Alma Mater Studiorum - Università di Bologna

DOTTORATO DI RICERCA IN

COMPUTER SCIENCE AND ENGINEERING

Ciclo 35

Settore Concorsuale: 01/B1 - INFORMATICA

Settore Scientifico Disciplinare: INF/01 - INFORMATICA

SUPPORTING REQUIREMENT ELICITATION AND ONTOLOGY TESTING IN KNOWLEDGE GRAPH ENGINEERING

Presentata da: Fiorela Ciroku

Coordinatore Dottorato	**Supervisore**
Ilaria Bartolini	Valentina Presutti

Esame finale anno 2023

Abstract

Knowledge graphs and ontologies are closely related concepts in the field of knowledge representation. A knowledge graph is a knowledge base that uses a graph-structured data model or topology to integrate data. In recent years, knowledge graphs have gained increasing popularity and are serving as essential components in many knowledge engineering projects that view them as crucial to their success. The conceptual foundation of the knowledge graph is provided by ontologies. An ontology is a formal, explicit specification of a shared conceptualization within a domain, providing a blueprint for the organization of information. The process of developing the associated ontologies is addressed by well-known methodologies. Ontology modeling is an iterative engineering process that consists of steps such as the elicitation and formalization of requirements, the development, testing, refactoring, and release of the ontology. The testing of the ontology is a crucial and occasionally overlooked step of the process due to the lack of integrated tools to support it. As a result of this gap in the state-of-the-art, the testing of the ontology is completed manually, which requires a considerable amount of time and effort from the ontology engineers. The lack of tool support is noticed in the requirement elicitation process as well. Long-established methods of collecting requirements, such as interactions with domain and application experts, investigation of datasets, and relevant literature, although highly functional, are performed manually. The rise in the adoption and accessibility of knowledge graphs allows for the development and use of automated tools to assist with the elicitation of requirements from such a complementary source of data. Moreover, despite the relevance of the requirements and the testing in the ontology development process, their documentation is often not complete and at times nonexistent, which impacts the reuse of the ontology. Therefore, this doctoral research is focused on developing methods and tools that support the requirement elicitation and testing steps of an ontology engineering process. To support the testing of the ontology, we have developed XDTesting, a web application that is integrated with the GitHub platform. The application is able to create, annotate, execute, and document a test case, and graphically summarize the testing status of the ontology. XDTesting is considered an acceptable, grade B, excellent tool according to the System Usability Scale questionnaire performed during the first evaluation. Concurrently, to support the elicitation and documentation of competency questions, we have defined and implemented RevOnt, a method to extract competency questions from knowledge graphs by firstly abstracting the triple verbalization, secondly generating questions, and lastly filtering competency questions. The elicited competency questions can be used to understand the knowledge graph itself, to drive the development of an ontology, and to provide input for the testing of the ontology. The implementation using data from the Wikidata knowledge graph has been evaluated, and the results show that the automatic abstraction of the triple verbalization and the generation of competency questions have a good to a high-quality based on the BLEU score. To conclude, the automation of the ontology testing process through XDTesting facilitates the workload of the ontology engineer in terms of time and effort. While the automation of an approach for eliciting requirements from knowledge graphs, RevOnt, is achievable and performs well with the help of language models.

Acknowledgements

I would like to express my deepest appreciation to my supervisor Valentina Presutti. Thank you so much for being an exceptional supervisor. Your guidance and support throughout the entire process was invaluable and made a significant impact on the quality of my work. Your expertise, patience, and encouragement helped me navigate the challenges and allowed me to bring my research to a successful conclusion. I am deeply grateful for your unwavering support and encouragement, and I feel honored to have had the opportunity to work with you. Thank you for being a true mentor and for having a profound impact on my life.

***

I would like to give my sincere thanks to Albert Meroño-Peñuela, Jacopo de Benardinis and Elena Simperl for the warm welcome you extended during my visit at King's College London. The opportunity to meet with you, learn about your research, and work together was an incredible and memorable experience. Your passion and dedication for your work were inspiring and made a lasting impression on me.

***

I would like to extend my gratitude to Prof. Paolo Ciancarini and Prof. Andrea Omicini for following my research throughout the doctorate. Your insights and suggestions were invaluable in helping me improve the quality of my work. Your dedication to the scholarly community is an inspiration and I am honored to have had the opportunity to receive your feedback.

***

I would like to take a moment to express my sincere appreciation to the STLab group. Your commitment to the scientific community and to advancing knowledge through research was evident in all of our interactions, and I am grateful for the support and feedback that you provided. Your dedication to quality, accuracy, and attention to detail has been a source of inspiration.

***

I would like to acknowledge the support from the Polifonia project group. Your collaboration and dedication to the success of the project were truly inspiring and I am honored to have had the opportunity to work alongside such talented and dedicated individuals. I am grateful for the lessons I have learned from each of you.

***

My dear family, your unwavering love, support, and encouragement throughout my doctoral journey were invaluable. Your belief in me and my abilities gave me the strength and motivation to keep going, even during the most challenging times. Your unwavering commitment to my success has meant the world to me and I could not have done it without you. Thank you!

***

I would like to dedicate this work to myself, as a show of strength and endurance beyond personal limits. During these last three years, I have been strong and weak, brave and coward, proud and humble, bright and gloomy, dedicated and neglectful, but most importantly, I have been me. We did it!

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Knowledge graphs are quickly becoming a standard for knowledge management. In recent years, there has been a change in the landscape of knowledge engineering projects. A decade earlier, such projects terminated with the development of the ontology, and the *population* of the ontology was an after-project task. Nowadays, an increasing number of knowledge engineering projects (e.g. Polifonia[1], SPICE[2], ArCo[3]) consider knowledge graphs the key to the project. In other cases, knowledge graphs are the starting point, making the development of ontologies take place after the creation of the knowledge graph. Most methodologies [1, 58, 76, 39, 59, 63] describe ontology development as a complex and iterative engineering process that generally consists of steps such as elicitation and formalization of requirements, development, testing, refactoring, and release of the ontology. Testing is the step of the process that assures the quality of the ontology and, indirectly, its knowledge graph. The testing of the ontology is an essential and often underestimated aspect of the process. In contrast to software testing practices, ontology testing is limited in terms of integrated tool support and occasionally protocols [23]. The techniques used for ontology testing vary based on the methodology used, the stage of the ontology's development when they are applied, and the layer and role that they test [34]. For instance, eXtreme Design (XD) is an ontology design methodology that defines an abundantly descriptive protocol for the testing of the ontology [63].

[1]`https://polifonia-project.eu/`

[2]`https://spice-h2020.eu/`

[3]`http://wit.istc.cnr.it/arco/`

The protocol dictates the creation, execution, and documentation of different types of tests that assess whether a requirement of the ontology has been implemented, if the inferences retrieved from a reasoner are correct, and if the ontology is able to handle erroneous data. Considering the lack of tools to support the process, the testing of the ontology is handled manually, which requires a considerable amount of time and effort from the ontology engineers. Another aspect to consider is that many methodologies delineate the development of the ontology as an iterative process. This indicates that the ontology engineer needs to rerun the tests every time a new development is integrated into the ontology to assure that the results are correct. The workload associated with this scenario may lead to testing being performed after the development of the ontology has been completed rather than as a parallel process with the modeling. Such an action might impede the detection of modeling issues during the development phase and cause further work for ontology engineers. This situation is not dependent on whether the ontology is developed before or after the knowledge graph.

The lack of tool support is noted in the requirement elicitation process as well. The requirements are mainly represented in the form of *Competency questions (CQs)* or *General constraints (GCs)*. The competency questions play an important role in the ontology development life-cycle, as they represent the ontology requirements and are criteria for the evaluation of the ontology [8, 63]. Generally, the requirements are collected by interacting with domain and application experts, investigating datasets, and consulting relevant literature. As noted from the elicitation techniques, the process is carried out manually, and similarly to the testing, it requires significant effort and time from the ontology engineers. A favorable aspect is the fact that the sources from which the ontology requirements are elicited can be extended. As mentioned above, since knowledge engineering projects also begin with knowledge graphs (at times even with collaboratively built knowledge graphs such as Wikidata [89] and DBPedia [7]), it provides an opportunity to design a bottom-up approach rather than the typical top-down approach. Thanks to the availability of technologies such as language models, we can define and implement methods to extract competency questions from the knowledge graph.

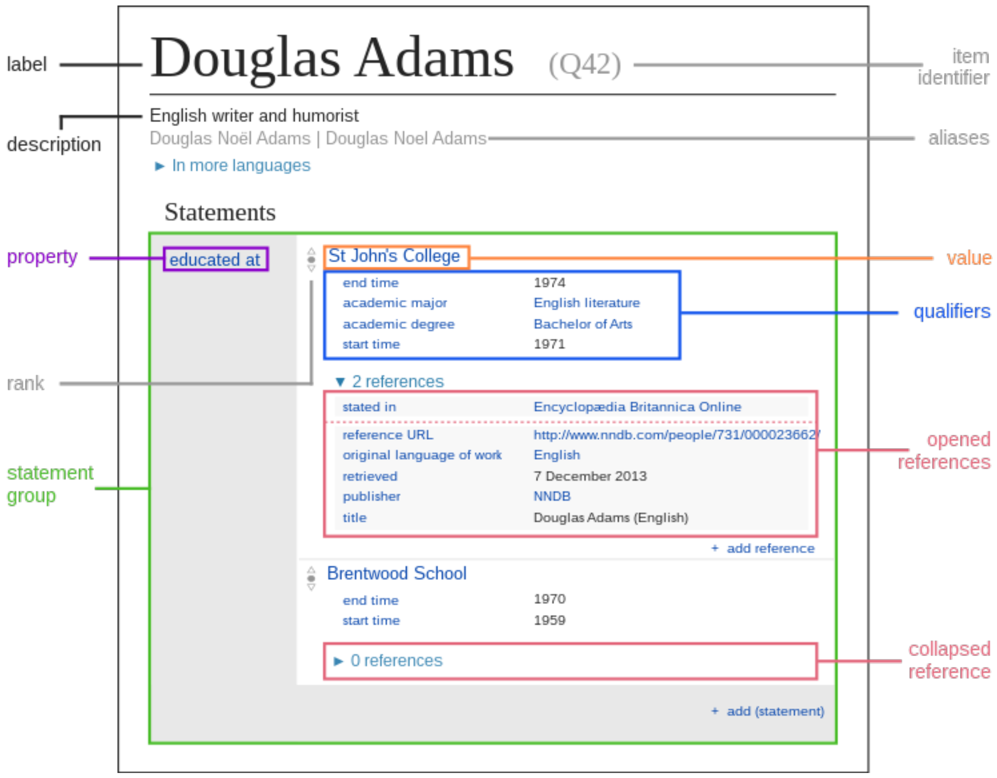
Moreover, besides representing a common understanding of a domain, an ontology's purpose is also to enable the reuse of the domain knowledge. The reuse of the ontology is highly dependent on the documentation, which optimally includes the competency questions, the design choices, the ontology, and the testing. Despite the relevance of the requirement elicitation and testing processes in the development of the ontology, their documentation is often not complete and at times non-existent. On the one hand, partial or missing documentation of the competency questions highly influences the testing of the ontology. In this scenario, the ontology engineers will not be able to fully test the ontology, given that they don't have documentation of what knowledge the ontology represents. On the other hand, partial or missing documentation of the testing may impact the reuse of the ontology because its quality cannot be guaranteed to an external party.

Considering the negative impact that issues such as lack of tool support and partial or missing documentation of the processes have on knowledge engineering projects, this doctoral research is focused on developing methods and tools that support the elicitation of the requirements needed to drive the development of ontologies, and the testing of ontologies. To support the elicitation of the requirements, we have defined RevOnt, an approach that enables the extraction of the competency questions from knowledge graphs. The knowledge graph that we use to experiment[4] and evaluate the method is Wikidata, which is a collaboratively-edited multilingual knowledge graph. It is a popular knowledge graph, it reflects common knowledge on many domains and provides resources that help to evaluate the approach, such as ID, label, description, and statements. The limitations of this use case remain in the fact that the knowledge graph is not domain-specific. An example of a data model in Wikidata is shown in Figure 1.1[5].

Meanwhile, to support the ontology testing, we have developed XDTesting, a tool for testing ontologies based on the eXtreme Design methodology. The tool is developed as a web application integrated with the GitHub platform. It offers numerous features that are in coherence with the testing methodology, mainly the annotation, creation, execution, and documentation of test cases. The work aims to open the path for competency question discovery in non-traditional sources and aspires to encourage the practice of ontology testing as a process that goes alongside ontology development practices.

[4]The implementation of the approach is found at `https://github.com/FiorelaCiroku/RevOnt`.

[5]Picture retrieved from `https://www.wikidata.org/wiki/Q16222597`



Douglas Adams (Q42)

English writer and humorist

Douglas Noël Adams | Douglas Noel Adams

► In more languages

Statements

educated at

St John's College

end time	1974
academic major	English literature
academic degree	Bachelor of Arts
start time	1971

▼ 2 references

stated in	Encyclopædia Britannica Online
reference URL	http://www.nndb.com/people/731/000023662/
original language of work	English
retrieved	7 December 2013
publisher	NNDB
title	Douglas Adams (English)

+ add reference

Brentwood School

end time	1970
start time	1959

► 0 references

+ add (statement)

Figure 1.1: A datamodel in Wikidata. The datamodel provides key information for a Wikidata item such as identifier, label, description, aliases, statements, and references.

The overview of the thesis is described below:

- The rest of **Chapter 1: Introduction** is structured as follows. Section 1.1 presents the motivation of the research with an example and an illustrated workflow description. Section 1.2 introduces the research questions with the respective hypotheses. In Section 1.3 we describe the methodology followed in this research work. Section 1.4 lists the contributions of the research and their potential impact in the field of ontology engineering.

- **Chapter 2: Related work** discusses related work in the fields of knowledge engineering, ontology learning, schema induction, requirement elicitation techniques, and cutting-edge ontology testing tools.

- **Chapter 3: Ontology engineering background** provides background knowledge of the eXtreme Design ontology development methodology, as well as a detailed understanding of the testing protocols.

- **Chapter 4: RevOnt - An ontology requirement elicitator** presents RevOnt, an approach for extracting competency questions from knowledge graphs. We discuss each step of the method with examples and describe the implementation choices.

- **Chapter 5: XDTesting - An ontology testing manager** presents the tool XDTesting, which manages the unit testing of ontology modules based on the XD methodology. The method and implementation of the tool are described in detail.

- **Chapter 6: Evaluation** reports the experimental setup and the results of the evaluation for the XDTesting tool and the RevOnt approach.

- **Chapter 7: Conclusions** discusses the limitations of the XDTesting and RevOnt, presents the planned future work, and concludes the research work.

1.1 Motivation

Generally, the process for engineering an ontology starts with the requirement elicitation phase. An initial set of requirements is formalized and prioritized by ontology engineers. Based on the priority given to the competency questions, ontology engineers develop an ontology in an iterative mode. Besides driving the development, the competency questions are simultaneously used for the testing of the ontology. When the ontology is built and tested, the engineers construct the respective knowledge graph. A synthesis of the main steps in an ontology development process is displayed in Figure 1.2.

Based on our experience with eXtreme Design, requirement elicitation is an engaging task that includes continuous interactions with experts and data investigation. In-person or online meetings with domain experts generally require substantial planning and time for the ontology engineer to thoroughly grasp the domain. Meanwhile, the current tool support, like Google forms and GitHub templates, is inadequate to provide a consistent framework for gathering information. The process still requires time and effort to analyze

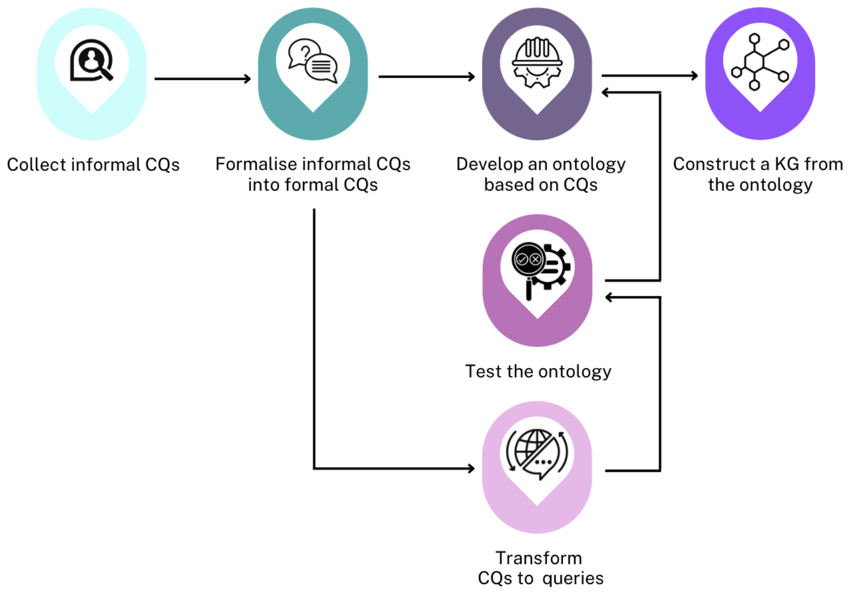

Collect informal CQs
Formalise informal CQs into formal CQs
Develop an ontology based on CQs
Construct a KG from the ontology
Test the ontology
Transform CQs to queries

Figure 1.2: A synthesis of the main steps in an ontology development process

the information, structure it, and comprehend the input. An illustrative example of the requirement collection procedure based on eXtreme Design comes from the Polifonia project. First, we asked domain experts to create stories. In the context of the project, a story is a template for collecting requirements that might include information about the persona, the goal, the scenario, competency questions, and resources, as described in detail by [12]. Considering that the domain experts are often not knowledge engineers, the competency questions that we received required a considerable amount of manual work to be transformed into formal competency questions. For instance, an informal competency question was *"In which historical documents is there evidence of a musical composition?"* [6]. This competency question was followed up by additional interaction with the domain experts to understand what the concept of evidence means to them, which in turn was defined as *"any direct linguistic sign that refers to concepts (name, or part of it)"*. This information was used to reformulate and formalize the competency question as follows *"Which historical document mentions a musical composition?"*. The newly formulated competency question follows an in-house standard for the vocabulary

[6]The complete story is found at `https://github.com/polifonia-project/stories`

used in the modeling of the overall ontology. This process is performed with most of the competency questions retrieved in this modality, which illustrates repeatedly that the requirement elicitation process lacks standardization, is time-consuming, and requires a significant amount of effort. The same techniques and constraints apply to extracting competency questions from data sets.

Moreover, based on XD, the requirements are used as input for different types of tests. Normally, in order to create a test case, the ontology engineer has to create a SPARQL query with its expected results and the dataset to run the test case against, using real-life or fictive data. This information is then used to fill in the template of the test case. Once the test case has been created, the engineer prepares the environment for the testing, which includes downloading a Java jar that enables the execution or usage of Protégé[7], an open-source ontology editor. The test case is executed, and its results are documented in a separate file. This complete process of creating, annotating, executing, and documenting the test case is done manually for each requirement, demonstrating the immense amount of effort required from the ontology engineers.

Thus, considering the above-mentioned issues, we have developed a support system that includes two main components, RevOnt, the requirement elicitation method, and XDTesting, the testing support tool. In Figure 1.3 we show how both of these components fit into the ontology development process and assist with specific steps. The entry point to the framework is the input, which is obtained from two different sources: the domain expert and a knowledge graph. When a domain expert provides a story or a dataset, it is the responsibility of the ontology engineer to break it down into competency questions and general constraints. Meanwhile, when the RevOnt method extracts competency questions from a knowledge graph relevant to the domain, the competency questions are already extracted from the source, formulated, and documented. Once the competency question and the general constraints are documented and prioritized, the process of ontology modeling can start. The ontology testing initiates when a set of requirements has been modelled into an ontology. In this step of the process, it is the implemented XDTesting method that deals with the management of the unit testing of the requirements. XDTesting, with the support of OWLUnit, creates, executes, and

[7]`https://protege.stanford.edu/`

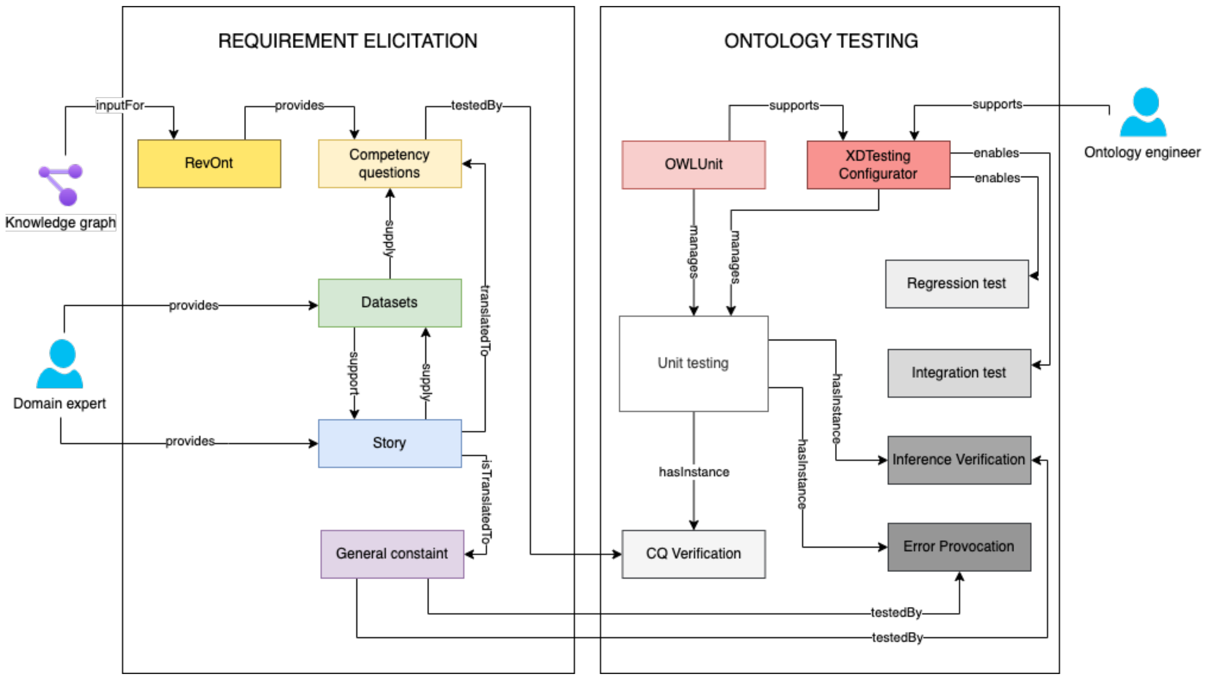


Figure 1.3: Overview of the component diagram for the eXtreme Design tool support system.

documents five types of test cases: Competency Question Verification test, Inference Verification test, Error Provocation test, Integration test, and Regression test. We describe each of the types of tests in Chapter 3, Section 3.3. This framework connecting two methods, one that extracts requirements from a knowledge graph and another assisting with the testing of the ontology, is an added value to the ontology engineering process. On the one hand, the RevOnt method provides a new source of requirements for the development of an ontology and consequently a knowledge graph. On the other hand, the XDTesting method reused the documented requirement provided by RevOnt to test the ontology.

1.2 Research questions

Motivated by the need for tool support for the knowledge engineering process, the opportunities that knowledge graphs offer in terms of application, and by leveraging recent advances in natural language processing, we raise the following research questions and formulate the respective hypothesis:

1. Can an approach for eliciting requirements from knowledge graphs be defined with the means of language models?

    - What are the quality features that a requirement must have in order to be classified as a competency question?
        - H1: The quality feature that an elicited requirement from a knowledge graph must show to be classified as a competency question is its ability to be transformed into a query.
    - What is the quality of the elicited requirements generated with means of language models from a knowledge graph?
        - H2: The quality of the elicited requirements that are generated with the means of language models from a knowledge graph is comparable to human-generated requirements.

2. Can the automation of the testing phase of an ontology engineering process reduce the workload of the ontology engineers/testers?

    - H3: The automation of the testing phase of an ontology engineering process can reduce the workload of the ontology engineer/tester.

1.3 Methodology

As mentioned above, this research is highly motivated by the gaps that are present in the state-of-the-art regarding the methodology and especially tool support for certain stages of an ontology engineering process such as requirement elicitation and ontology testing.

Concerning RQ1, the work is firstly focused on defining a method to extract requirements (competency questions) from a knowledge graph, independent of the latter, by using language models. Next in the line of work is the implementation of the method using a sample knowledge graph in order to evaluate the defined method. For the implementation, two very important tasks need to be performed. The first is an extensive literature review to determine the consensus on the concept of a competency question and any features that they show. This research offers a criterion on how to filter competency questions out of *simple* questions that are elicited from a knowledge graph and prove that what is being extracted from the knowledge graph are indeed competency questions. Secondly, the work includes an assessment of the quality of the elicited competency questions. This demonstrates that the defined method is able to automatically generate requirements that are comparable in quality to human-formulated ones.

With respect to RQ2, we speculate that the testing phase of an ontology engineering process is in great need of automated tool support. Therefore, the work includes a method defining how automation of the ontology testing tool support should work and which feature it must include in order to reduce the workload of an ontology engineer/tester; research on the state-of-the-art of collaborative platforms to understand what is already available in terms of automation and the necessity for automation in the platform; and lastly, an implementation of the method based on the fore-mentioned research. The evaluation of the implementation provides insight if the automation of the ontology testing phase has reduces the workload of the engineer and to what extent.

1.4 Contribution and impact

As described above, the objective of the research is to define methods and develop tools to support the requirement elicitation and testing phases of an ontology engineering process. Hence, the contribution of this research is:

1. *RevOnt*: an approach for automatically extracting competency questions from a knowledge graph and a dataset containing competency questions extracted from a knowledge graph and competency question templates mapped to SPARQL query templates.

2. *XDTesting*: a method for automating the ontology testing phase and an implementation in the form of a web application for managing ontology test case creation, annotation, execution, and documentation.

A direct application of the RevOnt method is being able to query the knowledge graph of origin in order to understand its content. Modern knowledge graphs are built using a combination of human and automated actions (e.g., information extraction from text, data input from databases). Usually, these actions are not well-documented and derive an ontology that is not adequately structured [60]. Therefore, understanding what competency questions a knowledge graph answers may support the development of its ontology, the knowledge graph's testing, documentation, and reuse. Another application of the method can be to support the knowledge engineering process. We speculate that the requirement elicitation and evaluation phases can be enhanced, both in terms of procedure, and in terms of the quality of the outcomes, with such a method. Regarding the requirement elicitation, the data from a knowledge graph can provide a different view of the domain of the ontology that is being constructed, complementary to the view provided by domain experts that are involved in the process or other sources. This additional information can improve the coverage of the domain and, in turn, of the ontology model. At the same time, these competency questions, if properly formulated and documented, can later be used as criteria for the evaluation of the ontology. Meanwhile, the XDTesting tool provides a well-integrated approach for facilitating the process of unit testing of the ontology requirements. Additionally, it offers automated, standardized documentation of the testing of an ontology, which might augment the reuse of the ontology.

Chapter 2

Related work

This chapter analyzes works that describe ontology engineering methodologies and tools that support the development process described in Section 2.1. In Section 2.2, we discuss the concept of competency questions and their role in the ontology engineering process. Section 2.3 describes methodologies that assist ontology testing and associated tools. Section 2.4 presents research in ontology learning and schema induction that is highly related to the work of extracting competency questions from knowledge graphs.

2.1 Ontology engineering methodologies and related tools

The first works in **ontology engineering** appeared in the early 1990s with projects such as TOVE by [25], Enterprise Model Approach by [20], METHONTOLOGY by [18], Ontolingua by [32], etc. These works intend to define methodologies that are closer to engineering practices than subjective modeling. An enhanced methodology in [81] and OntoEdit in [54] were presented in the early 2000s. After a decade, ontology engineering methodologies that include cyclic processes, continuous integration, and testing [1, 58, 76, 39, 59, 63] emerge and become a shared practice among engineers. Alongside many ontology engineering methodologies, there were also tools developed to assist with different ontology life-cycles. For instance, tools such as Protégé[1], OWLAPI[2], CENtree Ontology Manager[3], VocBench[4], OntoBee[5] support the engineering and management of ontologies. These tools serve as ontology editors, with capabilities such as running a reasoner, creating import modules, running reports, and various other tasks. Other tools such as ZOOMA[6], Prodigy[7], Ontotext[8], PoolParty[9] support the ontology annotation. They are offered in the form of web applications, APIs and Python packages. It is important to emphasise that most of the tools are specifically trained and cater to the medical domain. The tool OxO[10] is a web service that helps users locate mappings, also known as cross-references, between terms derived from ontologies, vocabularies, and coding standards. OxO imports mappings from a variety of sources, one of which is the Ontology Lookup Service[11]. This tool can be helpful in the process of ontology alignment. Lastly, OOPS[12] is a web application for detecting some of the most common pitfalls that appear when developing ontologies [62]. Interestingly, this tool signals pitfalls from a a catalog of the 40 most common issues in ontology modelings as creating polysemous elements, creating unconnected ontology elements, missing domain and range, etc. Although the feedback that the tool provides is very helpful, it is particularly related to the development of the ontology rather than the testing of it in specific. Certainly, this is not an exhaustive list of tools, as there are many other web applications, APIs, Java jars, and repositories that offer different functionalities related to the engineering, management, annotation, mapping, and matching of ontologies [55].

[1]`https://protege.stanford.edu/`
[2]`https://github.com/owlcs/owlapi`
[3]`https://www.scibite.com/platform/centree-ontology-management-platform/`
[4]`http://vocbench.uniroma2.it/`
[5]`https://ontobee.org/`
[6]`https://www.ebi.ac.uk/spot/zooma/`
[7]`https://prodi.gy/`
[8]`https://www.ontotext.com/`
[9]`https://www.poolparty.biz/`
[10]`https://www.ebi.ac.uk/spot/oxo/`
[11]`https://www.ebi.ac.uk/ols/index`
[12]`https://oops.linkeddata.es/`

2.2 Competency questions and their role in the ontology engineering process

The concept of competency questions, first defined in [33], is one of the common denominators among ontology engineering methodologies. According to the authors, each of the ontologies and microtheories (a necessary and sufficient set of axioms for describing and solving tasks [46]) is described using competency questions. The use of competence questions serves two functions: (1) they characterize the ontologies and microtheories that have been developed for each activity, and (2) they give guidance for the construction of new ontologies and microtheories. Meanwhile, the authors of [21], a work describing the generic enterprise resource ontology, define competency questions as the starting point for ontology creation; the needs for the ontology. The most apparent approach to showing competence is to establish a set of questions that the ontology can answer. Given a representation and a theorem prover, questions can be submitted in the form of queries that the theorem prover can answer. The authors in [87] support the same statement by concluding that competency questions may be beneficial in the later stages of assessing the ontology code and developing scenarios (microtheories/stories). Similarly, in [86], the authors state that the engineering of ontologies is motivated by scenarios that arise in the application, a principle that is also found in [63]. Furthermore, the notions of informal and formal competency questions are introduced. In [26], a competency question is defined as "a typical query that an expert might want to submit to a knowledge base of its target domain, for a certain task." In addition, the authors specify that a domain ontology should specify *all and only* the conceptualizations required in order to answer all the competency questions formulated by or acquired from experts. The author emphasizes in [88] that in order to enable automatic evaluation of competency questions, the competency questions must be formalized in a query language that can be used by the tool the ontology is developed for, posing a constraint on what type of property a question must have in order to be considered a competency question. In [69], the authors define a competency question as "a natural language sentence that expresses a pattern for a type of question people expect an ontology to answer". They consider competency questions to be invalid when they are redundant, incomplete, or ambiguous, going beyond the expressive power of a DL-based ontology language. To summarize, the role of a competency question is twofold: (1) to drive the modeling of the ontology by serving as a requirement; and (2) to assist in the evaluation of the ontology by being expressed as a query. This literature review regarding the concept of competency questions and their role in the ontology engineering process supports the significance of hypothesis H1 of the research. This hypothesis claims that the quality feature that an elicited requirement from a knowledge graph must have to be classified as a competency question is its ability to be transformed into a query. This criterion is used to evaluate the quality of the questions extracted from knowledge graphs with the RevOnt method.

2.3 Relevant ontology testing related tools

There are many ontology engineering methodologies that support the testing of ontologies. In [34], the authors provide a classification of methods and tools for the evaluation of ontologies for industrial practice, which gave us a basis for research about the field and a sense of classification of the tools that support the testing. Methods such as OntoMetric [43], OntoClean [31], EvaLexon [78] and tools such as ODEval [17], OntoManager [80], etc are described in the report. OntoMetric offers a series of procedures that the user should follow in order attain metrics of the adequacy of existing ontologies in relation to the requirements of a certain system. OntoClean focuses on taxonomy cleaning and is used to clean the upper level of the WordNet taxonomy. Meanwhile, EvaLexon's goal is to be simple enough for non-specialists to grasp, to be objective, automatable, and simply applicable to any text that represents an application domain and serves as input for the ontology mining/creation process. As for the tools, ODEval does syntactic evaluations of RDF(S), DAML+OIL, and OWL ontologies, as well as concept taxonomies evaluations from the standpoint of knowledge representation. Inconsistencies and redundancies in ontology concept taxonomies are detected using this tool. OntoManager is concerned with an ontology's truthfulness in relation to its target domain, regarding the question whether the ontology accurately represent a piece of reality and the users' requirements (users are end-users of ontology-based portals or applications).

Another survey of the state-of-the-art of ontology evaluation is presented in [13], where the authors classify the methodologies found into the following categories: those based on comparing the ontology to a “golden standard”, those based on using the ontology in an application and evaluating the results, those involving comparisons with a source of data about the domain to be covered by the ontology, and those where evaluation is done by humans who try to assess how well the ontology meets a set of predefined criteria, standards, and requirements. Similar classifications can be found in another survey by Raad in [64]. Meanwhile, in [27], the authors present a model that consists of a meta-ontology that characterizes ontologies as semiotic objects and an ontology of ontology validation called oQual that provides the means to devise the best set of criteria for choosing an ontology over others in the context of a given project. In more recent work, the authors in [24] present a web-based tool called Themis, independent of any ontology engineering environment, for validating ontologies by means of the application of test expressions that follow lexico-syntactic patterns. While, in [41] the authors introduce the approach of test-driven development (TDD) for ontology authoring. Their tool is implemented as a Protégé plugin so that one can perform a TDD test as a black box test. In another work, [45], the authors present the development of TDDOnto, which implements a subset of TDD tests.

Lastly, eXtreme Design is a test-driven approach to ontology engineering [10, 63, 9]. This methodology is aligned with the Software Testing Life Cycle, meaning that the process followed for the testing of an ontology is very similar in concept to the life cycle of a software testing process. XD follows a white box approach for the testing, similar to software testing. White box testing is particularly effective since it tests not only the functionality of the software but also the internal structure of the application [37]. Furthermore, the testing principles from software testing [75] are adopted into the eXtreme Design methodology. Such principles include testing a program/ontology to try to make it fail (similar to the Error Provocation Verification type of test in XD), starting testing early (the testing of the ontology is highly encouraged early in the development and the modular principle of the modeling is favorable to this aspect), define a test plan (the competency question are a good start to kick off the testing of the ontology), testing must be done by different persons at different levels (this is a crucial aspect of the XD methodology as is described in Chapter 3), etc. Despite the close alignment in methodology between software testing and ontology testing with eXtreme Design, the tool support for each field of work is very different. While software testing is supported extensively by tools [77], ontology testing is not in the same plan. The testing process of the eXtreme Design methodology is to some extent supported by three tools NeOn toolkit[13], TESTaLOD[14] and the OWLUnit jar[15].

In terms of testing and automated testing, NeOn recommends an iterative approach, stating that "knowledge acquisition, documentation, configuration management, *evaluation, and assessment*" should take place throughout the entire ontology network development, that is, "in any scenario used for developing the ontology network" [82]. Over the years, various NeOn plugins have been proposed for automating parts of ontology testing; for example, [11] proposes the XD Tool, which integrates with NeOn and offers support for SPARQL-based competency question verification. Yet, the tool is not maintained and accessible any longer.

TESTaLOD is a Web application that provides a testing toolbox for the XD methodology to support knowledge graph testing [15]. It leverages the TestCase OWL metamodel[16] as the standard schema for describing unit tests, as well as a way to validate ontologies and data commitments. The tool is a two-step workflow powered by a web-based user interface that allows a user to pick and execute an arbitrary number of defined test cases using the TestCase OWL meta model. A user must first provide one or more test cases as input in the first phase. Those test cases can be simply uploaded from a local file system or fetched from a Github repository. TESTaLOD gets all test cases available in a GitHub repository by recursively traversing the subfolders reachable from the repository root when the user selects it. After all of the test cases have been retrieved, the user is presented with a view that allows them to choose whether or not to run all of the test cases in the repository. When files are uploaded from a local file system, however, all uploaded local files are tested directly. Both options provide a visual representation of the output of the automated execution of the selected test cases. The execution of a test case with TESTaLOD can result in one of three outcomes: (i) The

[13]`http://neon-toolkit.org/wiki/Main_Page.html`

[14]`https://github.com/TESTaLOD/TESTaLOD`

[15]`https://github.com/luigi-asprino/owl-unit`

[16]`http://www.ontologydesignpatterns.org/schemas/testannotationschema.owl`

test case is fully successful, and the corresponding record in the user interface is green; (ii) The test case is partially successful (the test results do not match completely the expected results), and the corresponding record in the user interface is yellow; (iii) The test case is fully unsuccessful, and the corresponding record in the user interface is colored red. To conclude, TESTaLOD is a tool that provides support for the testing process by executing test cases that have already been constructed by the ontology testers. To our knowledge, TESTaLOD has not seen active development in the last two years and does not currently provide certain features that would ameliorate the testing process, such as not being able to upload files or access GitHub repositories. Furthermore, it does not provide support for the creating of the test cases, their annotation and the documentation of their execution. All these task are highly relevant and more importantly time-consuming.

Lastly, OWLUnit is a Java jar that allows the ontology tester to run unit tests for ontologies defined according to the OWLUnit Ontology[17]. OWLUnit runs four kinds of test cases: annotation verification, competency question verification, inference verification, and error provocation verification. For the annotation verification test, it verifies that the tested ontology complies with its default ontology shape[18]. Alternatively, you can define your shape and specify it by using the `owlunit:hasShapes` property. While for the competency question verification test, OWLunit makes sure that: 1. the IRIs used within the SPARQL query are defined either in the tested ontology or in the input test data (if provided); the IRIs that don't meet this condition are printed in the console; 2. If input data is provided, the result of the SPARQL unit test query evaluated over the input data is isomorphic to the expected result. The expected result can be specified either as a JSON serialization of the result set of the query or according to this vocabulary[19]. Moreover, you can also test multiple ontologies at a time. For the inference verification test, OWLunit makes sure that: 1. the tested ontology is consistent; 2. If input data is provided, the ontology and input data together don't lead to any inconsistencies; and 3. If a SPARQL unit test is provided, the result of the SPARQL unit

[17]`https://raw.githubusercontent.com/luigi-asprino/owl-unit/0.2.0/ontology/ontology.owl`

[18]`https://raw.githubusercontent.com/luigi-asprino/owl-unit/main/shapes/ontology.ttl`

[19]`https://www.w3.org/2001/sw/DataAccess/tests/result-set#`

test is equivalent to the expected result. Lastly, for the error provocation verification test, OWLunit makes sure that ontology and input data are inconsistent together. The drawback of OWLUnit is the fact that it is *just* a Java jar, so its use is not the easiest process for a non-technically savvy tester. Moreover, the jar provides very general feedback regarding the failure of a test case execution, making it very difficult to understand what the cause of the failure is.

Mostly all the above-mentioned works describe methodologies for testing and briefly refer to any tool support that exists to support them. We emphasize that, to our knowledge, the literature on the state-of-the-art primarily refers to dated tools, some of which are no longer actively maintained, such as OntoManager[20] or, in the worst case, are not accessible online, such as ODEVal[21]. Meanwhile, the current tool support is either not integrated with developing platforms (e.g. GitHub, GitLab) or it assists with only one specific task, and does not work as an overall manager for the testing of the ontology. So, to perform the testing of an ontology, an engineer might need to use several tools for different types of tests, adding to their workload more time and effort. A strong emphasis is put on the fact that besides TESTaLOD and OWLUnit, all the other tools do not support the unit testing of the ontology. This enforces, even more, our motivation to support the ontology testing process with tools and guarantee a certain level of ontology quality. XDTesting, our proposed method to support ontology testing, offers a complete testing manager for unit testing of an ontology, including the automatic creation of a test case, preparation of the execution environment, the execution of the test case, and the annotation and documentation of the execution. Besides the execution of the test case, all the other tasks are not currently supported by other tools, ergo are performed manually. In addition, XDTesting is integrated with an external platform such as GitHub.

[20]https://github.com/IvS-KULeuven/OntoManager
[21]`http://minsky.dia.fi.upm.es/odeval`

2.4 Relevant techniques from ontology learning and schema discovery

In consideration of the method and techniques that are used in the definition of an approach to extract competency questions from knowledge graphs, relevant related fields of research are ontology learning, and schema induction and discovery. **Ontology learning** is a multidisciplinary field that extracts terms, concepts, properties, and relationships from unstructured text using approaches from several disciplines such as knowledge representation, natural language processing, machine learning, etc. Surveys in ontology learning, such as [5], classify ontology learning techniques into three categories: linguistic, statistical, and logical. *Linguistic techniques* are based on language features and are commonly used for data preparation (speech tagging, parsing, and lemmatization) as well as various other ontology learning tasks such as knowledge extraction. Prime examples of such techniques are Text2Onto[22], CRCTOL[23]. *Statistical techniques* rely entirely on statistics from the underlying corpus and overlook the underlying semantics. The majority of statistical approaches make substantial use of probabilities and are commonly used in the stages of ontology learning following linguistic preprocessing. Relevant tools that use statistical techniques are OntoGain[24], OntoLearn[25], ASIUM[26]. Lastly, *Inductive logic programming* is a machine learning discipline that employs logic programming to generate hypotheses based on prior knowledge and a set of examples. Significant tools are Syndikate[27] and TextStorm[28].

Meanwhile, surveys such as [38] and [42] present an overview of the state-of-the-art in **schema induction** and discovery, a research field dealing with the extraction or discovery of semantic schemas from unstructured or semi-structured data [28]. The research is motivated by the fact that the data in the semantic web, whether expressed in RDF or JSON, is not based on a predefined schema. The most widely used techniques

[22]http://neon-toolkit.org/wiki/1.x/Text2Onto.html
[23]http://nlp.cs.berkeley.edu/
[24]https://github.com/Neuw84/CValue-TermExtraction
[25]https://github.com/dice-group/Ontolearn
[26]http://www-ai.ijs.si/ilpnet2/systems/asium.html
[27]http://pyke.sourceforge.net/
[28]https://dwijottam-dutta.github.io/TextStorm/about/about.html

in the schema discovery approaches are machine learning (classification, clustering, and frequent pattern mining) and formal techniques (Formal Concept Analysis, bisimulation). The surveys discuss valuable works regarding implicit and explicit schema discovery approaches by taking into consideration the target problem, techniques, features, input, output, and quality aspects.

Relevant surveys such as [61] and [16] describe **ontology summarisation** approaches that use centrality metrics (e.g., PageRank) to identify the most informative concepts or nodes or extract important subgraphs to facilitate query-testing for verifying requirements against accessible data. In contrast, recent research related to the extraction of Common Conceptual Components[29] from multiple ontologies using Ontology Design Patterns[30] is presented in [6]. The authors present a method that employs a non-extractive method to assist in the comprehension and comparison of different ontologies. Starting with a corpus of ontologies, it uses community detection, word sense disambiguation, frame recognition, and clustering to automatically produce a catalogue of conceptual components and observable ontology design patterns. Further, in [53], the authors present a study for discovering Encyclopedic Knowledge Patterns (EKP)[31] from Wikipedia[32] page links. The patterns, according to the authors, may be used as lenses for exploring DBpedia[33] or for developing new ontologies that inherit the data and textual grounding offered by DBpedia and Wikipedia. Data linking can also benefit from EKPs by modularizing the datasets to be linked.

Our research contributes to the field of knowledge engineering by enhancing the requirement elicitation and ontology testing processes. The extraction of competency questions from a knowledge graph supports ontology learning tasks since the approach retrieves terms and relations with the help of natural language processing models and machine learning. In addition, RevOnt can support the extraction of a schema from a knowledge graph through its abstraction stage, where the natural language verbalisation

[29]A conceptual component (CC) is a complex (cognitive) relational structure that a designer implements in an ontology by using classes, properties, axioms, etc.

[30]`http://ontologydesignpatterns.org/wiki/Main_Page`

[31]EKPs are Knowledge Patterns that are grounded in encyclopedic knowledge and expressed as linked data and as natural language text.

[32]`https://en.wikipedia.org/wiki/Main_Page`

[33]`https://www.dbpedia.org/`

of a triple is abstracted from the instance level to the class level. In terms of the XDTesting tool, we claim that it can fill a gap in the state-of-the-art and provide a useful solution for ontology testing management.

Chapter 3

Ontology engineering background

The chapter covers the theoretical background of eXtreme Design, the ontology modeling methodology on which we have founded the development of the XDTesting tool. Section 3.1 describes the principles of the methodology and the role of each phase. Section 3.2 presents examples of techniques for eliciting requirements based on XD. Lastly, Section 3.3 focuses on the testing protocol of the methodology.

3.1 The eXtreme Design methodology

eXtreme Design is an ontology design methodology that puts the reuse of Ontology Design Patterns (ODPs) at its core, both as a principle and as an explicit activity. The main characterizing principles of the method are the intensive use of ODPs, the modular design, and the test-driven approach [63]. Ontology Design Patterns address recurring modeling issues. They are ontologies that serve as a bridge between use cases (problem types) and design solutions. They are used as modeling components: an ontology should ideally be the result of a composition of ODPs with appropriate dependencies between them, as well as the necessary design expansion based on specific needs [26]. ODPs can be found in catalogs, such as The Ontology Design Patterns Portal[1], The workshop on Ontology Design Patterns series[2], and The University of Manchester catalogue[3].

[1]`http://ontologydesignpatterns.org/wiki/Main_Page`

[2]`http://ontologydesignpatterns.org/wiki/WOP:Main`

[3]`http://www.gong.manchester.ac.uk/odp/html/`

The principle of modular design consists of separating the modeling of the requirements into independent, interchangeable modules. Each of the modules contains everything required to perform only one component of the desired requirement. In addition, XD is focused on unit testing of the ontology, making it analogous to software testing, while other ontology modelling methodologies include tests that have a more semantic nature. Considering these aspects of the methodology and the fact that it is based on a test-driven approach, it is a great foundation for the development of a tool to manage the testing.

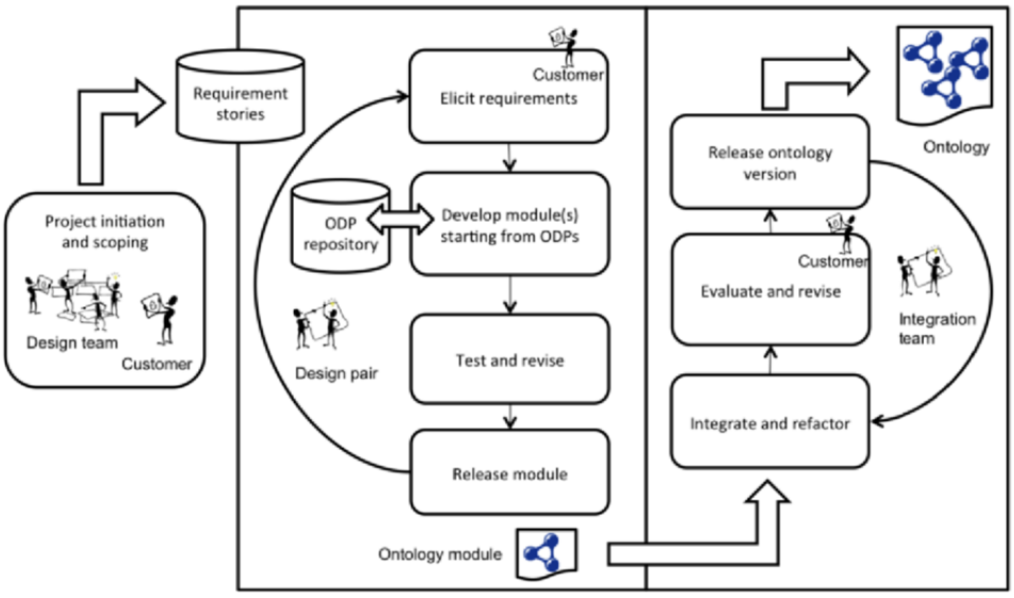


Figure 3.1: An overview of the eXtreme Design framework

An overview of the eXtreme Design framework is shown in Figure 3.1, retrieved from [9]. In this process, there are several actors that operate in different phases. There is the *customer team*, including domain and application experts, that provides and defines the requirements for the ontology. Another team is the *design team* which is comprised of ontology engineers that develop the ontologies. The *testing team*, a separate team from the design team, assists with the testing of the ontology module. The *integration team*, a group of ontology engineers, deals with the integration and refactoring of the ontology. In [9], the authors describe the process of developing an ontology with the following steps:

1. **Collect requirements**: Domain experts on the customer team elicit requirements that drive the design and testing processes. While the design team manages the design process by defining and executing the Ontology Design Patterns that best meet the requirements, the design team generalizes the user stories in collaboration with the customer team, outlining the primary principles for eliciting the CQs.

2. **Formalize requirements**: The customer team is directed to break down potentially complex stories into smaller, easier ones. The design team, in collaboration with the customer team, establishes general constraints and formulates one or more competency questions from the generalization of user stories.

3. **Develop ontology module**: The design team investigates existing ontology design patterns that may be able to answer the same CQ collected from the requirement collection. ODP's CQs are frequently more general than an ontology project's domain-specific CQs. In this situation, the designer will further generalize the CQs to determine whether the candidate ODP can be reused, and then specialize its properties and/or classes.

4. **Test and revise**: The testing team decides how to test each requirement. The testing includes the definition of the test case, the creation of test data, and the determination of expected results. Then, they execute the test run OWL-file with its test data and record the results. Once the testing team has completed the testing of the ontology module, the module is revised if needed.

5. **Release**: If there are no more revisions, the ontology module is released with the proper annotations, examples, and documentation.

6. **Integration and refactoring**: The integration team deals with the integration of the newly released ontology module into the whole ontology. To ensure that the new ontology module has not caused inconsistencies or other issues, the team needs to evaluate the ontology after the integration. Then, it revises the ontology based on the results of the evaluation. Lastly, they release the ontology.

3.2 Requirement elicitation techniques

The requirements in eXtreme Design are collected from different sources in the form of user stories. “A user story is a set of sentences that describes, by example, the kind of facts that the knowledge graph is required to encode.” [63]. The design team, in cooperation with the customer team, extracts requirements, Competency questions and General constraints from the user stories. There are several techniques that can be used in the interaction with the customer team, such as structured and unstructured interviews, introspection, document and protocol analysis, ethnographic techniques, etc. Structured interviews are interviews that have a set of definite questions that are asked in a specific order, whereas unstructured interviews are interviews that have neither the question nor the category of the answer predefined, but rather rely on social interaction [96]. Introspection offers the possibility to “imagine” the ideal outcome based on the needs and the means of the experts [29]. Document analysis consists of consulting the available documentation or literature for the knowledge that is to be represented. A protocol analysis requires a person to perform a task while also talking aloud about his or her cognitive process. Lastly, ethnographic techniques are beneficial when addressing contextual aspects like usability and when researching collaborative work scenarios where understanding interactions between multiple users with the system is critical [36]. To collect requirements, a technique or a combination of techniques is chosen based on the domain. Below is presented a user story[4] provided by the domain experts of the Polifonia project with the means of an asynchronous interview technique.

> Frank regularly comes together with his friends from church. They want to plan a little weekend trip to the other side of the country and go to the church mass on Sunday in the city nearby where they are staying. To prepare for the visit, he wants to learn about the church and the organ. In general, Frank wants to find out more about the history of the church, but also what the similarities are between how the organ he usually plays and the one he will listen to sound. For this, Frank is primarily interested in the disposition and

[4]`\textbf{https://github.com/polifonia-project/stories/blob/main/Frank:\%20Organist/Frank\%231_OrganKnowledge.md}`

who built the organ.

The competency questions extracted from the user story are shown in Table 3.1.

ID	Requirement
CQ1	Who built and/or renovated an organ?
CQ2	What is the disposition of the organ at a specific point in time?
CQ3	What are the art-historical features of the front of the organ?
CQ4	What are the decorative elements of the organ?

Table 3.1: A list of competency questions extracted from a user story in the Polifonia project

3.3 Ontology testing spectrum in eXtreme Design

Considering that eXtreme Design is a test-driven methodology, its testing protocol is quite descriptive and clear. The methodology includes different types of tests described below [10].

- **Competency Question Verification test** allows verifying if the ontology can answer the competency questions that have been collected during the requirement collection.
- **Inference Verification test** allows verifying that the inference mechanisms are in place, to ensure the correct fulfillment of the inference requirement.
- **Error Provocation test** allows verifying how the ontology acts when it is fed random or incorrect data.
- **Integration test** allows verifying if the integration of an ontology fragment into the ontology module has been successful.
- **Regression test** allows verifying if the integration of an ontology fragment to the ontology module has been successful network-wide, i.e., if it has caused any inconsistencies relating to other modules that import the module that was updated.

In Listing 3.1, 3.2 and 3.3 are displayed the templates for the test cases of the Competency Question Verification, Inference Verification and Error Provocation tests respectively.

```
@prefix owlunit: <https://w3id.org/OWLunit/ontology/> .
@prefix on: <https://w3id.org/myOntology/> .
@prefix ex: <https://w3id.org/myOntology/example> .

ex:xx a owlunit:CompetencyQuestionVerification ;
      owlunit:hasCompetencyQuestion " " ;
      owlunit:hasSPARQLUnitTest " " ;
      owlunit:hasInputData ex:xx ;
      owlunit:hasInputTestDataCategory owlunit:ToyDataset ;
      owlunit:hasExpectedResult " ";
      owlunit:testsOntology on: .
```

Listing 3.1: Template of a Competency Question Verification test

```
@prefix owlunit: <https://w3id.org/OWLunit/ontology/> .
@prefix on: <https://w3id.org/myOntology/> .
@prefix ex: <https://w3id.org/myOntology/example> .

ex:xx a owlunit:InferenceVerification ;
      owlunit:hasInputData ex:xx ;
      owlunit:hasSPARQLUnitTest " " ;
      owlunit:hasReasoner owlunit:HermiT ;
      owlunit:hasExpectedResult true/false ;
      owlunit:testsOntology on: .
```

Listing 3.2: Template of an Inference Verification test

```
@prefix owlunit: <https://w3id.org/OWLunit/ontology/> .
@prefix on: <https://w3id.org/myOntology/> .
@prefix ex: <https://w3id.org/myOntology/example> .

ex:xx a owlunit:ErrorProvocation ;
      owlunit:hasInputData ex:xx ;
      owlunit:testsOntology on: .
```

Listing 3.3: Template of an Error Provocation test

To formulate and execute the test cases, the testing team is assigned in pairs for each module. The protocol that is followed for the testing of the ontologies based on XD is presented in Table 3.2. As seen in the figure, the team needs to gather all relevant requirements for a specific type of test and select them one by one for testing. Then, they have to determine how to test it, e.g., with a SPARQL query. When the testing procedure is formulated, the ontology testers create a test case, add the test data, and determine the expected results of the test. With the test case ready, the ontology testers are ready to execute it. Later, they compare the actual result of the test case with the expected one. If they are not the same, the test case needs to be analyzed, and any changes should be documented. If the actual result matches the expected result, the test case is documented using the test meta-model. The documentation of a test case execution must include the test case ID, the test category, the requirement, the test, the input test data, the expected result, the actual result, the execution time, the execution environment, the execution result, and comments. If there are any requirements left from the module, the process is iterated.

Step	Task	Task description
1	Gather requirements	For a specific type of test, retrieve all the requirements of the current module that are relevant to this type of test.
2	Select requirement	Following the principle of unit testing, select one requirement to test of each test case.
3	Formulate test procedure	Determine how to test that particular requirement.
4	Create test case	Create the test case OWL-file, and an additional OWL-file for storing the first test run and describe both using the test case metamodel and its properties.
5	Add test data	Add the test data, needed to perform the procedure according to step 3, in the test run OWL-file.
6	Determine expected results	Depending on the test data, what would be the output of a correct test run?
7	Run test	Execute the procedure from step 3 on the test run OWL-file with its data from step 5, and record the result.
8	Compare results	Verify the expected output (step 6) against the actual result (step 7).
9	Analyze unexpected result	If the result is not the expected one, analyze and document any change suggestions or issues.
10	Document	Store all information about the test run and its related test case by using the properties of the test metamodel.
11	Iterate	If there are more requirements of this module to test, return to step 2.

Table 3.2: An overview of the testing protocol in eXtreme Design

Chapter 4

RevOnt - An ontology requirement elicitator

Chapter 4 presents RevOnt, an approach for extracting competency questions from knowledge graphs. Section 4.1 provides an overview of the RevOnt method with a description of each of the steps. In Section 4.2 we describe in detail the implementation of the RevOnt method. In Subsection 4.2.1 introduces the WDV dataset that is used for the implementation of the approach. Later, Subsections 4.2.2, 4.2.3, and 4.2.4 explain respectively the Verbalization Abstraction, Question Generation, and Question Filtration steps of the framework with examples.

4.1 RevOnt method

RevOnt is a framework for extracting competency questions from a knowledge graph. As depicted in Figure 4.1, the framework is divided into three stages: 1) Verbalization Abstraction, 2) Question Generation, and 3) Question Filtration, each with specific objectives.

The purpose of the *Verbalization Abstraction* stage is to transform the verbalization from the instance level to the class level. For example, given the triple verbalization *"Michael Jackson is a member of the Michael Jackson discography."*, without the abstraction, the generated questions are *"Who is a member of the Michael Jackson*

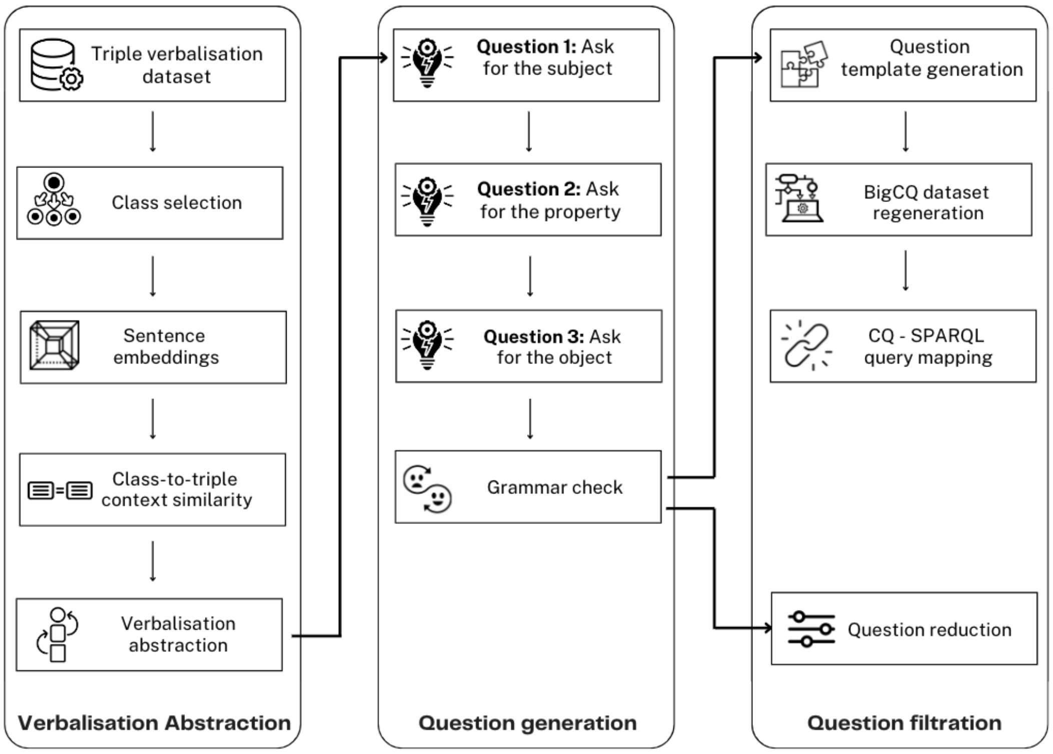

Triple verbalisation dataset
Class selection
Sentence embeddings
Class-to-triple context similarity
Verbalisation abstraction
Verbalisation Abstraction
Question 1: Ask for the subject
Question 2: Ask for the property
Question 3: Ask for the object
Grammar check
Question generation
Question template generation
BigCQ dataset regeneration
CQ - SPARQL query mapping
Question reduction
Question filtration

Figure 4.1: An overview of the RevOnt framework. The first stage, Verbalization Abstraction, generates the abstraction of a triple verbalization. The abstraction is used as input in the second stage, Question Generation, to generate three questions per triple and perform a grammar check. Lastly, the third stage, Question Filtration, filters the questions by performing different techniques.

discography?" or *"What is Michael Jackson a member of?"*. These questions are not competency questions because they ask for specific instances, and not classes or properties. Thus, there is a need to abstract the verbalization into a more general form. To complete the task, the *Verbalization Abstraction* stage begins with a dataset, which contains the verbalizations of triples from a knowledge graph, as an input. The dataset should include data about the subject, predicate, and object of the triple, descriptions of the instances, IDs that align them to a knowledge graph; and most importantly, the verbalization of the triple. This information is necessary for the selection of the class in which the subject and the object of the triple is an instance of or a subclass of.

For the purpose of abstracting the triple verbalizations from the instance level to the class level, we designed a novel method to abstract the subject and object of a triple to the most similar-to-context Wikidata classes. The procedure is shown in Algorithm 1. Firstly, the algorithm creates the sentence embedding of the description of the subject and the object. Next, it retrieves the classes that the subject/object is a instance of or a subclass of. After the classes are retrieved, for each class it gets the related sunsets. For each synset, it gets the definition of the synset and creates its sentence embedding. Lastly, the cosine similarity between the embedding of the synset definition and the embedding of the subject/object description. The algorithm returns the most similar synset. This synset is chosen as the most similar-to-context class where the subject/object belongs.

Algorithm 1 Abstraction of a triple

```
 1: procedure Triple verbalization abstraction
 2:     Create the sentence embedding of the subject description
 3:     Create the sentence embedding of the object description
 4:     Retrieve the Wikidata classes
 5:     for each class do
 6:         Get the synsets
 7:         for each synset do
 8:             Get the synset definition
 9:             Create the sentence embedding of the synset definition
10:             Calculate the cosine similarity between the synset definition
    and the descriptions
11:     return Most similar synset
```

The second stage of the framework is *Question Generation*. The aim of this stage is to generate three questions for each triple verbalization. This choice of design is made to understand the types of questions that are generated when the method asks the model about different parts of the verbalization sentence. Essentially, this stage generates questions based on the abstraction of the triple verbalization, as shown in the example above. Part of this stage is a grammar check task that corrects errors that are found in the questions.

Lastly, the *Question Filtration* stage deals with the quality check and reduction of the questions generated in the previous step. The two main tasks of this stage are 1) mapping question templates to query templates, and 2) reduction of questions based on their similarity. The task of mapping question templates to query templates addresses the question *"When is a question a competency question?"*. As discussed in Section 2, the answer is that a competency question must be used to query the ontology. Therefore, such a mapping would confirm whether the generated question is indeed a competency question. While the question reduction task filters redundant questions by searching for semantically similar or exact competency questions. This stage is needed to identify a set of core competency questions from the whole set of questions that were generated in the previous stage.

4.2 RevOnt implementation

In this section, we describe the implementation details of the RevOnt method. In Table 4.1 we show a list of Natural Language Processing (NLP) models, modules, datasets, and services used for the implementation of the framework, their category, and the stage where they are used.

Model	Category	Stage
WDV	Dataset	Input
MiniLM	LM	verbalization abstraction
Wikidata query service	Service	verbalization abstraction
T5	LM	Question generation
T5 Grammar Correction	LM	Question generation
BigCQ	Dataset	Question filtration

Table 4.1: A list of the language models, modules, datasets and services used in the RevOnt framework

4.2.1 The WDV dataset

There are several datasets that provide verbalizations of data such as the T-REx[1] and WDV dataset[2] for Wikidata entries, WebNLG[3] for DBPedia[4] entries, NYT-FB by [51] and FB15K-237 by [84] for Freebase[5], and so on.

A first implementation of the RevOnt method was achieved for Wikidata, by leveraging the WDV dataset [4]. This dataset provides verbalizations of Wikidata claims and contains 7.6K unique triples. According to [3], WDV has considerably more entity types and predicates than comparable datasets, and it is intended to serve as a benchmark dataset for data verbalization models used on Wikidata. WDV enables a tight coupling between single claims and text by directly connecting a triple-based claim to a natural language phrase. The WDV dataset contains verbalizations of claims from 20 different themes (or domains), from the most common like artist, sport teams, university to celestial body, chemical compounds, taxon, et cetera. The diversity of triple verbalizations in the dataset contributes to interesting results and imposes challenges as well. An example of a verbalization from the *Artist* theme is shown in Figure 4.2.

4.2.2 Verbalization Abstraction

The first stage of the RevOnt framework is the *Verbalization Abstraction*. Its role is to generate an abstraction of a triple verbalization. To perform the abstraction, it is necessary to recognize and categorize the entities present in the verbalization. The initial intuition for this task was to use a Named Entity Recognition (NER) model. We experimented with language models such as Camembert-ner[6], Camembert-base-multilingual-cased-ner-hrl[7], Ner-english-large[8], Bio-Ner[9] and the `SpaCy` library. The performance of each of these models was not satisfactory for 50% of the themes in the dataset. For

[1]`https://hadyelsahar.github.io/t-rex/`
[2]`https://github.com/gabrielmaia7/WDV`
[3]`https://gitlab.com/shimorina/webnlg-dataset`
[4]`https://www.dbpedia.org/`
[5]`https://developers.google.com/freebase`
[6]`https://huggingface.co/Jean-Baptiste/camembert-ner`
[7]`https://huggingface.co/Davlan/bert-base-multilingual-cased-ner-hrl`
[8]`https://huggingface.co/flair/ner-english-large`
[9]`https://github.com/librairy/bio-ner`

```
{
  "claim_id": "q2831$7404E524-E58B-4C48-AAA2-DF3F6C36FFB9",
  "rank": "normal",
  "subject_id": "Q2831",
  "property_id": "P358",
  "subject_label": "Michael Jackson",
  "property_label": "discography",
  "object_label": "Michael Jackson discography",
  "subject_dec": "American recording artist; singer and songwriter (1958-2009)",
  "property_desc": "item for list pages with discography of artist or band",
  "object_desc": "Wikimedia artist discography",
  "subject_alias": [
    "MJ",
    "The King of Pop",
    "Michael Joe Jackson",
    "Michael Joseph Jackson",
    "M.J.",
    "M. J.",
    "King of Pop",
    "King of Music",
    "The King"
  ],
  "property_alias": [
    "discography link",
    "recording catalog"
  ],
  "object_alias": "no-alias",
  "object_datatype": "wikibase-item",
  "object": {
    "value": {
      "entity-type": "item",
      "numeric-id": 259448,
      "id": "Q259448"
    },
    "type": "wikibase-entityid"
  },
  "theme_root_class_id": "Q483501",
  "theme_label": "Artist",
  "verbalisation": "Michael Jackson is a member of the discography.",
  "verbalisation_unk_replaced": "Michael Jackson is a member of the discography.",
  "sampling_weight": 245.4,
  "annotations": {
    "fluency_scores": [
      4,
      5,
      5,
      2,
      4
    ],
    "fluency_mean": 4.0,
    "fluency_median": 4.0,
    "adequacy_scores": [
      0,
      0,
      2,
      0,
      0
    ],
    "adequacy_majority_voted": 0.0,
    "adequacy_percentage": 0.8
  }
},
```

Figure 4.2: A claim verbalization from the WDV dataset. For the claim, the dataset provides the ID, rank, theme, and verbalization of the claim. There are present also the label, description, aliases and ID of the subject, property and object of the triple.

example, in themes like Celestial body, Astronaut and Chemical compound, any of the fore mentioned language model had issues with identifying and categorizing entities in questions such as *"What is the orbital eccentricity of Tau1 Gruis b?"*, *"What is the name of the device that encodes Hsa-mir-1180?"*, *"Where is PR00122 found?"*.

Based on the these observations, we defined Algorithm 1, shown in Section 4.1. For each triple, the method extracts the sentence embeddings of the descriptions of the subject and the object using the MiniLM language model[10]. MiniLM is a sentence-level transformer model [71] that maps sentences to a 384-dimensional dense vector space. By leveraging a pre-trained language model[11], MiniLM is fine-tuned on a number of datasets[12] using a contrastive objective. Intuitively, a contrastive loss is used to minimize the cosine similarity between similar sentence pairs while maximizing that of other sentences in the same batch.

In the next step, Revont selects the English label of Wikidata classes where an entity is an instance of or a subclass of. The query that we use for selecting this information from the knowledge graph is shown below. For this task, we use the Wikidata Query Service[13]. Wikidata Query Service is an implementation of a SPARQL server, based on Blazegraph[14] engine, to service queries for Wikidata and other data sets.

```
SELECT DISTINCT ?cLabel
WHERE {{
    wd:{id} wdt:P31/wdt:P279? ?c .
    ?c rdfs:label ?cLabel .
FILTER(LANG(?cLabel) = "en") }}
```

For instance, for the triple verbalization *"Michael Jackson is a member of the Michael Jackson discography."*, according to the dataset shown in Figure 4.2, the subject is *Michael Jackson* and the object is *Michael Jackson discography.* The respective descriptions are *American recording artist; singer and songwriter (1958-2009)* and *Wikimedia*

[10]https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[11]https://huggingface.co/nreimers/MiniLM-L6-H384-uncased

[12]https://www.sbert.net/docs/pretrained_models.html

[13]https://query.wikidata.org/

[14]https://github.com/blazegraph/database/wiki/Main_Page

artist discography. RevOnt extracts the sentence embeddings of the subject and object description and queries the Wikidata knowledge graph to select the classes of the subject and object of the triple. The result for the subject is shown in Example 4.2.1.

Example 4.2.1. *The Wikidata classes for the subject "Michael Jackson"*

```
['human', 'natural person', 'omnivore', 'person',
'mammal', 'Homo sapiens']
```

Once RevOnt retrieves the classes, it gets the corresponding WordNet synsets for each class. Wordnet is a large electronic lexical database for English [22]. Cognitive synonyms (synsets) are groups of nouns, verbs, adjectives, and adverbs that each communicate a separate notion. Synsets are linked together via conceptual-semantic and linguistic relationships. In Example 4.2.2 we present the respective synsets of the classes where *"Michael Jackson* is an instance of/subclass of. For each synset, it retrieves the definition and computes its sentence embeddings using the MiniLM model.

Example 4.2.2. *The synsets of the Wikidata classes*

```
{[Synset('homo.n.02')], [],
[Synset('omnivore.n.01'), Synset('omnivore.n.02')],
[Synset('person.n.01'), Synset('person.n.02'),
Synset('person.n.03')],  [Synset('mammal.n.01')], []}
```

In the last step, RevOnt calculates the cosine similarity between the definition of the synsets of each class and the description of the subject or object. The cosine similarity is calculated with the help of the Natural Language Toolkit (NLTK). NLTK is a suite of Python modules providing many NLP data types, processing tasks, corpus samples, and readers, together with animated algorithms, tutorials, and problem sets [47]. For the example shown above, the Wikidata class that is the most similar to the description of the subject *Michael Jackson* is *human.* As for the object of the triple, the Wikidata class is *discography.*

The values that the algorithm returns populate a Python dictionary that is used for the abstraction task. More specifically, the subjects/objects and their corresponding most similar-to-description class are added to a dictionary, as shown in Example 4.2.3.

Example 4.2.3. *The pattern dictionary*

```
{'Michael Jackson': 'human',
'Michael Jackson Discography': 'discography'}
```

This dictionary is used to replace the subject and the object in a triple with the respective class. To continue with the same example above, the result of the verbalization abstraction is presented in Example 4.2.4. This concludes the first stage of the framework, and the abstracted verbalizations are passed on to the second stage, Question generation.

Example 4.2.4. *The abstraction of the verbalization*

```
Verbalization: Michael Jackson is a member of the
               Michael Jackson discography.
Abstraction:   Human is a member of the discography.
```

4.2.3 Question Generation

After the abstraction of the verbalization, the next stage of the approach is to generate questions. For this task, we have used the Text-to-Text Transfer Transformer (T5) [73] fine-tuned on the Stanford Question Answering Dataset (SQuAD) [66] for question generation. This was achieved by prepending the answer to the context. The specific T5 model instance is `t5-base-finetuned-question-generation-ap` [15]. Notably, [74] demonstrated how the model can be applied successfully to generative tasks such as abstractive summarization, classification tasks such as natural language inference, and even regression tasks.

To generate questions, the model requires as input a context (sentence) and an answer. When an answer is not provided, the model will generate a question that is answered by the object of the sentence. For each triple, we have provided the model with the abstraction of the verbalization as the context, and three answers: the class of subject, the property, and the class of the object. In most observed cases, when the class of subject and class of the object are the answers, the questions that are generated are inverse. As for the cases when the property is the answer, the questions that are generated are often

[15]`https://huggingface.co/mrm8488/t5-base-finetuned-question-generation-ap`

the same as the ones when the answer is the class of the object, but there are also many cases when the questions are quite interesting and not so straightforward. This behavior of the model can be explained by the distance between the triple and the verbalization. As noticed in the example below, while the property of the triple is `discography`, this property is not present in the verbalization. This is the case for the majority of the verbalizations present in the dataset. In Example 4.2.5, we show the results of the Question Generation stage given the abstraction from Example 4.2.4.

Example 4.2.5. *Question generation*

```
Context: Human is a member of the discography.
Answer 1: human
Question 1: What is a member of the discography?
Answer 2: is a member of
Question 2: What is the relation between a human and a discography?
Answer 3: discography
Question 3: What is a human a member of?
```

Once the questions are generated, RevOnt performs a grammar check to detect and fix errors. This task is performed by the T5 Grammar Correction model[16]. Trivially, the model generates a revised version of the given text with the goal of addressing grammatical errors. It is trained with Happy Transformer[17] using the JFLEG dataset [52].

4.2.4 Question Filtration

The third and last stage of RevOnt consists in the filtration of the questions that are generated from the second stage. To address **H2**, regarding the quality of the extracted questions from a knowledge graph, we filter the questions that are generated from the abstraction of the verbalization with the help of the BigCQ dataset[18] by [93].

[16]https://huggingface.co/vennify/t5-base-grammar-correction
[17]https://github.com/EricFillion/happy-transformer
[18]https://github.com/dwisniewski/BigCQ

The dataset contains a set of Competency Question templates paired with SPARQL-OWL query templates, and it has been built upon previous work, CQ2SPARQLOWL by [94]. The method and dataset in CQ2SPARQLOWL are a benchmark of schema-level Competency Questions and corresponding SPARQL-OWL patterns. Therefore, in this task, we select the questions that are mapped to SPARQL queries, proving that they are indeed competency questions. To perform this task, we need to transform questions into templates. The transformation is shown in Example 4.2.6, where EC1 and EC2 are entity chunks.

Example 4.2.6. *The transformation from question to question template*

```
Question: What is a member of the discography?
Question template: What is EC1 of EC2?
```

The next step is to add the new templates to the BigCQ dataset and regenerate the dataset. The result of the regeneration is a file with a list of unique competency question templates, a file with a list of unique SPARQL-OWL query templates, and a folder with mappings from SPARQL-OWL templates to CQ templates. Each file is a JSON document containing a different SPARQL-OWL query template that follows the schema in Example 4.2.7.

Example 4.2.7. *Question template to SPARQL query template mapping*

```
{
    'query': 'SPARQL_OWL_QUERY',
    'cqs': ['CQ1', 'CQ2', ...]
}
```

From the new dataset, we are able to select the competency questions which have templates mapped to SPARQL queries. A present SPARQL query template assures that the extracted question from the knowledge graph is indeed a competency question. In conclusion, this stage filtrates only competency questions from the pool of the generated questions.

Chapter 5

XDTesting - An ontology testing manager

Chapter 5 present XDTesting, a tool developed to provide support for the testing of an ontology based on the eXtreme Design methodology. Section 5.1 provides an overview of the method, and Section 5.2 describes the requirement collection process and design choices that were made for its implementation. Section 5.3 presents the XDTesting method in detail with the means of the artifacts that were produced during the inception phase. Section 5.4 discusses the iterations of the development and describes each of the features that are implemented. Lastly, Subsections 5.4.1 and Subsection 5.4.2 presents the GitHub workflow and the XDTesting Configurator, a web application that serves as the interface of the tool.

5.1 Managing ontology testing with XDTesting

XDTesting is a method and tool created to support ontology testing based on the eXtreme Design methodology. The tool is implemented as an automated workflow on the GitHub platform and is comprised of numerous features that are in line with the testing methodology. The features are identified from personal experience, interactions with fellow ontology engineers and testers, and also from investigations of state-of-the-art tools and literature. To enhance the usability of the tool, we have developed an interface in the form of a web application named XDTesting Configurator, described in Section 5.4.2.

As described in Chapter 3, eXtreme Design is an ontology modeling methodology based on the intensive use of ODPs, modularity, and the test-driven approach. XDTesting is a realization of the test-driven approach of the methodology. Furthermore, the modularity principle has been incorporated into XDTesting by allowing ontology testers to create an ontology module and an ontology fragment that belongs to the module. Once the testing of the ontology fragment is complete, the tool integrates it with the ontology module and runs the necessary tests. An illustration of an ontology module and its fragments is shown in Figure 5.1. In this figure, the ontology module is the Instrument ontology module from the Polifonia Ontology Network. The fragments are the ontology design patterns such as AgentRole, Project, Parthood, and TimeInterval.

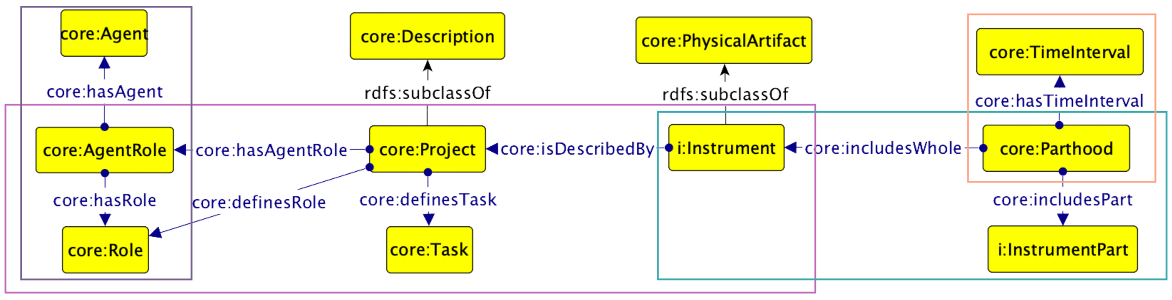


Figure 5.1: An illustration of an ontology module with its fragments.

5.2 Requirement collection and surveys

In this section, we describe how we collected the initial set of requirements for the refinement of the existing tool support, OWLUnit, and the request for new features. In addition to the requirement collection, we present two small-scale surveys that we conducted to assess the state-of-the-art of testing tool support and the need for testing automation in the GitHub platform.

To collect requirements for the testing automation, we had a brainstorming meeting with researchers from STLab[1], a laboratory dedicated to the representation and processing of knowledge. The participants are researchers that are specialized in ontology engineering and testing based on eXtreme Design and have experience with the OWLUnit jar. The aim of the meeting was to discuss the improvement of the current tool support and new features that are necessary to ease the testing process. The outcome of the meeting resulted in the identification of the following action points.

1. The tool should enable the testing of an ontology module under development and not published under the official URI of the ontology.
2. The tool should enable the automatic rerun of all test cases when a competency question, SPARQL query, or expected result is updated.
3. The tool should enable the addition of an annotation property or a datatype property to specify the version of the test case.
4. The tool should enable the provision of detailed information in the response messages to understand why the tests fail.[2]
5. Create a test to check whether a SPARQL query, that is necessary to execute Competency Questions Verification and Inferences Verification tests, is executable.

After identifying these requirements, we investigated the possibility of using GitHub actions for the automation of the testing process. As standard practice, we surveyed existing actions and apps that are available in the GitHub marketplace. The actions and apps that we took into consideration for analysis are under the *testing* category. A selection criterion besides the category is also the certification of the creators, which guarantees a certain quality of the tools. In total, in the GitHub marketplace, there are 1329 actions and apps under the testing category, but merely 109 are verified by Github. The information that we retrieved for each action and app are the URL, action name, description, creator, and review. After analyzing each of the 109 tools, it was concluded

[1]`http://stlab.istc.cnr.it/stlab/`

[2]Currently the information that the ontology tester receives in case of failed tests are orderly Java errors or highly general messages that make it difficult to pinpoint the problem in the test case or dataset.

that there are no actions or apps that provide support for ontology testing. Investigating beyond these restrictions, we searched for uncertified actions or apps in the Marketplace, and still, none support ontology testing.

To justify our decision to implement tool automation in GitHub, we conducted another survey to determine the presence of repositories containing ontologies or tools for interacting with ontologies in GitHub. We searched the keyword *ontology* on GitHub, and as a result, we got 11,065 repositories. From these 11,065 repositories, we investigated the Top 900 ranking of *Best matched* results with the purpose of categorizing them to assess the percentage of repositories that store ontologies. We classified the repositories into two categories: 1) repositories that store ontologies; and 2) repositories of tools for handling, editing, and visualizing ontologies. The categorization was done manually based on the description of the repository. If the description is missing, the categorization was done based on the name of the repository, e.g., /example-ontology, or the tag of the repository, e.g., ontology. As for the description of the repository, to categorize it under the ontology repository, we took into consideration phrases such as ontology of, *example* ontology, ontology for, ontology to describe. While for the categorization under the tool repository, we searched for phrases such as tool, package, visualization, code, manager, editor, etc. Based on the sample under consideration, which is roughly 10% of the total results, it resulted in approximately 66% being repositories that store ontologies and 34% repositories of tools for handling, editing, and visualizing ontologies. Ergo, we decided to develop from scratch an automation, comprised of several actions, in GitHub to provide tool support for the testing of the numerous ontologies that are stored on this platform.

Considerably, there are many online repositories and libraries, such as BioPortal[3], Ontology Design Patterns [4], OLS[5], Obo-Foundry[6], Ontobee[7], etc, whose purpose is to store ontologies of a multitude of domains. Ontologies are also stored locally on private servers, public websites, and collaborative spaces. In contrast to these repositories,

[3]`https://bioportal.bioontology.org/`
[4]`http://ontologydesignpatterns.org/wiki/Main_Page`
[5]`https://www.ebi.ac.uk/ols/index`
[6]`https://obofoundry.org/`
[7]`https://www.ontobee.org/`

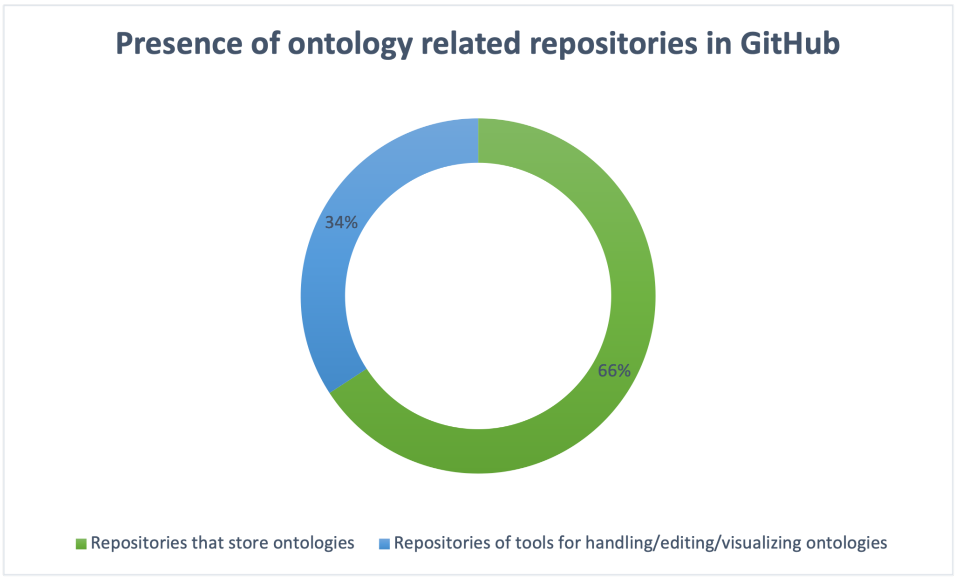


Figure 5.2: A donut chart representing the presence of ontology related repositories in the GitHub platform.

GitHub offers features that enable the possibility of having a tool support that can reduce the workload of ontology testers instead of building APIs and Java jars to interact with the foregoing repositories.

5.3 XDTesting method

In the present section, we describe the XDTesting method which aims to reduce the workload of ontology engineers in the testing phase of an ontology engineering process. The method is described using the following artifacts: a use case diagram, a workflow diagram, and a component diagram. It stands to reason that the workflow diagram that we will describe in this section is heavily based on the use case diagram in Figure 5.3. The XDTesting method is then implemented as described in Section 5.4.

The artifact depicted in Figure 5.4 is a diagram illustrating the workflow of the XDTesting tool automation based on XD methodology. As shown in the diagram, there are two actors that interact within the workflow and perform tasks that are in line with the use case diagram. The automation is triggered to start whenever an ontology

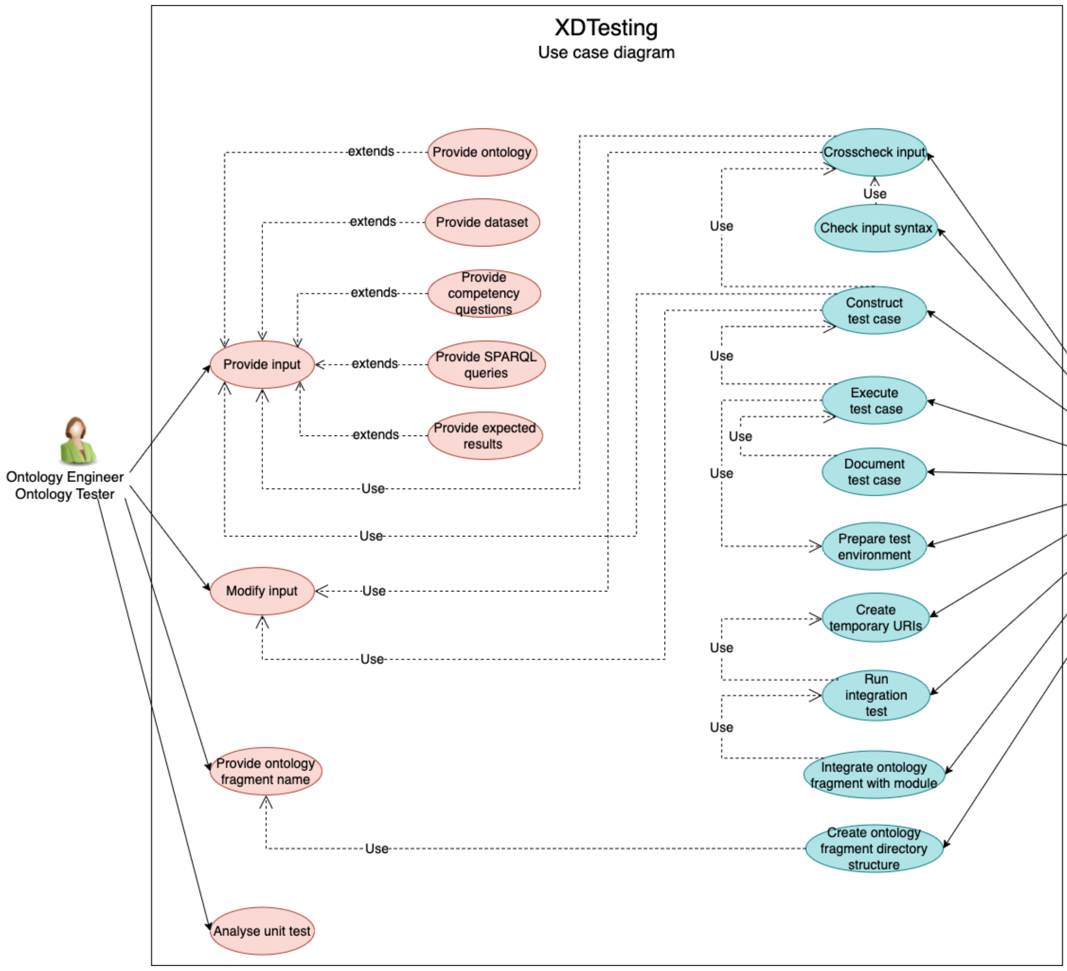


Figure 5.3: An illustration of the use case diagram of the XDTesting tool

engineer or tester needs an ontology fragment to be tested. To do so, he/she adds the ontology module, the ontology fragment, and test case data. This action triggers the GitHub workflow, which pulls the data to the platform and stores it in a specific directory structure. Once this step is completed, the system checks if the input provided is sufficient for constructing a unit test case. If this is the case, it checks the syntax of the input, more specifically of the SPARQL query, and the expected results in JSON and the Turtle datasets. If not, it requires the complete data for the test case. If the syntax check is passed, the system constructs the unit test, prepares the testing environment, and executes it. If the syntax is not correct, then the system reports that the input

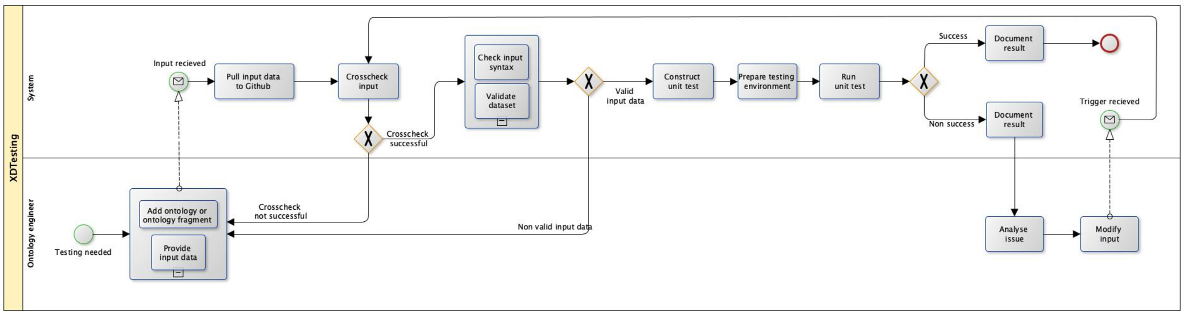


Figure 5.4: The workflow of the XDTesting tool automation

data are not syntactically correct. The course of the workflow depends on the result of the test case execution. If it is successful, the test case is documented, and the process ends. If it is not successful, the system documents the test case execution and waits for the issue to be analyzed by the ontology engineer. Once the input for the test case is modified, the system starts the process again at the cross-check step. The workflow terminates when all test cases have been passed successfully.

Lastly, the component diagram in Figure 5.5 depicts how individual components are interconnected to create the overall system. The components are `User input retrieval`, `Ontology fragment directory structure`, `GitHub`, `Input cross-check`, `OWLUnit jar`, `Setup testing environment`, `Check syntax`, `Validate toy dataset`, `Construct test case`, `Execute test case`, and `Document test case`. The entry-point component is the `User input retrieval` which enables the access to the input data. This components is used by the components that perform the input cross check, the syntax check, the toy dataset validation, the construction of the test case, and the creation of the ontology fragment directory structure. Meanwhile, the component `Execute test case` uses the output of the `Construct test case` component and the `Setup testing environment` which prepares the environment for the test by using the OWLUnit jar. Lastly, the `Document test case` uses the results of the test case execution to document it and pulls the files to GitHub.

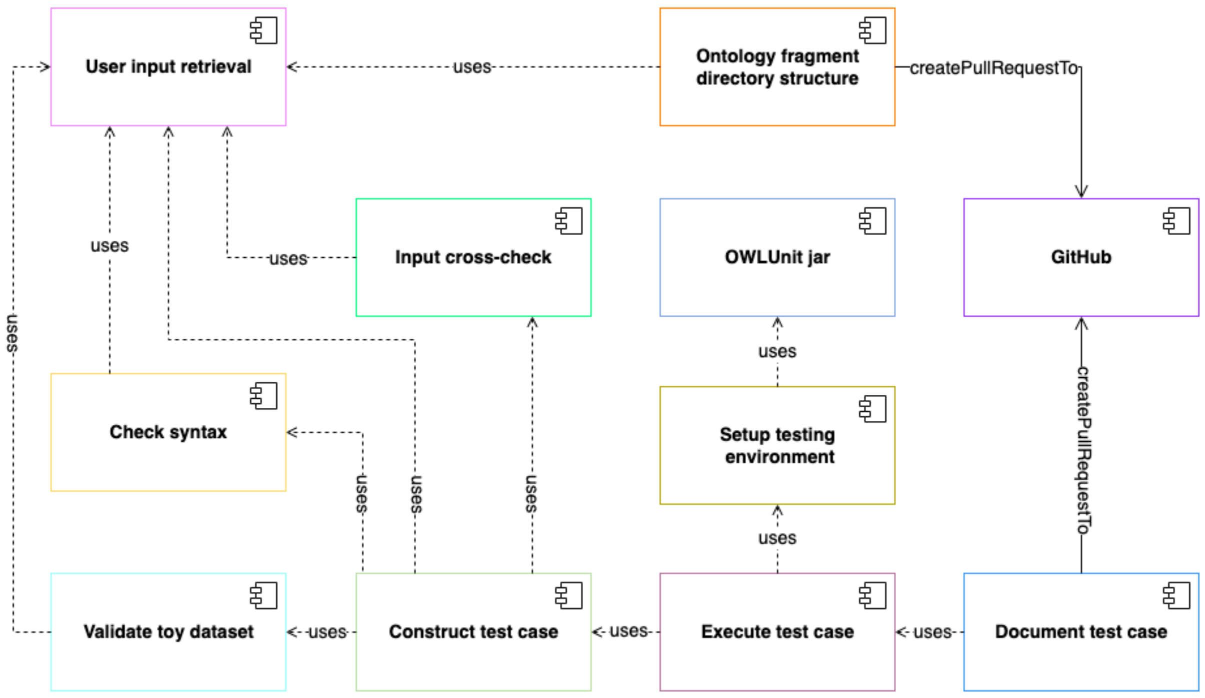


Figure 5.5: A component diagram showing the dependencies between the components of the XDTesting tool.

5.4 XDTesting development

In this section we describe the development of the XDTesting tool as a automatic workflow in the GitHub platform. Initially, we summarize the exploitability of the tool. In Subsection 5.4.1 we describe in detail the technical development of XDTesting, including pseudocode and a recap of the used libraries. In Subsection 5.4.2 we present the XDTesting Configurator that is the web application of XDTesting.

XDTesting is a ontology testing manager based on the eXtreme Design methodology. The tool is developed as a web application integrated with GitHub. The aim is to assist ontology engineers with the task of creating, annotating, executing, and documenting test cases. In addition, XDTesting provides a visual summary of the testing of the ontology module and a complete versioned documentation of the testing in the GitHub repository. The general information of XDTesting with links to the source code, documentation, release and running instance is shown in Table 5.1.

Information	
ID	`https://zenodo.org/badge/latestdoi/466713931`
Type	Software
Title	XDTesting
Description	XDTesting is an ontology testing manager integrated with the GitHub platform.
Source code	`https://github.com/FiorelaCiroku/XDTestingSession`
Documentation	`https://github.com/FiorelaCiroku/XDTestingSession/readme.md`
Release	`https://github.com/FiorelaCiroku/XDTesting-Configurator/releases/tag/v1.0.0`
Licence	CC BY
Running instance	`http://testing.extremedesign.info/`

Table 5.1: General information of the XDTesting tool

5.4.1 The XDTesting GitHub automated workflow

For the back-end of XDTesting we have used GitHub. The role of the platform is threefold: (1) it runs the workflow described in Algorithm 2; (2) it stores the data from the XDTesting Configurator; and (3) it stores the documentation of the test cases. The workflow listens to changes in the `UserInput.json` file, which is the file where the user input is stored. When a commit is completed on the file, the workflow is triggered to start. The workflow creates an Ubuntu environment and performs a checkout of the repository by using the `Checkout`[8] action. This action checks out the workspace of the repository workflow can access it.

Later, it install Python and a set of libraries needed to execute the commands that follow. One of the main libraries that we have used is the `Json` (JavaScript Object Notation) library, which is a collection of functions and methods that may be used in several computer languages, including Python, JavaScript, and others, to effortlessly parse, produce, and manipulate JSON data [40]. The library is used to parse data that

[8]`https://github.com/marketplace/actions/checkout`

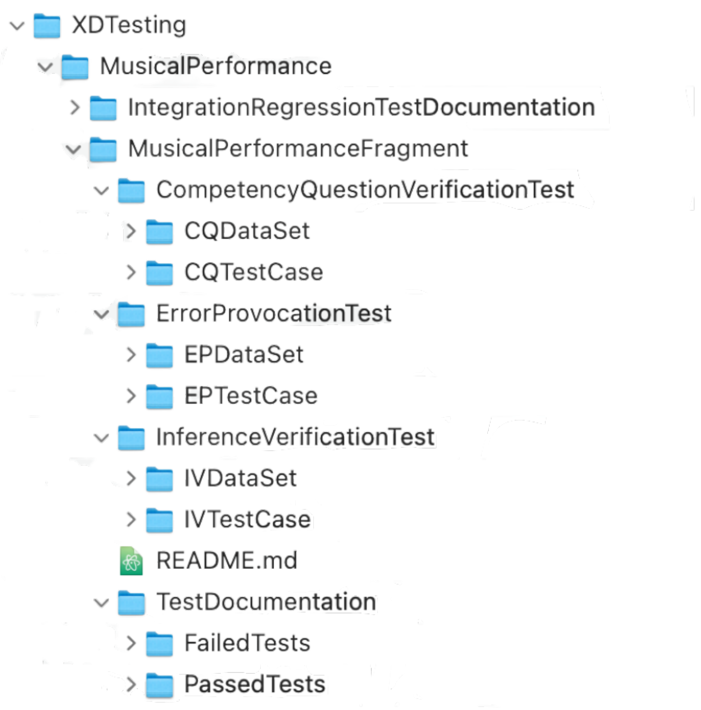
Algorithm 2 The GitHub workflow of XDTesting

```
 1: procedure GitHub Workflow
 2:     Trigger on push in UserInput.json
 3:     Run on Ubuntu
 4:     Checkout repository
 5:     Install Python
 6:     Install Python libraries
 7:     Create GitHub directory structure
 8:     Create test case and dataset
 9:     Commit to GitHub
10:     Create testing environment
11:     Execute test case
12:     Commit to GitHub
13:     return Clean environment
```

is exchanged between the front end and the back end of the XDTesting tool. More specifically, it is in a JSON file that the input is stored, and later on, the output results are stored. In addition, this package is also used to validate the expected results of a query in the JSON format. We have used the `os` library to create a structure of directories and files in the GitHub repository in order to guarantee a standard in documentation. The `os` library is a built-in library in Python that provides a way to interact with the operating system and perform various tasks such as creating, reading, and deleting files and directories, getting information about the system, and running shell commands [56]. To help with the process, we have also used the `sys` library, which is a built-in library in Python that provides access to various system-specific parameters and functions [83]. To validate the syntax of a SPARQL query, we use the `requests` and `rdflib`. The `requests` library allows to send `HTTP` requests and handle HTTP responses in a simple and elegant way [70]. In our implementation the HTTP requests are sent to a querying service to test the SPARQL query. The `rdflib` library is a third-party library in Python that allows you to parse, manipulate, and serialize RDF data in Python. It also provides support for several RDF serialization formats such as RDF/XML, N3, Turtle, N-Triples, and JSON-LD [67]. Lastly, to document the execution of the test cases as a Markdown

[48] table in GitHub we use the `pytablewriter` library. This library is a third-party library in Python that allows to write various formats of table data into files or strings [90]. Part of the documentation is also the date when the test case was executed, for which we use the `datetime` library, which is a built-in library in Python that provides a way to work with dates and times [19].

After the libraries are installed, the workflow runs a Python script to create the GitHub directory structure, as shown in Figure 5.6. The directories that are created are: *CompetencyQuestionVerificationTest*, *InferenceVerificationTest*, *ErrorProvocation-Test*, and *TestDocumentation*. Each of the first three directories includes two other directories named *TestCase* and *DataSet*. All the directories are briefly described in their respective *ReadMe.md* files.



```
XDTesting
  MusicalPerformance
    IntegrationRegressionTestDocumentation
    MusicalPerformanceFragment
      CompetencyQuestionVerificationTest
        CQDataSet
        CQTestCase
      ErrorProvocationTest
        EPDataSet
        EPTestCase
      InferenceVerificationTest
        IVDataSet
        IVTestCase
      README.md
      TestDocumentation
        FailedTests
        PassedTests
```

Figure 5.6: The directory structure automatically created in GitHub of the Musical Performance module of the Polifonia Ontology Network

At the same time, the workflow creates the files for the test cases and the datasets using the information stored in the `UserInput.json` file. Part of the creation of the test cases is the validation of the syntax of the expected results, the SPARQL query, and the toy dataset, if present. If the syntax of the data is valid, the script constructs the test case. An important aspect of the feature is that it makes it possible for the system to automatically generate the prefixes that are used in the test case. Then, it stores it in the correct directory in the GitHub repository and commits the changes. In the opposite case, it throws an error indicating incorrect syntax.

Next, it prepares the testing environment using a GitHub action named `XDTesting Setup environment`[9]. This action is developed and published in the GitHub marketplace. It deals with the installation in a GitHub hosted runner (server) of the technical components that are essential for the execution of the test case. The components are Java and the OWLUnit jar. Java is downloaded and installed by using a verified GitHub action named *Setup Java JDK*[10]. While the OWLUnit jar is downloaded from its original repository and installed.

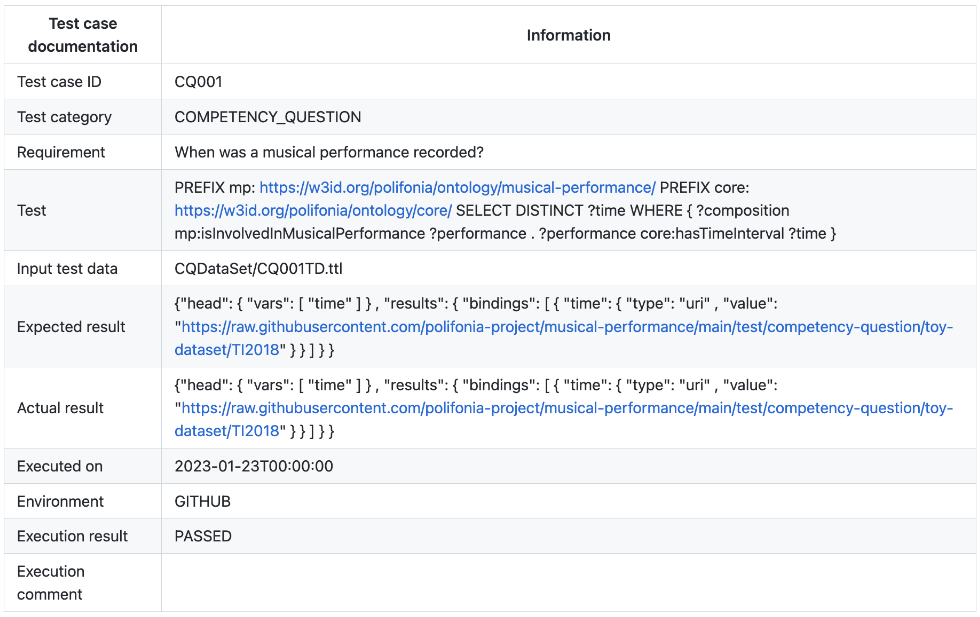
With the testing environment ready, the workflow executes the test case and commits the documentation of the results in the repository. The documentation is done based on a GitHub template that includes information such as the test ID, the requirement that is being tested, the category of the test, the test case description, the test itself, the input test data, the expected results, the actual results, time of the execution, environment of the execution, the execution result and comments. An example of the documentation of a test case execution is shown in Figure 5.7. After this step, the workflow cleans any remaining processes and completes.

5.4.2 The XDTesting Configurator

XDTesting Configurator is web application that serves as the front-end of the system. This interface is built using Angular 12[11] and can perform the actions listed below.

- The user can log in using their GitHub credentials.
- The user can select the repository in which it desires to work.

[9]`https://github.com/marketplace/actions/xdtesting-environment-setup`
[10]`https://github.com/marketplace/actions/setup-java-jdk`
[11]`https://angular.io/`



Test case documentation	Information
Test case ID	CQ001
Test category	COMPETENCY_QUESTION
Requirement	When was a musical performance recorded?
Test	PREFIX mp: https://w3id.org/polifonia/ontology/musical-performance/ PREFIX core: https://w3id.org/polifonia/ontology/core/ SELECT DISTINCT ?time WHERE { ?composition mp:isInvolvedInMusicalPerformance ?performance . ?performance core:hasTimeInterval ?time }
Input test data	CQDataSet/CQ001TD.ttl
Expected result	{"head": { "vars": ["time"] } , "results": { "bindings": [{ "time": { "type": "uri" , "value": "https://raw.githubusercontent.com/polifonia-project/musical-performance/main/test/competency-question/toy-dataset/TI2018" } }] } }
Actual result	{"head": { "vars": ["time"] } , "results": { "bindings": [{ "time": { "type": "uri" , "value": "https://raw.githubusercontent.com/polifonia-project/musical-performance/main/test/competency-question/toy-dataset/TI2018" } }] } }
Executed on	2023-01-23T00:00:00
Environment	GITHUB
Execution result	PASSED
Execution comment	

Figure 5.7: The documentation in GitHub of the execution of a test case from the Musical Performance module of the Polifonia Ontology Network

- The user can create new ontologies or select from ontologies that are found in the GitHub repository.
- The user can create new ontology fragments or edit existing ones.
- The user can create Competency Question Verification tests, Inference Verification tests, and Error Provocation tests for each ontology fragment.
- The user can provide the requirement, SPARQL query, expected results, and dataset for the competency question verification test.
- The user can provide a SPARQL query, select a reasoner, and upload a dataset for the inference verification test.
- The user can provide the dataset for the error provocation test.
- The user can create the SPARQL query, dataset, and expected results on the fly instead of uploading the files.

- The user can edit or delete the test cases.
- The user can check the status of the execution of the test case.
- The user can view the test case that is created and the documentation associated with it.
- The user can view the list and progress of the actions running in the selected repository.
- The user can find information about the tool's instructions in the Help section.

In order to use XDTesting Configurator, the user must log in with their GitHub account, as seen in Figure 5.8. When accessing the application for the first time, GitHub is going to require the confirmation of the password or another means of identity confirmation.

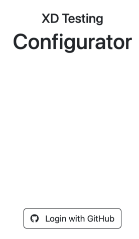
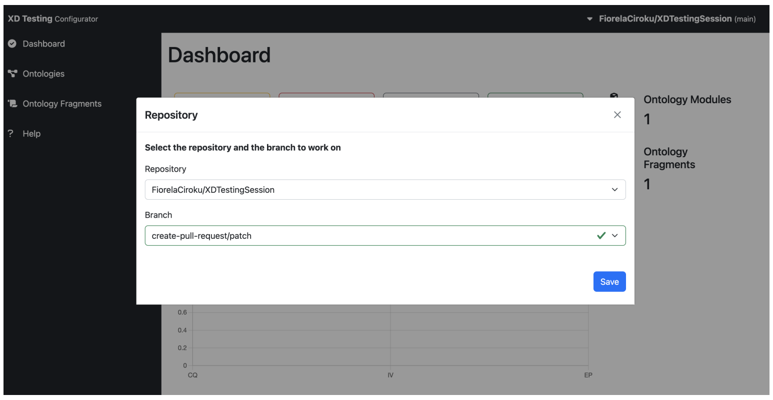


Figure 5.8: The login view of the XDTesting configurator

After the user has logged in, is required to select a GitHub repository and the branch to work on. The view of the selection is displayed in Figure 5.9.



Repository

Select the repository and the branch to work on

Repository: FiorelaCiroku/XDTestingSession

Branch: create-pull-request/patch

Figure 5.9: The repository selection view of the XDTesting configurator

When the repository is selected, the user is directed to the dashboard view of the XDTesting configurator, as shown in Figure 5.10. This dashboard displays statistics for the testing of the selected ontology fragments, such as the number of tests that have errors, have failed, are running, or are successful. On the right side of the dashboard, is projected the number of ontology modules and ontology fragments present in the repository. On the lower center section of the dashboard is shown a graph that summarizes the types of testing performed for each modules and their respective number. In the left side panel are the sections of Dashboard, Ontologies, Ontology fragments, and Help.

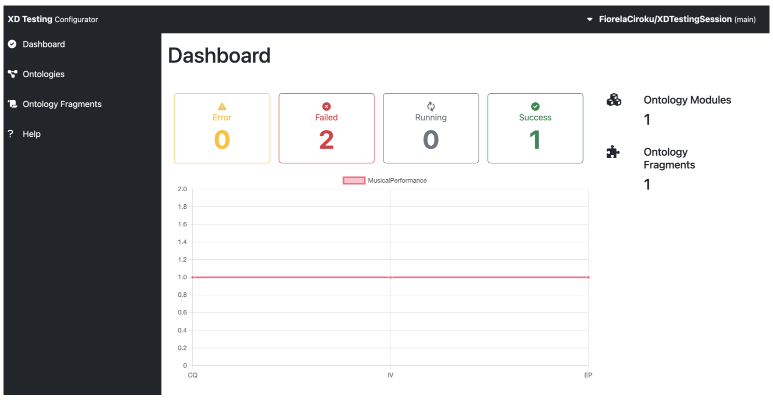


Figure 5.10: The Dashboard view of the XDTesting configurator

If one selects Ontologies, the application will redirect to the Ontology view, shown in Figure 5.11. The user can choose to create a new ontology or to select one from the list shown in the view. In addition, it can view the list of ontologies present in the repository and delete ontologies.

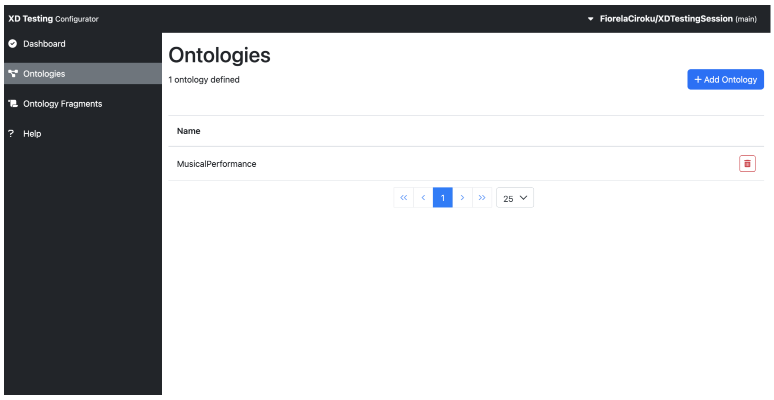


Figure 5.11: The Ontologies view of the XDTesting configurator

The Ontology Fragments section is displayed in Figure 5.12. In this view of the interface, the user can create a new ontology fragment or use the ones on the list. Similarly to the ontology section, the user can view a list of the fragments, with information about their belonging ontology. An ontology fragment can be searched by a keyword or with filters, e.g., by ontology name. Lastly, an ontology fragment can be edited and deleted.

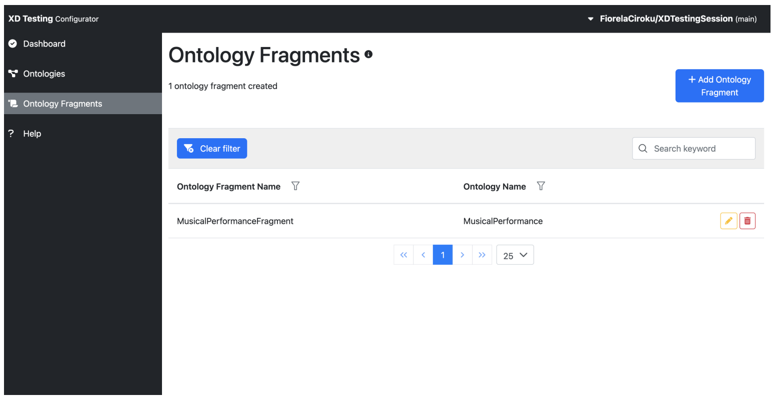
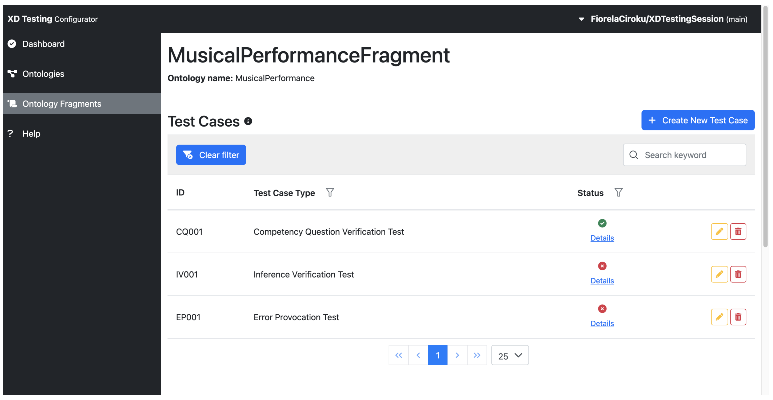


Figure 5.12: The Ontology Fragments view of the XDTesting configurator

If the user decides to edit an ontology fragment, then it is directed to the view shown in 5.13. Here, the user can create a new test case or view a list of existing test cases. Each of the test cases present in the repository is identified by an automatically assigned

ID, the type of the test case, and the status of the execution. The Configurator displays three possible outcome from the test case execution: (1) test case is executed and the result is success, (2) test case is executed and the result is fail, and (3) test case is not executed. The first type of outcome is notated with a green pass check, while the remaining types are notated with a red fail check. Under the details, the tool shows further information about the outcome of the execution.



MusicalPerformanceFragment

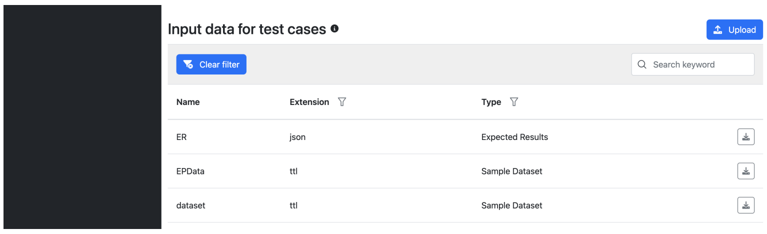
Ontology name: MusicalPerformance

Test Cases

ID	Test Case Type	Status
CQ001	Competency Question Verification Test	✓ Details
IV001	Inference Verification Test	✗ Details
EP001	Error Provocation Test	✗ Details

Figure 5.13: Inside the Ontology Fragments view of the XDTesting configurator for displaying test cases

In Figure 5.14, is shown another feature present in the ontology fragment view. This feature enables the user to upload data files by specifying the type of file. The user can also search files by keyword of by filtering the extension, or the type. Additionally, it can download the files.



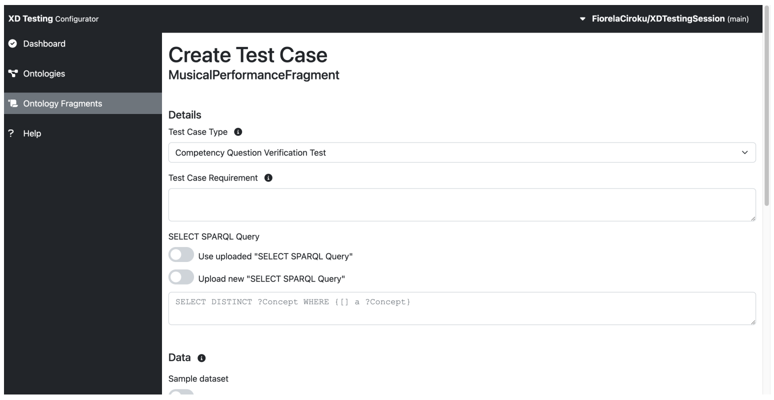
Input data for test cases

Name	Extension	Type
ER	json	Expected Results
EPData	ttl	Sample Dataset
dataset	ttl	Sample Dataset

Figure 5.14: Inside the Ontology Fragments view of the XDTesting configurator for uploading data files

If the user wants to create a new test case, they will be directed to the view shown in Figure 5.15 after clicking the `Create New Test Case` button on the right upper corner.

By default, the type of the test case that is selected is the Competency Question Verification test, but it can be changed by selecting another option from the dropdown list. Based on the selected type of the test case, different input is required. The SPARQL query, expected results, and sample dataset can be provided in three modalities: (1) by selecting a previously uploaded file, (2) by uploading a file, and (3) by directly writing it in the textbox.



Create Test Case
MusicalPerformanceFragment

Details
Test Case Type
Competency Question Verification Test
Test Case Requirement

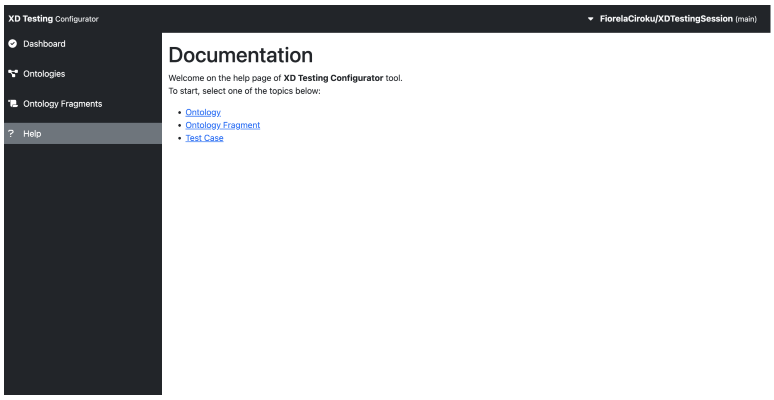
SELECT SPARQL Query
Use uploaded "SELECT SPARQL Query"
Upload new "SELECT SPARQL Query"

```
SELECT DISTINCT ?Concept WHERE {[] a ?Concept}
```

Data
Sample dataset

Figure 5.15: The test case view of the XDTesting configurator

In each of the views, the user can find quick information regarding important terms and concepts by hovering on the information bubbles. For more detailed information, he can redirect to the Help section for information on how to use the tool for creating ontologies, ontology fragments, and test cases.



Documentation
Welcome on the help page of **XD Testing Configurator** tool.
To start, select one of the topics below:

- Ontology
- Ontology Fragment
- Test Case

Figure 5.16: The Help view of the XDTesting configurator

Chapter 6

Evaluation

Chapter 6 reports the evaluation of the RevOnt method and the XDTesting method developed and implemented in this research. Section 6.1 and 6.2 describe the experimental setup and the results of the evaluation for RevOnt and XDTesting respectively.

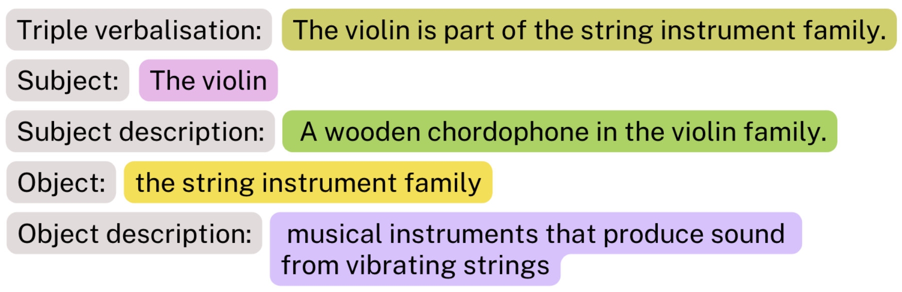
6.1 Evaluation of RevOnt

In this section, we describe the end-to-end and component evaluation of the RevOnt framework. The evaluated components are the Verbalisation Abstraction and the Question Generalisation stage. The end-to-end evaluation provides a meaningful quantification of system's effectiveness. While, with component evaluations, it is possible to observe the impact that each component has on the system effectiveness [72]. One of the challenges of ontology learning and natural language processing is the dependency between the tools and techniques used in a pipeline; therefore, such an evaluation would provide meaningful insight into the performance of the components and the approach overall. The objective of the evaluation is to prove the hypothesis H2 which claims that the quality of the elicited requirements that are generated with the means of language models from a knowledge graph is comparable to human generated requirements.

6.1.1 Experimental setup

We have conducted a two-step user-based experiment including 15 participants with an engineering background and familiar with ontology development. The experiment consisted of manually reproducing the stages of Verbalisation Abstraction, and Question Generation in order to assess the quality of the components individually. In total, we created 20 forms[1] (one for each theme), with a varying number of tasks. The tasks were created with data from the WDV dataset, and their number depends on the number of different triple verbalizations (properties) existing in a theme. An overview of the themes and the number of tasks is shown in Table 6.1.

Firstly, we introduced the participants to the theme and the data that they needed to complete the tasks. For each task, we provide the triple verbalization, the subject, the subject description, the object, and the object description. The illustration used to describe the data is shown in Figure 6.1.



Triple verbalisation: The violin is part of the string instrument family.
Subject: The violin
Subject description: A wooden chordophone in the violin family.
Object: the string instrument family
Object description: musical instruments that produce sound from vibrating strings

Figure 6.1: An illustration of example data including the triple verbalisation, the labels and descriptions of the subject and object of the triple. The example serves to explain the data needed to perform the tasks.

Then, using examples, we describe the two tasks that they must complete. The first task is to abstract a triple verbalization given the information provided, as shown in Figure 6.2. The second task is to formulate questions based on the abstraction that they have created. The participants are asked to formulate 3 questions, where the answers

[1]https://drive.google.com/drive/folders/1M7LCmqw4dc33U73GTauec02h3JNnNxv4?usp=sharing

Theme	No of tasks	Annotation/Theme
Airport	27	7%
Artist	65	17%
Astronaut	57	16%
Athlete	53	13,7%
Building	67	17,4%
Celestial body	25	6,4%
Chemical compound	33	8,6%
City	73	19%
Comics character	79	21%
Food	64	17,3%
Mean of transportation	58	15,4%
Monument	62	16,3%
Mountain	23	5,9%
Painting	29	7,5%
Politician	56	14,5%
Sports team	49	12,7%
Street	22	5,7%
Taxon	27	7%
University	62	16,4%
Written work	21	5,4%
Total	952	

Table 6.1: An overview of the number of tasks based on themes. The number of tasks corresponds to the number of distinct properties of the triples of theme.

to the questions will be: 1) the subject of the abstraction; 2) the property; and 3) the object of the abstraction. The tasks are represented in Figure 6.3.

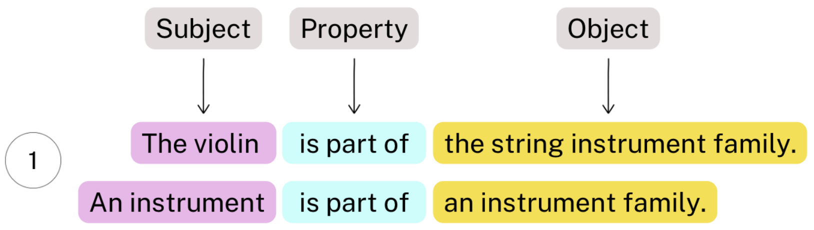


Figure 6.2: Example illustration for task 1. In this illustration, we describe the first task that is to abstract the verbalisation using the given data. The abstraction is completed by generalizing the subject and the object of the triple, and not the property.

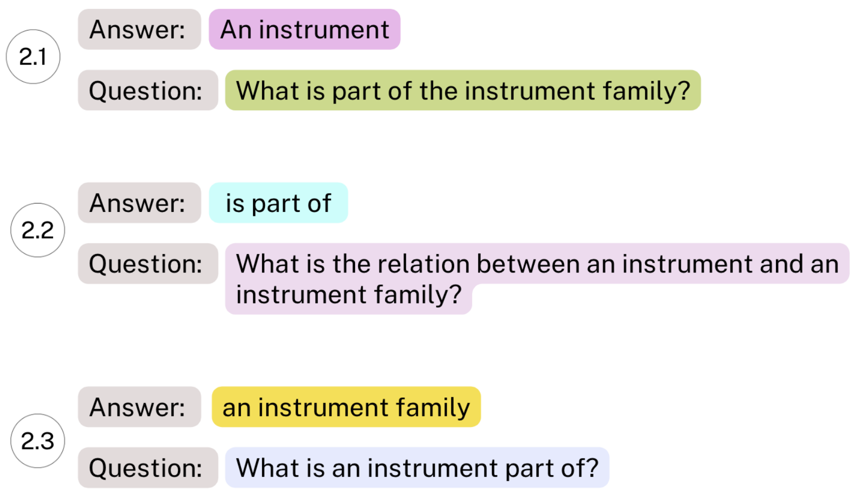

2.1
Answer: An instrument
Question: What is part of the instrument family?
2.2
Answer: is part of
Question: What is the relation between an instrument and an instrument family?
2.3
Answer: an instrument family
Question: What is an instrument part of?

Figure 6.3: Example illustration for task 2. The example describes task 2 that is the generation of three questions, when the answer is provided. The answers are the subject, property, and object of the triple.

We arranged for each theme to be covered by two participants. In total, we have gathered 40 responses from the forms, containing approximately 1.9K annotations. Meanwhile, the WDV dataset has, on average, 380 triple verbalizations per theme. The coverage of each theme with annotations can be found in the Table 6.1. We used the annotations from the user-based experiment to calculate the BLEU score, a metric for automatically evaluating machine-translated text. The BLEU score is a number between zero and one that measures the similarity of the machine-translated text to a set of high-quality reference translations [57]. The annotations serve as grams, each with a weight of 0.5. The number of reference human translations influences the results. Usually, more references result in better and more accurate scores. According to [44] and [30], scores over 0.3 generally reflect understandable translations, and scores over 0.5 generally reflect high-quality translations. The interpretation of BLEU scores is shown in Table 6.2. We have chosen this metric for the evaluation of the RevOnt implementation because according to a set of experiments presented in [57], BLEU correlates highly with human judgments. The experiments consisted of two groups of people, named the monolingual and the bilingual group, rating from 1 to 5 the translations of a set of questions. The high correlation coefficient of 0.99 shows that BLEU accurately tracks human judgment. Moreover, as mentioned above, the participants of our experiment have an engineering background, which insinuates that the annotations that they have done are of good quality.

BLEU Score	Interpretation
< 0.10	Almost useless
0.10 - 0.19	Hard to get the gist
0.20 - 0.29	The gist is clear, but has significant grammatical errors
0.30 - 0.40	Understandable to good translations
0.40 - 0.50	High quality translations
0.50 - 0.60	Very high quality, adequate, and fluent translations
> 0.60	Quality often better than human

Table 6.2: Interpretation of BLEU scores

6.1.2 Results

In this subsection, we present the results of the evaluation of the RevOnt framework. In the following subsections, we interpret the results for each of the stages of the approach and summarize the results for the whole system. To interpret the distribution of the BLEU scores for the Verbalisation Abstraction and the Question Generation stages, we have created Box and Whisker Plots.

The distribution of the BLEU scores for the Verbalisation Abstraction stage is presented in Figure 6.4. As seen from the plot, the mean of the scores is 0.41, which is interpreted as a high-quality translation. The 75th percentile is 0.55 and the 25th percentile is 0.3. Meanwhile, the highest and the lowest data points are respectively 1 and 0. These results mean that 75% of the abstractions generated from RevOnt have a good to high-quality.

By observing the differences between the abstractions generated by RevOnt and those created by the user, we can conclude that the abstractions generated by RevOnt are more general. Usually, the system abstracts the subject and the object of the verbalisation into more general classes than a user. As shown in Example 6.1.1, the system abstracts Michael Jackson as a human, while a user abstracts it as a singer.

Example 6.1.1. *The difference between the abstraction generated by RevOnt and by the users*

```
Verbalisation:   Michael Jackson is a member of the
                 Michael Jackson discography.
RevOnt:          Human is a member of the discography.
User annotation: A singer is a member of the discography.
```

As for the second component of Revont, Question Generation, we display the distribution of BLEU scores in Figure 6.5. The mean is 0.3, which is on the border of being interpreted as a good translation. The 75th percentile is 0.47 and the 25th percentile is 0.21. The highest and lowest data points are 0.81 and 0.1 respectively.

The non-satisfactory evaluation of this stage is explained in Figure 6.6. In this plot, we have evaluated individually the generation of each type of question. The red plot represents the generation of the questions when the answer is the subject of the

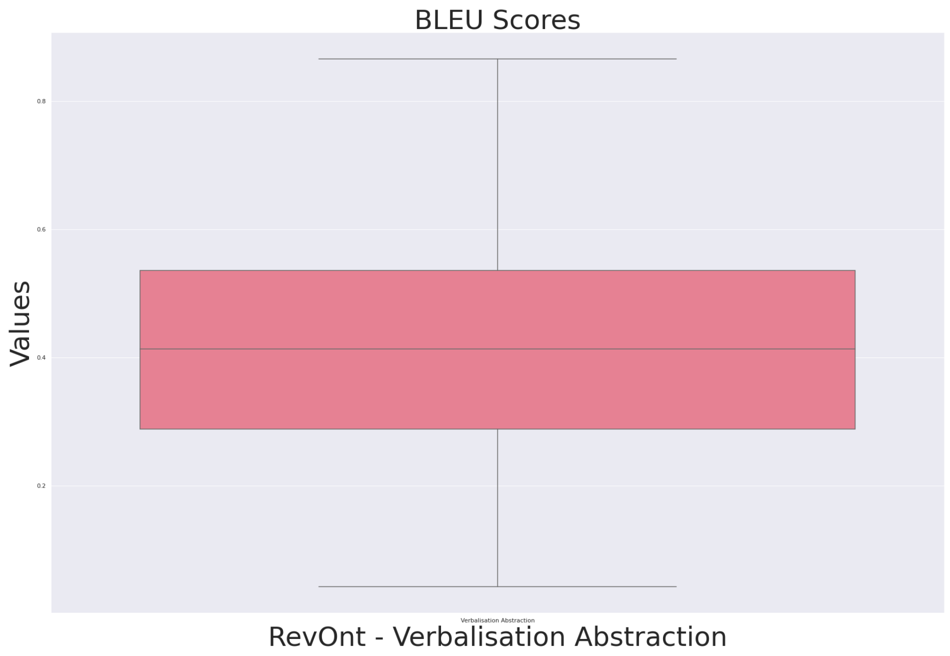


Figure 6.4: BLEU score for the Verbalisation Abstraction stage. In the plot is shown the distribution of BLEU scores for the Verbalisation Abstraction stage for all the domains of the dataset.

abstraction, the yellow plot when the answer is the property of the triple, and the green plot when the answer is the object of the abstraction.

Visibly, the generation of the question when the answer is a property of the triple performs poorly. This result supports the hypothesis that the questions that T5 generates in this case are generally identical to the ones that it generates when the answer is the object of the abstraction. In Example 6.1.2, we show the difference between the question that is generated by RevOnt and the one generated by the user when the answer is the property of the triple.

Example 6.1.2. *The difference between the questions generated by RevOnt and the users when the answer is the property of the triple.*

```
Verbalisation:   Michael Jackson is a member of the
                 Michael Jackson discography.
Answer:          discography/member of
```

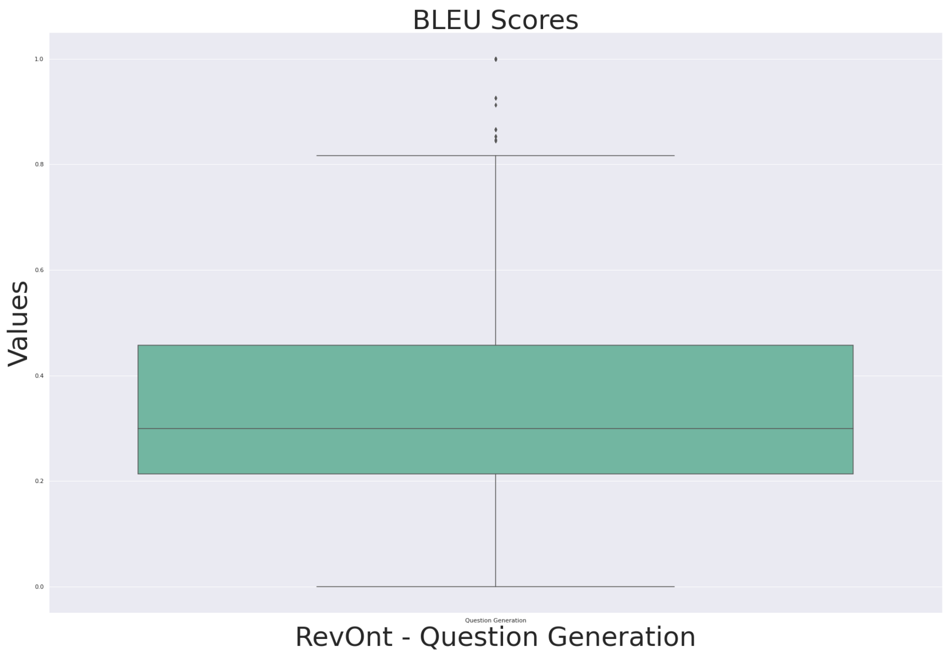


Figure 6.5: BLEU score for the Question Generation stage. In the plot is shown the distribution of BLEU scores for the Question Generation stage for all the domains of the dataset including the three types of questions.

```
RevOnt:           What is a human a member of?
User annotation: Which is the relation between a singer
                  and the discography?
```

Comparing the scenarios when the subject and the object of the abstraction are the answer, we observe that the latter is more accurate with a mean of 0.42, which is a high quality translation. The 75th percentile is 0.61, the 25th percentile 0.28, and the highest point is 1.0 and lowest is 0.08. 75% of the scores for this category of questions is of good to high quality.

The results provide valuable insight into the Question Generation stage in particular. The evaluation of the quality of each type of question is certainly a matter that is to be taken into account for the refinement of the RevOnt framework. As mentioned earlier, the design choice to include all three types of questions is for experimentation only and to provide feedback for future development. In Figure 6.7 is the RevOnt Box and

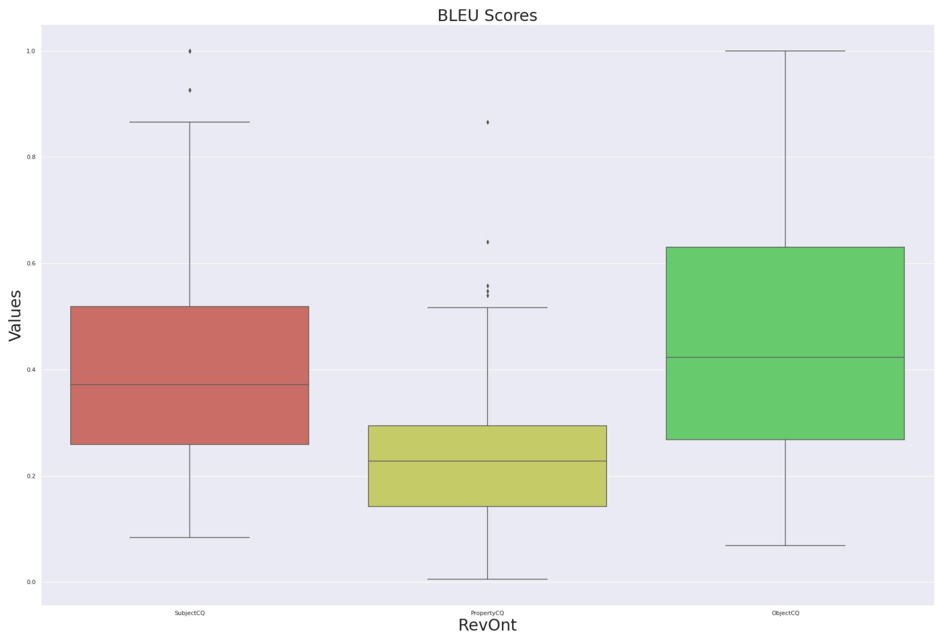


Figure 6.6: BLEU score for the Question Generation stage individualized. In the plot is shown the distribution of BLEU scores for each type of question that is produced by the Question Generation stage for all the domains of the dataset.

Whisker plot if we consider that the Question Generation stage produces only questions where the object of the abstraction is the answer (which is the best performing category of questions). Overall, the system's BLEU scores have a mean of 0,4, indicating good translation quality.Â

6.2 Evaluation of XDTesting

In this section, we describe the experimental setup and results of the evaluation of the XDTesting tool. The goal of the evaluation is to prove the hypothesis H3. Hypothesis H3 claims that the automation of the testing phase of an ontology engineering process can reduce the workload of the ontology engineer. To prove it, we have designed an experiment that focuses on the efficiency and usability of the XDTesting tool.

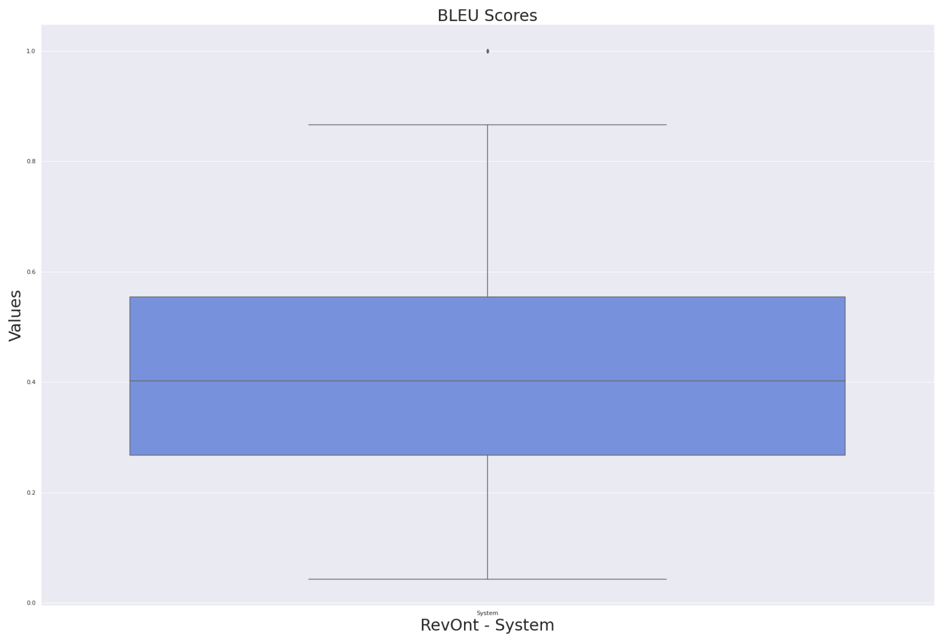


Figure 6.7: BLEU score for the RevOnt framework. In the plot is shown the distribution of BLEU scores for the RevOnt framework containing the Verbalisation Abstraction and the Question Generation stage with only one type of question (the question that is answered by the object of the triple).

6.2.1 Experimental setup

The evaluation of the XDTesting tool is realized in the form of a session in collaboration with ontology engineers and testers. We have selected a group of 7 researchers from the STLab research lab. Each of the participants is familiar with or an expert in ontology development and testing and the eXtreme Design methodology. The goals of the experiment are to evaluate the efficiency and usability of XDTesting and to identify action points for further development. The task for the participants is to complete a competency question verification test, an inference verification test, and an error provocation test by using the XDTesting tool. Completing a test case consists of creating, executing, and documenting a test case. The data that the participants need are an ontology, a competency question with the respective SPARQL query and expected result, general constraints, and datasets for each of the tests. The only prerequisite for the participants was for them to have an active GitHub account.

Initially, we presented a quick background related to the types of tests in eXtreme Design and described to the participants the task, the data that were provided to them, and the overall instructions. To share the data files and the scripts, we created a demo GitHub repository[2] that they could fork into their own accounts. In addition, we instructed the participants to grant *Read and write permission* to the actions workflow and enable the workflow that they forked from the demo repository. Lastly, in order to test some of the features of the tool, we asked the participants to download the data files locally. The ontology that the participants tested is the *Musical Performance* ontology module[3] developed in the Polifonia project. For the Competency Question Verification test, the participants had to test the requirement "When was a musical performance recorded?", with the SPARQL query and expected results shown respectively in Listings 6.1 and 6.2.

```
PREFIX mp: <https://w3id.org/polifonia/ontology/musical-performance/>
PREFIX core: <https://w3id.org/polifonia/ontology/core/>
SELECT DISTINCT ?time
WHERE {
?composition mp:isInvolvedInMusicalPerformance ?performance .
?performance core:hasTimeInterval ?time }
```

Listing 6.1: SPARQL query for the Competency Question Verification test

```
{"head": {"vars": ["time"]} ,
"results": {"bindings":
    [{"time": {
        "type": "uri",
        "value": "https://raw.githubusercontent.com/polifonia-project/
    musical-performance/main/test/competency-question/toy-dataset/TI2018
    " } } ] } }
```

Listing 6.2: Expected results for the Competency Question Verification test

After the completion of the task, the participants were asked to provide their feedback in different modalities. To document the problems that they encounter during the

[2]https://github.com/FiorelaCiroku/XDTestingDemo

[3]https://w3id.org/polifonia/ontology/musical-performance/

session, we had created two GitHub issue templates[4]. One template is to document bug reports, and the other is for requesting features that they would like to have. We have also opened several discussion boards[5] to light up conversations regarding features, bugs, documentation, and general feedback. Mainly, to document the insights of the participants for the tool, we created a questionnaire[6] which is divided into three sections. The goal of the first section is to gather information about the ontology engineers' experience with testing. More specifically, the questions asked are:

- Are you an ontology engineer?
- Do you perform ontology testing?
- How long does it take to create, execute and document a test case?
- What are the most common issues that you encounter while creating, executing and documenting a test case?
- Which tools/applications do you use for the creation and execution of a test case?
- Which platforms do you use for the documentation of the testing?

The goal of the second section is to gather feedback on the functionality of specific features of the XDTesting tool and new requirements for the further development of the tool through the comment section of the survey. The questions asked are:

- Does the dashboard provide useful statistics about the testing of an ontology?
- What other statistics would you prefer to be displayed in the dashboard?
- Does XDTesting provide an efficient documentation structure in GitHub? If not, what can be improved?
- Does XDTesting have a good selection of features that meet your needs?

[4]`https://github.com/FiorelaCiroku/XD-TestingSession/issues`
[5]`https://github.com/FiorelaCiroku/XD-TestingSession/discussions`
[6]`https://xdtesting.limesurvey.net/493542`

- Is there any feature that you would like to have as part of XDTesting? If yes, which?
- Does XDTesting tackle any of the issues that you encounter while completing the testing manually? If yes, which?
- Would you prefer XDTesting to be integrated with other platforms other than GitHub? If yes, which?
- Does XDTesting have a complete documentation, including clear instructions and troubleshooting guides?

Lastly, the third section is a standard System Usability Scale (SUS) which is a commonly used, standardized questionnaire designed to assess a system's perceived usability [14]. The SUS is frequently used as a benchmark to compare different systems or to track usability changes over time. It consists of ten assertions, as shown in the list below, that users must rate on a 5-point Likert scale, with values ranging from 1 (strongly disagree) to 5 (strongly agree). The scores for each statement are then totaled up and multiplied by 2.5 to yield a final score ranging from 0 to 100.

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.

10. I needed to learn a lot of things before I could get going with this system.

In Figure 6.8, the grade ranking of SUS scores is displayed, where the scores are categorized based on acceptability ranges, grade scale, and adjective ratings. The acceptability ranges are not acceptable, low and high marginal, and acceptable. While the grade scale starts from the F grade, which is the lowest, to the A grade, which is the highest. Lastly, the adjective ratings vary from worst imaginable to best imaginable.

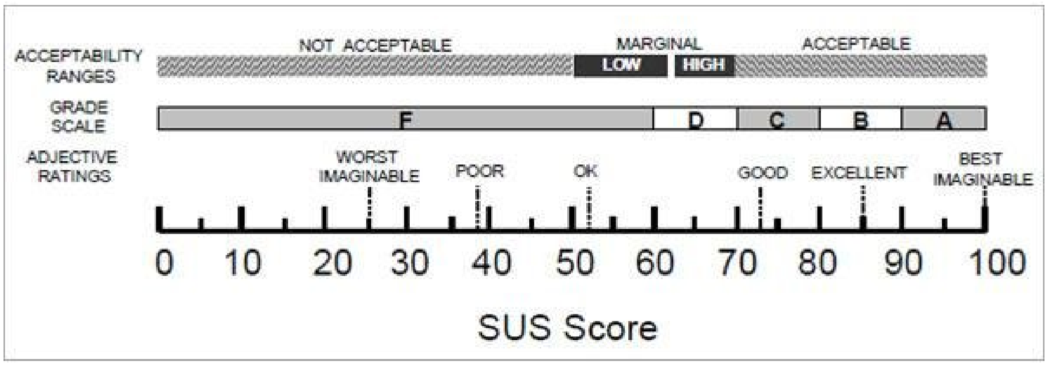


Figure 6.8: Grade rankings of SUS scores. Figure retrieved from [14]

In Figure 6.9 is shown the percentile ranking of the SUS scores. In accordance to Figure 6.8, scores below 50 is a grade F and fall under 12%, and scores above 85 are a grade A and are above 90%.

6.2.2 Results

In this subsection, we present the results of the evaluation of XDTesting. The results are gathered through a questionnaire, direct feedback, and GitHub issues. Firstly, we summarize the experience that the ontology engineers have relating to testing. Secondly, we outline their feedback regarding the features of XDTesting. Thirdly, we present the overall SUS score of the system and interpret it.

Seven ontology engineers participated in the XDTesting tool evaluation. Five of the engineers are experts in ontology testing, while the remaining two are familiar with the process. The findings of the first section of the questionnaire uphold the claims that we have made throughout the thesis regarding the tediousness of creating test cases,

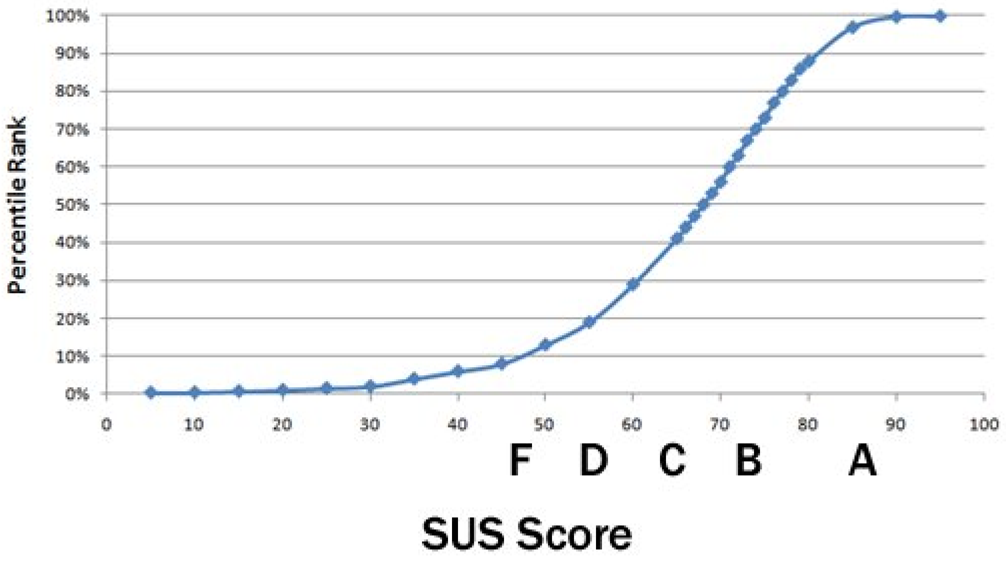


Figure 6.9: Percentile rankings of SUS scores. Figure retrieved from [14]

the most common issues when executing test cases, and the main tools that are used to assist with the process. Based on the survey, an ontology engineer spends from 10 to 60 minutes to create a test case, execute it, and document it, which is a substantial amount of time when testing ontologies. The main issues they encounter during the testing are the syntax errors with the SPARQL and expected results, the conversion of a SPARQL query written from a common text editor into a *one-liner escaped* form (that is, removing new lines, escaping spaces, and so on), and formatting errors. They notice that the lack of tools to automatically assist with the filling of the template, the execution, and documenting the executions is particularly a concern. To complete a test case, the engineers use a combination of disconnected tools such as TESTaLOD for the creation of test cases, the OWLUnit jar for the execution of the test case, Protégé for Inference Verification and Error Provocation tests, LODE[7] for testing the ontology module by means of its generated documentation, LodView[8] for assessing the quality of data associated with the ontology module, and text editors for creating data and queries to triplestores for storing test data and executing queries. For the documentation of the

[7]`https://essepuntato.it/lode/`
[8]`https://lodview.it/`

testing, almost all of the participants use GitHub with manually created files, while others do not document because of the lack of an integrated tool to automatically generate it.

The second section of the survey provides valuable insights regarding the features of XDTesting. All the participants have responded that the dashboard of the application provides useful statistics about the status of the testing of the ontology. Suggested improvements that can be made to the dashboard are the addition of a clickable button to direct to failed tests or their documentation, the replacement of the line plot with a scatter or bar plot, the addition of a list of fragments with a "fully completed" and "error presence" flag, and lastly, the possibility to select a specific kind of test. An interesting statistic that can be added to the dashboard is the calculated average for fails and successes, and the coverage of the test, which can be computed as the number of classes and properties in the tested ontology and the number of them involved in any test. When asked about the documentation structure on GitHub, the participants agreed that it is efficient. Though it can be improved by adding more documentation regarding the structure of the repository in the readme.md file and by adding links between files. Regarding the selection of features in XDTesting, the ontology engineers acknowledged that it meets their needs and tackles issues that they encounter while completing the testing manually. According to the feedback, the XDTesting tool facilitates the elimination of formatting issues, the automation of the test documentation with the versioning of the files, and it provides a coherent, homogeneous, and integrated tool for supporting ontology testing that reduces the effort. Considering that the main platform used for the documentation of the ontologies and their testing is GitHub, only two of the participants responded that they would like for XDTesting to be integrated into another platform. The suggested options are BitBucket[9] and GitLab[10]. Lastly, when asked if XDTesting has complete documentation, including clear instructions and troubleshooting guides, the participants unanimously agreed. One suggestion is for the documentation in the application to be integrated with the documentation in the GitHub repository.

The last section of the evaluation survey was the System Usability Scale questionnaire. As previously described, the results of the questionnaire are transformed into a score.

[9]`https://bitbucket.org/`

[10]`https://gitlab.com/`

The mean of the scores received is 83.57, and according to Figure 6.8 XDTesting can be evaluated as acceptable and a grade B tool. The mean score of the SUS questionnaire for the XDTesting tool is above the 90 percentile rank, as shown in Figure 6.9. The standard deviation is 6.79, which means that the data is relatively close to the mean. The margin of error is 2.56. The skewness of the data is calculated to be -0.092588, indicating that the data are roughly symmetrical and that we can use the t-test. We performed a one-sample t-test and raise the following hypothesis $H_0 : \mu = 78.5$ and $H_1 : \mu > 78.5$, where 78.5 is the hypothesized population mean of SUS scores. The calculated value is 1.9755. Based on the t-test score, we calculated the p-value with a degree of freedom of 6 (sample number - number of parameters from t-test). Its value is 0.047 and we can reject H_0 at the significance level 0.05, because the p-value does not exceed 0.05. The fact that the null hypotheses is rejected, does not necessarily mean that the alternative hypothesis is true. If we test whether the mean of SUS scores for the population is 80, the t-test value is 1.391, and the p-value is 0.10. There is not enough evidence to reject H_0 at the significance level 0.05, because the p-value is greater than 0.05. This result is expected considering the small sample (seven). A new evaluation session will be carried out after the second iteration of the XDTesting development.

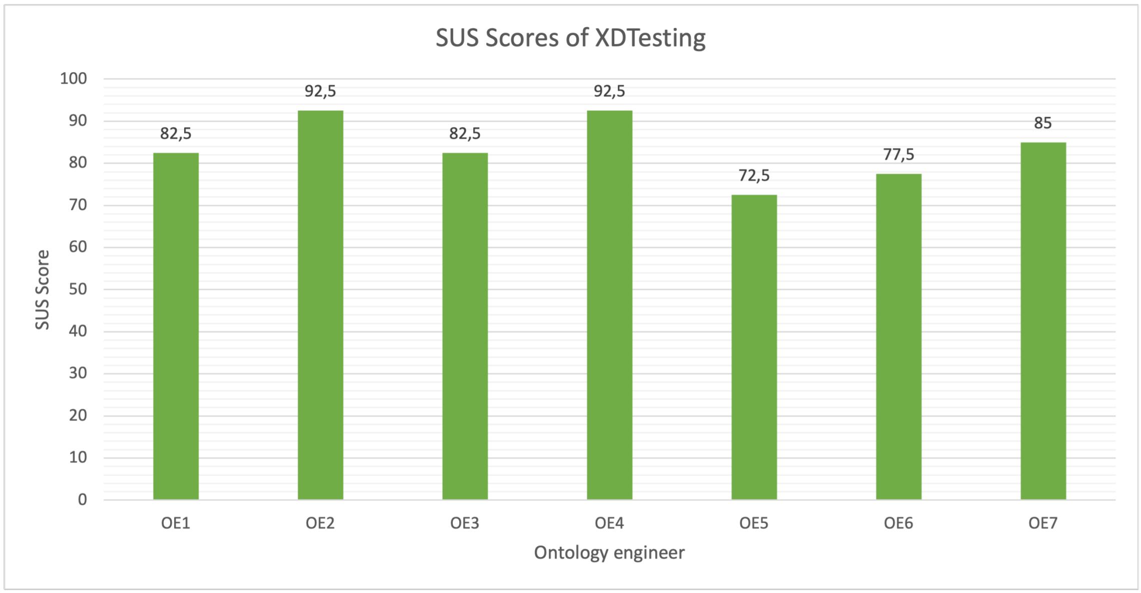


Figure 6.10: SUS scores of the participants in the experiment

Chapter 7

Conclusions and future work

In Chapter 7, we discuss the limitations of the work in Section 7.1, plans for future work in Section 7.2 and present the overall conclusions of the research in Section 7.3.

7.1 Discussion

Even though the results from the evaluation of RevOnt are satisfying, the work extensively depends on state-of-the-art natural language processing methods to achieve good results in extracting competency questions from knowledge graphs. These methods come with well-known issues and limitations, which inevitably propagate into Revont's pipeline. Below is our discussion regarding issues in the fields of natural language processing, ontology learning, and schema induction and discovery, which are reflected to some extent in our work.

1. Methods based on language models are highly sensitive to the input data [44]. This issue is generally observed in the generation of questions by RevOnt, where the distance between the triple and its verbalization plays a significant role. The quality of the extracted competency questions is decidedly related to the quality of the dataset that is used in the implementation. This is clearly noted in the difference in quality between the different types of questions that are generated when the answer is the subject/object of the verbalisation and when the answer is the predicate of the verbalisation. Given that there is a higher distance between

the property of the triple with its verbalisation compared to the distance between the subject/object and their verbalisation, it is understandable that the quality of the generated questions is going to differ. An important emphasis should be put on the fact that for the implementation, we did not extract the data directly from the knowledge graph and rather from a verbalization dataset of the knowledge graph. To extract data directly from a knowledge graph, it is necessary to retrain the language models that were reused with new data.

2. The WDV dataset showed limitations in the reuse of NER models considering that the ones that were analysed performed poorly for specific themes (50% of the WDV dataset). Their performance is explained by the heterogeneous data used to train these models, which is comprehensively different compared to the dataset. These models are often trained with data from news articles, and emails/chat data, which explains the good performance with themes such as Artist, Athlete, City, etc. This issue prompted the need to create an algorithm to classify the subject and object of the triple to the nearest class they belong to. The concern is that the implementation of the algorithm is dependent on the Wikidata query service, which does not make the method robust if one prefers to use another knowledge graph as input. By training a NER model with data specific to the domain of interest, it is possible to make the approach independent from the Wikidata query service, and more available to be reused.

3. The dependency between the language models and methods used in such a system poses another limitation. The performance of the system is impacted by the performance of each component. For natural language processing systems, the effect of errors is multiplicative. Because each component is dependent on the preceding, errors propagate along a processing pipeline, resulting in a final output that may be unsatisfactory despite the good performance of each component [72]. A solution to this problem could be to add a manual validation phase between each passage in the system, which would incur high costs in terms of time and effort. In most cases, the validation phase proposed in the above point would consist of a human intervention, which is not always desirable. Although interesting ideas such as us-

ing crowd-sourcing and social media to validate the results of language models are proposed by [5].

4. The scalability of a system with combines language models and querying services is one of the main drawbacks. Firstly, language models need time to perform their tasks and high processing capabilities. Secondly, knowledge graph querying services have limitations in usage, meaning that one is not allowed to make more than a certain number of requests per minute. So, despite a high computational power in use, the execution time is still not satisfying even for small datasets.

As for the XDTesting tool, the implementation of the method and its evaluation brought to the surface several limitations related to design choices and feature inclusion. Below we discuss these issues together with possible solutions.

1. The main limitation is the dependency on the GitHub platform. Even though GitHub is a highly used platform for collaborating on projects and showcasing work, there are as well other platforms that have relatively the same function such as GitLab, BitBucket, etc. Moreover, considering that the automation of several tasks of the ontology testing process is executed in GitHub, it might cause delay issues if several tasks are trying to access the workflow at the same time.

2. The evaluation session brought several features to our attention that might boost its usability. Most of the suggestions mention the integration of other platforms and APIs into the XDTesting Configurator. The main feature requested by the ontology testers was for the tool to assist with the creation of SPARQL queries and datasets. A suggested API to support the creation of SPARQL queries is YASGUI[1], which is a query editor that offers syntax highlighting, syntax validation, auto-completion, and a variety of different SPARQL result visualizations [95]. Currently, the users are requested to either write the SPARQL query in the text field or upload a file, which might be a redundant task. Meanwhile, for the creating and validating the dataset on-the-fly, `rdf-editor`[2] is suggested. It is a text editor

[1]`https://yasgui.triply.cc/`

[2]`https://www.npmjs.com/package/@rdfjs-elements/rdf-editor`

custom element which parses and serializes RDF/JS Quads using a selected RDF format. Another suggestion is to have to XDTesting tool as a Protégé plug-in that would provide ontology engineers with an all-in-one type of application for the creation, management, and testing of an ontology. By implementing these features, XDTesting might reduce the workload of the engineer to a greater extent and increase its usability within the community.

7.2 Future work

In its first implementation, RevOnt generated questions based on the abstraction of triple verbalizations, rather than the triple itself. In particular, the question generation model (T5) performs better when provided with the sentence and the answer. This limits the usability of the method with datasets where the answer is not explicit. It requires additional steps such as Part-Of-Speech tagging, parsing, and lemmatization to be able to provide an entity as an answer. We plan to refine this aspect of the method by 1) detecting entities that might serve as answers, and 2) not providing an answer. Furthermore, we are currently working on applying the RevOnt approach in a different setting by using AMR graphs[3] as input. The chosen AMR graph is constructed from a multilingual corpus produced by the Polifonia project. This extension requires the annotation and preparation of data to be used as a training set for the language models and the retraining of the models with multilingual data. The results of the method will be used as a testbed for a Question Answering system.

As for XDTesting, as previously stated, the tool is developed based on the testing protocols of the eXtreme Design methodology. Even though XD complies with other ontology engineering methodologies, a direct future work is the experimentation of the tool with ontologies that are not built using this methodology. This experimentation would help identify new requirements for a third iteration, detect bugs, and make the tool more inclusive of other methodologies. A second direction is the extension of the tool automation beyond the GitHub platform. We are considering options such as self-hosted

[3]AMR graphs are a graph-based representation that aims to preserve semantic relations.

servers, cloud-hosted management, or expansion to other platforms such as GitLab[4]. Recent development of the tool attempts to dissociate from the platform by integrating all the automation in one script independent from the GitHub marketplace.

7.3 Conclusion

The use of knowledge graphs and ontologies has increased significantly in recent years, due to the growing need for better data management and integration. Many organizations are recognizing the advantages of implementing these technologies to increase data accessibility, consistency, and quality, as well as to aid in the creation of knowledge-based systems.

On the one hand, collecting requirements for ontologies is an important step in the ontology development process since it defines the ontology's structure and content. Traditional techniques of extracting requirements, such as surveys and interviews, take time and are prone to inaccuracies and misunderstandings. The creation of methods that use natural language processing algorithms and language models aids in extracting requirements from large volumes of (un)structured data (e.g., knowledge graphs). RevOnt is an approach that can extract competency questions from knowledge graphs with the means of multiple natural language processing models. The approach is based on the reverse engineering of an ontology development process and uses the verbalization of knowledge graph triples as input. RevOnt abstracts the verbalization, generates three questions for each triple, and filtrates the questions. The end result of the approach is a set of competency questions that represent the triples from which they were extracted, and the templates of the competency questions are mapped to SPARQL query templates. The RevOnt approach has been implemented using the WDV dataset. Based on a first evaluation, 75% of the abstractions generated by the first component of the approach, Verbalization Abstraction, have a good-to-high quality. Meanwhile, the questions generated by the second component, Question generation, have a wider range of quality, starting from poor to high. The type of question that received a higher score quality-wise is the one that RevOnt generates when the answer is the object of the triple. The

[4]`https://about.gitlab.com/`

RevOnt method directly answers RQ1 which asks whether an approach for eliciting requirements from knowledge graphs can be defined with the means of language models. To answer H1 regarding the quality features that a requirement must have in order to be classified as a competency question, the literature research concludes in the fact that a competency question is considered as such if it is able to be translated into a query that is used to question the respective ontology. As for H2, the evaluation through the implementation of the RevOnt method confirms that the quality of the elicited requirements that are generated with the means of language models from a knowledge graph is comparable to human-generated requirements.

On the other hand, despite the growing importance of ontologies in knowledge management, there is still a scarcity of comprehensive and user-friendly ontology testing tools. The test procedure is generally laborious and time-consuming, including checks for the ontology's completeness, consistency, and accuracy. As a result, ontology testing continues to be a bottleneck in knowledge graph engineering processes, and tools that automate the testing process are required. To fill a gap in the state-of-the-art, we defined the XDTesting method and implemented it in the form of a web application integrated with the GitHub platform. Referring RQ2, XDTesting reduces the workload of the ontology testers because it can create, annotate, execute, and document test cases automatically. The results of the test cases' execution are displayed graphically in the XDTesting Configurator and in more detail in the ontology fragment view. The complete documentation of the test case execution is stored in a standardized directory structure in GitHub. XDTesting has been evaluated as an acceptable, grade B, excellent tool according to the System Usability Scale questionnaire. The tool fulfills an immediate need for support in ontology testing. Furthermore, it contributes to the research community by encouraging the ontology testing process and making it a standard practice.

To conclude, requirement collection and ontology testing are critical steps in the knowledge graph engineering process, but they are frequently impeded by a lack of tool support. Traditional requirements elicitation methods and ontology testing require significant time and effort, and at times can result in errors and inconsistencies in the final ontology. The lack of tools for automating these steps hinders the effective application of knowledge graphs and ontologies across multiple domains. The development of tools to support the requirements elicitation process with the help of natural language processing algorithms, such as RevOnt, as well as to automate ontology testing, such as XDTesting, improves the quality and efficiency of the ontology engineering process. The value of these tools in the ontology engineering process cannot be overstated, as they might be vital to the development, documentation, and reuse of knowledge graphs and ontologies.

Bibliography

[1] Abdelghany Salah Abdelghany, Nagy Ramadan Darwish, and Hesham Ahmed Hefni. An agile methodology for ontology development. *International Journal Of Intelligent Engineering And Systems*. Volume **12**. pp. 170-181 (2019). `https://doi.org/10.22266/ijies2019.0430.17`.

[2] Aggarwal, Charu C., Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. *Lecture Notes in Computer Science*. pp. 420-434 (2001). `https://doi.org/10.1007/3-540-44503-x_27`.

[3] Amaral, Gabriel, Odinaldo Rodrigues, and Elena Simperl. ”WDV: A Broad Data Verbalisation Dataset Built from Wikidata.” *Lecture Notes in Computer Science*. pp. 556â574 (2022). `https://doi.org/10.1007/978-3-031-19433-7_32`.

[4] Amaral, Gabriel. WDV. `https://figshare.com/articles/dataset/WDV/17159045`. (2022) 10.6084/m9.figshare.17159045.v1 .

[5] Asim, Muhammad Nabeel, Muhammad Wasim, Muhammad Usman Ghani Khan, Waqar Mahmood, Hafiza Mahnoor Abbasi. A survey of ontology learning techniques and applications. *Database*. Volume **2018**. pp. 1-24 (2018). `https://doi.org/10.1093/database/bay101`.

[6] Asprino, Luigi, Valentina Anita Carriero, and Valentina Presutti. Extraction of common conceptual components from multiple ontologies. *Proceedings Of The 11th On Knowledge Capture Conference*. pp. 185-192 (2021). `https://doi.org/10.1145/3460210.3493542`.

[7] Auer, Sören, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. *Lecture Notes in Computer Science.* pp. 722-735 (2007). `https://doi.org/10.1007/978-3-540-76298-0_52`.

[8] Bezerra, Camila, Fred Freitas, and Filipe Santana. Evaluating ontologies with competency questions. *IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT).* Volume **3**. pp. 284-285 (2013). `https://doi.org/10.1109/wi-iat.2013.199`.

[9] Blomqvist, Eva, Karl Hammar, and Valentina Presutti. Engineering Ontologies with Patterns-The eXtreme Design Methodology. *Ontology Engineering With Ontology Design Patterns.* pp. 23-50 (2016).

[10] Blomqvist, Eva, Valentina Presutti, Enrico Daga, and Aldo Gangemi. Experimenting with eXtreme design. *International Conference On Knowledge Engineering And Knowledge Management.* pp. 120-134 (2010). `http://dx.doi.org/10.1007/978-3-642-16438-5_9`.

[11] Blomqvist, Eva, Azam Seil Sepour, and Valentina Presutti. Ontology testing-methodology and tool. *International Conference On Knowledge Engineering And Knowledge Management.* pp. 216-226 (2012). `http://dx.doi.org/10.1007/978-3-642-33876-2_20`.

[12] Bottini, Thomas, Valentina Anita Carriero, Jason Carvalho OU, Philippe Cathé, Fiorela Ciroku, Enrico Daga OU, Marilena Daquino et al. *Polifonia: a digital harmoniser for musical heritage knowledge, H2020.* (2021).

[13] Brank, Janez, Marko Grobelnik, and Dunja Mladenic. A survey of ontology evaluation techniques. *Proceedings Of The Conference On Data Mining And Data Warehouses (SiKDD 2005).* pp. 166-170 (2005).

[14] Brooke, John. SUS: a retrospective. *Journal Of Usability Studies.* Volume **8**, pp. 29-40 (2013).

[15] Carriero, Valentina Anita, Fabio Mariani, Andrea Giovanni Nuzzolese, Valentina Pasqual, and Valentina Presutti. Agile Knowledge Graph Testing with TESTaLOD. *ISWC Satellites*. pp. 221-224 (2019).

[16] Čebirić Šejla, François Goasdoué, Haridimos Kondylakis, Dimitris Kotzinos, Ioana Manolescu, Georgia Troullinou, and Mussab Zneika. Summarizing semantic graphs: a survey. *The VLDB Journal*. Volume **28**, pp. 295-327 (2019). https://doi.org/10.1007/s00778-018-0528-3.

[17] Corcho, Óscar, Asunción Gómez-Pérez, Rafael González-Cabero, and M. Carmen Suárez-Figueroa. ODEval: a tool for evaluating RDF (S), DAML+ OIL, and OWL concept taxonomies. *IFIP International Conference On Artificial Intelligence Applications And Innovations*. pp. 369-382 (2004). https://doi.org/10.1007/1-4020-8151-0_32.

[18] Corcho, Óscar, Mariano Fernández-López, Asunción Gómez-Pérez, and Angel López-Cima. Building legal ontologies with METHONTOLOGY and WebODE. *Law And The Semantic Web*. pp. 142-157 (2005). https://doi.org/10.1007/978-3-540-32253-5_9.

[19] Datetime - Basic Date and Time Types. Python Documentation, https://docs.python.org/3/library/datetime.html. Accessed 19 Jan. 2023.

[20] Dietz, Jan LG. What is Enterprise ontology?. *Springer Berlin Heidelberg*. pp. 7-13 (2006). https://doi.org/10.1007/3-540-33149-2_2.

[21] Fadel, Fadi George, Mark S. Fox, and Michael Grüninger. A generic enterprise resource ontology. *Proceedings Of 3rd IEEE Workshop On Enabling Technologies: Infrastructure For Collaborative Enterprises*. pp. 117-128 (1994). https://doi.org/10.1109/enabl.1994.330496.

[22] Fellbaum, Christiane. WordNet. *Theory And Applications Of Ontology: Computer Applications*. pp. 231-243 (2010). https://doi.org/10.1007/978-90-481-8847-5_10.

[23] Fernández-Izquierdo, Alba and Raúl García-Castro. Ontology verification testing using lexico-syntactic patterns. *Information Sciences*. Volume **582**. pp. 89-113 (2022). https://doi.org/10.20868/upm.thesis.66728.

[24] Fernández-Izquierdo, Alba and Raúl García-Castro. Themis: a tool for validating ontologies through requirements. *Proceedings of the 31st International Conference on Software Engineering and Knowledge Engineering*. pp. 573-753 (2019). https://doi.org/10.18293/seke2019-117.

[25] Fox, Mark S., Mihai Barbuceanu, and Michael Gruninger. An organisation ontology for enterprise modeling: Preliminary concepts for linking structure and behaviour. *Computers In Industry*. Volume **29**. pp. 123-134 (1996). https://doi.org/10.1016/0166-3615(95)00079-8.

[26] Gangemi, Aldo and Valentina Presutti. Ontology design patterns. *Handbook On Ontologies*. pp. 221-243 (2009), https://doi.org/10.1007/978-3-540-92673-3_10.

[27] Gangemi, Aldo, Carola Catenacci, Massimiliano Ciaramita, and Jos Lehmann. Modelling ontology evaluation and validation. *The Semantic Web: Research and Applications*. pp. 140-154. (2006). https://doi.org/10.1007/11762256_13.

[28] Gick, Mary L. and Keith J Holyoak. Schema induction and analogical transfer. *Cognitive Psychology*. Volume **15**. pp. 1-38 (1983). https://doi.org/10.1016/0010-0285(83)90002-6.

[29] Goguen, Joseph A. and Charlotte Linde. Techniques for requirements elicitation. *[1993] Proceedings Of The IEEE International Symposium On Requirements Engineering*. pp. 152-164 (1993). https://doi.org/10.1109/isre.1993.324822.

[30] Google Evaluating models - AutoML translation documentation. *Google*. https://cloud.google.com/translate/automl/docs/evaluate#:~:text=BLEU\%20(BiLingual\%20Evaluation\%20Understudy)\%20is,of\%20high\%20quality\%20reference\%20translations. (2022).

[31] Guarino, Nicola, and Christopher A Welty. An overview of OntoClean. *Handbook On Ontologies*. pp. 151-171 (2004). `https://doi.org/10.1007/978-3-540-24750-0_8`.

[32] Gruber, Thomas R. Ontolingua: A mechanism to support portable ontologies. *Citeseer*. (1992).

[33] Grüninger, Michael and Mark S. Fox. The role of competency questions in enterprise engineering. *Benchmarking - Theory And Practice*. pp. 22-31 (1995). `https://doi.org/10.1007/978-0-387-34847-6_3`.

[34] Hartmann, Jens, Peter Spyns, Alain Giboin, Diane Maynard, Roberto Cuel, Mc Suarez-Figeroa, and York Sure. Methods for Ontology Evaluation. *KnowledgeWeb Deliverable D1.2.3, Karlsruhe* (2005).

[35] Ioffe, Sergey and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference On Machine Learning*. pp. 448-456 (2015).

[36] Iqbal, Tabbassum and Mohammad Suaib. Requirement elicitation techniques: a review paper. *International Journal of Computer Mathematics*. Volume **3**, pp. 1-6 (2014).

[37] Jamil, Muhammad Abid, Muhammad Arif, Normi Sham Awang Abubakar, and Akhlaq Ahmad. Software testing techniques: A literature review. *In 2016 6th international conference on information and communication technology for the Muslim world (ICT4M)*. pp. 177-182 (2016).

[38] Ji, Qiu, Guilin Qi, Huan Gao, and Tianxing Wu. Survey on Schema Induction from Knowledge Graphs. *China Conference On Knowledge Graph And Semantic Computing*. pp. 136-142 (2018). `https://doi.org/10.1007/978-981-13-3146-6_12`.

[39] John, Santhosh, Nazaraf Shah, and Craig Stewart. Towards a Software Centric Approach for Ontology Development: Novel Methodology and its Application. *2018*

IEEE 15th International Conference On E-Business Engineering (ICEBE). pp. 139-146 (2018). `https://doi.org/10.1109/icebe.2018.00030`.

[40] Json - JSON Encoder and Decoder. Python Documentation, `https://docs.python.org/3/library/json.html`. Accessed 19 Jan. 2023.

[41] Keet, C. Maria, and Agnieszka Ławrynowicz. Test-driven development of ontologies. *European Semantic Web Conference*. pp. 642-657 (2016).

[42] Kellou-Menouer, Kenza, Nikolaos Kardoulakis, Georgia Troullinou, Zoubida Kedad, Dimitris Plexousakis, and Haridimos Kondylakis. A survey on semantic schema discovery. *The VLDB Journal*. Volume **31**. pp. 675-710 (2022). `https://doi.org/10.1007/s00778-021-00717-x`.

[43] Lantow, Birger. OntoMetrics: Putting Metrics into Use for Ontology Evaluation. *Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*. pp. 186-191 (2016). `https://doi.org/10.5220/0006084601860191`.

[44] Lavie, Alon. Evaluating the output of machine translation systems. *Proceedings Of The 9th Conference Of The Association For Machine Translation In The Americas: Tutorials*. (2010).

[45] Lawrynowicz, Agnieszka and C. Maria Keet. The TDDonto tool for test-driven development of DL knowledge bases. *CEUR-WS Proceedings*. (2016).

[46] Lenat, DB and RV Guha. Building large knowledge-based systems: Representation and inference in the CYC project. *Artificial Intelligence*. Volume **61**. pp. 4152 (1993).

[47] Loper, Edward and Steven Bird. Nltk: The natural language toolkit. *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*. (2002). `https://doi.org/10.3115/1118108.1118117`.

[48] Markdown Guide. `https://www.markdownguide.org/`. Accessed 30 Jan. 2023.

[49] McInnes, Leland, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*. Volume **2**. pp. 205 (2017). `https://doi.org/10.21105/joss.00205`.

[50] McInnes, Leland, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *Journal of Open Source Software*. Volume **3**. pp. 861 (2018). `https://doi.org/10.21105/joss.00861`.

[51] Mintz, Mike, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. *Proceedings Of The Joint Conference Of The 47th Annual Meeting Of The ACL And The 4th International Joint Conference On Natural Language Processing Of The AFNLP*. Volume **2**. pp. 1003-1011 (2009). `https://doi.org/10.3115/1690219.1690287`.

[52] Napoles, Courtney, Keisuke Sakaguchi, and Joel Tetreault. JFLEG: A fluency corpus and benchmark for grammatical error correction. *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*. Volume **2**. (2017). `https://doi.org/10.18653/v1/e17-2037`.

[53] Nuzzolese, Andrea Giovanni, Aldo Gangemi, Valentina Presutti, and Paolo Ciancarini. Encyclopedic knowledge patterns from wikipedia links. *Lecture Notes in Computer Science*. pp. 520-536 (2011). `https://doi.org/10.1007/978-3-642-25073-6_33`.

[54] Öhgren, Annika and Kurt Sandkuhl. Towards a methodology for ontology development in small and medium-sized enterprises. *IADIS International Conference Applied Computing*. pp. 369-376 (2005).

[55] Ontology-Related Tools and Services. `https://fairplus.github.io/the-fair-cookbook/content/recipes/interoperability/ontology-operations-tools.html`. Accessed 30 Jan. 2023.

[56] Os - Miscellaneous Operating System Interfaces. Python Documentation, `https://docs.python.org/3/library/os.html`. Accessed 19 Jan. 2023.

[57] Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. *Proceedings Of The 40th Annual Meeting Of The Association For Computational Linguistics*. pp. 311-318 (2001). https://doi.org/10.3115/1073083.1073135.

[58] Paschke, Adrian and Ralph Schäfermeier. OntoMaven-maven-based ontology development and management of distributed ontology repositories. *Synergies Between Knowledge Engineering And Software Engineering*. pp. 251-273 (2018). https://doi.org/10.1007/978-3-319-64161-4_12.

[59] Peroni, Silvio. A simplified agile methodology for ontology development. *OWL: Experiences And Directions - reasoner Evaluation*. pp. 55-69 (2017). https://doi.org/10.1007/978-3-319-54627-8_5.

[60] Piscopo, Alessandro and Elena Simperl. Who models the world? Collaborative ontology creation and user roles in Wikidata. *Proceedings Of The ACM On Human-Computer Interaction*. Volume **2**. pp. 1-18 (2018). https://doi.org/10.1145/3274410.

[61] Pouriyeh, Seyedamin, Mehdi Allahyari, Krys Kochut, and Hamid Reza Arabnia. A comprehensive survey of ontology summarization: measures and methods. *ArXiv Preprint ArXiv:1801.01937*. (2018).

[62] Poveda-Villalón, MarÃa, Asunción Gómez-Pérez and Mari Carmen Suárez-Figueroa. OOPS! (OntOlogy Pitfall Scanner!): An Online Tool for Ontology Evaluation. *International Journal On Semantic Web And Information Systems (IJSWIS)*. **10**, 7-34 (2014). https://doi.org/10.4018/ijswis.2014040102.

[63] Presutti, Valentina, Enrico Daga, Aldo Gangemi, and Eva Blomqvist. eXtreme design with content ontology design patterns. *Proc. Workshop On Ontology Patterns*. pp. 83-97 (2009).

[64] Raad, Joe and Christophe Cruz. A survey on ontology evaluation methods. *Proceedings Of The International Conference On Knowledge Engineering And Ontology Development, Part Of The 7th International Joint Conference On Knowledge*

Discovery, Knowledge Engineering And Knowledge Management. (2015). https://doi.org/10.5220/0005591001790186.

[65] Raffel, Colin, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J. Liu, and others. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research.* Volume **21**. pp. 1-67 (2020).

[66] Rajpurkar, Pranav, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 100,000+ questions for machine comprehension of text. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing.* (2016). https://doi.org/10.18653/v1/d16-1264.

[67] Rdflib 6.2.0 - Rdflib 6.2.0 Documentation. https://rdflib.readthedocs.io/en/stable/. Accessed 19 Jan. 2023.

[68] Re - Regular Expression Operations. Python Documentation, https://docs.python.org/3/library/re.html. Accessed 19 Jan. 2023.

[69] Ren, Yuan, Artemis Parvizi, Chris Mellish, Jeff Z Pan, Kees van Deemter, and Robert Stevens. Towards competency question-driven ontology authoring. *European Semantic Web Conference: Trends and Challenges.* pp. 752-767 (2014). https://doi.org/10.1007/978-3-319-07443-6_50.

[70] Requests: HTTP for Humans - Requests 2.28.2 Documentation. https://requests.readthedocs.io/en/latest/. Accessed 19 Jan. 2023.

[71] Reimers, Neils and Irina Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP).* (2019). https://doi.org/10.18653/v1/d19-1410.

[72] Resnik, Philip and Jimmy Lin. 11 evaluation of NLP systems. *The Handbook Of Computational Linguistics And Natural Language Processing.* Volume **57**. pp. 271-295 (2010). https://doi.org/10.1002/9781444324044.ch11.

[73] Roberts, Adam and Colin Raffel. Exploring transfer learning with t5: the text-to-text transfer transformer. *Google AI Blog - ArXiv Preprint ArXiv:arXiv:1910.10683.* (2020).

[74] Romero, Manuel. T5 (base) fine-tuned on SQUAD for QG via AP. *Hugging Face Hub.* `https://huggingface.co/mrm8488/t5-base-finetuned-question-generation-ap` (2021).

[75] Sawant, Abhijit A., Pranit H. Bari, and P. M. Chawan. Software testing techniques and strategies. *International Journal of Engineering Research and Applications (IJERA).* Volume **2**. pp. 980-986 (2012).

[76] Schekotihin, Konstantin, Patrick Rodler, Wolfgang Schmid, Matthew Horridge, and Tania Tudorache. Test-Driven Ontology Development in Protégé. *International Conference on Biological Ontology.* (2018).

[77] Sneha, Karuturi, and Gowda M. Malle. Research on software testing techniques and software automation testing tools. *In 2017 international conference on energy, communication, data analytics and soft computing (ICECDS).* pp. 77-81 (2017). `https://doi.org/10.1109/icecds.2017.8389562`.

[78] Spyns, Peter. Validating EvaLexon: validating a tool for evaluating automatically lexical triples mined from texts. *Lecture Notes in Computer Science.* pp. 11-12 (2007). `https://doi.org/10.1007/978-3-540-76888-3_6`.

[79] Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal Of Machine Learning Research.* Volume **15**. pp. 1929-1958 (2014).

[80] Stojanovic, Ljiljana, Nenad Stojanovic, Jorge Gonzalez, and Rudi Studer. OntoManager - a system for the usage-based ontology management. *OTM Confederated International Conferences "On The Move To Meaningful Internet Systems".* pp. 858-875 (2003). `https://doi.org/10.1007/978-3-540-39964-3_54`.

[81] Sure, York, Michael Erdmann, Juergen Angele, Steffen Staab, Rudi Studer, and Dirk Wenke. OntoEdit: Collaborative ontology development for the semantic web.

International Semantic Web Conference. pp. 221-235 (2002). https://doi.org/10.1007/3-540-48005-6_18.

[82] Suárez-Figueroa, Mari Carmen, Asunción Gómez-Pérez, and Mariano Fernández-López. The NeOn methodology for ontology engineering. *Ontology Engineering In A Networked World.* pp. 9-34 (2012). https://doi.org/10.1007/978-3-642-24794-1_2.

[83] Sys - System-Specific Parameters and Functions. Python Documentation, https://docs.python.org/3/library/sys.html. Accessed 19 Jan. 2023.

[84] Toutanova, Kristina and Danqi Chen. Observed versus latent features for knowledge base and text inference. *Proceedings Of The 3rd Workshop On Continuous Vector Space Models And Their Compositionality.* pp. 57-66 (2015). https://doi.org/10.18653/v1/w15-4007.

[85] Urllib - URL Handling Modules. Python Documentation, https://docs.python.org/3/library/urllib.html. Accessed 19 Jan. 2023.

[86] Uschold, Michael and Michael Grüninger. Ontologies: Principles, methods and applications. *The Knowledge Engineering Review.* Volume **11**. pp. 93-136 (1996). https://doi.org/10.1017/s0269888900007797.

[87] Uschold, Michael and Martin King. Towards a methodology for building ontologies. *Edinburgh: Artificial Intelligence Applications Institute, University of Edinburgh.* (1995).

[88] Vrandečić, Denny. Ontology evaluation. *Handbook On Ontologies.* pp. 293-313 (2009). https://doi.org/10.1007/978-3-540-92673-3_13.

[89] Vrandečić, Denny and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications Of The ACM.* Volume **57**. pp. 78-85 (2014). https://doi.org/10.1145/2629489.

[90] Welcome to Pytablewriter's Documentation! - Pytablewriter 0.64.1 Documentation. https://pytablewriter.readthedocs.io/en/stable/. Accessed 19 Jan. 2023.

[91] Wang, Zhichun, Qingsong Lv, Xiaohan Lan, and Yu Zhang. Cross-lingual knowledge graph alignment via graph convolutional networks. *Proceedings Of The 2018 Conference On Empirical Methods In Natural Language Processing.* pp. 349-357 (2018). `https://doi.org/10.18653/v1/d18-1032`.

[92] Wang, Zhiguo, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching for natural language sentences. *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence.* (2017). `https://doi.org/10.24963/ijcai.2017/579`.

[93] Wisniewski, Dawid, Jkedrzej Potoniec, and Agnieszka Lawrynowicz. BigCQ: A large-scale synthetic dataset of competency question patterns formalized into SPARQL-OWL query templates. *arXiv preprint arXiv:2105.09574.* (2021).

[94] Wisniewski, Dawid, Jkedrzej Potoniec, Agnieszka Lawrynowicz, and Maria Keet. Competency questions and SPARQL-OWL queries dataset and analysis. *ArXiv Preprint ArXiv:1811.09529.* (2018).

[95] Yasgui Documentation. Triply. `https://triply.cc/docs/yasgui`. Accessed 30 Jan. 2023.

[96] Zhang, Yan and Barbara M. Wildemuth. Unstructured interviews. *Applications Of Social Research Methods To Questions In Information And Library Science.* pp. 222-231 (2009).