model a tricolor light system, where each light (red, yellow, and green) represents different conditions and actions to control traffic flow. The system requirements are rules governing the lights and how they manage cars at intersections. Our objective is to ensure that these requirements do not conflict or overlap unnecessarily. Specifically, we aim to detect: (i) *Requirements Redundancy*: Two or more requirements that essentially describe the same thing. (ii) *Requirements Conflict*: Requirements that, under the same conditions, lead to opposing actions (e.g., one says "stop" while another says "go").

Textual system requirements in this scenario are as follows:
***(R1)**: The system shall allow cars to go when the light is green.*
***(R2)**: The system shall ensure cars come to a complete stop when the light is red.*
***(R3)**: The system shall require cars to slow down when the light is yellow.*
***(R4)**: The system shall ensure cars move quickly when the light is green.*
***(R5)**: The system shall require cars to stop when the light is yellow.*

Analyzing requirements shows that each requirement specifies a condition (the state of the traffic light) and an action (the behavior expected of cars). Using our framework (see Section 3.2), we needs to identify when these requirements are either redundant and/or inconsistent.

4.1 System Requirements Ontology Structure

After NLP processing (Subsection 3.2.2) and retrieving requirements tokens to populate boilerplates placeholders in requirements ontology (RDO) (Subsection 3.2.3), we define the OWL ontology structure with the following entities:
3 Main Classes: (C1) **Requirement**: Represents a system requirement (e.g., R1, R2); (C2) **Condition**: Represents the state of the traffic light placeholders (e.g., green, yellow, red); (C3) **Action**: Represents actions placeholders, the behavior that cars should follow (e.g., go, stop, slow down);
2 Main Object Properties: (OP1) **hasCondition**: Associates a requirement with the condition placeholder it depends on (e.g., R1 hasCondition "green light"); (OP2) **hasAction**: Associates a requirement with the action placeholder it requires (e.g., R1 hasAction "go");

2 Main Data Properties: (DP1) **conditionType**: Describes the specific traffic light color for a requirement's condition (e.g., green, yellow, red); (DP2) **actionType**: Describes the type of action prescribed by the requirement (e.g., go, stop, slow down);
2 Relationships to Infer: hasSimilarFunctionalityAs: Indicates that two requirements have similar functionality if they apply to the same condition and suggest closely related actions (redundancy); **hasConflictingOutcomeWith**: Indicates that two requirements are conflicting if they apply to the same condition but suggest opposing actions (contradiction).

4.2 SWRL Rules Definition

To automatically infer redundancy and conflicts, we use SWRL (cf. Subsection 3.2.4) rules.
Rule 1 (Inferring *hasSimilarFunctionalityAs*): This rule infers that two requirements have similar functionality if they share the same condition and require similar actions:

```
Requirement(?r1) ^ Requirement(?r2) ^
hasCondition(?r1, ?c1) ^ hasCondition(?r2, ?c2) ^
conditionType(?c1, ?type) ^ conditionType(?c2, ?type) ^
hasAction(?r1, ?a1) ^ hasAction(?r2, ?a2) ^
```

```
actionType(?a1, ?actionType1) ^ actionType(?a2, ?actionType2)
^ swrlb:equal(?actionType1, ?actionType2)
-> hasSimilarFunctionalityAs(?r1, ?r2)
```

Example: R1 (“go on green”) and R4 (“move quickly on green”) share the same condition (green light).

Rule 2 (Inferring *hasConflictingOutcomeWith*): This rule identifies conflicts between two requirements that share the same condition but require opposing actions:

```
Requirement(?r1) ^ Requirement(?r2) ^
hasCondition(?r1, ?c1) ^ hasCondition(?r2, ?c2) ^
conditionType(?c1, ?type) ^ conditionType(?c2, ?type) ^
hasAction(?r1, ?a1) ^ hasAction(?r2, ?a2) ^
actionType(?a1, ?actionType1) ^ actionType(?a2, ?actionType2)
^ swrlb:notEqual(?actionType1, ?actionType2)
-> hasConflictingOutcomeWith(?r1, ?r2)
```

Example: If R1 and R2 both applied to the same condition (e.g., “yellow light”), and one required cars to go while the other required cars to stop, the reasoner would infer a conflict.

4.3 Inferences & Implications

The ontology reasoner performs two services: (i) it checks the ontology consistency, (ii) it infers new knowledge. In our scenario, it detects redundancy and inconsistency inference such as:

R1 and R4 (*Similar Functionality*)

Condition : “Green light”;

Actions : Both involve allowing cars to move, but R4 is more specific (move quickly);

Inferred Relationship : R1 and R4 are flagged as having similar functionality (*redundancy*).

R1 and R4 have similar functionality, then R1 or R4 can be deleted to eliminate redundancy.

R3 and R5 (*Potential Contradiction*)

Condition : If both requirements applied to the same light (e.g., “yellow light”), they conflict, as R3 says “slow down” and R5 says “stop”.

Inferred Relationship : R3 and R5 conflict because they apply to same conditions and require different actions.

Then, the reasoner can explain where the conflicts come from.

In brief, the different implications are:

1. **Requirement Redundancy:** R1 and R4 both apply to the green light and allow cars to move forward, though one is more specific. They are flagged as having similar functionality (*redundancy*).
2. **Requirement Conflict:** R3 and R5 conflict because they apply to same light condition (“yellow light”). Since one instruction is to “slow down” and the other is to “stop”, there is a conflict (i.e. *contradiction*).
3. **System Flexibility:** The system requirements’ domain-specific ontology and rules are designed to be flexible. As new requirements are added, the system automatically checks for redundancy or inconsistency using the defined SWRL rules.
4. **Automatic Inference:** The reasoner Pellet[6] used in this scenario can infer relationships such as *hasSimilarFunctionalityAs* and *hasConflictingOutcomeWith*. This helps system engineers quickly identify and address potential issues in system requirements.

For instance, explanations axioms are represented with Manchester Syntax[7] which is an ontology user-friendly syntax for OWL ontologies description and development.

R1 *hasSimilarFunctionalityAs* **R4**:

```
- hasCondition Domain Requirement
- R1 hasAction Action_Go
Requirement(?r1) ^ Requirement(?r2) ^
- hasCondition(?r1, ?c1) ^ hasCondition(?r2, ?c2) ^
conditionType(?c1, ?type) ^ conditionType(?c2, ?type) ^
hasAction(?r1, ?a1) ^ hasAction(?r2, ?a2) ^
actionType(?a1, ?actionType1) ^ actionType(?a2, ?actionType2)
^ swrlb:equal(?actionType1, ?actionType2)
-> hasSimilarFunctionalityAs(?r1, ?r2)
- Condition_Green conditionType "green light"
- R4 hasAction Action_Go
- Action_Go actionType "go"
- R4 hasCondition Condition_Green
- R1 hasCondition Condition_Green
```

R3 *hasConflictingOutcomeWith* **R5**:

```
- R5 Type Requirement
- Action_Stop actionType "stop"
- Action_SlowDown Type Action
- Condition_Yellow conditionType "yellow light"
- R3 Type Requirement
- Requirement(?r1) ^ Requirement(?r2) ^
hasCondition(?r1, ?c1) ^ hasCondition(?r2, ?c2) ^
conditionType(?c1, ?type) ^ conditionType(?c2, ?type) ^
hasAction(?r1, ?a1) ^ hasAction(?r2, ?a2) ^
actionType(?a1, ?actionType1) ^ actionType(?a2, ?actionType2)
^ swrlb:notEqual(?actionType1, ?actionType2)
-> hasConflictingOutcomeWith(?r1, ?r2)
- R5 hasAction Action_Stop
- R5 hasCondition Condition_Yellow
- R3 hasAction Action_SlowDown
- Action_Stop Type Action
- Action_SlowDown actionType "slow down"
```

[6] www.github.com/stardog-union/pellet - visited 22/10/2024

[7] www.w3.org/TR/owl2-manchester-syntax/ - visited 22/10/2024

5 DISCUSSION

This research provides a logical answer to all the research issues stated in Section 1. The first research question on system requirements extraction approach or technique is addressed in Subsections 3.2.1 and 3.2.2, by POS tagging and syntactic dependencies analysis (NLP Layer). The second research question regarding the way of using ontologies to federate specification document knowledge and system requirements is answered in Subsection 3.2.3, by defining an ontology-based framework for textual system requirements extraction and analysis support. For the third research question, the resulting ontology enables logical reasoning to check the ontological requirements model and perform inferences to detect ambiguities, redundancies incompleteness and inconsistencies in requirements boilerplates, as detailed in Subsection 3.2.4. We have applied a traffic light system scenario to better illustrate the reasoning-based part about the resulting ontology of system requirements (cf. Subsection 3.2.3). The resulting explanations seems to be clear and explicitly describe the reasons behind scenario requirements redundancy and contradiction. Additionally, we can verbalize the logical axioms produced to obtain a more seamless representation, enabling system engineers to understand ontological axioms and support them during the system requirements analysis phase.

6 CONCLUSION AND PERSPECTIVES

The proposed framework is still in implementation phase, which is why we have not been able to reveal any measurable or qualitative evaluation results by engineers that would enable us to fully evaluate it.

In future work, we aim to improve requirements extraction using the entire COOPANS specification and enhance the ontology consistency by reasoning.

REFERENCES

Bittner, T., Donnelly, M., and Winter, S. (2005). Ontology and semantic interoperability. In *Large-scale 3D data integration*, pages 139–160. CRC Press.

Bossche-Marquette, M. V., Guizol, L., and Brouster, R. L. (EasyChair, 2024). Ontologies and semantic rules in real life. EasyChair Preprint 15236.

Castañeda, V., Ballejos, L., Caliusco, M., and Galli, M. (2010). The use of ontologies in requirements engineering. *Glob J Res Eng*, 10.

Dori, D. (2016). *Model-based systems engineering with OPM and SysML*. Springer New York, NY.

Fahad, M., Qadir, M. A., and Shah, S. A. H. (2008). Evaluation of ontologies and dl reasoners. In Shi, Z., Mercier-Laurent, E., and Leake, D., editors, *Intelligent Information Processing IV*, pages 17–27, Boston, MA. Springer US.

Fuchs, N. E., Kaljurand, K., and Schneider, G. (2006). Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces. In *FLAIRS 2006*.

Glimm, B. and Kazakov, Y. (2019). Classical algorithms for reasoning and explanation in description logics. In *Reasoning Web*.

IEEE (2018). Iso/iec/ieee international standard - systems and software engineering – life cycle processes – requirements engineering. *ISO/IEC/IEEE 29148:2018(E)*, pages 1–104.

IEEE (2023). Iso/iec/ieee international standard - systems and software engineering–system life cycle processes. *ISO/IEC/IEEE 15288:2023(E)*, pages 1–128.

Mejdoul., Z. and Lortal., G. (2022). Gluon: A reasoning-based and natural language generation-based system to explicit ontology design choices. In *Proceedings of the 14th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2022) - KEOD*, pages 228–236. SciTePress.

Mokos, K., Nestoridis, T., Katsaros, P., and Bassiliades, N. (2022). Semantic modeling and analysis of natural language system requirements. *IEEE Access*, 10:84094–84119.

Pohl, K. and Rupp, C. (2015). *Requirements Engineering Fundamentals*. Rocky Nook Inc., San Rafael, CA, 2nd edition.

Rousseau, D., Wilby, J., Billingham, J., and Blachfellner, S. (2016). A typology for the systems field. *Systema: connecting matter, life, culture and technology*, 4(1):15–47.

Roussey, C., Pinet, F., Kang, M. A., and Corcho, O. (2011). *An Introduction to Ontologies and Ontology Engineering*, pages 9–38. Springer London, London.

Sarder, M. B. and Ferreira, S. (2007). Developing systems engineering ontologies. In *2007 IEEE International Conference on System of Systems Engineering*, pages 1–6.

Sillitto, H. (2011). Sharing systems engineering knowledge through incose: Incose as an ultra-large-scale system? *INSIGHT*, 14:20–22.

Stachtiari, E., Mavridou, A., Katsaros, P., Bliudze, S., and Sifakis, J. (2018). Early validation of system requirements and design through correctness-by-construction. *Journal of Systems and Software*, 145:52–78.

Studer, R., Benjamins, V., and Fensel, D. (1998). Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1):161–197.

Yang, L., Cormican, K., and Yu, M. (2019). Ontology-based systems engineering: A state-of-the-art review. *Computers in Industry*, 111:148–171.