

Hybrid Human-Machine Ontology Verification

Identifying Common Errors in Ontologies by Integrating Human Computation with Ontology Reasoners

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Alexander Prock, BSc

Matrikelnummer 01529065

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Projektass.(FWF) Reka Marta Sabou, MSc PhD

Mitwirkung: Ao.Univ.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Stefan Biffli

Wien, 2. September 2021

Alexander Prock

Reka Marta Sabou



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Hybrid Human-Machine Ontology Verification

Identifying Common Errors in Ontologies by Integrating Human Computation with Ontology Reasoners

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Alexander Prock, BSc

Registration Number 01529065

to the Faculty of Informatics

at the TU Wien

Advisor: Projektass.(FWF) Reka Marta Sabou, MSc PhD

Assistance: Ao.Univ.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Stefan Biffi

Vienna, 2nd September, 2021

Alexander Prock

Reka Marta Sabou



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Alexander Prock, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 2. September 2021

Alexander Prock



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

First, I would like to thank my supervisor Marta Sabou, for supporting me with her expertise, continuous guidance and valuable feedback during all stages of this thesis. I also want to express my gratitude to Stefan Biffel, for his scientific advice and constructive feedback, as well as his role in the initiation of this thesis.

I want to thank everybody involved in the MDRE research project, out of which the ideas for this thesis developed. Besides Stefan, I therefore want to thank Kristof Meixner and Dietmar Winkler, as well as my peers from the development team, Christian Engelbrecht, Christoph Burger, Dominik Kretz and Mustafa Isikoglu - those also for their companionship in the latter stages of my studies for this master's degree.

Furthermore, I want to thank the students that participated in the empirical study, and the researchers of the SemSys research group that provided feedback following the study's pilot.

Finally, I want to express my gratitude to my parents and friends, and especially my partner Lisi, for their ongoing support during my work on this thesis.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Ontologien sind eine Art semantische Ressource, die in Systemen basierend auf künstlicher Intelligenz bzw. wissensbasierten Systemen zum Einsatz kommen. Ontologien können auch als die Schemata von Knowledge Graphs betrachtet werden, die beispielsweise im Semantic Web genutzt werden, um Daten und Wissen zu integrieren. Fehler in Ontologien können Systeme, die entweder auf ihnen oder auf Knowledge Graphs basieren, zum Scheitern bringen oder dazu führen, dass diese inkorrekte Ergebnisse produzieren. Nachdem Fehler in Ontologien daher sehr kostspielige Konsequenzen haben können, ist es notwendig Ontologien zu verifizieren. Während einige Arten von Fehlern in Ontologien mittels (Reasoning-basierter) Algorithmen automatisch identifiziert werden können, ist häufig zusätzlich eine Verifizierung durch Menschen notwendig. Dafür kommt zumeist manuelle Batchverarbeitung zum Einsatz, wobei Techniken der Human Computation und Crowdsourcing verwendet werden. Diese Ansätze sind allerdings nicht effizient und skalieren nicht gut.

Diese Arbeit stellt eine kostensparende und besser skalierbare Methode für die Identifizierung von häufigen Modellierungsfehlern in Ontologien vor, die einen zweistufigen, hybriden Mensch-Maschine-Verifizierungsprozess verwendet. Im ersten Schritt wird ein Ontology Reasoner in Kombination mit speziell entwickelten Heuristiken verwendet, um automatisch Fehlerkandidaten zu entdecken. Diese Fehlerkandidaten werden dann im zweiten Schritt von Menschen unter der Verwendung von Techniken der Human Computation bzw. Crowdsourcing verifiziert. Der automatische erste Schritt stellt eine Vorauswahl von Klassen bzw. Kombinationen von Klassen dar, sogenannten “Bad Smells”, von denen es wahrscheinlich ist, dass sie Fehler enthalten, um den manuellen menschlichen Aufwand zu reduzieren.

Diese Arbeit leistet die folgenden Beiträge: (i) das Konzept von hybriden Mensch-Maschine-Verifizierungsprozessen für die Identifizierung bestimmter Arten von Modellierungsfehlern in Ontologien, (ii) ein Design für Human Computation Tasks, das geeignet ist, um in solchen Prozessen menschliches Urteilsvermögen einfließen zu lassen, (iii) Heuristiken für die Entdeckung von Fehlerkandidaten für vier ausgewählte Fehlertypen, (iv) den Entwurf einer Studie für die Evaluierung der präsentierten Methode und (v) Erkenntnisse über Einflussfaktoren auf die Effektivität der Methode. Um diese Beiträge zu leisten werden die Methoden der Literaturrecherche, der Algorithmusentwicklung, des Designs von Human Computation Tasks, der Entwicklung von Prototypen und des Studi-

endesigns angewandt, die entworfene empirische Studie ausgeführt und eine anschließende Datenanalyse basierend auf deskriptiven Statistiken durchgeführt.

Die Evaluierung dieses neuen Ansatzes anhand des Prototypen ist auf den Human Computation-Teil fokussiert, wobei die empirische Studie zeigt, dass 80,9 Prozent der gestreuten Modellierungsfehler und falsch-positiven Fehlerkandidaten korrekt von Menschen erkannt werden. In den Studienergebnissen werden Einflüsse der Art des vorhandenen Fehlers und des Vorwissens bzw. der Qualifikation der menschlichen Prüfenden auf die Verifizierungsgenauigkeit beobachtet. Weiters wird gezeigt, dass durch die Aggregation mehrerer Antworten mittels Mehrheitsentscheid eine signifikante Verbesserung der Verifizierungsgenauigkeit erreicht werden kann.

Abstract

Ontologies are a type of semantic resource, which are utilized in knowledge-based artificial intelligence systems, and can be seen as schemata for knowledge graphs, which are used to integrate data and knowledge, e.g. in the Semantic Web. Defects in ontologies can therefore cause systems based on either them, or knowledge graphs, to fail or to produce incorrect output, thus defects in ontologies may have very expensive consequences, implying the necessity of ontology verification. While several types of ontology defects can be identified through automatic (reasoning) algorithms, often additional human-based ontology verification is required. This is mostly achieved through batch processes using Human Computation (HC) and Crowdsourcing techniques, which however are not efficient and do not scale well.

This thesis proposes a cost-effective and more scalable method for identifying common modeling errors in ontologies, using a two-step hybrid human-machine verification process. In the first step, this process facilitates an ontology reasoner together with specifically designed heuristics to automatically detect defect candidates. These defect candidates are then verified by human workers in the second step using HC and Crowdsourcing techniques. The automatic first step performs a preselection of classes or class combinations that are likely to contain errors, so-called “bad smells”, reducing the amount of human labor needed.

This thesis makes the following contributions: (i) the concept of hybrid human-machine workflows for identifying specific types of ontology modeling errors, called *Defect Identification Workflows*, (ii) an HC task design suitable for collecting human judgement in these workflows, (iii) heuristics for detecting defect candidates for four selected error types, (iv) a study design for evaluating the proposed approach, and (v) insights on factors that influence the effectiveness of the approach. To make these contributions, a literature review is conducted, the methods of algorithm and HC task design, prototyping and study design are applied, the designed empirical study is executed, and subsequent data analysis is performed based on descriptive statistics.

The evaluation of this novel approach, using the prototype, focuses on the HC part, where the empirical study shows that 80.9 percent of the seeded modeling errors and false positives are correctly identified by human workers. Analyzing the evaluation results, influences of the error type present in a task and the qualification of the human verifiers

on the verification performance are observed. Furthermore, it is shown that aggregating multiple answers via majority voting significantly improves the verification performance.

Contents

Kurzfassung	ix
Abstract	xi
Contents	xiii
1 Introduction	1
1.1 Motivation	1
1.2 Research Questions	3
1.3 Approach, Methods and Contributions	3
1.4 Thesis Structure	6
2 Background and Related Work	9
2.1 Ontology Evaluation and Verification	9
2.2 Common Errors, Bad Smells and Anti-Patterns in Ontologies	13
2.3 Human Computation & Crowdsourcing	15
2.4 Hybrid Human-Machine Processes	20
2.5 Ontology Debugging	22
3 Hybrid Human-Machine Ontology Verification Method	25
3.1 Selected Error Types	25
3.2 Defect Identification Workflows	30
3.3 Defect Candidate Detection Heuristics	31
3.4 Human Defect Candidate Verification	39
3.5 Hybrid Human-Machine Workflows beyond Defect Identification	43
3.6 Prototype Implementation	45
4 Evaluation Setup	49
4.1 Evaluation Approach Overview	49
4.2 Participants	50
4.3 Data Set	50
4.4 Seeded Defects	52
4.5 Execution Overview	52
4.6 Measured Variables and Metrics	53
	xiii

5	Evaluation Results	59
5.1	Result Data	59
5.2	Influence of Prior Knowledge	60
5.3	Influence of Error Type	62
5.4	Problematic HITS	64
5.5	Time Spent per Task	66
5.6	Influence of Number of Aggregated Judgements	68
5.7	Cost Estimation	69
5.8	Feedback	71
6	Conclusion & Future Work	73
6.1	Summary	73
6.2	Conclusion	75
6.3	Limitations & Future Work	76
	List of Figures	81
	List of Tables	83
	List of Algorithms	85
	Acronyms	87
	Bibliography	89
	Appendices	95
	Appendix A: Quiz Guidelines Page	95
	Appendix B: Self-Assessment Form	96
	Appendix C: Qualification Test	98
	Appendix D: Instructions for Tutorial and Quiz	100
	Appendix E: Feedback Questionnaire	103
	Appendix F: Results of the Self-Assessment Form	104
	Appendix G: Results of the Feedback Questionnaire	107

Introduction

1.1 Motivation

Ontologies play a fundamental role in the semantic web technology stack and as such they are the basis of many modern intelligent systems. Studer et al. [57] define an ontology to be a “formal, explicit specification of a shared conceptualization”. Brank et al. [7] describe ontologies as explicit formal conceptualizations of domains of interest.

Artificial intelligence systems are often classified into the two categories of (i) systems that learn from data and (ii) systems based on encoded domain knowledge. Knowledge-based systems, and therefore ontology-based systems, fall under the second category. The correctness of the output, e.g. decisions and suggestions, of such systems depends on the correctness of the underlying knowledge base [43].

More recently, as an extension of ontologies, there has been increased interest in *knowledge graphs*. Paulheim [36] defines a minimum set of characteristics for a collection of knowledge to be considered a knowledge graph: “A *knowledge graph* (i) mainly describes real world entities and their interrelations, organized in a graph, (ii) defines possible classes and relations of entities in a schema, (iii) allows for potentially interrelating arbitrary entities with each other, and (iv) covers various topical domains.” According to this definition, ontologies without instances are not to be considered knowledge graphs, but can be seen as their schemata, consisting of the classes and relations of entities.

Ontologies are either created by ontology engineers or by automatic ontology learners, which create ontologies from source material, such as natural language text corpora [10]. Ontology engineers can make mistakes, in particular, novice ontology engineers typically introduce modeling errors of certain classes, e.g. omitting disjointness axioms or incorrectly using existential and universal quantification [42, 61]. Furthermore, the correctness of automatically learned ontologies depends on the quality of the source material and learning tool.

Ontologies may therefore contain errors. Errors in ontologies have a negative impact on the intelligent systems based on them, increasing the risk of faulty behaviour of such systems, or causing them to fail at reflecting diversity and therefore being perceived as biased. Ontologies thus need to be verified [7].

While the errors belonging to some error classes can be identified automatically, e.g. using structural pattern matching to detect specific erroneous patterns, or facilitating ontology reasoners to detect unsatisfiable classes, some errors require inspection by humans [40]. Rector et al. [42] give such an example with an ontology about pizzas and their toppings. As the classes `Meat` and `Vegetable` are not disjoint without the presence of an explicit disjointness axiom, a reasoner would not detect an error if another class `MeatyVegetable` was introduced, that is both a subclass of `Meat` and `Vegetable`. The reasoner would not detect an error, while a human could intuitively detect the missing disjointness axiom between `Meat` and `Vegetable`.

Humans are therefore needed to verify certain aspects of ontologies. As letting experts perform verification tasks can be expensive [40], in recent years HC techniques and Crowdsourcing have been used for ontology verification, as well as other data intensive tasks in the field of the semantic web [47].

Human Computation generally refers to having humans solve tasks that computers cannot (yet) solve reliably. Crowdsourcing is the concept of outsourcing tasks otherwise performed by designated workers, e.g. employees, to a large unknown population, often through the means of Crowdsourcing platforms [25].

While the application of Crowdsourcing to ontology verification may be less expensive than to employ experts, there still is a cost associated to it by the compensation payed to the human workers. This cost scales linearly with the size of the ontology under verification, as all ontology elements of relevant types need to be considered [31, 64]. As there exist ontologies with over 100.000 elements [31], verification of these large ontologies remains infeasible even when Crowdsourcing is applied.

In fields related to ontology verification, hybrid human-machine, resp. semi-automatic, approaches have emerged to improve scalability by combining automatic pre-processing steps with Crowdsourcing. The pre-processing steps thereby reduce the number of tasks to be crowdsourced, while the approach still benefits from human judgement. For example, such approaches exist for entity linking [13], ontology alignment [49] and linked data quality assessment [66].

This thesis therefore aims to integrate automatic machine computation, based on ontology reasoning, with HC and Crowdsourcing into hybrid human-machine workflows, in order to reduce the amount of human labor needed while still profiting from the strengths of HC, and thus to increase the scalability of ontology verification.

1.2 Research Questions

The first research question thus is concerned with finding an approach enabling the usage of automatic tasks combined with HC tasks for identifying specific types of errors that cannot (yet) be identified automatically.

RQ-1 *How can specific modeling errors in ontologies be identified using hybrid human-machine workflows?*

To answer this question, first a specific set of common ontology modeling error types is selected based on literature study of common errors in ontologies. Secondly, a method for the semi-automatic identification of selected error types is proposed as one of the novel contributions of this thesis, and thirdly, example workflows, each for the the identification of one of the selected error types, are designed and implemented as part of a prototype system.

The second research question concentrates on the HC part of the hybrid human-machine workflows, specifically on finding suitable HC interfaces that enable human verifiers to give their judgment in the workflows.

RQ-2 *What are suitable Human Computation interfaces to enable the verification of specific error types in hybrid human-machine workflows?*

Therefore, considerations regarding the task design inferred from literature are taken into account, an HC interface is designed, and the designed HC interface is used in an empirical study to assess its suitability.

The third and final research question aims to gain an understanding of the influence of different factors on the error detection rate of the hybrid human-machine workflows at detecting modeling errors of specific error types, as well as the time spent by human verifiers.

RQ-3 *Is there an influence of certain factors, such as (a) prior knowledge and qualification of the human workers, (b) the type of modeling error under verification, or (c) the number of human votes aggregated for crowdsourced judgements, on the error detection rate of the hybrid human-machine workflows for identifying specific types of modeling errors in ontologies, or the time spent by human verifiers thereby?*

To gain insights regarding possible influences of various factors on the error detection rate and time spent, data about these factors is collected during an empirical study using the implemented prototype and the combination of this data with the measured performance metrics is analyzed based on descriptive statistics.

1.3 Approach, Methods and Contributions

This thesis follows a *design science* approach [15, 22, 63] to answer its research questions and make its contributions, in particular the *Design Science Research Methodology* by

Peffers et al. [37] is applied. Hevner et al. [22] defines design science as follows: “*Design science [...] creates and evaluates IT artifacts intended to solve identified organizational problems*”.

The *Design Science Research Methodology* process [37] consists of six activities, which are not necessarily sequential, but rather can be iterated upon. These six activities and their realization in the scope of this thesis is described below and visualized in Figure 1.1.

1. The first activity is to identify the problem and to motivate the research. For this thesis, the identified problem is the lacking scalability of ontology verification approaches for types of errors that cannot (yet) be automated.
2. The second process step is to define the objectives of the new solution to be designed, in this thesis the objective is to reduce the amount of human labor needed for the verification of specific error types in ontologies to improve the scalability and cost-efficiency of ontology verification.
3. Thirdly, a new solution is designed and an artifact created. This thesis contributes the conceptual design of *DIWs*, which are semi-automatic workflows consisting of a heuristics-based detection of defect candidates and an HC-based verification of these detected defect candidates. Furthermore, such heuristics are designed to detect candidates for four selected types of ontology modeling errors, suitable HC interfaces are designed and these newly designed concepts are implemented in a prototype.
4. The fourth component is the demonstration of the newly designed solution by solving one or more problem instances. In this thesis, defects are seeded into the well-known pizza ontology, which is then used as the example problem instance.
5. The fifth process activity is the evaluation of the new solution, which is realized by means of an empirical study in this thesis.
6. The sixth step is the *Communication* of the new research contributions. This thesis communicates its findings and contributions in the following chapters and, as a master’s thesis, will be publicly available.

Out of the four possible research entry points discussed by Peffers et al. [37], this research falls into the category of seeking an objective-centered solution, as it focuses on finding a solution that is more scalable than existing approaches for identifying errors of specific modeling error types in ontologies.

The contributions made by this thesis, the methods applied to accomplish them and the relation to the research questions, stated in Section 1.2, are summarized in Figure 1.2.

RQ-1 is concerned with finding an approach to support an HC-based verification process for identifying specific error types with automatic tasks to reduce the amount of human

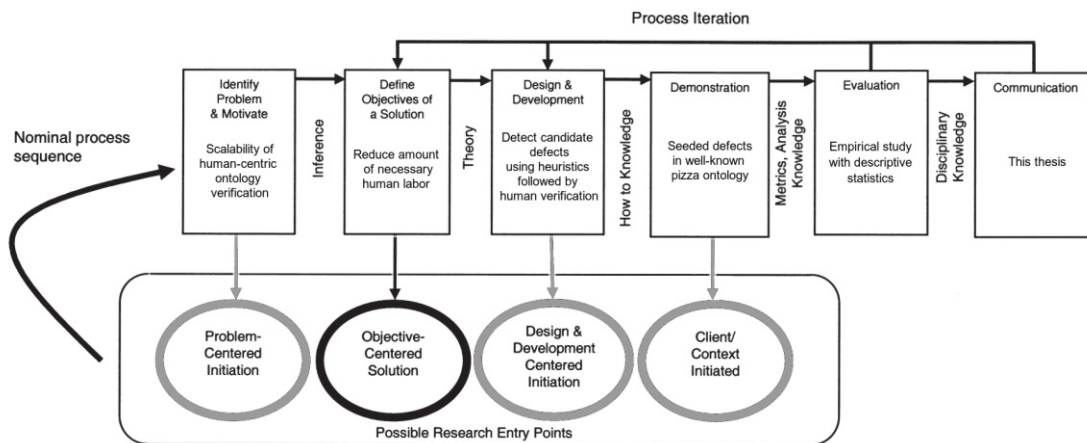


Figure 1.1: Design science research process of this thesis, based on [37, Fig.1]

labor required. Therefore a literature review was conducted, firstly to identify types of common ontology modeling errors, which should be targeted in the new hybrid processes and secondly to assess the state-of-the-art in ontology verification and to get inspiration from hybrid human-machine processes from other fields. After selecting the error types to target, heuristics were designed for the automatic detection of error candidates of the selected error types. To integrate these heuristics with HC techniques in hybrid human-machine workflows, the concept of DIWs was designed. The contributions made thereby are on the one hand the heuristics designed for the detection of defect candidates for four selected error types, described in Section 3.3, and on the other hand the concept of DIWs for the integration of these heuristics with an HC-based verification, see Section 3.2.

RQ-2 is concerned with suitable HC interfaces to be used in these hybrid workflows. Such interfaces were designed based on considerations derived from both the problem and literature. The arising contribution thus is a concrete task design for human computation tasks that allows the integration of human judgement into the hybrid human-machine workflows, described in detail in Section 3.4.1.

RQ-3 aims to assess the influence of certain factors on the error detection rate and time spent by humans in the designed workflows. Therefore, the proposed DIWs, including the heuristics for the selected error types and the designed HC interfaces, were implemented in a prototype, which was in turn used in an empirical study, which was designed for measuring the impact of the considered factors. Following the execution of the empirical study, descriptive statistics were calculated and the resulting data was analyzed to gain insights on the influences of the considered factors. The resulting contributions are the study design for the evaluation of the proposed approach, described in Chapter 4, and the insights on influencing factors gained thereby, discussed in detail in Chapter 5.

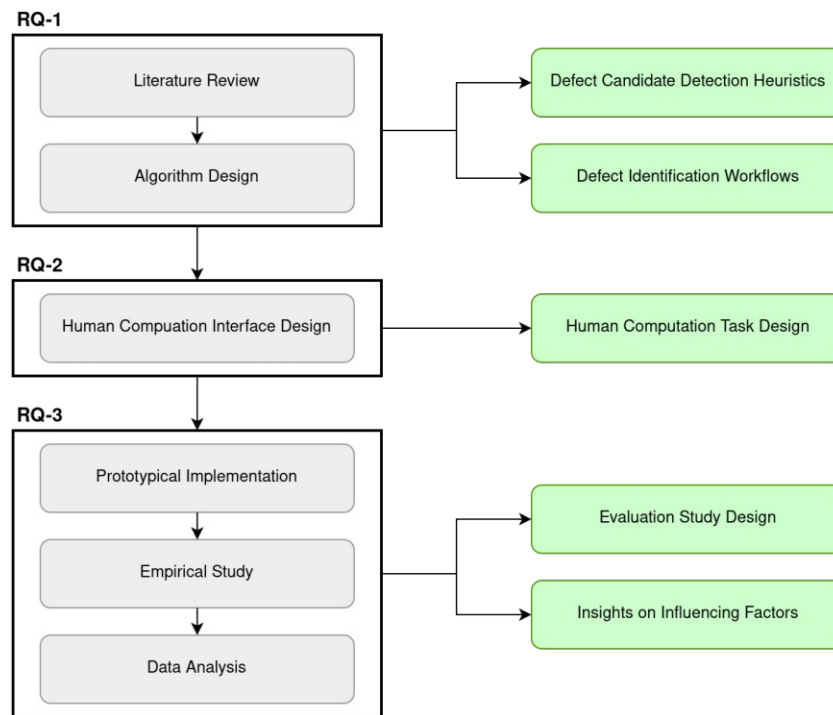


Figure 1.2: Methods and contributions of this thesis

1.4 Thesis Structure

This thesis is structured as follows:

- Chapter 2 *Background and Related Work* details the state-of-the-art and related work regarding ontology verification, common error types in ontologies, the application of HC and Crowdsourcing in ontology verification and existing hybrid human-machine approaches.
- Chapter 3 *Hybrid Human-Machine Ontology Verification Method* introduces the concept of DIWs, i.e. the hybrid human-machine workflows for detecting selected types of ontology modeling errors, consisting of heuristics for the detection of defect candidates and a Crowdsourcing-based verification of these candidates using specifically designed HC interfaces. Four common error types were selected and such heuristics, resp. workflows, were designed for their identification.
- Chapter 4 *Evaluation Setup* describes the approach to evaluate the proposed method, i.e. the details of the study design.
- Chapter 5 *Evaluation Results* provides the results of the executed study and a discussion thereof.

- Chapter 6 *Conclusion & Future Work* summarizes and concludes the findings of this thesis, sets them in the context of the research questions and discusses limitations and possible future research opportunities.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Background and Related Work

Ontologies are predominantly specified using description logics, a subset of first-order logic, using the *Web Ontology Language (OWL)* [35, 61]. Another notable semantic web technology mentioned in the following chapter is the *Resource Description Framework (RDF)* for representing linked data.

In order to design, implement and assess a novel hybrid human-machine ontology verification process, the existing work on ontology verification is analyzed, see Section 2.1.

Literature on common errors and other quality issues found in ontologies is elaborated upon in Section 2.2, with the goal of selecting relevant common error types for a pilot implementation of the concepts proposed in this thesis, i.e. to be able to design a semi-automatic detection approach for selected common error types.

To gain an understanding of the techniques and proven approaches to HC and Crowdsourcing in ontology verification and related fields, relevant literature is presented in Section 2.3.

Finally, existing hybrid human-machine processes are considered in Section 2.4 and the field of ontology debugging is analyzed regarding existing semi-automatic, resp. hybrid human-machine, approaches in Section 2.5.

2.1 Ontology Evaluation and Verification

Ontology evaluation is defined by Gómez-Pérez et al. [19] as “*a technical judgment of the content of the ontology with respect to a frame of reference*”. Ontology verification is a part of ontology evaluation, defined as “*building the ontology correctly, that is, ensuring that its definitions implement correctly the ontology requirements and competency questions, or function correctly in the real world.*” [19].

A survey of ontology evaluation techniques [7] identifies four main categories in which most evaluation approaches can be classified. The four identified categories are - each with example approaches from literature that can be classified in the respective category, to be discussed in detail below:

- comparing an ontology to a gold standard, which may also be an ontology, prepared by experts or taken from a corpus of documents [9, 30]
- using an ontology in an application and evaluating the application's results, therefore indirectly evaluating the ontology [16, 39, 56]
- comparing an ontology to source data from the covered domain [2, 8]
- evaluation by humans who verify whether, resp. how well, an ontology fulfills specified criteria: [18, 20, 62]

Gómez-Pérez, in another work [18], state that ontology verification includes the verification of: (i) each individual definition and axiom, (ii) the collection of explicitly stated definitions, resp. axioms, (iii) definitions imported from other ontologies, and (iv) axioms that can be inferred from the ontology. To guarantee the correctness of an ontology, they list that the following criteria, resp. properties, need to be checked: (i) soundness of ontology architecture, (ii) lexical and syntactical correctness, (iii) consistency, completeness, conciseness, expandability and sensitiveness of the content.

Soundness of architecture thereby means that the structure conforms to given design principles, *consistency* refers to the impossibility to infer contradictions, *completeness* means that all definitions are complete and all desired content is defined or can be inferred, *conciseness* is defined as “*whether all the information gathered in the ontology is useful and precise*”, *expandability* is concerned with the effort required for the addition or alteration of definitions without altering the properties guaranteed by previous verification and *sensitiveness* is given if small changes do not have a large effect on the other described properties.

Welty and Guarino [20, 62] proposed the *OntoClean* methodology for the ontological analysis of taxonomies based on notions of the philosophical discipline of ontology. Their work uses the term *property* in the sense of first-order logic predicates describing instances, an example for a such a property is *being an apple*. A class is the set of instances that have a certain property, e.g. the set of all things that are apples.

An example for such a philosophical notion is *rigidity*. A property is rigid if it is essential to all its instances, e.g. *being a human* is essential to all instances with that property, as they can never cease to be human. In contrast, a property is anti-rigid if it is essential to none of its instances, e.g. *being a student* is not essential to all its instances, as the instances will stop being students at some point.

The philosophical notions are expressed using meta-properties and assigned to a taxonomy's properties either by using a question/answer mode or by manually writing

assertions. In the question/answer mode the user is asked questions regarding the meta-properties of a property, to be answered with yes or no, e.g. whether a specific property is rigid. Inconsistencies regarding the assigned notions can be detected based on logical constraints, e.g. an anti-rigid property cannot subsume a rigid one. The proposed methodology helps to make the intended meaning of properties explicit, and to detect inconsistencies thereof, but requires human effort to assign the meta-properties to a taxonomy's properties.

Strasunskas and Tomassen [56] introduce the *Evaluation of Ontology Quality for Searching* framework to assess the fitness of ontologies for web search tasks, consisting of three steps: (i) generic quality evaluation, to filter out irrelevant ontologies and those of bad quality, including checking the syntactical correctness using a parser and the domain fitness using the AKTiveRank algorithm [2], (ii) search task fitness for three different types of search task (fact-finding, exploratory and comprehensive search), by calculating metrics based on the ratios of different relevant ontology elements present, and (iii) search enhancement capability to assess improvements using an ontology could have on search precision and recall, by calculating metrics based on the number of relevant synonyms and related terms present.

AKTiveRank, proposed by Alani et al. [2], is a metric calculated for an ontology stating how well it represents a set of given search terms to enable ranking ontologies by relevance. The total AKTiveRank score is based on four metrics: (i) *Class Match Measure*, based on the number of search terms (partially) present in an ontology, (ii) *Density Measure*, based on the number of direct subclasses, superclasses, relations and siblings of classes corresponding to search terms, (iii) *Semantic Similarity Measure*, based on the lengths of the shortest paths between the classes corresponding to search terms, and (iv) *Betweenness Measure*, based on the number of shortest paths between classes in the ontology passing through the classes corresponding to search terms, thus how central these are in an ontology. AKTiveRank does thus not evaluate the correctness of an ontology, but rather how well it represents a set of terms or concepts.

A similar data-driven approach is proposed by Brewster et al. [8], to evaluate the congruence of ontologies with text corpora and thus how well ontologies represent the domain of the given text corpora. This automatic domain-fitness evaluation consists of three steps: (i) identification of keywords from the given text corpus - in clusters, to preserve their closeness with each other, (ii) expansion of the clusters of keywords with hypernyms, and (iii) mapping of the keywords to the ontology to determine the coverage.

While both data-driven approaches [2, 8] described above represent automatic ways of ontology quality assessment, they do not perform ontology verification in the sense of detecting and correcting errors.

Burton-Jones et al. [9] propose a metrics suite for ontology auditing, including metrics for the syntactic, semantic, pragmatic and social quality. For syntactic quality, the syntactical correctness and the ratio of used to available syntactical features is taken into account. Regarding semantic quality, the *interpretability*, given by the ratio of

terms from the ontology's classes and properties that are also present in WordNet¹, and *consistency*, given by the ratio of inconsistent classes and properties, are taken into account. Pragmatic quality is measured by the *comprehensiveness*, derived from the ontology's size, *accuracy*, given by the number of true statements in the ontology by comparison to a gold standard, and *relevance*, given by the ratio of statements in the syntax of interest. Social quality is given by *authority*, measured as the number of links to an ontology, and *history*, as the number of accesses to the ontology. A sum of the metrics from the different quality levels, resp. their attributes, gives the overall quality score. Similar to the data-driven approaches above, the aim of this work is not to detect and repair errors, but to assess the quality of ontologies to make decisions about their usage in applications and to support the development of ontologies of high quality.

Maedche and Staab [30] present ontology similarity measures that can be used to quantify the lexical and conceptual similarity of two ontologies. Their approach can be used to measure the similarity to a gold standard ontology to verify that the contained concepts are modeled correctly, or to evaluate the relevance of other ontologies to a core ontology of interest. This approach, however, is not feasible for general ontology verification, since such a gold standard ontology will not be available in most cases.

Porzel and Malaka [39] propose an approach for evaluating ontologies based on their usage in applications to perform specific tasks, allowing the output of these applications after performing the tasks to be compared to a gold standard, resulting in a quantitative performance measure. By comparing the output to the gold standard, three types of errors can be detected: (i) insertion errors, i.e. superfluous concepts, or relations, (ii) deletion errors, i.e. missing concepts or relations, and (iii) substitution errors, i.e. replaced or ambiguous concepts or relations. The proposed approach is applied with the example task of tagging ontological relations in given text, whereas the entities are already marked-up. A gold standard consisting of texts and the correct tags is provided and an application that uses an example ontology to generate the relation tags is executed. Using this example task, superfluous, substituted and missing relation tags were detected with regard to the gold standard, whereas these errors have to be manually checked if they correspond to problems in the ontology.

Fernández et al. [16] measure the quality of an ontology by the quality of its individual relations, compared to a gold standard of triples. This can be categorized as a task-based, resp. application-based, approach, considering the task of assessing relation types. For each triple of the form *subject - relation - object* from the gold standard, the ontology under evaluation is checked if it has the correct type of relation between *subject* and *object*, whereas the possible types of relation are equivalence, subsumption, disjointness, sibling relationship and named relation.

In this section, ontology evaluation and verification were defined and the dimensions of ontology verification discussed. Existing approaches to ontology evaluation and verification were considered to give an overview on the state-of-the-art in traditional

¹<https://wordnet.princeton.edu/>

ontology evaluation and, more specifically, verification. Most of the presented literature is not concerned with identifying and repairing errors in ontologies, but rather presents general frameworks, assesses the fitness of ontologies for specific tasks and calculates metrics for quality assessment.

Those approaches that directly verify ontologies either rely on a gold standard for comparison [9, 30], which in most cases will have to be manually created, or let ontology engineers perform the verification [18, 20, 62]. Both of these options do not scale well as with the growing size of the ontologies to be verified, the manual verification, or the creation of a gold standard, become infeasible relying on a (likely small) group of ontology engineers. To give an example, ontologies with more than 100,000 elements are available in the biomedical domain [31], reaching a magnitude in which manual verification is impossible.

Therefore, more recent work utilizes HC and Crowdsourcing in more cost-effective approaches to ontology verification. Literature following this paradigm is discussed in Section 2.3. Before discussing said literature, the next section presents existing work on common errors, bad smells and anti-patterns in ontologies, to select worthwhile targets for identification.

2.2 Common Errors, Bad Smells and Anti-Patterns in Ontologies

Since the approach of this thesis is to improve the scalability of ontology verification by finding automatic processing steps that reduce the amount of human labor needed for identifying common types of ontology modeling errors, literature regarding ontology modeling errors needs to be considered. Related work on ontology modeling errors is considered to be able to select a set of relevant error types to design automatic processing steps for.

Rector et al. [42] describe a set of common errors made by novice users of OWL when constructing an ontology in the domain of pizzas. They list and discuss a number of common mistakes, classified by their cause of either confusion about the open world assumption, confusion about domain, range and other axioms, or logical issues. Furthermore, they developed guidelines for avoiding the described common errors during ontology modeling, including the recommendation to paraphrase class definitions using proposed sentence components, s.t. the precise meaning of OWL constructs can be understood more easily, and therefore errors spotted.

Warren et al. [61] present three studies that analyzed users' difficulties with understanding description logics, in particular with the popular Manchester OWL Syntax (MOS) [24, 32], a successor of the *Manchester House Style* used in [42]. MOS is also used in the popular Protégé ontology editor. The authors applied insights from cognitive psychology and language theory to understand the discovered difficulties with description logics, resp.

MOS, and proposed changes to keywords that were shown to be ambiguous in their studies.

Poveda-Villalón et al. [40] provide a categorized list of common pitfalls in ontology engineering based on an empirical analysis, ranked by their importance, as well as an online tool, called *OOPS! - Ontology Pitfall Scanner!* that scans uploaded ontologies for these pitfalls. A pitfall is defined to be a pattern of characteristics, which often represent a problem and include or lead to an error, which however must not always be the case in every ontology. The pitfall catalogue contains 40 pitfalls, out of which those 32 that do not require outside knowledge or information external to the ontology are implemented in the tool. OOPS! relies on structural pattern matching, lexical content analysis and search for specific characteristics to detect the selected pitfalls, but does not facilitate inference by the means of an ontology reasoner.

In a subsequent work [26], Keet and Poveda-Villalón evaluated ontologies using the OOPS! pitfall scanner to check the prevalence of different pitfalls and the correlation between the occurrence of pitfalls and other factors. They evaluated ontologies from three sets, (i) ontologies created by novice ontology engineers, (ii) well-known, resp. mature, ontologies, and (iii) random ontologies which were uploaded to and scanned with the pitfall scanner by users. The study showed that the ontologies created by novices tended to contain more pitfalls than those of the well-known set and that the size and complexity generally do not correlate with the number of pitfalls in them in a statistically significant manner, with the exception of those created by novice engineers, for which this is the case. In addition to this analysis, they provide guidelines for the prevention of the pitfalls from the catalogue. The pitfall catalogue is designed as a live and online collection², and since the publication of [40], one pitfall has been appended.

Roussey et al. [46] provide a catalogue of ontology anti-patterns found in inconsistent ontologies. The anti-patterns are classified in three groups:

- Logical anti-patterns, which can be detected by reasoners, e.g. adding an existential restriction to a class conflicting with an existing universal restriction on it
- Non-logical anti-patterns, which do not cause inconsistencies but are modeling errors, e.g. expressing synonyms as equivalent classes
- Guidelines, for complex expressions that can be simplified, e.g. grouping all restrictions on a class that use the same property into a single restriction

For this thesis, only the anti-patterns from the category of non-logical anti-patterns are of relevance, since logical anti-patterns can be detected automatically and guidelines for the simplification of ontology constructs do not imply errors in these constructs.

Sales and Guizzardi [23] define an “*ontological anti-pattern*” as a “*modeling pattern that, despite producing syntactically valid conceptual models, [...] is prone to be the source of*

²<http://oops.linkeddata.es/catalogue.jsp>

domain-related ontological misrepresentations”. An anti-pattern thus is a model structure that requires further attention, but not necessarily always represents, resp. leads to, an error. Furthermore, they define that, in addition to a defined structure, an anti-pattern also needs to have refactoring options or rectification plans associated with it. They note that this definition is based on both the definition of an anti-pattern in software engineering, as well as the concept of *code smells*, resp. *bad smells*.

This definition of anti-pattern differs from the usage in [46] and the concept of pitfalls in [40], as the intention behind the anti-patterns in [23] is not to support the correct building of models, but to support correctly modeling a domain. The authors of [23] furthermore extend an existing catalogue of anti-patterns for the OntoUML ontology modeling language.

Bad smells, resp. code smells, were introduced by Beck and Fowler [3], as certain structures of program code that suggest the need for their refactoring. Smells are used as an analogy, as in case of something smelling bad, there might be something wrong with it, which however must not always be the case, much like the code structures described by the authors, which are likely to require changes to improve understandability and maintainability, as well as to reduce the risk for bugs.

The concepts of *bad smells* and *pitfalls* are therefore roughly equivalent in ontology engineering.

This section discussed literature on ontology modeling errors, including a discussion of the used terminology. For this thesis four error types were selected based on the work by Rector et al. [42] and Warren et al. [61], the pitfall catalogue by Poveda-Villalón et al. [40], and the anti-pattern catalogue by Roussey et al. [46], see Section 3.1. The hybrid human-machine ontology verification process proposed in this thesis will be illustrated using these four selected error types, for which example verification processes were designed.

2.3 Human Computation & Crowdsourcing

HC and Crowdsourcing are often applied to ontology verification as a more scalable approach compared to human-centric batch processes executed by ontology engineering experts. This section gives definitions of the relevant terminology and proceeds to discuss related work that facilitates HC and Crowdsourcing in ontology verification.

The introduction in Section 1.1 already established the need for human reviewers in ontology verification.

The usage of HC techniques, as well as the benefits thereof for ontology verification, is addressed in multiple papers. Before discussing these papers, the established definitions for HC and Crowdsourcing are given.

Quinn and Bederson [41] list several definitions of HC, which can be condensed to it being the concept of letting humans solve tasks using their abilities that computers cannot

perform. Crowdsourcing is defined as “the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call” [25].

The two most popular approaches of HC are mechanised labor and Games with a Purpose (GWAP). With mechanised labor, human contributors are payed money to perform their tasks, making mechanised labor a similar concept to Crowdsourcing. The GWAP approach works by humans playing games and contributing through their actions, resp. their side-effects, within these games [48].

Given these definitions, relevant work that makes use of HC and Crowdsourcing techniques to increase the scalability of ontology engineering is elaborated upon.

Mortensen et al. [31] use Crowdsourcing as a scalable method to verify the hierarchical relationships in biomedical ontologies. They report good results for their approach, whereas the verification results of the Crowdsourcing approach are compared to a gold standard created by manual verification of the used ontology. They also compare the performance of workers that passed different qualification tests, or none at all. Furthermore, they estimate that between 5 and 10 non-spam responses are needed to verify a relation.

Wohlgenannt et al. [64] created a plugin for the popular ontology editor Protégé, integrating verification by means of Crowdsourcing into the ontology engineering process. The plugin allows the ontology engineer to send various verification tasks to a Crowdsourcing platform directly from the editor and presents the verification results to the ontology engineer as soon as they are available, also directly in the editor. Using this plugin, tasks for the verification of (i) domain relevance, (ii) subsumption relations, (iii) *instanceOf* relations, and (iv) domain and range of axioms, can be generated, as well as (v) tasks to collect suggested relation types for unlabeled relations. The evaluation’s results show that the overall costs of the ontology engineering project can be reduced using the plugin.

Nuzzolese et al. [33] automatically extracted Encyclopedic Knowledge Patterns (EKPs), which are a restricted form of ontologies, from Wikipedia and used them in an application for exploratory search. The EKPs themselves were evaluated by comparison to human-crafted EKPs as a gold standard and the application was evaluated regarding its ability to provide relevant information, whereas that aspect was compared to two other existing tools.

Roengsamut et al. propose using a gamification approach to construct and improve knowledge bases, creating a GWAP in the form of a quiz to construct and augment a multi-lingual knowledge base about rental apartments [45]. They also propose a Crowdsourcing-based protocol for ontology refinement in the same domain in another paper [44].

While the described approaches based on HC and Crowdsourcing are more cost-effective than traditional manual verification by ontology engineers, they still lack scalability regarding very large ontologies, as all ontology elements and axioms of relevant types

need to be considered. This thesis in contrast aims to reduce the amount of ontology elements and axioms that need to be verified by humans by performing an automatic preselection of error candidates.

2.3.1 Crowdsourcing Linked Data Quality Assessment

This section discusses existing approaches to the application of Crowdsourcing to linked data quality assessment, especially patterns used, which can to some extent be translated to the related field of ontology quality assessment, i.e. verification.

BetterRelations [21] is a GWAP for rating RDF triples by their importance to a topic. These relevance ratings are necessary to enable querying ranked result sets from linked data. Players are matched in pairs and are presented a topic, e.g. “Facebook”. The two players matched together are then shown two facts from the linked data source that have the presented topic as its subject, e.g. “Facebook has subject Online social networking” and “Facebook has key person Chris Hughes”, and have to select those of the two facts that the other player will likely think of first. In case of agreement, i.e. both players choosing the same fact, the relevance ratings of the two facts are updated. The evaluation showed that the ranked result lists produced from queries using the calculated relevance ratings are of similar or better quality than manual rankings by single humans.

Zaveri et al. [66] present a methodology for assessing the data quality of linked data resources using four steps, comprised of manual, semi-automatic and automatic workflows. In the first step, the resource to be evaluated is chosen, in the second step the evaluation mode is selected to either be manual, semi-automatic or automatic. The third step is the actual resource evaluation using the evaluation mode selected in the second step, followed by the fourth and final step of data quality improvement, either by directly editing linked data triples that contain problems, or indirectly by gathering user feedback in a *Patch Request Ontology*³, proposed in [27].

The process is partly implemented in a tool, allowing the evaluation of the manual and semi-automatic evaluation modes using Crowdsourcing. For the evaluation, resources from DBpedia [5], are assessed, and quality issues mapped to a defined quality problem taxonomy. The quality problem taxonomy contains types of linked data issues, for example issues caused by incorrect extraction from source data, incorrect links to websites or other data sets, redundant attribute values and irrelevant attribute values like layout information. In the manual evaluation mode, researchers experienced with RDF are presented a resource as a table with all of the triples belonging to the resource, as well as the link to the Wikipedia page of the presented resource. For each triple, the user has to check if it contains a problem, and in case of a problem, select the present problem from the defect taxonomy, with the possibility of adding problems to the taxonomy in case they are not present yet.

The semi-automatic evaluation mode consists of two steps, firstly schema axioms are learned from the data set using machine learning, and secondly, the generated axioms

³<http://141.89.225.43/patchr/ontologies/patchr.ttl#>

are verified by human experts. Once these axioms are verified, an ontology reasoner or SPARQL queries can be facilitated to find violations of these axioms. In the evaluation, the semi-automatic approach is executed and certain characteristics of properties are learned with a 95 percent confidence value.

From a sample of evaluation results, 81 percent of the evaluated triples were correctly evaluated using the manual evaluation mode. For each resource, and therefore triple, up to two humans gave their judgement. The measured inter-rater agreement (Cohen's kappa [11]) was quite low and the authors therefore conclude that more than two judgements should be collected in future evaluations.

The semi-automatic process yielded useful schema axioms, with all 24 resulting axioms for irreflexive properties being correct. While 81 axioms for property asymmetry were correct, these were also in some instances suggested for properties, which asymmetry is not correct for in all cases. Regarding the functionality of properties, many invalid facts could be detected using the 76 generated axioms for that characteristic. Furthermore, 13 properties were proposed to have inverse functionality.

Building on the work by Zaveri et al. [66], Acosta et al. [1] study the assessment of three specific linked data quality issues using Crowdsourcing. Aiming towards enabling future work to apply the *Find-Fix-Verify* pattern, in which a complex human task is broken up into three successive stages of simpler tasks, the paper focuses on applying two-step *Find-Verify* processes.

The *Find-Fix-Verify* pattern was originally introduced by Bernstein et al. [4]. The pattern “*splits complex crowd intelligence tasks into a series of generation and review stages that utilize independent agreement and voting to produce reliable results*”. In [4], the application area is text editing. The stages of the pattern are as follows:

1. *Find*: Workers identify parts that need further attention
2. *Fix*: Workers create a patch to fix a part that was identified in the previous stage
3. *Verify*: Workers assess the quality of the patches created in the *Fix* stage

Applying the pattern leads to two desirable properties:

- Small tasks: The tasks can be very small or even atomic, which reduces the risk of both lazy workers putting in minimal effort and skipping parts of larger tasks, and over-zealous workers providing too much detail or options, leading to increased effort by the task's requesters.
- Aggregation per stage: The answers for each stage can be aggregated, e.g. after the *Find* section only those parts identified by multiple workers could be fed into the next stage. From related work and the experimental evaluation, around 30 percent of the answer to Crowdsourcing tasks are found to be of poor quality. An

aggregation after each stage also decreases the impact of poor answers, e.g. if multiple answers are aggregated using majority voting.

The two-step *Find-Verify* process in [1] is designed in order to identify the three linked data quality issues (i) incorrect object, (ii) incorrect datatype or language tag, and (iii) incorrect link in RDF triples. Two such Crowdsourcing processes are compared, both of which use microtask Crowdsourcing with a layman crowd for the *Verify* stage, but one using a contest-based approach with experts and the other one again using microtask Crowdsourcing with a layman crowd for the *Find* stage. Following an experiment, which evaluated quality issues in DBpedia [5], a large data set of structured data extracted from Wikipedia, the authors concluded that Crowdsourcing is a feasible, cost-effective way to detect the studied quality issues.

Furthermore, the results suggest that experts should be assigned tasks that require specific-domain knowledge beyond common knowledge or technical knowledge, e.g. knowledge on the semantics of specific data types. However, the layman crowd workers showed to be skilled regarding the verification of the given language tags and comparisons between data values and given contextual information. For the quality issue of determining whether an external link is relevant to a resource, both Crowdsourcing using experts and laymen was infeasible, with the layman crowd even outperforming the experts. Thus both approaches, i.e. using either laymen or experts in the *Find* stage has its advantages, with them having complementary strengths.

In addition to the proposed Crowdsourcing-based processes a semi-automatic process is introduced, in which the *Find* stage is automated. The automated step used a *Test-Driven Quality Assessment* approach, originally introduced in [28] allowing the creation of test cases (i) automatically from schema constraints, (ii) semi-automatically derived from manually entered ontology constraints, and (iii) manually as SPARQL queries.

In the conclusion the authors of [1] summarize that while their Crowdsourcing approach is feasible, it may lack scalability for detecting linked data quality issues in large data sets, as triples are assessed individually. They suggest that the full potential of their approach could be reached when it is combined with automatic approaches, in order to reduce the amount of Crowdsourcing tasks necessary.

This section presented approaches to linked data quality assessment using Crowdsourcing and semi-automatic processes, which inspire the design of the semi-automatic, resp. hybrid human-machine, approach to ontology verification in this thesis. Especially the work of Zaveri et al. [66] and Acosta et al. [1] is highly relevant to this thesis, as they substantiate the approach of supporting Crowdsourcing-based processes with automatic preselection steps, e.g. by applying the *Find-Verify* pattern proposed by Bernstein et al. [4].

The next section therefore considers hybrid human-machine processes from other fields to gain further insight on the successful design of such processes.

2.4 Hybrid Human-Machine Processes

The previous section included literature concerned with semi-automatic approaches to linked data quality assessment, which provides valuable inspiration for the design of the hybrid ontology verification process. This section discusses related work on hybrid human-machine processes from other fields, to gain insights on design considerations, challenges and limitations.

Demartini [12] summarize the challenges, e.g. quality assurance, a cost/quality trade-off and the design of incentives for the involved humans and other limitations of Crowdsourcing, and opportunities, e.g. improving efficiency, effectiveness and scalability, of hybrid human-machine processes after providing an overview of existing hybrid human-machine systems.

Furthermore, they note that all existing hybrid human-machine systems that were considered used a human component for either pre-processing data or post-processing the output of automatic components.

In case a Crowdsourcing-based, resp. hybrid human-machine, application cannot rely on a general public crowd, e.g. because of sensitive content or the necessity of expert knowledge - the latter of which might be a concern with ontology verification - the work notes two possibilities. Firstly, Crowdsourcing can be carried out with internal crowds, e.g. the members of an organisation or the employees of a company [60] and secondly, combating the problem of required expert knowledge, crowds can be built by checking social media profiles to select suitable workers [6, 14].

In another work [13], Demartini et al. propose *ZenCrowd*, a system that combines automatic algorithmic and manual matching techniques for entity linking. Entity linking is the task of finding a linked open data Uniform Resource Identifier (URI) for an entity.

While both techniques for automatic and manual matching are available from related work, both have their strengths and weaknesses - automatic approaches being scalable and therefore suited for large-scale tasks, while manual approaches are more reliable but also less scalable, as human labor is required. The specific task in question is to annotate entities in HTML documents with a corresponding linked data URI.

Therefore, the system takes HTML pages as an input, automatically extracts entities using state-of-the-art tools and then uses algorithmic matching to find the most relevant URIs for the extracted entities, from a given set of linked data sources. Alongside the most relevant URIs, confidence values are yielded by the algorithmic matching. These confidence values are used by a decision making component, which may decide on one of three outcomes for each entity and its list of relevant URIs. For entities with URIs of very high confidence values, these links are stored in the database, i.e. treated as valid links. Possible links with low confidence values are discarded, thus in case all candidates for an entity are of low confidence, no entity links are produced.

If on the other hand, the confidence values of possible links are promising but uncertain, a microtask is created to decide on the validity of the links by means of Crowdsourcing.

These microtasks are carried out by humans, which are presented the entity, accompanied by context information (given a text snippets from the HTML pages), and are asked to check all candidate URIs if they match, with the possibility to select none. The results of the Crowdsourcing tasks are then handed back to the decision making component, which facilitates a probabilistic model to decide if links are deemed valid and saved, or not and thus discarded. The probabilistic model can also be adapted given preliminary results, e.g. to take the reliability of individual workers into account. Experimental results show that ZenCrowd achieved higher precision than both manual and automatic matching on their own. Furthermore, the proposed system constitutes a trade-off between large-scale automatic matching and high quality manual matching.

CrowdMap [49] is a hybrid human-machine approach to ontology alignment. Given two ontologies, CrowdMap uses an automatic mapping algorithm to produce a set of candidate mappings between the ontologies. These candidate mappings each represent a possible correspondence between two concepts, one from each ontology, and a proposed relation between them, e.g. equivalence, subsumption, or a domain-specific relation. Concepts can thereby be classes, properties or axioms. These candidate mappings are then verified by humans using Crowdsourcing, whereas a task is generated for each candidate mapping. Depending on the candidate generation algorithm, the tasks are either *validation tasks*, whereas human workers are asked to verify a proposed type of relation between two concepts, or *identification tasks*, whereas the type of relation between two concepts needs to be chosen. For validation tasks, CrowdMap requests three workers to verify the same mapping, for identification tasks up to seven votes are collected, until two workers agree on the type of relation.

One such HC task thereby consists of (i) title, explanation and instructions of the task, (ii) information about the concepts to be compared, including contextual information, and (iii) the form to be filled out by the workers, consisting of one or more input elements. As contextual information all available labels, definitions, super-, sub- and sibling concepts and instances of the two presented concepts are given.

For the experimental evaluation, seven such alignment tasks were combined into one Human Intelligence Task (HIT) on Amazon Mechanical Turk (MTurk). These combined HITs include quality assurance measures, i.e. each contains a question for which a gold standard is available to assess individual worker performance, as well as verification questions that force the workers to type out or select the name of a presented concept to reduce spam answers. The experiments showed that this hybrid human-machine approach to ontology alignment is feasible and a scalable, cost-effective alternative that can also improve accuracy compared to state-of-the-art approaches.

Shabani and Sokhn [51] present a hybrid human-machine approach to fake news detection. For the specific task of telling fake news and satirical stories from each other, the proposed hybrid approach achieves higher accuracy than both a Crowdsourcing-based and a purely machine learning-based one. The hybrid process consists of a machine learning step, a decision making model and an optional Crowdsourcing step. Firstly, the data is classified using several machine learning algorithms and the probability confidence is measured. If

the confidence values are below a certain threshold, crowdworkers are asked to classify the sample.

All hybrid human-machine approaches described in this section use an automatic pre-processing step that yields candidates of some sort, followed by a Crowdsourcing-based human verification of these candidates. The described approaches, however, extend beyond the *Find-Verify* pattern described in the previous section, as they facilitate confidence values to decide if a candidate needs to be verified by humans [13, 51], or perform quality assurance measures in their Crowdsourcing part [49]. Furthermore, the considerations regarding the HC task design presented in [49] are highly relevant to *RQ-2*.

2.5 Ontology Debugging

This section is concerned with ontology debugging and existing interactive, i.e. hybrid human-machine, approaches to it to get inspiration from applied techniques.

Ontology debugging is concerned with finding, understanding and therefore enabling the repair of undesired entailments in ontologies, e.g. unsatisfiable classes, resp. inconsistencies, and is therefore a related field to ontology verification [29].

According to Rodler et al. [43], the two main knowledge base debugging techniques are (i) model-based and (ii) heuristic approaches. Heuristic approaches use handcrafted pattern matching procedures or search techniques and while they are more efficient than model-based approaches, they may be incomplete and/or unsound. Model-based approaches yield a set of axioms for a given inconsistency, called a diagnosis. This diagnosis is the explanation, resp. justification, for the inconsistency, calculated by a reasoner. The reasoning task of calculating justifications is called *axiom pinpointing* [38].

ORE [29] is a tool for ontology enrichment and repair, using ontology debugging techniques to allow the repair. It facilitates automated learning of subsumption and equivalent class axioms from the knowledge base's individuals to enrich ontologies with. For the repair functionality, ORE relies on the Pellet reasoner [52] and its incremental reasoning feature, as well as an algorithm that can be configured to stop loading the knowledge base after the schema part with sample individuals are loaded, in order to enable reasoning on very large web ontologies, such as DBpedia [5], which similar tools cannot do.

Rodler et al. [43] evaluate query-based and test case based ontology debugging techniques in a series of user studies. Test case based debugging allows ontology engineers to specify test cases, i.e. certain entailments, that a repaired ontology should satisfy, s.t. the debugging process can focus on solutions that fulfill this requirement. Query-based ontology debugging on the other hand poses queries for the user to answer, from which test cases are then automatically formulated. Both query-based and test case based ontology debugging are types of model-based debugging. The studies concluded that both types of model-based debugging are effective, as users were able to find large fractions of faults in a given ontologies using either variant. Query-based ontology debugging was however found to be more efficient than the test based approach. In addition the

user studies reveal that users sometimes provide wrong information during interactive debugging, which should be taken into account in future work.

While ontology debugging has a different goal than the ontology verification approach central to this thesis, as it does not try to identify errors, but to find the reasons for specific errors at hand, its model-based approaches rely on ontology reasoners and axiom pinpointing, which can also be used in automatic error candidate detection.

To conclude this chapter, the main findings from the literature study are recapitulated. Section 2.1 defined ontology evaluation and verification and discussed traditional human-centric approaches to them, which all lack scalability, especially regarding the verification of very large ontologies. Section 2.2 discusses common errors found in ontologies, considers multiple collections of errors and anti-patterns to select the error types targeted in this thesis from, and clears up the terminology in this regard. Section 2.3 discussed the application of HC and Crowdsourcing in ontology verification and linked data quality assessment, concluding that these are still not scalable enough for the verification of very large ontologies, as the whole ontologies need to be considered. In the field of linked data quality assessment, there however exist semi-automatic approaches, from which the *Find-Verify* pattern is taken for the design of the processes in this thesis. Section 2.4 considered hybrid human-machine processes from other fields, providing considerations on the challenges and design of such processes and further substantiating the suitability of the *Find-Verify* pattern. Finally, Section 2.5 briefly discussed ontology debugging and applied techniques, from which the inspiration to rely on ontology reasoners and axiom pinpointing for the automatic part of the hybrid human-machine processes is taken, as well as delimiting the scope of this thesis from ontology debugging.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Hybrid Human-Machine Ontology Verification Method

This chapter describes the novel *Hybrid Human-Machine Ontology Verification (HOV) Method*, central to this thesis, for detecting common errors in ontologies in order to answer *RQ-1*, which seeks a concept to enable the identification of specific types of ontology modeling error using hybrid human-machine workflows.

As shown in Figure 1.2, input for the design of such a method comes from literature as discussed in Chapter 2.

The proposed HOV method aims to improve the scalability of ontology verification by reducing the amount of human labor necessary for the detection of common types of ontology modeling errors.

Therefore, four common error types were selected to be targeted, see Section 3.1. Section 3.2 describes the concept of *Defect Identification Workflows (DIWs)*, which are the workflows that integrate the automatic pre-processing with HC and Crowdsourcing elements for the verification of the automatically detected defect candidates. Section 3.3 then describes the heuristics that were designed to automatically detect *defect candidates* of the four chosen types. Section 3.4 describes the designed HC interfaces for this verification. Finally, Section 3.5 discusses how the proposed workflows could be extended to go beyond defect identification and Section 3.6 describes the implemented prototype.

3.1 Selected Error Types

Based on the related work regarding common errors in ontology modeling, presented in Section 2.2, four error types were selected to be targeted for identification using the new hybrid verification method. In the following the four chosen error types are described and the rationale for their selection explained.

The error types were selected based on them occurring in multiple considered sources [40, 42, 46, 61] and their importance ratings from that sources where available. While the fourth selected error type “*Missing Closure Axioms*” is only present in one source [42], it was selected because the other sources contain multiple error types regarding quantified restrictions and this error type is an example of these.

In order to illustrate examples for the four selected error types below, a base example is introduced. The example is inspired by the well-known pizza ontology [42], which is also used in the empirical study, see Section 4.3. Consider the class `CardinalePizza`, representing a specific type of pizza called *Cardinale* which has tomatoes, mozzarella and ham as its toppings. A properly modelled example is given in Visual Notation for OWL Ontologies (VOWL) [59] in Figure 3.1. This proper model is simplified, as within a pizza ontology, this class would have other axioms defining it in addition to the presented ones, e.g. it would be declared as a subclass of the class `Pizza`. There are three existential, resp. `someValuesFrom`, restrictions on the `CardinalePizza`, each on the property `hasTopping`, for each one of the required toppings. These existential restrictions ensure that an individual is topped with the three required toppings to be considered a *Cardinale*. Furthermore, there is one universal, resp. `allValuesFrom`, restriction, with the union of the three toppings as its filler, i.e. a *Cardinale* can only have toppings from the union of ham, mozzarella and tomatoes, thus no other kind of topping like pineapples or onions.

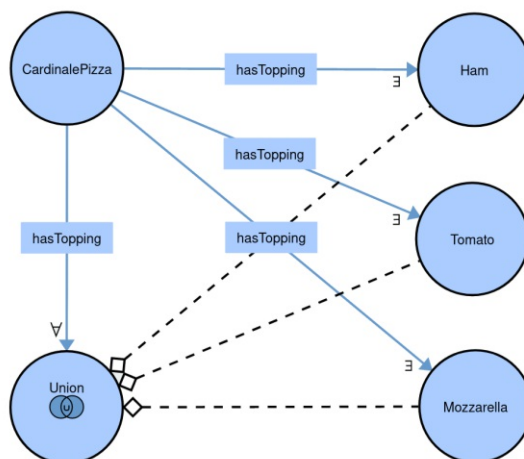


Figure 3.1: Correctly modelled class `CardinalePizza`

3.1.1 Missing Disjointness Axioms

This type of error is described by Rector et al. [42] and part of the pitfall catalogue by Poveda-Villalón et al. [40]. In the pitfall catalogue, this type of error is labeled “P10” and categorized as “Important”, which is the medium importance level used. Furthermore it is the fifth-most detected pitfall in the OOPS! ontology pitfall scanner [40].

In OWL *Open World Reasoning* is applied, meaning that as long as something is not unsatisfiable given the explicitly stated facts and inferences, it is assumed to be true. This also holds for disjointness, which can be used to declare that no individual can be part of given two or more classes simultaneously, i.e. that the sets of instances of the given classes are disjoint from each other. Two classes are therefore overlapping by default, i.e. without any explicit disjointness axiom between them. Rector et al. [42] state that the omission of disjointness axioms is one of the most common errors in building ontologies.

An example for this is given by Rector et al. [42], again from the domain of pizzas. If the two classes *Meat* and *Vegetable* are not explicitly declared as disjoint by a disjointness axiom, they are assumed to be overlapping based on open world reasoning. Therefore an individual could be both meat and vegetable at the same time, or a class *MeatyVegetable* could be introduced as a subclass of both classes *Meat* and *Vegetable*. As nothing is both a meat and a vegetable at the same time, this disjointness should be declared to avoid unwanted classifications and inferences when using the ontology. This example is visualized in Figure 3.2, on the left side the disjointness axiom is correctly included, on the right side it is missing.

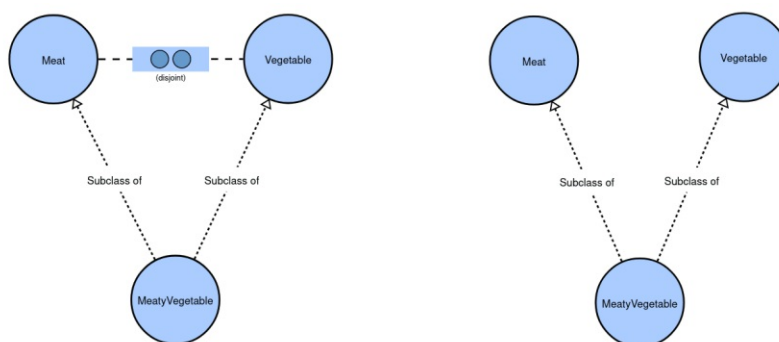


Figure 3.2: Example for missing disjointness axioms

3.1.2 Confusion between logical and linguistic “and”

The confusion between the logical and linguistic meaning of “and”, and also “or”, is discussed by Rector et al. [42], Warren et al. [61] - where it is part of the broader problem labeled “*the ambiguity of natural language*” - and the anti-pattern catalogue by Roussey et al. [46], where it is called “*AndIsOr*”.

This error is caused by the fact that “*In common linguistic usage, “and” and “or” do not correspond consistently to logical conjunction and disjunction respectively*” [42].

Furthermore, Rector et al. [42] give the sentence “*Find all of the Pizzas containing Fish and Meat*” as an example. It is ambiguous whether the answer should contain all pizzas that contain *either Meat or Fish, or both* - corresponding to disjunction - or all pizzas containing *both Meat and Fish* - corresponding to conjunction.

Errors of this type may occur in case a concept is translated too literally from natural language into OWL definitions. In this thesis, the focus is on confusions between the logical and linguistic meaning of “and”.

As another example, if the `CardinalePizza` introduced above was described as a pizza “that has only ham, mozzarella and tomatoes on it” and this “and” was directly translated into a conjunction, i.e. an intersection, and assuming that the classes for ham, mozzarella and tomatoes are properly declared as disjoint, the class `CardinalePizza` would become unsatisfiable. This incorrect approach to modeling the *Cardinale* is depicted in Figure 3.3. The unsatisfiability is caused by the universal restriction requiring each topping to be ham, mozzarella and tomato *at the same time*, which is impossible due to these classes being disjoint and nonsensical from a domain point-of-view.

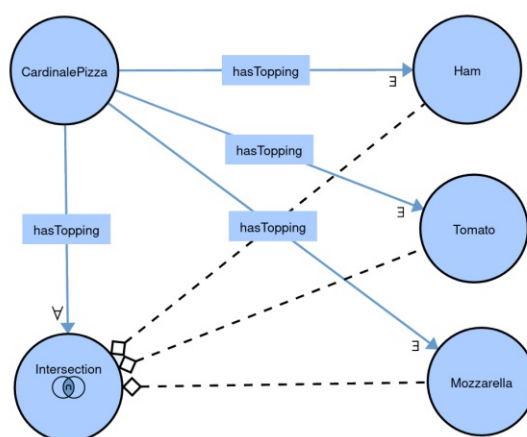


Figure 3.3: Class `CardinalePizza` with a confusion between the logical and linguistic meaning of “and”

3.1.3 Trivially satisfiable allValuesFrom Restrictions

This type of error is discussed by both Rector et al. [42] and Warren et al. [61], and is part of the pitfall catalogue by Poveda-Villalón et al. [40] as one of the possibilities causing pitfall “P14” (in addition to incorrect closure axioms). Said pitfall “P14” is classified as “Critical”, the highest of the used importance levels. This is one of the pitfalls from the catalogue which the ontology pitfall scanner does not detect automatically.

If a class is defined using a universal, i.e. `allValuesFrom`, restriction on a property with a certain filler, this means that instances of this class can only be linked with classes from the filler using the given property. Such an `allValuesFrom` restriction is satisfied by either all occurrences of the property on an instance having a filler conforming to the restriction’s filler, or the absence of instances of this property. The latter case is called the *trivial satisfiability* of the `allValuesFrom` restrictions. In most cases this trivial satisfiability is undesired and indicates an error.

A possible explanation for the frequent occurrence of this error type is given in [42], suggesting that the implication of an existential restriction by a universal restriction may be assumed by novice ontology engineers.

Revisiting the `CardinalePizza` as an example, if it was modelled without the existential restrictions, a pizza without any toppings would be considered a valid *Cardinale*, as the universal restriction only defines which classes are allowed as toppings, but there is no restriction that actually requires the existence of any topping at all. See Figure 3.4 for a graphical representation of this example.

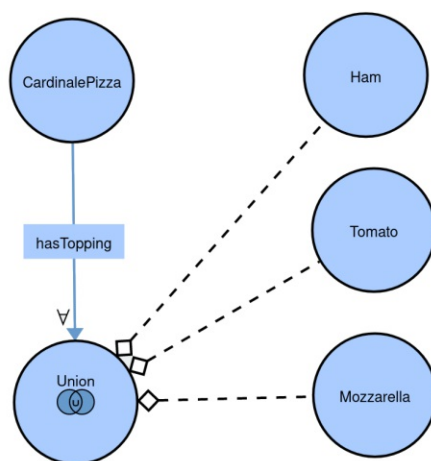


Figure 3.4: Class `CardinalePizza` without existential restrictions allowing the trivial satisfiability of its universal restriction

3.1.4 Missing Closure Axioms

Revisiting the `CardinalePizza` once again, modeling it without the universal restriction, which is a so-called *closure axiom*, would allow any pizza with ham, mozzarella and tomatoes on it to be a valid *Cardinale*, regardless of possible other toppings. Thus a *Hawaiian* pizza, which is topped with pineapples in addition to ham, mozzarella and tomatoes is a valid *Cardinale* in the absence of the closure axiom. This incorrect approach to modeling the `CardinalePizza` is depicted in Figure 3.5.

This error can again be caused by disregarding the *Open World Assumption*. The often incorrectly assumed opposite *Closed World Assumption* would imply that no other toppings were possible, as this possibility is not explicitly stated.

This explanation and example are inspired by Rector et al. [42] who deem problems with open world reasoning to be one of the largest sources of difficulties for new users of OWL.

This section described the four error types selected as the targets for the hybrid human-machine error detection. The next section describes the method for their identification

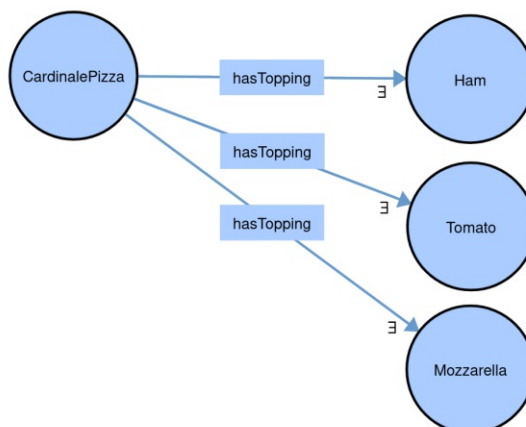


Figure 3.5: Class `CardinalePizza` without a closure axiom

and Section 3.3 explains the heuristics that were designed to detect possible errors of the described types.

3.2 Defect Identification Workflows

The previous section discussed the selection of the error types targeted for verification. This section describes the approach of *Defect Identification Workflow (DIWs)* for their identification by integrating automatic defect candidate detection with HC and Crowdsourcing tasks for the verification of the defect candidates to utilize human judgement for the classification of the defect candidates as either true defects or false positives. The automatic defect candidate detection is thereby based on heuristics and described in detail in Section 3.3. Section 3.4 describes the HC step in more detail.

As described in Section 2.3.1, Acosta et al. [1] designed a two-step *Find-Verify* process to identify three specific types of linked data quality issues in semantic web resources. The authors implemented this process in two manners: (i) Crowdsourcing in both the *Find* phase and the *Verify* phase, and (ii) an automated *Find* phase with a Crowdsourcing-based *Verify* phase.

In their conclusion they suggested that an automated approach in the *Find* stage of a quality assessment process should be used to improve the scalability of the process, reducing the amount of human labor, resp. HC tasks, needed.

The process designed in this thesis for defect identification by means of HOV follows a similar concept to the *Find-Verify* process used in [1] for detecting linked data quality issues, facilitating an automated approach in the *Find* stage.

It is also conceptually similar to the CrowdMap [49] approach to ontology alignment, which also consists of an automated candidate generation, followed by a Crowdsourcing-based verification of these candidates.

The designed process thus consists of the following two conceptual steps:

1. **Automatic Defect Candidate Detection** - analogous to the *Find* stage
2. **HC Defect Candidate Verification** - analogous to the *Verify* stage

The process therefore starts with an ontology as the input and in the first step utilizes the defect candidate detection heuristics described in the previous section to automatically detect ontology classes, resp. class combinations, that are likely to contain errors. These defect candidates are then verified by human judgement using HC and Crowdsourcing techniques, classifying each defect candidate either as a true defect or a false positive.

A realization of this process for the detection of one specific error type is called a *Defect Identification Workflow*. Such a workflow has the purpose of identifying errors of one selected error type using the heuristic that was designed for the chosen error type, see the next section.

See Figure 3.6 for a visualization of the proposed process. The first step receives the ontology as its input and uses the defect candidate detection heuristics to detect defect candidates, i.e. possible errors, and yields them as its result. The second step creates HC tasks, possibly on a Crowdsourcing platform, following a specific task design to present the defect candidates - one by one - to human verifiers, in order to let them give their judgement to distinguish the defect candidates into verified defects and false positives.

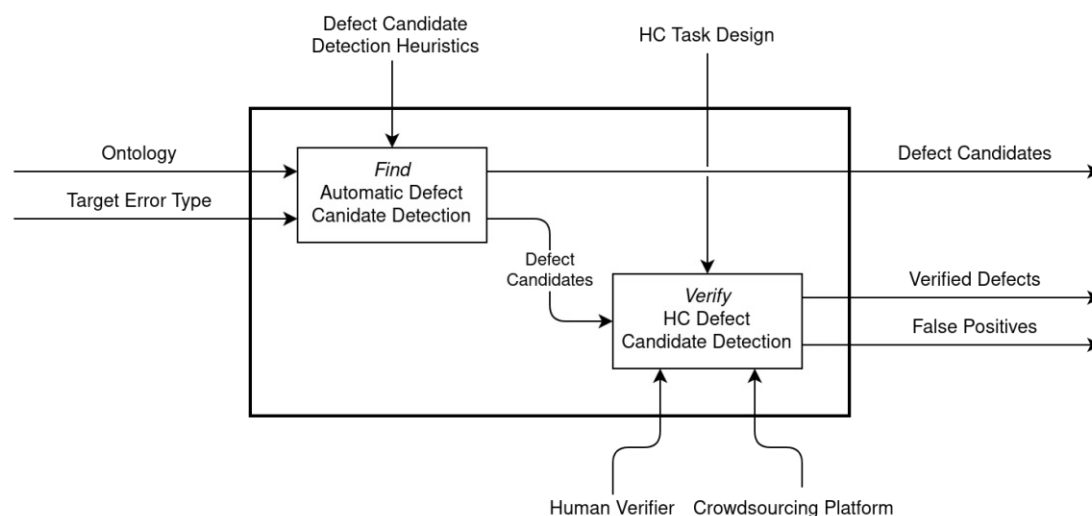


Figure 3.6: Conceptual overview of Defect Identification Workflows as an IDEF-0 diagram

3.3 Defect Candidate Detection Heuristics

In order to detect possible ontology modeling errors, so-called *defect candidates*, of the four chosen error types, a heuristic approach is chosen, as these error types cannot be

detected automatically.

A defect candidate thereby is either a single ontology class or a combination of ontology classes that likely contain an error. As noted in Section 2.2, the term “defect candidate” may therefore be regarded as synonymous to the terms “bad smell” and “pitfall”.

As described in Section 3.2, these defect candidates will be verified by humans in a subsequent step. To enable human verification of the detected defect candidates, they should include details about the classes that are suspected to cause a defect, such as the class axioms and labels, as well as human-readable context information to allow humans to understand the intended domain information encoded, and therefore enable a more informed verification.

The inclusion of context information is recommended by a research manifesto concerned with Crowdsourcing for the semantic web [50]. The context information may either be sourced from the ontology itself, e.g. using annotation property values, or from external sources, e.g. glossaries or encyclopedias.

A heuristic for the detection of a specific error type may for example be based on structural pattern matching, search for specific characteristics on ontology elements, facilitate an ontology reasoner, perform any other kind of computation, or any combination of these to select defect candidates.

In the following, for each one of the four selected error types from Section 3.1, a heuristic detecting defect candidates for that type is presented.

As all four heuristics described below rely on the capabilities of an ontology reasoner to some extent, the consistency of the ontology is a precondition to proper defect candidate detection, because reasoning cannot be performed in a useful manner on inconsistent ontologies.

3.3.1 Missing Disjointness Axioms

The basic idea behind the heuristic for detecting candidates for missing disjointness axioms is to check all possible pairs of satisfiable primitive classes that are not disjoint from each other. This by itself would already be a useful heuristic, which however would yield a large amount of defect candidates in case a disjointness axiom is missing between classes at the top of subclass hierarchies. Therefore, the proposed heuristic reduces the resulting set of defect candidates by aggregating multiple candidates that share common superclasses. Furthermore, a defect candidate is only emitted if the introduction of the disjointness axiom that it describes as missing would not make the ontology inconsistent or increase the number of unsatisfiable classes in the ontology.

The conceptual components of the heuristic are visualized in Figure 3.7 to provide an overview and the heuristic is formalized in Algorithm 3.1.

Using this heuristic for the example of the classes `Meat` and `Vegetable` from the pizza ontology if they were not declared disjoint, but the rest of the pizza ontology

Algorithm 3.1: Heuristic for detecting candidates of missing disjointness axioms

Input: A consistent ontology O , an ontology reasoner R and a function $superclasses(c)$ that returns the set of superclasses of the given class c from the ontology O inferred by R

Output: Defect candidates for missing disjointness axioms as a set of pairs of classes

```

1  $C =$  set of satisfiable primitive class from  $R$  with input  $O$ ;
2  $P = \emptyset$ ;
3 for  $i \leftarrow 1$  to  $|C|$  do
4   for  $j \leftarrow i + 1$  to  $|C|$  do
5      $c_i = get(C, i)$ ,  $c_j = get(C, j)$ ;
6      $D_{c_i} =$  set of classes disjoint from class  $c_i$  inferred by  $R$ ;
7     if  $c_j \notin D_{c_i} \wedge c_i \notin superclasses(c_j) \wedge c_j \notin superclasses(c_i)$  then
8       | add pair  $(c_i, c_j)$  to  $P$ ;
9     end
10  end
11 end
12  $D = \emptyset$ ;
13 for  $i \leftarrow 1$  to  $|C|$  do
14    $c_i = get(C, i)$ ;
15    $C_{c_i} = \{c_j | (c_i, c_j) \in P \vee (c_j, c_i) \in P\}$ ;
16   for  $j \leftarrow 1$  to  $|C_{c_i}|$  do
17     for  $k \leftarrow j + 1$  to  $|C_{c_i}|$  do
18        $c_j = get(C_{c_i}, j)$ ,  $c_k = get(C_{c_i}, k)$ ;
19       add  $superclasses(c_j) \cap superclasses(c_k)$  to  $C_{c_i}$ ;
20     end
21   end
22    $C_{c_i} = \{c | c \in C_{c_i} \wedge \neg \exists s. s \in C_{c_i} \wedge s \in superclasses(c)\}$ ;
23   add  $\{(c_i, x) | x \in C_{c_i}\}$  to  $D$ ;
24 end
25  $D = \{(c_i, c_j) | (c_i, c_j) \in D \wedge ((c_i, c_j) \notin D \vee i < j) \wedge (\neg \exists c_k. c_k \in$ 
    $superclasses(c_j) \wedge ((c_i, c_k) \in D \vee (c_k, c_i) \in D))\}$ ;
26 foreach  $(c_i, c_j) \in D$  do
27   add disjointness axiom between  $c_i$  and  $c_j$  to  $O$ ;
28    $cons =$  check if  $O$  is consistent using  $R$ ;
29    $C' =$  set of satisfiable primitive class from  $R$  with input  $O$ ;
30   if  $\neg cons \vee |C'| < |C|$  then
31     | remove  $(c_i, c_j)$  from  $D$ ;
32   end
33   remove disjointness axiom between  $c_i$  and  $c_j$  from  $O$ ;
34 end
35 return  $D$ 

```

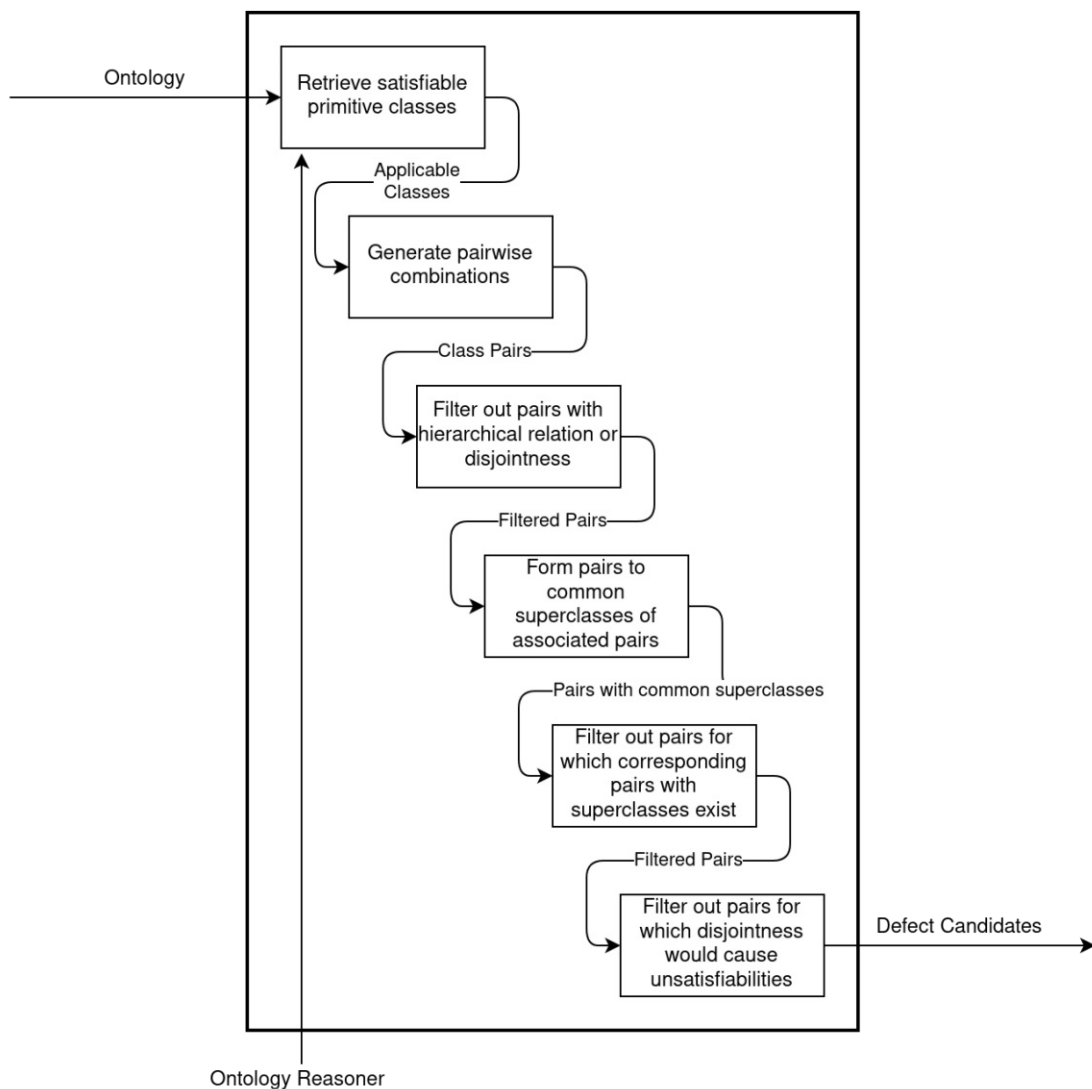



Figure 3.7: Overview on the components of the heuristic for detecting candidates for missing disjointness axioms as an IDEF-0 diagram

properly modelled, a defect candidate for the class combination of these two classes would be emitted. As they are not disjoint from each other and not related via an inferred superclass property, pairs for the two classes and all combinations of their subclasses would be generated. Pairs of one of the two classes and a subclass of the other would be filtered out, and either the pair $(Meat, Vegetable)$ or $(Vegetable, Meat)$ would remain, depending on the comparison of an arbitrary identifying property (indicated by $i < j$ in line 25 of Algorithm 3.1). Under the assumption that the rest of the pizza ontology is correctly modelled, the introduction of the disjointness axiom between *Meat* and *Vegetable* would not cause an inconsistency or lead to more unsatisfiable classes, and

therefore a defect candidate consisting of these two classes would be returned.

3.3.2 Confusion between logical and linguistic “and”

While the keyword “and” in the Manchester OWL Syntax and many query languages represents a conjunction, this is not always the case in natural language, i.e. the linguistic meaning can differ from the logical semantics in some situations.

In case the definition of a concept uses “and” in its linguistic meaning but a direct translation to a conjunction in the definition of an OWL class leads to the intended class definition differing from the actual one, this will likely lead to an unsatisfiable intersection. The idea of this heuristic for detecting such confusions is therefore to detect classes that contain unsatisfiable intersections.

The heuristic is formalized in Algorithm 3.2.

Algorithm 3.2: Heuristic for detecting candidates of confusions between the logical and linguistic meaning of “and”

Input: A consistent ontology O , an ontology reasoner R , which is capable of calculating explanations, and a function $satisfiable(expr)$ that returns \top or \perp if the given expression $expr$ is satisfiable in the ontology O using R

Output: List of classes that are defect candidates for a confusion between the logical and linguistic meaning of “and”

```

1  $U$  = set of unsatisfiable classes from  $R$  with input  $O$ ;
2  $D = \emptyset$ ;
3 foreach  $u \in U$  do
4    $A$  = set of axioms comprising a minimal explanation of the unsatisfiability of
    $u$  using  $R$ ;
5   foreach  $a \in A$  do
6      $I$  = set of intersections contained in  $a$ ;
7     if  $\exists i. i \in I \wedge \neg satisfiable(i)$  then
8       | add  $u$  to  $D$ ;
9     end
10  end
11 end
12 return  $D$ 

```

The example of the `CardinalePizza` with an intersection in its closure axiom instead of a union depicted in Figure 3.3 would be reported as a defect candidate by this heuristic. Assuming that the classes for ham, tomato and mozzarella are correctly declared as disjoint from each other, the universal restriction with the intersection and therefore the `CardinalePizza` would be unsatisfiable. The heuristic would detect the unsatisfiable `CardinalePizza`, retrieve a minimal explanation for its unsatisfiability, which will always contain the universal restriction with the intersection and one of

the disjointness axioms between the toppings and as the intersection contained in the universal restriction is itself unsatisfiable, the heuristic will return a defect candidate for the class `CardinalePizza`.

3.3.3 Trivially satisfiable `allValuesFrom` Restrictions

The idea behind the heuristic for detecting trivially satisfiable `allValuesFrom`, i.e. universal, restrictions is to find all satisfiable primitive classes that have an `allValuesFrom` restriction, but no “corresponding” `someValuesFrom`, i.e. existential, restriction. To be deemed “corresponding”, such a `someValuesFrom` restriction must be on the same property as the `allValuesFrom` restriction, or a subproperty of it, and have a filler containing a class that is either equal, a subclass or a superclass of a class occurring in the filler of the `allValuesFrom` restriction.

It should be noted that the definition of “corresponding” `someValuesFrom` restrictions does not take semantics into account, e.g. a defect candidate would be emitted even if the negated form of the `someValuesFrom` restriction’s filler occurs in the `allValuesFrom` restriction’s filler.

The described heuristic is formalized in Algorithm 3.3.

The example error for trivially satisfiable `allValuesFrom` restrictions from Section 3.1 depicted in Figure 3.4, i.e. the `CardinalePizza` without the existential restrictions, is detected by the described heuristic, as the class has an `allValuesFrom` restriction, but no corresponding `someValuesFrom` restriction (as it has none at all).

3.3.4 Missing Closure Axioms

The heuristic for detecting candidates for missing closure axioms is similar to the one for trivially satisfiable universal restrictions, as it also obtains classes with one type of quantified object property restriction and looks for corresponding ones of the other type. For this heuristic, all primitive satisfiable classes with `someValuesFrom` restrictions are checked for corresponding `allValuesFrom` restrictions, i.e. so-called closure axioms. `someValuesFrom` restrictions on functional properties are excluded from the defect candidate detection of missing closures as for them the range of the property defines the possible values, and there cannot be any instances of the property apart from the one required by the `someValuesFrom` restriction on the inspected class. An `allValuesFrom` restriction thereby is deemed corresponding if it is on the same property as the `someValuesFrom` restriction, or a superproperty of it. If no corresponding universal restriction is found, the class is emitted as a defect candidate for missing closure.

The filler of the `someValuesFrom` and `allValuesFrom` restrictions do not need to be compared in this heuristic because an incompatible filler would lead to the class being unsatisfiable and thus not considered by this heuristic.

A formalization of the described heuristic is given in Algorithm 3.4.

Algorithm 3.3: Heuristic for detecting candidates of trivially satisfiable universal restrictions

Input: A consistent ontology O , an ontology reasoner R , a function $subproperties(p)$ that returns the set of subproperties of the given property p from the ontology O inferred by R and functions $subclasses(c)$ and $superclasses(c)$ that return the set of subclasses, resp. superclasses, of the given class c from the ontology O inferred by R

Output: List of classes that are defect candidates for trivially satisfiable allValuesFrom restrictions

```

1  $C$  = set of satisfiable primitive classes from  $R$  with input  $O$ ;
2  $D = \emptyset$ ;
3 foreach  $c \in C$  do
4    $U$  = set of allValuesFrom restrictions on  $c$ , inferred by  $R$ ;
5    $E$  = set of someValuesFrom restrictions on  $c$ , inferred by  $R$ ;
6   foreach  $u \in U$  do
7      $prop_u$  = property of restriction  $u$ ;
8      $F_u$  = classes in signature of filler of  $u$ ;
9      $F'_u = F_u \cup \{c_{sub} | \exists c_f.c_f \in F_u \wedge c_{sub} \in subclasses(c_f)\} \cup \{c_{sup} | \exists c_f.c_f \in F_u \wedge c_{sup} \in superclasses(c_f)\}$ ;
10     $corr = \perp$ ;
11    foreach  $e \in E$  do
12       $prop_e$  = property of restriction  $e$ ;
13       $F_e$  = classes in signature of filler of  $e$ ;
14      if  $(prop_e = prop_u \vee prop_e \in subproperties(prop_u)) \wedge F'_u \cap F_e \neq \emptyset$ 
15        then
16           $corr = \top$ ;
17          break
18        end
19    end
20    if  $\neg corr$  then
21      add  $c$  to  $D$ ;
22      break
23    end
24 end
25 return  $D$ 

```

Algorithm 3.4: Heuristic for detecting candidates of missing closure axioms

Input: A consistent ontology O , an ontology reasoner R , a function $superproperties(p)$ that returns the set of superproperties of the given property p from the ontology O inferred by R and a function $functional(p)$ that returns \top or \perp if the given property p is functional

Output: List of classes that are defect candidates for missing closure axioms

```
1  $C$  = set of satisfiable primitive classes from  $R$  with input  $O$ ;  
2  $D = \emptyset$ ;  
3 foreach  $c \in C$  do  
4    $E$  = set of someValuesFrom restrictions on  $c$ , inferred by  $R$ ;  
5    $U$  = set of allValuesFrom restrictions on  $c$ , inferred by  $R$ ;  
6   foreach  $e \in E$  do  
7      $prop_e$  = property of restriction  $e$ ;  
8     if  $\neg functional(prop_e)$  then  
9        $corr = \perp$ ;  
10      foreach  $u \in U$  do  
11         $prop_u$  = property of restriction  $u$ ;  
12        if  $prop_u = prop_e \vee prop_u \in superproperties(prop_e)$  then  
13           $corr = \top$ ;  
14          break  
15        end  
16      end  
17      if  $\neg corr$  then  
18        add  $c$  to  $D$ ;  
19        break  
20      end  
21    end  
22  end  
23 end  
24 return  $D$ 
```

The example class `CardinalePizza` visualized without a closure axiom in Figure 3.5 would be detected by this heuristic and a defect candidate for the class would be returned. The class would be detected because it has a `someValuesFrom` restriction on the property `hasTopping`, but no `allValuesFrom` restriction at all, thus none corresponding to the `someValuesFrom` restriction.

This section has introduced heuristics for the automatic detection of defect candidates for each one of the four selected error types.

3.4 Human Defect Candidate Verification

The defect candidates detected in the automatic heuristic-based first step are then verified by humans by the means of Crowdsourcing. For each of the defect candidates, human judgement decides whether the candidate is a true defect, or a false positive.

Each defect candidate is therefore presented in an HC task to be verified by humans.

This section describes the task design used and the considerations thereof in Section 3.4.1. Furthermore, the number of judgments to collect for each task and the strategy for their aggregation is discussed in Section 3.4.2.

3.4.1 Human Computation Task Design

There are multiple options for the design of an HC task that verifies defect candidates. In any case, the defect candidate must be presented to the human verifiers, whereas the affected classes, their definitions and context information should be included.

There are three main decisions to be made for the task design, based on the task design used in CrowdMap [49], discussed in Section 2.4:

- **Question phrasing and answer options**

The answer options need to be discrete, or at least from a defined range, in any case, to enable for them to be easily processed in later steps, as well as the result aggregation in case of multiple votes collected per task, see below. Collecting answers using free-text fields would make such automated processing and aggregation more complex or even impossible.

The answer options could be a binary choice, to create a *validation task* by the terms of [49], i.e. if the defect candidate represents a true defect of a specific error type or not. Alternatively, there could be one answer option per error type (and one to mark false positives), to form a *identification task* by the terms of [49], allowing human verifiers to select an error type different from the one for which the heuristic produced the defect candidate, e.g. in case a combination of classes contain multiple errors, a human verifier could select the most important one, or the one that is the root cause of possible other problems. Furthermore, letting the human verifiers select the type of error present (or mark the task as a false

positive) can be useful to test if the human verifiers are able to distinguish between certain errors types, in order to assess the general ability of the proposed method and actual implementation to detect true defects on a specific data set.

- **Presentation of ontology elements**

A task needs to include a presentation of the ontology elements which are part of the defect candidate. There are multiple options:

- Axioms in an OWL syntax: A simple approach to presenting OWL axioms in a human-readable form is to render them in an OWL syntax. There are multiple OWL syntaxes available [34], most notably the functional-style syntax, which is used in the OWL specification [35], and MOS [24, 32] which has the explicit purpose of making it easy for humans to read/write description logic ontologies [34].
- Natural language paraphrases: Both Rector et al. [42] and Warren et al. [61] suggest paraphrasing OWL constructs, as they are easier to comprehend than “raw” axioms rendered in an OWL syntax.
- Graphical visualizations: Ontologies can be represented graphically, e.g. using *concept diagrams* and *property diagrams*, introduced by Stapleton et al. [55], or using VOWL [59].

- **Context information**

As described previously, [50] recommends presenting context information in Crowdsourcing tasks. This context may either come from the ontology itself, e.g. from various annotation properties, or from external sources such as online glossaries, encyclopedias, or semantic resources like DBpedia [5]. The presented context information can either be embedded into the task itself, e.g. by displaying text values of annotation properties or loading referenced resources like linked images. Alternatively, links can also be given as context, e.g. to images or Wikipedia articles.

After the consideration of these design options, a design for HC interfaces used for the tasks in the verification part of the DIWs was created.

Figure 3.8 shows an example task for an HC task for the verification of a defect candidate consisting of a single class and Figure 3.9 shows one for a defect candidate consisting of two classes. Both examples are taken from the study described in Chapter 4 using the well-known pizza ontology.

In the presented task design, each class definition - there may be one or more per defect candidate, resp. task - is rendered in three parts:

- **Class name**

The value of the metadata annotation property `skos:prefLabel` is used as the class name here. For a description of the used property see Section 4.3.2.

[View instructions](#)

Class Name: Mushroom

Description and Synonyms for Mushroom

- Mushroom Pizza
- A pizza topped with tomatoes, mozzarella and mushrooms.

Class Axioms

- Mushroom SubClassOf hasTopping some MozzarellaTopping
- Mushroom SubClassOf hasTopping some MushroomTopping
- Mushroom SubClassOf NamedPizza
- Mushroom SubClassOf hasTopping some TomatoTopping
- Mushroom DisjointWith (each of) Rosa, American, Veneziana, Capricciosa, FruttiDiMare, LaReine, SloppyGiuseppe, Napoletana, VegetarianaPizza, Siciliana, Fiorentina, Parmense, Margherita, QuattroFormaggi, Cajun, Caprina, Soho, PrinceCarlo, PolloAdAstra, Giardiniera, FourSeasons, AmericanHot

Which of the following statements holds for the given class(es): Mushroom?

- A disjointness axiom is missing for the given class(es).
- The given class(es) contain(s) a confusion between the logical and linguistic meaning of "and".
- The given class(es) contain(s) a trivially satisfiable universal (allValuesFrom) restriction.
- The given class(es) is/are missing a closure axiom (missing allValuesFrom restriction).
- The given class(es) do(es) not contain any of the given modeling errors, or is/are modelled correctly.

Comment (optional)

In case you have remarks, please note them here

[Submit](#)

Figure 3.8: Example HIT from the quiz with one class definition, in this case the fourth option is the correct answer.

- **Description and synonyms**
Following the name, the other metadata annotation values are listed, i.e. the alternative labels, synonyms and definitions.
- **Class axioms**
The axioms of the class are listed, rendered in MOS [32, 24]. MOS was chosen for a combination of reasons, (i) it is one of the syntaxes defined in the OWL specification overview [34] as being easy to read and write for humans, (ii) a renderer is readily available from OWL API, and (iii) rendering classes as paraphrases or graphical visualizations, as discussed above, is out of scope of this thesis and would have posed significant additional effort. As some classes have many disjointness axioms and therefore the list of axioms would be very long, all disjointness axioms of a class are merged together, using the term `DisjointWith (each of)`. For example,

[View instructions](#)

Class Name: WhiteOnionTopping

Description and Synonyms for WhiteOnionTopping

- White Onion
- White onions have white skin and flesh and a distinct light and mild flavour profile.

Class Axioms

- WhiteOnionTopping SubClassOf OnionTopping
 - WhiteOnionTopping DisjointWith (each of) ChoppedOnionTopping, OnionRingTopping
-

Class Name: RedOnionTopping

Description and Synonyms for RedOnionTopping

- Red Onion
- Red onions have purplish-red skin and white flesh tinged with red and a sharp flavor.

Class Axioms

- RedOnionTopping SubClassOf OnionTopping
 - RedOnionTopping DisjointWith (each of) OnionRingTopping
-

Which of the following statements holds for the given class(es): WhiteOnionTopping and RedOnionTopping?

- A disjointness axiom is missing for the given class(es).
- The given class(es) contain(s) a confusion between the logical and linguistic meaning of "and".
- The given class(es) contain(s) a trivially satisfiable universal (allValuesFrom) restriction.
- The given class(es) is/are missing a closure axiom (missing allValuesFrom restriction).
- The given class(es) do(es) not contain any of the given modeling errors, or is/are modelled correctly.

Comment (optional)

In case you have remarks, please note them here

[Submit](#)

Figure 3.9: Example HIT from the quiz with two class definitions, in this case the first option is the correct answer.

the two axioms `ClassA DisjointWith ClassX` and `ClassA DisjointWith ClassY` would be merged into `ClassA DisjointWith (each of) ClassX, ClassY`.

Below the class rendering(s) the task's question is asked, "*Which of the following statements holds for the given class(es): <CLASS NAMES>?*", whereas the class names of the given one or more classes are mentioned again.

As the answer one of five radio buttons must be selected, one for each of the four selected error types and another one for either no error or none of the above error types. In case the task was unclear to the worker, or they have any remarks, all tasks include the option to leave a comment.

Above all described elements, a button allows to show the task's instructions, including a description of all answer options, a summary of the Manchester OWL syntax and examples for the four error types. The instructions can be seen in *Appendix D: Instructions for Tutorial and Quiz*.

3.4.2 Collection of Judgements

Related work, see Section 2.3, has shown that for certain tasks there may be disagreement among human verifiers, and therefore collecting multiple votes per task may be advisable. In case multiple judgements are collected, the individual votes need to be aggregated.

This aggregation may be performed using simple majority voting, i.e. the most common judgement among all collected judgements is selected as the aggregated result. In case of majority voting as the aggregation strategy, an odd number of judgements should be collected to reduce the risk for tied most common answers, resp. completely avoid the risk thereof in case of binary choice of judgement.

An alternative approach would be a weighted aggregation strategy, meaning that not all judgements are equal, rather some are of higher weight than others. The weight of a judgement can be determined by the qualifications or skill levels of the human verifier (which have to be assessed before), or any other characteristics of the verifier or the task, e.g. the prior performance of the verifier or the task difficulty, or any combination of factors.

The ideal number of votes to collect, as well as the aggregation strategy to use, likely depend on the definite task design. The number of votes to collect is subject of *RQ-3c*, aiming to assess the influence of the number of votes aggregated using majority voting, see Section 5.6 for a discussion of the results.

While collecting multiple votes may improve the quality of the results, increasing the number of votes collected per task increases the overall cost of this process step. Therefore a trade-off between cost and quality of results needs to be made. A discussion thereof based on the results of the empirical study can be found in Section 5.7.

3.5 Hybrid Human-Machine Workflows beyond Defect Identification

The DIWs described in Section 3.2 each aim to identify ontology modeling errors of one specific error type. The proposed concept of DIWs can be generalized to the broader concept of *verification workflows* to enable more complex ontology verification workflows,

e.g. of the pattern *Find-Fix-Verify* to include a step for repair, as discussed in Section 2.3.1 for linked data quality assessment.

A verification workflow thus would consist of an arbitrary number of steps, executed in succession analogously to the two-step DIWs. Each step passes its results and all data available to it to the next workflow step, until the workflow comes to an end.

Using this approach of passing all data available to a step to the next step enables more complex topologies, including splitting and joining the workflow. This approach comes at the cost of possibly passing unnecessary data, which is a trade-off between flexibility and memory usage that will have to be adapted in future applications depending on their use cases.

A *Find-Fix-Verify* verification process for a specific error type would thus consist of three steps. The first *Find* step would employ the same defect candidate detection heuristics as the *Find* stage of the two-step DIWs. In the second step HC tasks for collecting possible repair options would be generated, followed by an HC-based verification of the repaired defect candidate, similar to the verification tasks generated by the DIWs.

This *Find-Fix-Verify* process could for example be used to repair classes with trivially satisfiable *allValuesFrom* restrictions, whereas in the first step the heuristic from Section 3.3 would be used, in the second step the resulting defect candidates would be presented in HC tasks allowing humans to enter possible repairs, e.g. the introduction of *someValuesFrom* restrictions or modifications to the *allValuesFrom* restriction, and in the third step, the defect candidate, together with the suggested repairs, would be verified by humans using a similar HC interface as is used in the DIWs, for which an example is depicted in Figure 3.8.

An alternative, more complex workflow for a more detailed diagnosis of trivially satisfiable *allValuesFrom* restrictions, beyond their identification as described in the previous sections, that is enabled by the multi-step verification workflow concept, is shown as an example in Figure 3.10.

Analogously to the two-step DIWs and the three-step *Find-Fix-Verify* workflows, the more complex workflow also requires the ontology to be consistent, as it relies on the inferences of an ontology reasoner (*STEP-1*). If the ontology is inconsistent, this fact is reported and the workflow is finished. In the next step *STEP-2*, trivially satisfiable classes are detected by the heuristic described in Section 3.3.

Where the DIWs would now present the detected candidate classes to human verifiers to let them judge if the classes contain trivially satisfiable *allValuesFrom* restrictions, the more complex diagnosis workflow creates HC tasks to verify the correctness of the *allValuesFrom* restrictions (*STEP-3*).

If an *allValuesFrom* restriction is judged to be incorrect, another HC task is created in step *STEP-4a* to let human judgement decide if the *allValuesFrom* restriction should be converted to a *someValuesFrom* restriction, reporting the result of this task as the final result of the workflow, i.e. either that the *allValuesFrom* restriction should be changed

to a `someValuesFrom` restriction - possibly because the ontology engineer confused the types of quantification - or just that the `allValuesFrom` restriction is incorrect.

If on the other hand the result from step *STEP-3* is that the `allValuesFrom` restriction is correct, an HC task is created in step *STEP-4b* to inquire if the class from the defect candidate should really be satisfiable without any existential restrictions. In case the resulting judgement is affirmative, the workflow concludes that either the class was reported as a false positive or that the trivial satisfiability is desired. Otherwise, the workflow reports that at least one existential restriction is missing on the class.

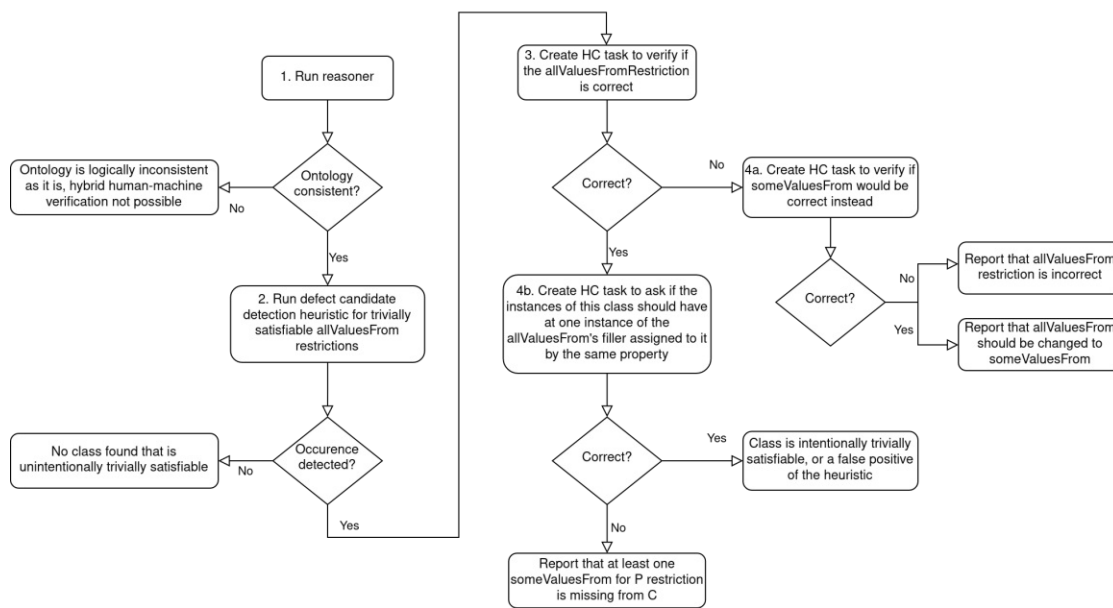


Figure 3.10: Complex verification workflow for the diagnosis of trivially satisfiable `allValuesFrom` restrictions

3.6 Prototype Implementation

The DIWs described in this chapter were implemented for the four selected error types using the proposed heuristics and HC task design.

Figure 3.11 presents an overview of the prototype's architecture. An ontology engineer feeds an ontology document to the prototype, which is deserialized using Apache Jena¹. The deserialized ontology is then given to the workflow engine, which is the component responsible for executing the defined workflows, by executing their steps in succession. The workflow engine first invokes the defect candidate detection heuristics and then passes the yielded candidates to the HC task component. The HC task component is

¹<https://jena.apache.org/>

responsible for creating HC tasks for the verification of the candidates, as well as the retrieval and aggregation of the results of these tasks.

The defect candidate detection heuristics use the library OWL API², which is used because it provides support for OWL2 semantics, which is limited in Apache Jena. ONT-API³ is used to bridge between Apache Jena and OWL API. For the reasoning functionality required by the heuristics, the Hermit ontology reasoner⁴ [17] is used, as it provides all required functionality and there exists an integration with OWL API.

MTurk⁵ was chosen as the platform for the HC tasks. It is a well-known Crowdsourcing platform, allowing *requesters* to create tasks, so called Human Intelligence Tasks, that are performed by *workers* for monetary rewards. As the empirical study used an internal crowd and did not rely on workers from the public marketplace offered by MTurk, using the MTurk developer sandbox environment⁶ sufficed.

In order to integrate MTurk HITs into the prototype for DIWs, new HITs are created by the HC task component via the MTurk REST API, which is then polled frequently for the current status of the open HITs, waiting for their finalization to return the collected judgements.

Once the judgements are retrieved from MTurk, they are aggregated and handed back to the workflow engine, which returns them as the output as the workflow is finished.

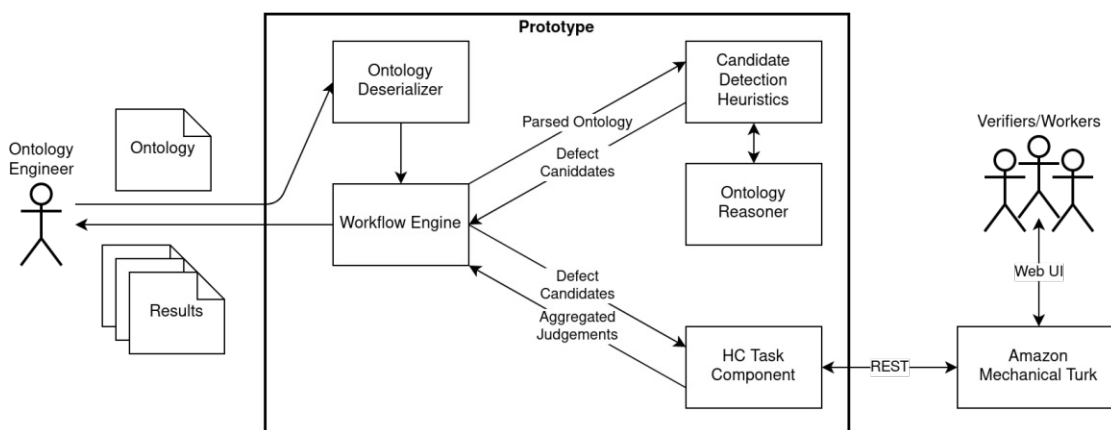


Figure 3.11: Prototype Architecture

In this chapter, four common types of ontology modeling error were chosen to be targeted for identification using hybrid human-machine workflows. The concept of DIWs was therefore introduced, combining an automatic heuristics-based detection of *defect candidates* relying on an ontology reasoner with an HC verification of the detected

²<https://owlcs.github.io/owlapi/>

³<https://github.com/owlcs/ont-api>

⁴<http://www.hermit-reasoner.com/>

⁵<https://www.mturk.com/>

⁶<https://requester.mturk.com/developer/sandbox>

defect candidates. For each of the four selected error types, a heuristic was designed for the detection of such defect candidates and an HC interface was presented with considerations thereby discussed. The combination of an automatic preselection of errors with Crowdsourcing to verify them aims to improve the scalability of Crowdsourcing-based ontology verification by reducing the amount of human labor needed. The last two sections discussed how the concept of hybrid human-machine workflows could be extended to complete tasks beyond error identification and the details of the implemented prototype, respectively. The following chapter describes the setup for evaluating the proposed solution, detailing the design of an empirical study with descriptive statistics.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Evaluation Setup

In order to evaluate the impact of the proposed HOV method, resp. the designed DIWs, which are described in Chapter 3, a study was designed using the prototype described in Section 3.6.

This chapter describes the designed study, starting with an overview on the evaluation approach in Section 4.1, followed by a description of the study's details, i.e. the participants in Section 4.2, the chosen data set and preparation steps performed on it in Section 4.3, the seeded defects and false positives in Section 4.4, an overview over the phases of the study's execution in Section 4.5, and finally the measured variables and metrics used in Section 4.6.

The evaluation results can be found in Chapter 5.

4.1 Evaluation Approach Overview

As described in Chapter 3, the workflows to be evaluated for the detection of the four selected error types each consist of two consecutive parts, starting with an automated detection of defect candidates using an ontology reasoner and the designed heuristics, followed by an HC step to either confirm or reject each of the defect candidates.

The evaluation approach consists of two parts, analogous to the phases of the process under evaluation:

- 1. Effectiveness of the Automatic Defect Candidate Detection**

The process steps that detect the defect candidates for each of the error types have to be checked for their effectiveness using sample ontologies, with the yielded defect candidates being checked against a gold standard of defects within the used ontologies. This part of the evaluation is out of this thesis' scope, see Section 6.3.

2. Effectiveness of the HC Candidate Verification

The HC part of the hybrid-human machine verification processes is evaluated regarding its effectiveness, i.e. whether workers are able to correctly judge defect candidates as either true defects or false positives, to answer *RQ-1*.

This ability of the workers to correctly judge candidates is evaluated both on an individual level for each worker and with different numbers of judgements aggregated using majority voting.

Furthermore, the influence on the effectiveness of the HC verification of both the prior knowledge of the workers regarding ontology modeling, as well as other skills, and the type of defect under verification is examined to gain insights regarding *RQ-3*.

In the following section the evaluation of the HC part of the novel ontology verification process is described.

According to Wohlin et al. [65], the designed evaluation is an *empirical study* with *descriptive statistics*.

In the following, in addition to *empirical study*, the term *quiz* is used as an alternative term, which was the term used in the communication with the participants.

4.2 Participants

The crowd of the study consisted of 16 master students and research staff of the Semantic Systems research group¹ at TU Wien. The students were taking the course *188.387 Semi-Automatic Information and Knowledge Systems* at the time the study took place. At that time the students had obtained ontology modeling skills from the course and could therefore be deemed (junior) ontology engineers.

The participants were awarded five bonus points in the course for participating in the study, and were further incentivized to give good answers by receiving five additional bonus points if they judged at least 75 percent of the tasks correctly.

4.3 Data Set

The ontology to be verified in the experiment is the pizza ontology.² This ontology is proven to be of successful use in teaching ontology engineering to western audiences [42]. The authors of [42] also note that pizzas are familiar and concrete subjects, while being both compositional and rich enough to illustrate key issues in ontology modeling.

¹<http://semsys.ifs.tuwien.ac.at/>

²<https://protege.stanford.edu/ontologies/pizza/pizza.owl>

4.3.1 Cleaning

The pizza ontology includes some errors for demonstration purposes, these were removed beforehand as they either were not correctly representing domain concepts or their construction and name made it obvious that they were not “natural” parts of the pizza ontology. Five classes were removed from the ontology because of this, see Table 4.1. Any disjointness axioms with a set of more than two classes, including one of the removed classes, were adapted to not include the removed class anymore in order to preserve the asserted disjointness between the classes remaining in the sets.

Removed Class	Demonstration Purpose
IceCream	Mistakes with property domains
UnclosedPizza	Mistakes with missing closures
VegetarianPizzaEquivalent1	Alternative definition of VegetarianPizza
VegetarianPizzaEquivalent2	Alternative definition of VegetarianPizza
CheesyVegetableTopping	Mistakes with disjoint parent classes

Table 4.1: Classes removed from the pizza ontology and the reason for their presence in the ontology according to accompanying comments

4.3.2 Metadata Annotations

The original pizza ontology already contains some metadata annotations using OWL annotation properties from the *Simple Knowledge Organization System (SKOS)* which are defined in a W3C recommendation [54]. The reference is accompanied by a primer [53], describing the recommended usage of the defined properties.

Specifically, the lexical labels `skos:prefLabel` and `skos:altLabel`, and the documentation property `skos:definition` are present in the original pizza ontology.

`skos:prefLabel` and `skos:altLabel` can be used to assign human-readable labels to ontology classes, with `skos:prefLabel` representing the preferred human-readable label and `skos:altLabel` declaring alternative labels, synonyms, abbreviations, acronyms or near-synonyms. The SKOS primer states that `skos:definition` should supply “*a complete explanation of the intended meaning of a concept*” [53].

For all classes preferred labels are available in the pizza ontology, with most also having alternative labels defined. The usage of `skos:definition` was sparse, with the property being available only for a few of the classes.

As these metadata annotations are used to give context about the presented classes to the human workers, the classes used in the quiz were checked for their labels and definitions, with a definition being added if not present and alternative labels supplemented where applicable. The definitions were added as short sentences, based on the first sentence of the Wikipedia articles of the concepts represented by the classes, or in case of the various named pizzas, the list of their toppings.

This was done for the empirical study under the assumption that ontologies will contain annotation properties for their classes that are not self-explanatory, written by ontology engineers in a similar manner as a software engineer leaves comments in their code to describe non-trivial code sections. This assumption is substantiated by the ontology pitfall catalogue [40], as it includes the pitfall “*P08. Missing Annotations*”, thus quality-cautious ontology engineers are recommended to include annotations on all their classes. The ontology pitfall scanner automatically checks for missing annotation properties [40].

In case an ontology does not contain any metadata annotations, they could be retrieved from Wikipedia, glossaries or dictionaries, based on the given labels, or the class names, if no labels are given.

The given definition of the class `VegetarianPizza` was shortened, as it included a hint to its trivial satisfiability.

4.4 Seeded Defects

In total each participant was presented 20 tasks, with three tasks containing an error per each error type and eight tasks containing no error, or at least none of the four defined error types.

To achieve this, the classes listed in Table 4.2 were added to, resp. altered in, the pizza ontology after cleaning and preparing it as described in the previous section. Table 4.2 gives a list of the added and altered classes with a description of their definition and the corresponding error type.

4.5 Execution Overview

To prepare for the quiz, participants were advised to create an account on the MTurk worker sandbox, where the main quiz took place, before their quiz session.

Participants were asked to register to a quiz session, which took place at a specific time and was supported by a Zoom³ call, to allow the interactive answering of possible questions.

The quiz sessions were started by briefly reiterating the phases and steps of the quiz (which the participants were already presented before signing up to the quiz). The overall structure of the quiz was inspired by the master’s thesis of Stefani Stoyanova Tsaneva [58] and large parts of the skill assessment part were reused from her thesis.

The quiz sessions were structured as follows, with a graphical overview presented in Figure 4.1:

1. Skill Assessment & Preparation Phase

³<https://zoom.us/>

a) Self-Assessment Test

Participants were asked to give subjective assessment of their skill level, resp. prior knowledge, in different fields, using a Google Forms questionnaire⁴, see *Appendix B: Self-Assessment Form*.

b) Qualification Test

The qualification test was designed as an extensive single task on MTurk, consisting of eleven questions in three sections. The questions checked the skill level of the participants, with the questions becoming harder in the later sections. *Appendix C: Qualification Test* shows all the questions asked.

c) Tutorial

The tutorial used the same task design as the quiz, to prepare participants for the actual quiz tasks and give them the opportunity to familiarize themselves with the mode and read the instructions and examples without any sort of pressure. There were four tasks, one for each error type, corresponding to the examples given. Before submitting each tutorial task, participants had the option to display the correct answer to the task. See *Appendix D: Instructions for Tutorial and Quiz* for the given instructions and examples. Figure 4.2 shows an example HIT from the tutorial.

2. Quiz Phase

The actual quiz consisted of the 20 tasks described above, presented and completed on MTurk. The same instructions and examples, see *Appendix D: Instructions for Tutorial and Quiz*, were available as in the tutorial.

3. Feedback Phase

After the actual quiz, participants were asked to give feedback on the quiz, including their subjective experience of the quiz and a rating of the easiest and hardest error types to classify. See *Appendix E: Feedback Questionnaire* for a screenshot of the used Google Forms questionnaire.

To aid the participants through the various phases and tasks, they were sent a link to a guidelines page, giving an overview on the phases and tasks, as well as the links to the forms and MTurk tasks. See *Appendix A: Quiz Guidelines Page* for a screenshot of said page.

4.6 Measured Variables and Metrics

This section describes the variables that are measured to gain insights regarding the research questions in Section 4.6.1 and the metrics that are used to assess them in Section 4.6.2.

⁴<https://www.google.com/forms/about/>

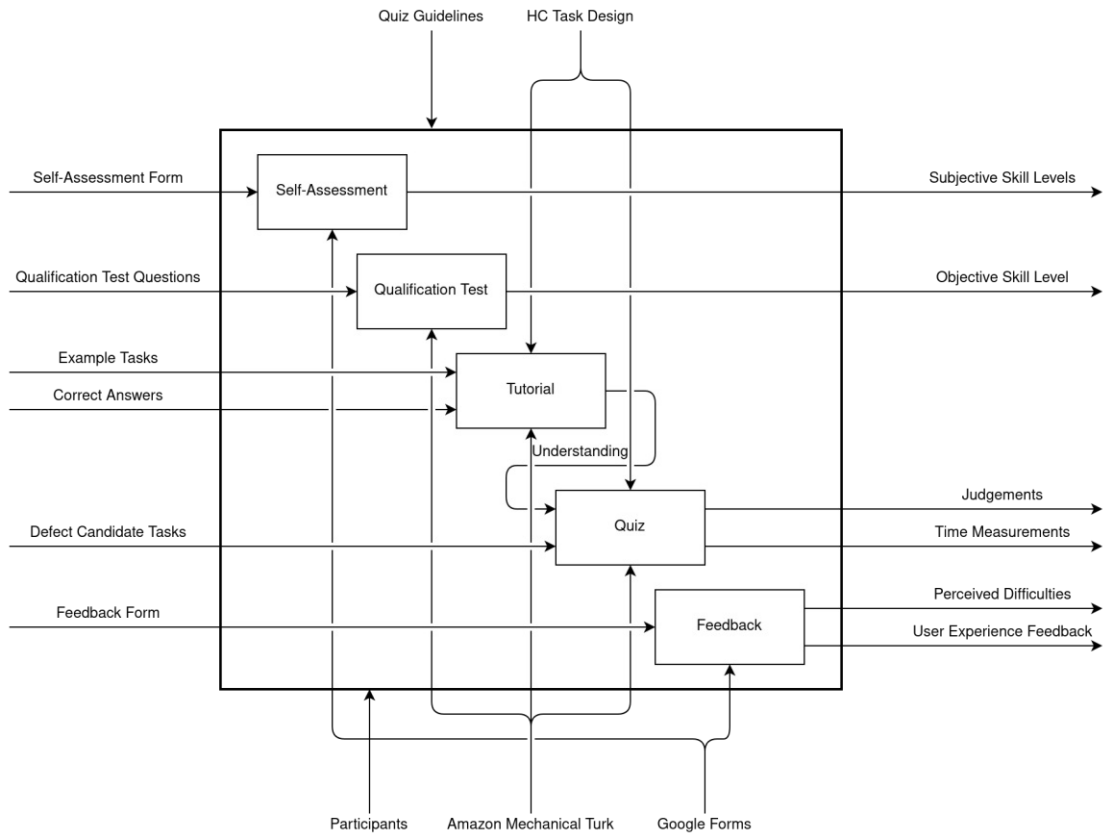


Figure 4.1: Quiz phases overview as an IDEF-0 diagram

4.6.1 Measured Variables

In order to gain insight to answer the research questions stated in Section 1.2, multiple variables are measured. The judgement accuracy is of interest to *RQ-1* and *RQ-2*, as the performance of the human verifiers is an indicator that the proposed HOV method, DIWs and the used HC task design are suitable. Measuring the judgement accuracy naturally is also required to answer *RQ-3*, as it is concerned with the influence of various factors on the judgement accuracy.

For *RQ-3a*, concerned with the influence of prior knowledge, various subjective skill levels are enquired and an objective skill level regarding ontology modeling is measured. *RQ-3b* and *RQ-3c*, concerned with the influence of the error type and judgement aggregation, respectively, do not require further variables for their assessment, as the error type is already given by the gold standard and the effect of judgement aggregation is calculated using random samples, see Section 5.6.

[View instructions](#)

Class Name: CardinalePizza

Description and Synonyms for CardinalePizza

- A CardinalePizza is a pizza with only tomatoes, mozzarella and ham.
- Cardinale
- HamPizza

Class Axioms

- CardinalePizza SubClassOf Pizza and (hasTopping only (TomatoTopping and MozzarellaTopping and HamTopping))
- CardinalePizza hasTopping some TomatoTopping
- CardinalePizza hasTopping some MozzarellaTopping
- CardinalePizza hasTopping some HamTopping

Which of the following statements holds for the given class(es): CardinalePizza?

- A disjointness axiom is missing for the given class(es).
- Correct Answer** The given class(es) contain(s) a confusion between the logical and linguistic meaning of "and".
- The given class(es) contain(s) a trivially satisfiable universal (allValuesFrom) restriction.
- The given class(es) is/are missing a closure axiom (missing allValuesFrom restriction).
- The given class(es) do(es) not contain any of the given modeling errors, or is/are modelled correctly.

Comment (optional)

In case you have remarks, please note them here

[See Correct Answer](#)

[Submit](#)

Figure 4.2: Example HIT from the tutorial, displaying the correct answer after choosing an option and clicking the respective button

Judgement Accuracy and Time Spent

For the quiz, the measured variables are on the one hand the answers from the participants to the HITs, as the combination of their judgements and optional comments, and on the other hand the time spent for each answer. By comparison to the correct answers per task, the correctness of each judgement can be determined, and the ratio of correct answers can be calculated on an individual basis, as well as for all judgements grouped by various parameters. The time measurement is carried out by MTurk, which reports the time spent by a participant on a task alongside the answers.

Subjective Skill Levels

The self-assessment task yields data points from its skill assessment questions, each asking for an assessment on a four-point Likert scale with 1="no knowledge" to 4="expert knowledge". This subjective assessment is collected for (Q1) the understanding of English

documents, the knowledge regarding (Q2) formal logics, (Q3) model-driven engineering, (Q4) ontology engineering, and (Q5) ontology engineering specifically using web-based knowledge representation languages, such as OWL and RDF Schema (RDFS). The final question inquires about the prior experience with Crowdsourcing platforms.

Objective Ontology Modeling Skill Level

While the data points from the self-assessment task naturally are subjective assessments, the qualification test and its evaluation reports an objective skill level regarding the usage of OWL. Therefore, for each participant the answers to the eleven questions from the qualification test are compared to the answer key, followed by a classification onto a four-point scale as described below.

Sections 1 and 3 of the qualification test have four questions each, while section 2 has three questions. The submission of a participant is classified as follows:

- Skill level 4=EXPERT KNOWLEDGE:
Achieved by having at least 10/11 correct answers overall (and therefore also the majority in section 3).
- Skill level 3=INTERMEDIATE KNOWLEDGE:
Achieved by having at least 7/11 correct answers overall and the majority in section 2.
- Skill level 2=LITTLE KNOWLEDGE:
Achieved by having at least 4/11 correct answers overall and the majority in section 1.
- Skill level 1=NO KNOWLEDGE:
Achieved by having less than 4 correct answers overall.

The rationale behind the thresholds is that one achieves expert knowledge with a maximum of one mistake, and the number of overall correct answers required to achieve levels 2 and 3 being equally distributed among the remaining possible scores, $threshold(level) = \lceil ((level - 1) * 10/3) \rceil$, i.e. at least 7 points for level 3 and at least 4 points for level 2.

4.6.2 Used Metrics

This section describes the metrics used for assessing the measured judgement accuracy.

The collection of judgements via HC tasks can be considered a form of classification and therefore metrics from the fields of information retrieval and machine learning can be used. A HIT is thus considered as a classification problem, during which a human assigns a label to an ontology class or a set of ontology classes, i.e. one of $\{Missing\ Disjointness, Logical\ vs.\ linguistic\ "and", Trivial\ Satisfiability, Missing\ Closure, No\ (known)\ Error\}$.

One such assignment of a label to a task can be considered either as a *true positive (TP)*, *false positive (FP)*, *true negative (TN)* or *false negative (FN)* regarding its equality to the correct answer according to the gold standard.

The following metrics are used in the report of the evaluation's results:

- $Precision = \frac{TP}{TP+FP}$
- $Recall = \frac{TP}{TP+FN}$
- F_1 measure, $F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$ (harmonic mean of precision and recall)
- Cohen's kappa [11], a metric for measuring inter-rater agreement while taking the probability of agreement by chance into account (in this case agreement between judgements and the gold standard is measured)

For precision, recall and the F_1 measure there exist different approaches for calculating the respective metric for multi-label values. *Micro-averaging* calculates the metrics using the global numbers of true/false positives/negatives among all classes and for *macro-averaging* in contrast the numbers of true/false positives/negatives are calculated per label and the unweighted means of those are taken to calculate the metrics.

A third approach, the *weighted* approach, is similar to macro-averaging, but instead of using an unweighted mean, the mean is weighted by the number of true instances for each label. This weighted approach is used in this thesis, as it considers the different frequencies of the labels (in contrast to macro-averaging) and is sensitive to low metrics for small labels (in contrast to micro-averaging).

This chapter has introduced a study design for the evaluation of the proposed HOV method, which uses the implemented prototype to evaluate the DIWs with a focus on the HC part. Therefore, an overview of the study was given, the crowd, data set and execution phases were described, and the measured variables and used metrics were defined.

The results of the empirical study are presented and discussed in the following chapter.

4. EVALUATION SETUP

Class(es)	Description	Error Type
Pizza, Hamburger	no disjointness between them	Missing Disjointness
CamembertTopping, ParmezanTopping	no disjointness between them	Missing Disjointness
WhiteOnionTopping, RedOnionTopping	no disjointness between them	Missing Disjointness
RedOnionTopping, ChoppedOnionTopping	not disjoint, but also should not be	No (known) Error
GlutenFreeBase, ThinAndCrispyBase	not disjoint, but also should not be	No (known) Error
SeafoodTopping, TunaTopping	not disjoint, TunaTopping should be a subclass of SeafoodTopping	No (known) Error
VegetarianTopping	using a conjunction of all non-Meat and -Seafood toppings in the closure axiom	Logical vs. Linguistic AND
NonVegetarianPizza	using existential restriction with the conjunction of Meat and Seafood	Logical vs. Linguistic AND
FourCheesesTopping	subclass of the conjunction of four different cheeses	Logical vs. Linguistic AND
QuattroFormaggi	existential restriction with the unsatisfiable FourCheesesTopping	No (known) Error
SpicyMargherita	subclass of MargheritaPizza and the existential restriction of having TobascoSauce	No (known) Error
Margherita	removed all existential restrictions, s.t. only closure axiom remains	Trivial Satisfiability
Siciliana	removed all existential restrictions, s.t. only closure axiom remains	Trivial Satisfiability
Fiorentina	removed all existential restrictions, s.t. only closure axiom remains	Trivial Satisfiability
VegetarianaPizza	given only existential restrictions, no closure axiom	Missing Closure
MushroomPizza	given only existential restrictions, no closure axiom	Missing Closure
FourSeasonsPizza	given only existential restrictions, no closure axiom	Missing Closure
GiardineraPizza	defined correctly with existential restrictions and closure axiom	No (known) Error
PolloAdAstraPizza	defined correctly with existential restrictions and closure axiom	No (known) Error
RosaPizza	defined correctly with existential restrictions and closure axiom	No (known) Error

Table 4.2: Classes presented in the quiz tasks

Evaluation Results

This chapter presents and discusses the results of the empirical study, described in Chapter 4.

Following a brief summary of the overall results in Section 5.1, Sections 5.2, 5.3 and 5.6 discuss the influence of prior knowledge (*RQ-3a*), the error type of the task (*RQ-3b*) and the number of aggregated votes for a final judgement (*RQ-3c*) on the ratio of correct answers, respectively. Section 5.4 goes into detail on three problematic tasks and discusses possible reasons. Section 5.5 discusses the time spent on each task, Section 5.7 provides a cost estimation for different numbers of votes aggregated and finally Section 5.8 presents the feedback given by the study’s participants.

5.1 Result Data

As mentioned already in Section 4.2, there were 16 participants in this empirical study. 319 HITs were submitted by the participants, with 80.9 percent of the judgements being correct. On average 78.8 seconds were spent on each task, with a standard deviation of 73.4 and with a median of 53 seconds.

The metrics discussed above are as follows: (i) precision: 0.8492, (ii) recall: 0.8088, (iii) F_1 score: 0.8113, and (iv) Cohen’s kappa: 0.7538.

These metrics indicate that the proposed concept is feasible and the used HC task design is suitable, as human verifiers are able to correctly verify a large portion of the presented defect candidates.

Considering that each of the 16 participants had 20 HITs to complete, it can be concluded that the submission of a single task for one of the participants failed. There may be multiple reasons for this, either the participant simply did not submit the HIT, or used the “Return” button in the MTurk user interface for exiting tasks which the user does not want to complete anymore (possibly by accident).

5.1.1 Free-Text Comments

As described in Section 3.4.1, participants had the option to leave a comment for each task. In total 23 comments were left on the 319 tasks, they were classified as follows:

- Stated reasoning behind correct answer: 10
- Recognized ambiguous task but made wrong decision: 2 (see Section 5.4 for a discussion of the ambiguous tasks)
- Documented incorrect assumption, leading to incorrect judgement: 4
- Documented uncertainty if only the two presented classes are relevant regarding missing disjointness axioms, or if other classes from the definition should be considered as well: 2
- Other (no information about decision making): 5

This may hint at two possible problems, (i) ambiguous tasks, as discussed in Section 5.4 and (ii) uncertainty on which classes to consider regarding missing disjointness axioms. Both should be taken into account in future work, whereas the instructions given to the participants should clear up the uncertainty about which classes to consider.

5.2 Influence of Prior Knowledge

In order to draw conclusions on the influence of prior knowledge on judgement accuracy, to address *RQ-3a*, subjective skill assessments of various skills and an objective skill assessment of ontology modeling using OWL were collected.

5.2.1 Various Subjective Skill Levels

As described in Chapter 4, the self-assessment form, see *Appendix B: Self-Assessment Form*, collected the participants' own assessment of their various skill levels on a four-point Likert scale with 1="no knowledge" to 4="expert knowledge". A visualization of the self-assessment results is presented in *Appendix F: Results of the Self-Assessment Form*.

On this scale from one to four, the average assessments per skill are as follows: (i) understanding of English documents: 3.88, (ii) formal logics: 2.81, (iii) modeling in general: 3.00, (iv) ontology modeling: 3.13, (v) ontology modeling using OWL or related web-based languages: 3.00. Nine out of 16 participants, i.e. 56.25 percent, state to have prior experience with Crowdsourcing platforms.

The average participant thus states to have a very good understanding of English documents, intermediate skills regarding formal logics and an at least intermediate skill level in ontology modeling.

No strong correlations of these subjective skill levels were found with the ratio of correct answers, with the correlation coefficients being: (i) English language understanding: 0.15, (ii) formal logics: 0.20, (iii) general modeling: -0.25, (iv) ontology modeling: 0.32, (v) ontology modeling using a web-base language: 0.06, and (vi) prior experience with Crowdsourcing platforms: -0.10. There were however strong correlations of subjective skill assessments with the amount of time spent, discussed in Section 5.5.

5.2.2 Objective Ontology Modeling Skills

The participant’s objective skill level was measured using a qualification test, see *Appendix C: Qualification Test*, assigning a skill level from 1=“no knowledge” to 4=“expert knowledge” to each participant. The results are shown in Figure 5.1.

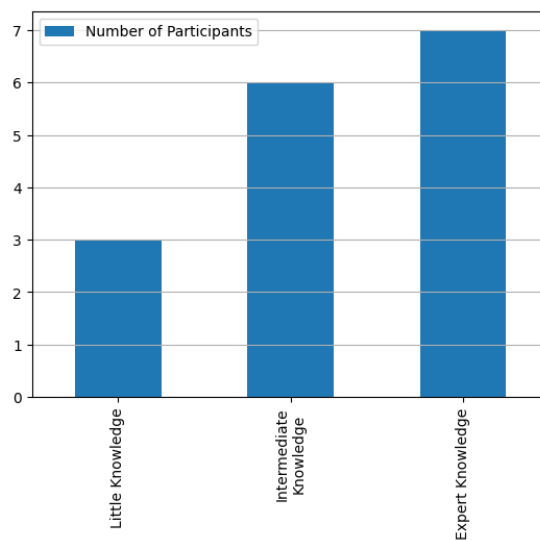


Figure 5.1: Calculated qualification levels

Not a single participant was assigned the lowest skill level, with a large majority of 81.25 percent being assigned one of the top two skill levels, with an average level of 3.25. The crowd thus can be considered to be skilled in ontology modeling using OWL.

Compared to the subjective skill assessment regarding ontology modeling and more specifically using web-based ontology modeling languages, the correlation coefficients are 0.38 and 0.50, respectively. While the average self assessments are 3.13 and 3.00, respectively, the average objective ontology modeling skill level is 3.25. The crowd overall therefore slightly underestimated its ontology modeling skill level.

Figure 5.2 sets the qualification levels in relation to the ratio and amount of correct judgements. It can be observed that there is a positive correlation between qualification levels and the ability to judge correctly, with notable improvements with higher qualification level. The correlation coefficient for the qualification level and the ratio of

correct answers is 0.45. The difference between the ratio of correctness between levels *Little Knowledge* and *Intermediate Knowledge* is 9.2 percent points, and 4.1 between *Intermediate Knowledge* and *Expert Knowledge*.

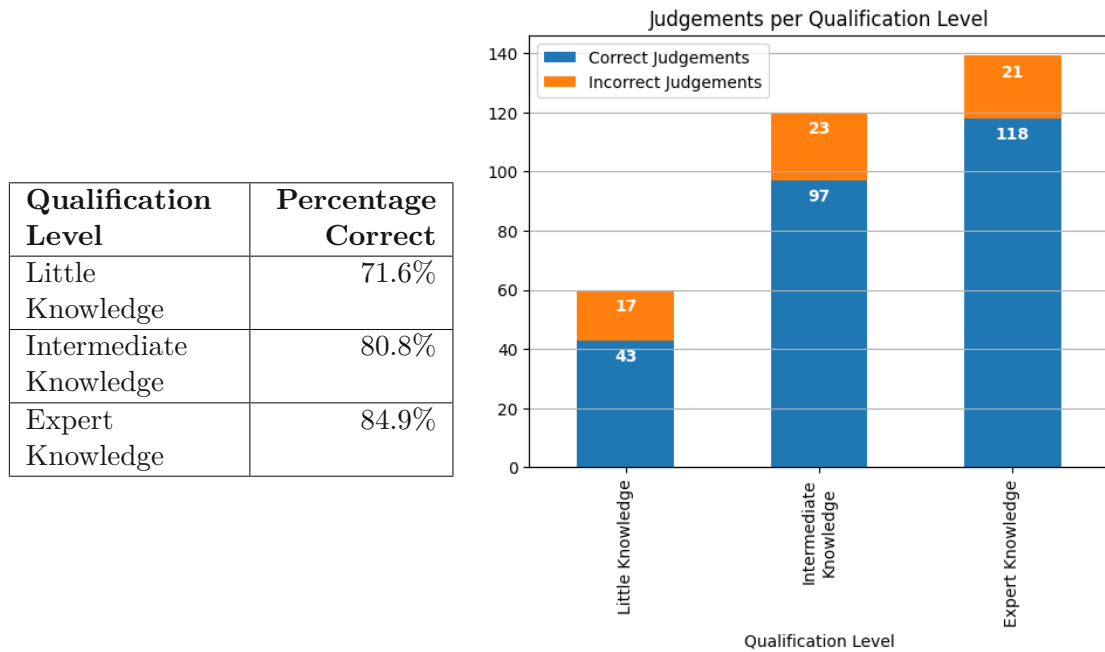


Figure 5.2: Correct judgements by qualification level

One thing to mention here is that eleven of the 16 participants previously took a similar quiz [58]. As this quiz also relied on MTurk for its HC tasks, it is notable that in the self-assessment only nine participants stated to have prior experience with Crowdsourcing platforms. A possible explanation of this fact would be that two of the participants that took both quizzes did not have experience apart from the previous quiz and did not consider their one-time usage in the previous quiz almost half a year prior to the later quiz as significant experience. Another reason might be that at least two participants did not associate MTurk as being a Crowdsourcing platform.

The results of this empirical study, when set in the context of the participants' qualification levels, therefore indicate that there is a significant influence of prior knowledge on performance, as a positive correlation between ontology modeling skills and judgement accuracy is observed, thus answering *RQ-3a*.

5.3 Influence of Error Type

In the quiz' results, there is notable difference between the results for tasks of different error types, thus answering *RQ-3b* which sought for the existence of such an influence of

the type of error on performance. Figure 5.3 shows both the ratio of correct answers and the counts of correct and incorrect judgements per error type.

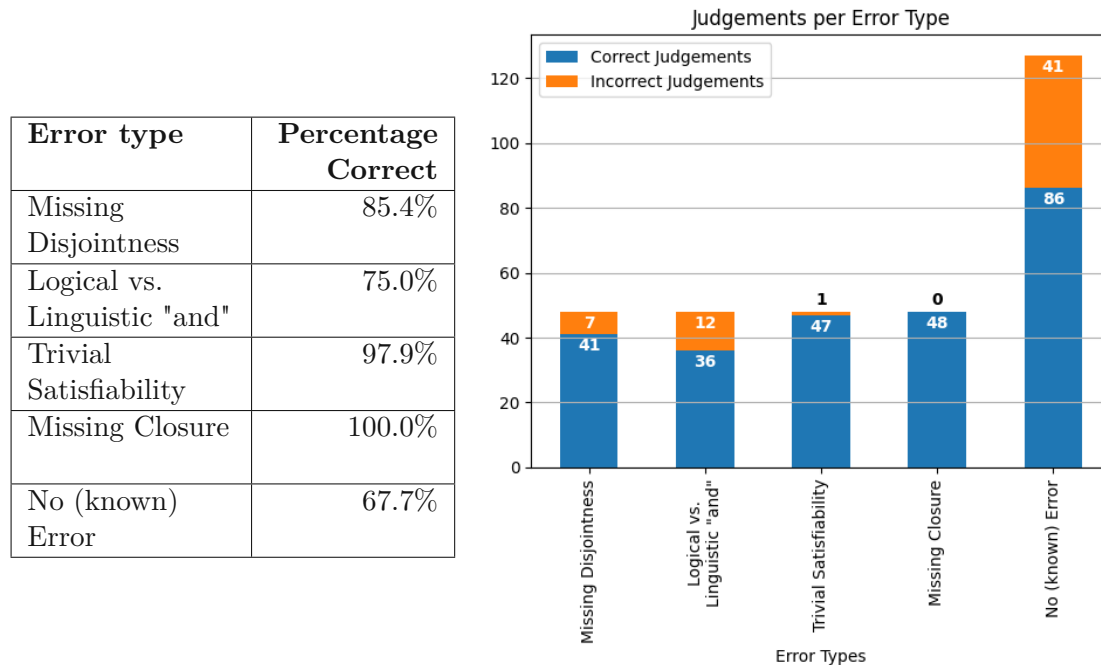


Figure 5.3: Correct judgements by error type

The error types *Trivial Satisfiability* and *Missing Closure* were judged almost perfectly, with a single task of the type *Trivial Satisfiability* being judged as *Missing Closure*. This may have multiple reasons, on the one hand these two error types may simply be easier for the participants to judge, or on the other hand prior knowledge may play a part, as eleven out of the 16 participants also took the quiz from Stefani Stoyanova Tsaneva's master's thesis [58] which focused on the verification of existential and universal restrictions. From this previous quiz there may have been some learning effect, as the two error types *Trivial Satisfiability* and *Missing Closure* are about universal restrictions without corresponding existential restrictions and vice versa, respectively.

Even if the strong performance on these two error types was to be fully attributed to a learning effect from the previous quiz, this should be viewed as a hint for the importance of training. If a previous quiz taking approximately two hours, or a similar form of training or preparation, leads to near perfect performance on some error types, this suggests that humans can acquire knowledge, resp. skills, in little time to greatly improve their performance in HOV of the discussed format.

The two error types *Missing Disjointness* and *Logical vs. linguistic "and"* also show reasonable performance, with 85.4 percent and 75.0 percent of correct judgements, respectively.

The remaining tasks, of the type *No (known) Error*, only show a performance of two out of three correct judgements. It should be noted that in future work, this type should be split up into two distinct types, one for unknown errors and one for the absence of any errors. This way a resulting classification of a defect candidate with one of these types would imply if the defect candidate should be further investigated, or was a false positive.

The next section discusses three problematic HITs with poor rates of correct answers, possible reasons and ways to avoid the problems. All three of these HITs belong to the *No (known) Error* type, in total amounting to 30 judgements for incorrect types, 20 of them being judged as *Missing Disjointness* and ten of them being judged as *Missing Closure*. The impact of this can be observed in the confusion matrix shown in Figure 5.4.

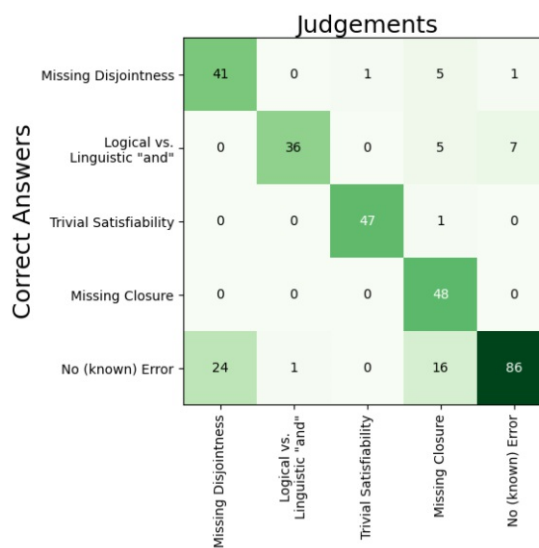


Figure 5.4: Confusion matrix of judged error types

5.4 Problematic HITs

Although the results overall were positive, three tasks had exceptionally bad results. In the following these problematic tasks and possible explanations are discussed.

5.4.1 Incorrect Disjointness Axioms Given

The first problematic HIT was concerned with the two classes `ChoppedOnionTopping` and `RedOnionTopping`, its rendering as presented to the study's participants is shown in Figure 5.5. This task was intended as a false positive, as from a domain point of view, these two classes should not be disjoint, as red onions can in fact be chopped, thus there may be individuals with both classes assigned, i.e. chopped red onions. The correct answer which the judgements were compared against was therefore *No (known) Error*.

Class Name: ChoppedOnionTopping**Description and Synonyms for ChoppedOnionTopping**

- Chopped Onions
- Onions chopped to small dices.

Class Axioms

- ChoppedOnionTopping SubClassOf OnionTopping
- ChoppedOnionTopping DisjointWith (each of) WhiteOnionTopping, OnionRingTopping

Class Name: RedOnionTopping**Description and Synonyms for RedOnionTopping**

- Red Onion
- Red onions have purplish-red skin and white flesh tinged with red and a sharp flavor.

Class Axioms

- RedOnionTopping SubClassOf OnionTopping
- RedOnionTopping DisjointWith (each of) OnionRingTopping

Figure 5.5: Problematic HIT #1, with two incorrect disjointness axioms given in the class definitions

A majority of 68.75 percent of the participant however judged this HIT as *Missing Disjointness*, while only 18.75 percent voted for the correct answer.

A possible explanation for this could be that participants drew conclusions, resp. made assumptions, based on the class definitions given. Two of the given disjointness axioms, however are incorrect: (i) red onions are declared disjoint from onion rings and (ii) white onions are declared disjoint from chopped onions. The participants may have assumed that red onions should be disjoint from chopped onions analogous to the given disjoint axiom between white onions and chopped onions.

The second problematic HIT was quite similar, concerned with the disjointness of the classes *GlutenFreeBase* and *ThinAndCrispyBase*, see Figure 5.6. These classes should not be disjoint, as thin and crispy pizza bases can be baked from gluten-free dough, thus a pizza base can be gluten-free, thin and crispy at the same time. Again, an incorrect disjointness axiom (*GlutenFreeBase* with *DeepPanBase*) was given, that may have led 56.25 percent of the participants to analogously answer with *Missing Disjointness*. The correct answer *No (known) Error* was given by 43.75 percent of the participants.

The impact of this problem of incorrect axioms given could possibly be weakened by explicitly noting that other modeling errors can be given in the definitions and instructing participants to primarily judge the classes based on the domain semantics, not relying on unverified information too much. This was not done for the study and should be considered in future work.

<p>Class Name: GlutenFreeBase</p> <p>Description and Synonyms for GlutenFreeBase</p> <ul style="list-style-type: none"> • A pizza base made from gluten-free dough. <p>Class Axioms</p> <ul style="list-style-type: none"> • GlutenFreeBase SubClassOf PizzaBase • GlutenFreeBase DisjointWith (each of) DeepPanBase
<p>Class Name: ThinAndCrispyBase</p> <p>Description and Synonyms for ThinAndCrispyBase</p> <ul style="list-style-type: none"> • A flattened base baked at high temperatures. <p>Class Axioms</p> <ul style="list-style-type: none"> • ThinAndCrispyBase SubClassOf PizzaBase • ThinAndCrispyBase DisjointWith (each of) DeepPanBase

Figure 5.6: Problematic HIT #2, with an incorrect disjointness axiom given in the class definitions

5.4.2 Ambiguous Errors

The third problematic HIT is depicted in Figure 5.7, showing the definition of the class `SpicyMargherita` as the subclass of `Margherita` and the existential restriction of `TobascoPepperSauce` as a topping.

The correct answer for this task was *No (known) Error*, as the class `SpicyMargherita` by itself does not contain any errors. If, however, it is (reasonably) assumed that `Margherita` is defined using proper existential and universal restrictions, i.e. two `someValuesFrom` restrictions on `hasTopping` for tomatoes and mozzarella, and an `allValuesFrom` closure axiom limiting the toppings of the pizza to just these two toppings, then the class `SpicyMargherita` would be unsatisfiable. The additional existential restriction of the `Tobasco` sauce and the closure axiom of the `Margherita` contradict each other.

62.5 percent of the participants voted for *Missing Closure* on this task and only 31.25 percent chose the correct answer *No (known) Error*.

To mitigate problems like these, the instructions given to the participants should include guidelines on how to vote in cases like this, where presented class definitions by themselves are not incorrect, only becoming problematic by making reasonable assumptions about the definitions of other classes.

5.5 Time Spent per Task

As mentioned in Section 5.1, on average 78.8 seconds were spent per task, with a standard deviation of 73.4 and a median of 53 seconds spent. The minimum time spent per task

Class Name: SpicyMargherita**Description and Synonyms for SpicyMargherita**

- A pizza Margherita with a spicy kick.

Class Axioms

- SpicyMargherita SubClassOf Margherita and (hasTopping some TobascoPepperSauce)

Figure 5.7: Problematic HIT #3, depending on the definition of Margherita, this class is unsatisfiable

was measured at 8 seconds and the maximum at 408 seconds (6 minutes 48 seconds).

Figure 5.8 shows two box plots of the time spent, one grouped by qualification level and one per error type. There are some outliers among all qualification levels and error types, which may be partly caused by distractions and breaks taken by the participants, which is reflected in the minimum and maximum values, as well as the standard deviation.

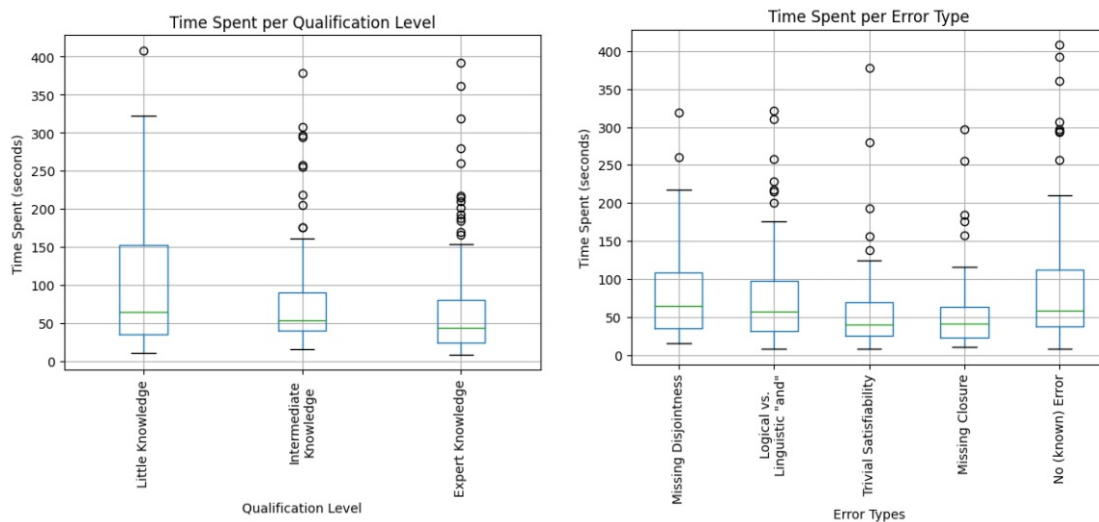


Figure 5.8: Time spent on tasks per qualification level and error type

It can be observed that the time spent differs by qualification level, with those participants classified as having little knowledge spending the most time. While the lower quartile and median do not differ very much compared to the higher qualification levels, the upper quartile is significantly larger. There also is a small difference between the higher two qualification levels, with those of the highest qualification level spending slightly less time per task than those with intermediate knowledge. This observation is further substantiated by the correlation coefficient of the qualification level and the time spent showing a notable negative correlation. The correlation coefficient of the qualification level with different time metrics: (i) total time spent: -0.54 , (ii) mean time spent: -0.54 ,

(iii) median time spent: -0.64.

Strong negative correlations with the median time spent are also observed for the two subjective skill levels: (i) ontology modeling using OWL: -0.68, (ii) prior experience with Crowdsourcing platforms: -0.57.

While the time spent per error type was comparable among the error types *Missing Disjointness*, *Logical vs. linguistic “and”* and *No (known) Error*, there was less time spent for tasks with the error types *Trivial Satisfiability* and *Missing Closure* axioms. Analogous to the high ratio of correct answers for these two error types, as discussed in Section 5.3, this may be attributed to either these two error types being easier to judge, a learning effect from the previous quiz concerned with universal and existential restrictions [58], or a combination of both.

5.6 Influence of Number of Aggregated Judgements

This section discusses the impact of aggregating multiple votes using majority voting, to explore possible answers to *RQ-3c*.

In order to obtain metrics of the results if multiple judgements had been collected, these aggregated judgements are simulated, inspired by the evaluation approach in [67]. Aggregated result metrics are calculated for different numbers of votes to aggregate. The maximum number of votes to aggregate is 15 because this is minimum number of votes available for any task - ideally there would have been 16 judgements for each task, one from each participant, but as described in Section 5.1, for one task only 15 votes are available.

For a number of votes $n = \{3, 5, 7, 9, 11, 13, 15\}$, the aggregated metrics are calculated by first randomly picking n judgements for each HIT and aggregating them using majority voting. The metrics precision, recall, F_1 score and Cohen’s kappa described in Section 4.6.2 are then calculated based on the aggregated results for all HITs. This process is repeated 1,000 times for each number of votes n and the mean value for each metric is used as the final result.

The calculated metrics can be seen in Table 5.1 and are visualized in Figure 5.9.

All four metrics show very similar differences between single judgements and the different numbers of votes aggregated. Comparing the result metrics of using single judgements and three votes aggregated, there is a notable improvement by the aggregation. Another small improvement is made by aggregating five instead of three votes. Beyond five votes the improvements gained when more votes are aggregated become increasingly minor, with the best results achieved by aggregating eleven votes. The results beyond eleven aggregated votes show very minor deterioration.

In case of a limited budget or workforce availability, a trade-off needs to be made if the limited resources should be committed to perform more tasks with fewer votes aggregated, or fewer tasks with more votes aggregated, thus if larger parts of an ontology, resp. more

No. of votes	Precision	Recall	F_1 score	Cohen's kappa
1	0.8492	0.8088	0.8113	0.7538
3	0.8922	0.8406	0.8407	0.7967
5	0.9021	0.8494	0.8485	0.8087
7	0.9046	0.8561	0.8549	0.8169
9	0.9064	0.8597	0.8586	0.8214
11	0.9071	0.8599	0.8589	0.8217
13	0.9063	0.8585	0.8575	0.8200
15	0.9025	0.8500	0.8488	0.8095

Table 5.1: Performance metrics for different numbers of votes aggregated by majority voting

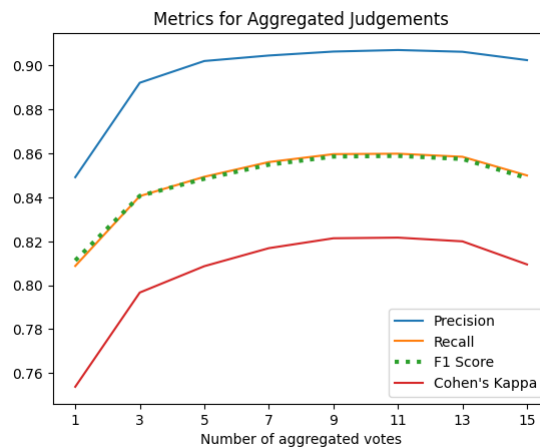


Figure 5.9: Visualization of the metrics after aggregation from Table 5.1

ontologies, should be evaluated with less accuracy, or smaller parts, resp. fewer ontologies, with higher accuracy. The next section provides a cost estimation based on the number of votes aggregated and discusses economical factors of the choice of the number of votes to aggregate.

The data from the empirical study therefore suggest that *RQ-3c* should be answered affirmatively, as a significant performance improvement is observed if votes are aggregated instead of using single votes.

5.7 Cost Estimation

This section gives an estimation of the cost of the evaluated DIWs, assuming that the participants would be employees. This assumption is made as human verifiers need prior knowledge in ontology engineering, which is a skill unlikely to be very prevalent on public

Crowdsourcing platforms such as Amazon Mechanical Turk. The cost estimation is therefore based on hourly rates of employees, assuming that if ontologies shall be verified using the proposed approach, staff trained in ontology engineering will be available or made available, e.g. via recruiting.

A majority of the study's participants are master's students, with only two participants being pre-doc researchers. As a base for the cost estimation, the standard personnel costs of the Austrian Science Fund (FWF)¹ are taken. Of the various employment categories, the category of doctoral candidates is used, as this would be the category the master's students would fall into once they finish their current studies.

The FWF's standard personnel costs put a doctoral candidate at € 2,237.60 gross salary per month with 30 hours per week. Approximating four weeks per month, this puts the gross hourly rate to € 18.65. It should be noted that this hourly rate serves as a basis for cost estimations, assuming that the human verifiers are employed and paid similarly to doctoral candidates. In the case of contractual work, hourly rates are more likely to fall into the range of € 50 to € 100, based on rates paid for contractual software engineering work by similarly qualified personnel in Austria.

Section 5.8 discusses the time spent per task, with a mean of 78.8 seconds and a median of 53 seconds per task. The substantial difference between the mean and median time is caused by a number of outliers, probably due to breaks taken by and distractions to the human verifiers. For the calculations below the mean of 78.8 seconds is taken, because breaks will be taken and distractions will occur too if the human verifiers are employed.

Assuming 78.8 seconds per task and an hourly rate of € 18.65, one candidate verification task approximately costs € 0.4082. Based on these assumptions, a human verifier could approximately judge 45.685 tasks per hour, or 1,371 tasks per week.

It should be noted that these calculations are based on the mean time spent per task from the empirical study, in which the participants performed 20 tasks. These calculations therefore do not consider a possible learning effect that would likely lead to a decrease of time spent per task by human verifiers that already performed considerable amounts of such tasks.

As Section 5.6 shows that aggregating multiple judgements per task leads to a significant increase of correct resulting votes, the costs are calculated for the numbers of votes to be aggregated, for which Table 5.1 shows the result metrics. The resulting costs for the various numbers of aggregated votes are shown in Table 5.2, in addition to the F_1 measure from Table 5.1, as a representative for the performance metrics (which all show comparable differences between different numbers of votes aggregated). Additionally, the delta between the F_1 scores between n and $n - 2$ votes is given, i.e. the improvement or decline of that metric in comparison to the next smaller number of votes aggregated.

It should be noted that the cost naturally increases linearly with the number of votes, as the cost per judgement is multiplied by the number of judgements to aggregate, while

¹<https://www.fwf.ac.at/en/research-funding/personnel-costs>

No. of votes	Estimated cost per task	F_1 score	ΔF_1
1	€ 0.4082	0.8113	-
3	€ 1.2247	0.8407	0.0294
5	€ 2.0411	0.8485	0.0078
7	€ 2.8576	0.8549	0.0064
9	€ 3.6741	0.8586	0.0037
11	€ 4.4905	0.8589	0.0003
13	€ 5.3070	0.8575	-0.0014
15	€ 6.1234	0.8488	-0.0087

Table 5.2: Estimated costs per task

the performance metric only shows significant improvements comparing the results of single votes with those of three aggregated votes with an improvement of almost three percent of accuracy, with little difference in the performance among three and 15 votes being aggregated. The optimal results are reached when aggregating eleven votes, which however amounts to eleven times the cost of taking the results of single votes and 3.7 times the cost of aggregating three votes.

As the performance differences between the different numbers of votes are rather small, from the point of view regarding cost-efficiency, the suggestion is to aggregate three votes for the final result, based on the data of this empirical study. Not performing any vote aggregation at all would also be a reasonable and more economical approach, accepting a loss of accuracy by approximately three percent.

5.8 Feedback

Following the quiz, participants were asked to provide feedback in a questionnaire, see *Appendix E: Feedback Questionnaire*. The questionnaire asks three questions about the participants' general enjoyment of the quiz (Q1), their willingness to take a similar quiz again (Q2) and how much the quiz improved their understanding of ontology verification (Q3), each on a four-point Likert scale. Questions number four and five inquired about the easiest (Q4) and hardest modeling errors (Q5) to identify, respectively. Multiple error types could be chosen for these two questions each. Finally, participants could optionally leave a comment. The results of the feedback questionnaire are visualized in *Appendix G: Results of the Feedback Questionnaire*.

81.25 percent of the participants at least slightly enjoyed the quiz, with 31.25 percent very much so. 81.25 percent would also like to take a similar quiz again, e.g. in a distance learning course, with 43.75 percent stating that they would like that very much. All participants stated that they thought that the quiz helped them better understand ontology verification, with 62.5 percent voting in strong agreement.

Trivial Satisfiability and *Missing Closure* were voted to be the easiest error types to identify, receiving 13 and 12 votes, respectively. As discussed in the previous sections, this could be partly attributed to prior experience with the verification of existential and universal restrictions from the quiz in [58].

Missing Disjointness was voted to be the hardest to identify by far, receiving 13 votes, while the other error types all received three or less votes.

Comparing these rankings with the actual performances per error type, as discussed in Section 5.3, for the two classes *Trivial Satisfiability* and *Missing Closure* the subjective assessment of the participants aligns with the ratio of correct answers, as the two error types have 97.9 and 100 percent correct answers, respectively.

While the performance for tasks of the error type *Missing Disjointness* is lower than for these two types, the performance was the worst at 75 percent of correct answers for the error type *Logical vs. Linguistic “and”*, which was neither voted as particularly easy, nor hard to identify, receiving five votes to be the easiest and 3 votes to be the hardest error type to identify.

As already discussed in Sections 5.3 and 5.4, two of the problematic tasks caused a confusion with the decision between judging the tasks as either the absence of an error or *Missing Disjointness*, attributing 20 incorrect judgements for *Missing Disjointness*, whereas *No (known) Error* would have been correct. These problematic HITs likely also are the explanation for *Missing Disjointness* being perceived as the hardest type to identify by far.

Regarding the easiest and hardest types to identify, there is a small inconsistency in the responses, as the error type *Missing Disjointness* has received four votes as the easiest modeling error to identify and 13 votes for it to be the hardest to identify, it follows from the pigeonhole principle that at least one participant has voted for *Missing Disjointness* to be both the easiest and the hardest type to identify.

Ten participants left a comment, with multiple comments mentioning that it was unclear if the classes `Cat` and `Dog` should have been considered to be disjoint in the last question in the qualification test. In each of the quiz sessions too, one participant asked if they should make this assumption - which they were answered that the disjointness was a reasonable assumption to make.

The other comments did not yield further insights, with one comment each (i) criticizing the too long and irrelevant domain descriptions, (ii) pointing out the ambiguity of the task with the class `SpicyMargherita`, (iii) stating that they felt to be forced to make too many assumptions, and (iv) noting that tasks with two class definitions are confusing. Two comments expressed a perceived need for more detailed domain descriptions.

Conclusion & Future Work

This chapter summarizes the main contributions of this thesis in Section 6.1, revisits the research questions and puts the findings in their context in Section 6.2 and discusses the limitations of the findings, as well as future research opportunities in Section 6.3.

6.1 Summary

In this thesis, the integration of automatic machine computations with HC processes into hybrid-human machine workflows for the detection of common ontology modeling errors was investigated as a solution paradigm for addressing ontology verification, in order to improve the scalability of Crowdsourcing-based ontology verification by an automatic pre-selection of possible errors.

Therefore, as a first step, four common error types in ontology modeling were selected based on literature review. The selected error types are *Missing Disjointness Axioms*, *Confusion between logical and linguistic “and”*, *Trivially satisfiable allValuesFrom Restrictions* and *Missing Closure Axioms*.

For each of these error types, a heuristic was designed to automatically detect possible errors of the type, i.e. *defect candidates*, relying on an ontology reasoner.

These detected defect candidates then need to be verified by humans, therefore, a specific type of hybrid human-machine workflow is proposed, called *Defect Identification Workflow (DIW)*. Each DIW aims to detect errors of a specific ontology modeling error type and consists of two steps.

The first step performs the automatic defect candidate detection using the designed heuristic for the targeted error type. The second step then presents the resulting defect candidates to human verifiers, to let human judgement distinguish between true defects and false positives.

Therefore, a suitable HC interface was designed, presenting the class or class combination of a defect candidate, whereas the defining axioms and context information are given. The axioms defining a class are rendered in MOS and for context information the values of the SKOS metadata annotation properties are used. The designed HC interface is used to show the defect candidates one at a time for the human verifier to classify, by either selecting the type of error present, or marking the task as a false positive.

The four DIWs were implemented in a prototype, whereas the designed heuristics were implemented and Amazon Mechanical Turk was integrated for the execution of the HC tasks.

This prototypical implementation of the DIWs, including the designed HC interfaces, was evaluated in an empirical study with descriptive statistics, for which 20 defects and false positives were seeded into the well-known pizza ontology to be used as the data set. Participants of the study completed five steps in succession: (i) self-assessment form, (ii) qualification test, (iii) tutorial, (iv) quiz and (v) feedback questionnaire.

The self-assessment form and qualification test gathered data on subjective assessments of prior knowledge and an objective qualification level regarding ontology modeling, respectively. The quiz was the main part of the study, in which the participants were asked to complete the 20 tasks presenting the defect candidates on MTurk.

16 students and research staff participated in the study, with 13 of them being assigned one of the higher two out of four qualification levels and no participant classified as having no prior knowledge regarding ontology modeling.

Overall, 80.9 percent of the tasks were judged correctly, with the performance depending on the qualification level, showing a positive correlation. It was observed that the performance on different tasks also varied by the present error type, with perfect, resp. almost perfect, scores for the error types *Trivially satisfiable allValuesFrom Restrictions* and *Missing Closure Axioms*. For the error types *Missing Disjointness Axioms* and *Confusion between logical and linguistic "and"* 85.4 percent and 75.0 percent of the answers were correct, respectively.

The fifth category of tasks, that of no or unknown errors, showed only two out of three correct answers. This must at least partially be attributed to three distinct tasks with poor results, for which either incorrect axioms were given in the class definitions, or the error type was ambiguous.

Furthermore, the data from the empirical study suggest that the aggregation of multiple votes for a conclusive human judgement on a task brings a significant improvement compared to only collecting single votes. Substantial differences between a single vote and three votes aggregated were observed. The differences among three and 15 aggregated votes were marginal, thus from the perspective of cost-efficiency either one or three judgements should be collected per task, depending on the budget and the size of the ontology to verify.

6.2 Conclusion

This section discusses the thesis' findings in relation to the research questions formulated in Section 1.2.

RQ-1 *How can specific modeling errors in ontologies be identified using hybrid human-machine workflows?*

This thesis introduced *Defect Identification Workflows*, which are hybrid human-machine workflows consisting of two steps. Each such workflow aims to identify errors of one specific type of modeling error. The first step is an automatic defect candidate detection, based on specifically designed heuristics for each error type relying on the capabilities of an ontology reasoner. In the second step, these automatically detected possible errors are verified by humans using Crowdsourcing to classify them as either true defects or false positives.

The empirical study's results suggest that the hybrid human-machine workflows are an effective approach to identifying specific types of errors, with a precision value of up to 0.9071 and a recall value of up to 0.8599 in case multiple human judgements are aggregated. This means that more than 90 percent of the reported true defects really are true defects, and that almost 86 percent of the true defects are reported as such.

RQ-2 *What are suitable Human Computation interfaces to enable verification of specific error types in hybrid human-machine workflows?*

To allow a verification of the automatically generated defect candidates by humans using Crowdsourcing, an HC interface was designed. The resulting HC task design presents the class definition and context information of the ontology class or class combination that comprises a defect candidate to a human verifier. The human verifier has the option to either select the type of error present in the shown ontology class or class combination, or to mark the task as a false positive.

The proposed HC task design was also used in the empirical study, and given the positive results thereof, can thus be deemed suitable.

RQ-3 *Is there an influence of certain factors, such as (a) prior knowledge and qualification of the human workers, (b) the type of modeling error under verification, or (c) the number of human votes aggregated for crowdsourced judgements, on the error detection rate of the hybrid human-machine workflows for identifying specific types of modeling errors in ontologies, or the time spent by human verifiers thereby?*

To gain insights on the influence of these factors, the empirical study was designed and executed. The results of the study indicate that all three of these factors have an influence on the effectiveness of the approach.

Regarding the *effect of prior knowledge*, a correlation between the ratio of correctly reported true defects and the objectively assessed qualification level of the human verifiers in ontology modeling is observed. In the study, participants were assigned one of four

qualification levels based on their answers to a qualification test. Those assigned the lower-middle out of four qualification levels achieved 71.6 percent of correct answers, those with the upper-middle level reached 80.8 percent and those with the highest level correctly judged 84.9 percent of the defect candidates.

Substantial differences in performances are also noted among *different error types*, with instances of two error types getting correctly reported every time, respectively almost every time (97.9 percent). The two other error types were also reported correctly at reasonable rates at 75 and 85.4 percent. Those instances where defect candidates did not contain one of the four selected error types showed a worse rate of correct judgements at 67.7 percent, whereas this must at least partially be attributed to weaknesses in the task design, which are discussed for future work to improve.

The third factor, the *number of human votes to aggregate* for a final judgement, also shows a strong influence on the error detection rate of the hybrid human-machine workflows, with substantial improvements of accuracy when aggregating three or five votes compared to reporting a single vote as the final result. Very minor improvements can be achieved by aggregating more than five votes, up to eleven, with the performance metrics slightly deteriorating if more than eleven votes are aggregated. As the difference in performance between aggregating three and five votes is also small, the study's results suggest to aggregate three votes per task for the best cost-efficiency.

All three of the considered factors are also observed to have an *influence on the time spent by human verifiers*, with those human verifiers of higher qualification levels spending less time to perform the verification tasks and time spent varying by error type. In case multiple votes are aggregated, the time spent by human verifiers naturally is multiplied by the number of votes collected for aggregation.

6.3 Limitations & Future Work

This section presents limitations of this thesis' findings and opportunities for future work.

Evaluation of the Defect Candidate Detection Heuristics

As the evaluation approach of this thesis was concerned with the effectiveness of the hybrid human-machine workflows for defect identification as a whole and the human verification part in particular, the four designed heuristics for defect candidate detection were themselves not evaluated beyond their proof-of-concept usage in the described study, in which they were used to identify the 20 seeded defects and false positives.

Evaluating the effectiveness of the heuristics-based detection of error candidates requires a gold standard to compare the actually generated defect candidates against. Thus, one or more ontologies that contain errors of the types detected by the heuristics and a list of these errors, checked or generated scientifically, or by experts, is required. Such a data set including a gold standard does not exist yet and creating one would require much

effort from ontology modeling experts, as for a significant large-scale evaluation following this design many of these errors would need to be detected or seeded.

The main contribution of this thesis is a novel approach for detecting ontology modeling errors using hybrid human-machine workflows. The machine part of the hybrid processes is based on heuristics, each specifically designed for one error type. While four heuristics for defect candidate detection are presented in this thesis, these pose as examples only and should they lack feasibility, effectiveness or efficiency, there would be no implication to the feasibility, effectiveness and efficiency of the proposed method in general.

The fact that ontology modeling pitfalls can be detected automatically has been shown with the ontology pitfall scanner, in which the automated detection of 32 out of 40 pitfalls from the accompanying pitfall catalogue was implemented [40, 26]. As described in Section 2.2, pitfalls can be viewed as equal, or at least similar to bad smells, both of which are concepts that indicate the heightened risk of an error, and therefore could both be considered as synonymous with the term *defect candidate*.

In addition to their effectiveness not being evaluated, the feasibility of the four designed defect candidate detection steps regarding their performance on large ontologies should be considered in future work.

Large-scale Empirical Evaluation

In the scope of this thesis only a small empirical study with descriptive statistics, a small number of participants and a small reference ontology was conducted. The proposed method was thus not evaluated with a large sample of typical subjects. Following the positive results of the study presented in this thesis, further research resources can be invested into a large-scale empirical evaluation of the proposed hybrid human-machine workflows for ontology verification with minimized risk.

To substantiate the positive results of this thesis' evaluation, a large-scale evaluation with real-world ontologies of considerable size with a larger number of ontology engineers, or alternatively domain experts, should be performed.

Extending the Set of Detected Error Types

The set of error types targeted for detection in the verification workflows is relatively small at the size of four. Given the abundant existence of literature on common ontology modeling errors, pitfalls and anti-patterns, as discussed in Section 2.2, defect candidate detection heuristics for further error types are an opportunity for future research.

Focus on Most Relevant Ontology Parts

The proposed workflows are executed on whole ontologies, not considering if any parts, resp. modules, of the ontology under verification are more relevant than others to intended applications. In case of limited resources, either by limited budget or workforce, or very large ontologies, focusing the verification efforts on the most relevant parts will

result in a larger quality improvement compared to partially executing the workflows on random parts of the ontology until the resources are exhausted. Such a value-, risk-, or usage-based approach could thus concentrate resources for the verification of the ontology parts most relevant to the use cases at hand. While focusing on specific parts of an ontology could be easily accomplished by restricting the defect candidate detection to only yield defect candidates from the chosen parts, it must be defined what *most relevant* means for each intended application.

If, for example, a question-answering system that uses an ontology is considered, usages of ontology elements by the executed queries, resp. asked questions, could be kept track of. The most relevant ontology elements, resp. parts, could in this example either be those that show the heaviest usage among all queries, or those that are used by specific types of queries, which e.g. are the most mission-critical or provide the most business value.

Future research could thus explore such focused approaches in conjunction with the method proposed in this thesis, to provide approaches for verifying the most relevant parts of an ontology with as little resources as possible.

Hybrid Human-Machine Workflows beyond Defect Identification

The framework for *verification workflows*, proposed in Section 3.5, supports workflows beyond the evaluated two-step DIWs. More complex workflows, e.g. as depicted in Figure 3.10, can be designed in future work, facilitating the full potential of the designed concept. One such opportunity would for example be the design of a hybrid human-machine workflow that goes beyond defect detection and tries to semi-automatically repair detected defects, applying a *Find-Fix-Verify* approach as discussed in Section 2.3.1 and Section 3.5.

The proposed DIWs only identify defects semi-automatically using heuristics-based candidate detection and Crowdsourcing-based verification of candidates, which still leaves the repair of the detected defects to be done manually by ontology engineers. Thus the scalability of the defect identification is improved by the proposed method, but not the repair. Designing a *Find-Fix-Verify* approach to defect identification with subsequent repair would bring the scalability improvements to the repair stage of ontology verification as well.

Usefulness of Laymen Crowds

Due to the fact that none of the study's participants lacked prior knowledge in ontology modeling, no conclusions can be drawn about the usefulness of laymen crowds in the proposed hybrid-human machine ontology verification workflows. Laymen crowds, in contrast to those experienced in ontology modeling, are more broadly available, e.g. on public Crowdsourcing platforms. As the proposed HC task design presumably requires at least some level of understanding of ontology modeling, future work could be concerned with proposing a task design enabling the usage of laymen crowds. This would in turn

allow for the execution of hybrid human-machine verification workflows regardless of the availability of ontology engineers and also imply reduced overall cost, assuming that laymen are compensated lower amounts than those experienced with ontology modeling. Furthermore, as laymen workforce is abundantly available on Crowdsourcing platforms, ontology verification could be parallelized by many laymen working at the same time, speeding up the verification process.

Finding an HC task design that enables laymen to perform the required tasks thus could eliminate the dependency to ontology engineers which reduces risk due to their limited availability, make the verification more cost-effective and also possibly faster.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

1.1	Design science research process of this thesis	5
1.2	Methods and contributions of this thesis	6
3.1	Correctly modelled class <code>CardinalePizza</code>	26
3.2	Example for missing disjointness axioms	27
3.3	Class <code>CardinalePizza</code> with a confusion between the logical and linguistic meaning of “and”	28
3.4	Class <code>CardinalePizza</code> without existential restrictions allowing the trivial satisfiability of its universal restriction	29
3.5	Class <code>CardinalePizza</code> without a closure axiom	30
3.6	Conceptual overview of Defect Identification Workflows as an IDEF-0 diagram	31
3.7	Overview on the components of the heuristic for detecting candidates for missing disjointness axioms as an IDEF-0 diagram	34
3.8	Example HIT from the quiz with one class definition	41
3.9	Example HIT from the quiz with two class definitions	42
3.10	Complex verification workflow for the diagnosis of trivially satisfiable <code>allValuesFrom</code> restrictions	45
3.11	Prototype Architecture	46
4.1	Quiz phases overview as an IDEF-0 diagram	54
4.2	Example HIT from the tutorial	55
5.1	Calculated qualification levels	61
5.2	Correct judgements by qualification level	62
5.3	Correct judgements by error type	63
5.4	Confusion matrix of judged error types	64
5.5	Problematic HIT #1	65
5.6	Problematic HIT #2	66
5.7	Problematic HIT #3	67
5.8	Time spent on tasks per qualification level and error type	67
5.9	Visualization of the metrics after aggregation	69



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

4.1	Classes removed from the pizza ontology	51
4.2	Classes presented in the quiz tasks	58
5.1	Performance metrics for different numbers of votes aggregated by majority voting	69
5.2	Estimated costs per task	71



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Algorithms

3.1	Heuristic for detecting candidates of missing disjointness axioms	33
3.2	Heuristic for detecting candidates of confusions between the logical and linguistic meaning of “and”	35
3.3	Heuristic for detecting candidates of trivially satisfiable universal restrictions	37
3.4	Heuristic for detecting candidates of missing closure axioms	38



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acronyms

- API** Application Programming Interface. 46
- DIW** Defect Identification Workflow. xi, 4–6, 25, 30, 31, 40, 43–46, 49, 54, 57, 69, 73–75, 78, 81
- EKP** Encyclopedic Knowledge Pattern. 16
- GWAP** Games with a Purpose. 16, 17
- HC** Human Computation. xi, 2–6, 9, 13, 15, 16, 21–23, 25, 30, 31, 39, 40, 44–47, 49, 50, 54, 56, 57, 59, 62, 73–75, 78, 79
- HIT** Human Intelligence Task. 21, 41, 42, 46, 53, 55, 56, 59, 64–68, 72, 81
- HOV** Hybrid Human-Machine Ontology Verification. 25, 30, 49, 54, 57, 63
- HTML** HyperText Markup Language. 20, 21
- MOS** Manchester OWL Syntax. 13, 14, 40, 41, 74
- MTurk** Amazon Mechanical Turk. 21, 46, 52, 53, 55, 59, 62, 74
- OWL** Web Ontology Language. 9, 13, 27–29, 35, 40, 41, 43, 46, 51, 56, 60, 61
- RDF** Resource Description Framework. 9, 17, 19
- RDFS** RDF Schema. 56
- REST** Representational State Transfer. 46
- SKOS** Simple Knowledge Organization System. 51, 74
- SPARQL** SPARQL Protocol and RDF Query Language. 18, 19
- URI** Uniform Resource Identifier. 20, 21
- VOWL** Visual Notation for OWL Ontologies. 26, 40



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

- [1] Maribel Acosta, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Fabian Flöck, and Jens Lehmann. Detecting linked data quality issues via crowdsourcing: A dbpedia study. *Semantic web*, 9(3):303–335, 2018.
- [2] Harith Alani, Christopher Brewster, and Nigel Shadbolt. Ranking ontologies with AKTiveRank. In *International Semantic Web Conference*, pages 1–15. Springer, 2006.
- [3] Kent Beck, Martin Fowler, and Grandma Beck. Bad smells in code. *Refactoring: Improving the design of existing code*, 1(1999):75–88, 1999.
- [4] Michael S Bernstein, Greg Little, Robert C Miller, Björn Hartmann, Mark S Ackerman, David R Karger, David Crowell, and Katrina Panovich. Soy lent: a word processor with a crowd inside. In *Proceedings of the 23rd annual ACM symposium on User interface software and technology*, pages 313–322, 2010.
- [5] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia-A crystallization point for the Web of Data. *Journal of web semantics*, 7(3):154–165, 2009.
- [6] Alessandro Bozzon, Marco Brambilla, Stefano Ceri, Matteo Silvestri, and Giuliano Vesci. Choosing the Right Crowd: Expert Finding in Social Networks. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, page 637–648, New York, NY, USA, 2013. Association for Computing Machinery.
- [7] Janez Brank, Marko Grobelnik, and Dunja Mladenic. A survey of ontology evaluation techniques. In *Proceedings of the conference on data mining and data warehouses (SiKDD 2005)*, pages 166–170. Citeseer Ljubljana, Slovenia, 2005.
- [8] C. Brewster, H. Alani, S. Dasmahapatra, and Y. Wilks. Data driven ontology evaluation. In *LREC*, 2004.
- [9] Andrew Burton-Jones, Veda C Storey, Vijayan Sugumaran, and Punit Ahluwalia. A semiotic metrics suite for assessing the quality of ontologies. *Data & Knowledge Engineering*, 55(1):84–102, 2005.

- [10] Philipp Cimiano. *Ontology Learning and Population from Text*. Springer US, 2006.
- [11] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.
- [12] Gianluca Demartini. Hybrid human–machine information systems: Challenges and opportunities. *Computer Networks*, 90:5–13, 2015.
- [13] Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. Zen-Crowd: Leveraging Probabilistic Reasoning and Crowdsourcing Techniques for Large-Scale Entity Linking. In *Proceedings of the 21st International Conference on World Wide Web, WWW '12*, page 469–478, New York, NY, USA, 2012. Association for Computing Machinery.
- [14] Djellel Eddine Difallah, Gianluca Demartini, and Philippe Cudré-Mauroux. Pick-a-Crowd: Tell Me What You like, and i'll Tell You What to Do. In *Proceedings of the 22nd International Conference on World Wide Web, WWW '13*, page 367–374, New York, NY, USA, 2013. Association for Computing Machinery.
- [15] Emelie Engström, Margaret-Anne Storey, Per Runeson, Martin Höst, and Maria Teresa Baldassarre. How software engineering research aligns with design science: a review. *Empirical Software Engineering*, 25(4):2630–2660, 2020.
- [16] Miriam Fernández, Chwhynny Overbeeke, Marta Sabou, and Enrico Motta. What makes a good ontology? A case-study in fine-grained knowledge reuse. In *Asian semantic web conference*, pages 61–75. Springer, 2009.
- [17] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. HermiT: an OWL 2 reasoner. *Journal of Automated Reasoning*, 53(3):245–269, 2014.
- [18] Asunción Gómez-Pérez. Towards a framework to verify knowledge sharing technology. *Expert Systems with applications*, 11(4):519–529, 1996.
- [19] Asunción Gómez-Pérez, Mariano Fernández-López, and Oscar Corcho. *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer Science & Business Media, 2006.
- [20] Nicola Guarino and Christopher A Welty. An overview of OntoClean. *Handbook on ontologies*, pages 151–171, 2004.
- [21] Jörn Hees, Thomas Roth-Berghofer, Ralf Biedert, Benjamin Adrian, and Andreas Dengel. BetterRelations: using a game to rate linked data triples. In *Annual Conference on Artificial Intelligence*, pages 134–138. Springer, 2011.
- [22] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, pages 75–105, 2004.

- [23] P Hitzler et al. Anti-patterns in ontology-driven conceptual modeling: the case of role modeling in OntoUML. *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, 25:161, 2016.
- [24] Matthew Horridge, Nick Drummond, John Goodwin, Alan L Rector, Robert Stevens, and Hai Wang. The Manchester OWL syntax. In *OWLed*, volume 216, 2006.
- [25] Jeff Howe. *Crowdsourcing: How the power of the crowd is driving the future of business*. Random House, 2008.
- [26] C Maria Keet, Mari Carmen Suárez-Figueroa, and María Poveda-Villalón. Pitfalls in ontologies and tips to prevent them. In *International Joint Conference on Knowledge Discovery, Knowledge Engineering, and Knowledge Management*, pages 115–131. Springer, 2013.
- [27] Magnus Knuth, Johannes Hercher, and Harald Sack. Collaboratively patching linked data. *arXiv preprint arXiv:1204.2715*, 2012.
- [28] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. Test-driven evaluation of linked data quality. In *Proceedings of the 23rd international conference on World Wide Web*, pages 747–758, 2014.
- [29] Jens Lehmann and Lorenz Bühmann. ORE-a tool for repairing and enriching knowledge bases. In *International Semantic Web Conference*, pages 177–193. Springer, 2010.
- [30] Alexander Maedche and Steffen Staab. Measuring similarity between ontologies. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 251–263. Springer, 2002.
- [31] Jonathan M Mortensen, Mark A Musen, and Natalya F Noy. Crowdsourcing the verification of relationships in biomedical ontologies. In *AMIA Annual symposium proceedings*, volume 2013, page 1020. American Medical Informatics Association, 2013.
- [32] OWL 2 Web Ontology Language Manchester Syntax (Second Edition). <https://www.w3.org/TR/2012/NOTE-owl2-manchester-syntax-20121211/>, 2012. Accessed: 2021-08-27.
- [33] Andrea Giovanni Nuzzolese, Valentina Presutti, Aldo Gangemi, Silvio Peroni, and Paolo Ciancarini. Aemoo: Linked data exploration based on knowledge patterns. *Semantic Web*, 8(1):87–112, 2017.
- [34] OWL 2 Web Ontology Language Document Overview (Second Edition). <https://www.w3.org/TR/2012/REC-owl2-overview-20121211/>, 2012. Accessed: 2021-08-27.

- [35] OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). <https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/>, 2012. Accessed: 2021-08-27.
- [36] Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 8(3):489–508, 2017.
- [37] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.
- [38] Rafael Penaloza and Barış Sertkaya. Understanding the complexity of axiom pinpointing in lightweight description logics. *Artificial Intelligence*, 250:80–104, 2017.
- [39] Robert Porzel and Rainer Malaka. A task-based approach for ontology evaluation. In *ECAI Workshop on Ontology Learning and Population, Valencia, Spain*, pages 1–6. Citeseer, 2004.
- [40] María Poveda-Villalón, Asunción Gómez-Pérez, and Mari Carmen Suárez-Figueroa. Oops!(ontology pitfall scanner!): An on-line tool for ontology evaluation. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 10(2):7–34, 2014.
- [41] Alexander J Quinn and Benjamin B Bederson. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1403–1412, 2011.
- [42] Alan Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch, Robert Stevens, Hai Wang, and Chris Wroe. OWL pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 63–81. Springer, 2004.
- [43] Patrick Rodler, Dietmar Jannach, Konstantin Schekotihin, and Philipp Fleiss. Are query-based ontology debuggers really helping knowledge engineers? *Knowledge-Based Systems*, 179:92–107, 2019.
- [44] Boonsita Roengsamut and Kazuhiro Kuwabara. Interactive refinement of linked data: toward a crowdsourcing approach. In *Asian Conference on Intelligent Information and Database Systems*, pages 3–12. Springer, 2015.
- [45] Boonsita Roengsamut, Kazuhiro Kuwabara, and Hung-Hsuan Huang. Toward gamification of knowledge base construction. In *2015 International Symposium on Innovations in Intelligent Systems and Applications (INISTA)*, pages 1–7. IEEE, 2015.

- [46] Catherine Roussey, Oscar Corcho, and Luis Manuel Vilches-Blázquez. A catalogue of OWL ontology antipatterns. In *Proceedings of the fifth international conference on Knowledge capture*, pages 205–206, 2009.
- [47] Marta Sabou, Lora Aroyo, Kalina Bontcheva, Alessandro Bozzon, and Rehab K Qarout. Semantic Web and Human Computation: The status of an emerging field. *Semantic Web*, 9(3):291–302, 2018.
- [48] Marta Sabou, Kalina Bontcheva, Arno Scharl, and Michael Föls. Games with a purpose or mechanised labour? A comparative study. In *Proceedings of the 13th International Conference on Knowledge Management and Knowledge Technologies*, pages 1–8, 2013.
- [49] Cristina Sarasua, Elena Simperl, and Natalya F. Noy. CrowdMap: Crowdsourcing Ontology Alignment with Microtasks. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *The Semantic Web – ISWC 2012*, pages 525–541, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- [50] Cristina Sarasua, Elena Simperl, Natasha F Noy, Abraham Bernstein, and Jan Marco Leimeister. Crowdsourcing and the semantic web: A research manifesto. *Human Computation*, 2(1), 2015.
- [51] S. Shabani and M. Sokhn. Hybrid Machine-Crowd Approach for Fake News Detection. In *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, pages 299–306, 2018.
- [52] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.
- [53] SKOS Simple Knowledge Organization System Primer. <https://www.w3.org/TR/2009/NOTE-skos-primer-20090818>, 2009. Accessed: 2021-08-27.
- [54] SKOS Simple Knowledge Organization System Reference. <https://www.w3.org/TR/2009/REC-skos-reference-20090818>, 2009. Accessed: 2021-08-27.
- [55] Gem Stapleton, Michael Compton, and John Howse. Visualizing OWL 2 using diagrams. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 245–253. IEEE, 2017.
- [56] Darijus Strasunskas and Stein L Tomassen. The role of ontology in enhancing semantic searches: the EvOQS framework and its initial validation. *International journal of Knowledge and Learning*, 4(4):398–414, 2008.
- [57] Rudi Studer, V.Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1):161 – 197, 1998.

- [58] Stefani Stoyanova Tsaneva. Human-Centric Ontology Evaluation. Master's thesis, TU Wien, 2021.
- [59] VOWL: Visual Notation for OWL Ontologies, Specification of Version 2.0. <http://vowl.visualdataweb.org/v2/>, 2014. Accessed: 2021-08-27.
- [60] Maja Vukovic and Arjun Natarajan. Operational Excellence in IT Services Using Enterprise Crowdsourcing. In *2013 IEEE International Conference on Services Computing*, pages 494–501, 2013.
- [61] Paul Warren, Paul Mulholland, Trevor Collins, and Enrico Motta. Improving comprehension of knowledge representation languages: A case study with Description Logics. *International Journal of Human-Computer Studies*, 122:145–167, 2019.
- [62] Christopher Welty and Nicola Guarino. Supporting ontological analysis of taxonomic relationships. *Data & knowledge engineering*, 39(1):51–74, 2001.
- [63] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.
- [64] Gerhard Wohlgenannt, Marta Sabou, and Florian Hanika. Crowd-based ontology engineering with the uComp Protégé plugin. *Semantic Web*, 7(4):379–398, 2016.
- [65] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.
- [66] Amrapali Zaveri, Dimitris Kontokostas, Mohamed A Sherif, Lorenz Bühmann, Mohamed Morsey, Sören Auer, and Jens Lehmann. User-driven quality evaluation of dbpedia. In *Proceedings of the 9th International Conference on Semantic Systems*, pages 97–104, 2013.
- [67] Markus Zlabinger, Marta Sabou, Sebastian Hofstätter, Mete Sertkan, and Allan Hanbury. DEXA: Supporting Non-Expert Annotators with Dynamic Examples from Experts. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2109–2112, 2020.

Appendices

Appendix A: Quiz Guidelines Page

SAIKS Experiment Guidelines

Phase 1 - Skill Assessment & Preparation

1. Self-Assessment Test (Google Forms, approx. 10 min.)
<https://forms.gle/uncCjnQ1D6RyTwRY8>
2. Qualification Test (Mechanical Turk, approx. 20 min.)
<https://workersandbox.mturk.com/projects/3J06FGISM2SS29GYAG7ZQ6FFESDK5V/tasks>
(In case the above link is broken, please try searching for the keyword "saiks2021-qualification" at <https://workersandbox.mturk.com/>)
3. Tutorial (Mechanical Turk, approx. 20 min.)
Please take your time to read through the instructions and examples, these are the same as for the quiz.
<https://workersandbox.mturk.com/projects/3787DLY3ZC4SHA5232JFO7PUKOYT1O/tasks>
(In case the above link is broken, please try searching for the keyword "saiks2021-tutorial" at <https://workersandbox.mturk.com/>.)

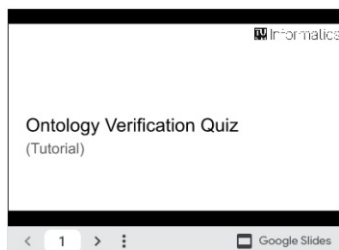
Phase 2 - Quiz

4. Quiz (Mechanical Turk, approx. 60 min.)
The instructions and examples are available during the quiz as well.
<https://workersandbox.mturk.com/projects/3XXLL7ZNOJSYRVAJX6E3JL3DC9Z52E/tasks>
(In case the above link is broken, please try searching for the keyword "saiks2021-quiz" at <https://workersandbox.mturk.com/>.)

Phase 3 - Feedback

5. Feedback Questionnaire (Google Forms, approx. 10 min.)
<https://forms.gle/VAomYBrx7pa5HJei8>

Slides



Appendix B: Self-Assessment Form

Self-Assessment Test

This self-assessment test is part of an experiment at TU Wien.

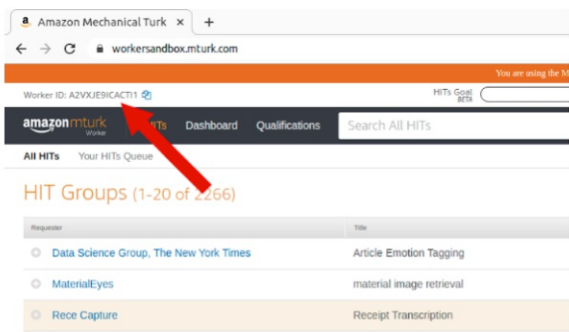
Your answers will be treated anonymously. Your student ID will only be used to connect your answers to your inspection records. No individual information will be made public in any form.

* Erforderlich

Student ID (Matrikelnummer) *

Meine Antwort _____

Mechanical Turk Sandbox Worker ID *



Meine Antwort _____

English Language Skills

For the question below, please consider the following levels:

- 1 - no understanding: My understanding of English is very basic.
- 2 - little understanding: I can understand and use familiar everyday expressions and very basic phrases.
- 3 - some understanding: I can understand the main points of clear standard input on familiar matters regularly encountered in work, school, leisure, etc.
- 4 - expert understanding: I can understand the main ideas of complex text on both concrete and abstract topics, including technical discussions in my field of specialization, and can recognize implicit meaning.

Q1: How would you rate your understanding of English documents? *

no understanding 1 2 3 4 expert understanding

Experience with Formal Logics

For the question below, please consider the following levels:

- 1 - no knowledge: I don't have experience in the area.
- 2 - little knowledge: I am aware of the basic symbolic notation and understand the meaning of logical conjunctions and quantifiers.
- 3 - some knowledge: I have an understanding of logical axioms and can derive the conveyed implications from them.
- 4 - expert knowledge: I fully understand logical axioms and can derive explicit and implicit implications from them.

Q2: How would you rate your knowledge of formal logic? *

	1	2	3	4	
no knowledge	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	expert knowledge

General Modeling Skills

For the question below, please consider the following levels:

- 1 - no knowledge = I have no knowledge in the area.
- 2 - little knowledge = I am aware of the basic model components and can recognise them in graphical and textual representations.
- 3 - some knowledge = I have performed modeling as part of my education/study assignments.
- 4 - expert knowledge = I have performed extensive modeling during my professional employment.

Q3: How would you rate your knowledge in Model-Driven Engineering? *

	1	2	3	4	
no knowledge	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	expert knowledge

Ontology Modeling Skills

For the questions below, please consider the following levels:

- 1 - no knowledge = I have no knowledge in the area.
- 2 - little knowledge = I am aware of the basic components of ontologies and can recognise them in graphical and textual representations.
- 3 - some knowledge = I have an understanding of the implications of ontology axioms and restrictions.
- 4 - expert knowledge = I can perform reasoning with ontology models, as well as compare and relate them to each other.

Q4: How would you rate your knowledge in ontology modeling? *

	1	2	3	4	
no knowledge	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	expert knowledge

Q5: How would you rate your knowledge of web-based knowledge representation languages (e.g., RDF(S), OWL)? *

	1	2	3	4	
no knowledge	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	expert knowledge

Crowdsourcing

Q6: Do you have any prior experience working with Crowdsourcing platforms? *

Yes
 No

Thank you for your participation!

Senden

Appendix C: Qualification Test

Qualification Test

Make sure to **accept the HIT** before you start answering the questions. Click on the button labelled "View Instructions" for helpful information regarding this task.
Please note that you **cannot go back to this section** once you continue to the next section!

[View Instructions](#)

Section 1: Basic Understanding

This section tests your understanding of basic ontology components and the ability to recognise them in textual representations.

Consider the model given below and answer questions 1 and 2.

PersonTypeA SubClassOf hasDaughter some Daughter
PersonTypeA SubClassOf hasSon some Son
PersonTypeA SubClassOf hasPet only Dog

1. Identify the main model components from the model

How many named classes can you identify from the model?

How many relations can you identify from the model?

2. Identifying the different quantifiers from the model

How many universal restrictions (allValuesFrom) can you identify in the model?

How many existential restrictions (someValuesFrom) can you identify in the model?

[Continue to Section 2](#)

Section 2: Intermediate Knowledge

This section tests your understanding of the implications of ontology axioms and restrictions.

Consider the model and answer question 3 below.

PetLoverTypeA SubClassOf hasPet only Dog

3. Select the statement that describes instances of PetLoverTypeA correctly.

- Instances of PetLoverTypeA must have a Dog pet and cannot have other types of pets.
- Instances of PetLoverTypeA might not have a Dog pet and cannot have other types of pets.
- Instances of PetLoverTypeA must have a Dog pet and can also have other types of pets.
- Instances of PetLoverTypeA might not have a Dog pet and can also have other types of pets.

Consider the model and answer question 4 below.

PetLoverTypeB SubClassOf hasPet some Cat

4. Select the statement that describes instances of PetLoverTypeB correctly.

- Instances of PetLoverTypeB must have a Cat pet and cannot have other types of pets.
- Instances of PetLoverTypeB might not have a Cat pet and cannot have other types of pets.
- Instances of PetLoverTypeB must have a Cat pet and can also have other types of pets.
- Instances of PetLoverTypeB might not have a Cat pet and can also have other types of pets.

Consider the model and answer question 5 below.

PetLoverTypeC SubClassOf hasPet only Cat
PetLoverTypeC SubClassOf hasPet only Dog
Cat DisjointWith Dog

5. Select the statement that correctly represents instances of PetLoverTypeC.

- Instances of PetLoverTypeC must have two pets - a Dog and a Cat.
- Instances of PetLoverTypeC might have two pets - a Dog and a Cat but also might not have any pets.
- Instances of PetLoverTypeC cannot have any pets.
- Instances of PetLoverTypeC could have 0 to n pets from type Cat or 0 to n pets from type Dog but not both.

[Continue to Section 3](#)

Section 3: Expert Knowledge

This section tests your ability to reason with ontology models, as well as compare and relate them to each other.

Consider models A and B both describing `PetLoverTypeE` and answer question 6 below.

Model A: `PetLoverTypeE`

```
PetLoverTypeE SubClassOf hasPet some Cat
PetLoverTypeE SubClassOf hasPet some Dog
PetLoverTypeE SubClassOf hasPet only (Cat or Dog)
```

Model B: `PetLoverTypeE`

```
PetLoverTypeE SubClassOf hasPet some Cat
PetLoverTypeE SubClassOf hasPet some Dog
```

6. Select the correct statement about models A and B describing `PetLoverTypeE`.

- Model A allows for instances of `PetLoverTypeE` to have a pet that is neither a Dog nor a Cat.
- Model B allows for instances of `PetLoverTypeE` to have a pet that is neither a Dog nor a Cat.
- None of the models allow for instances of `PetLoverTypeE` to have a pet that is neither a Dog nor a Cat.
- Both models allow for instances of `PetLoverTypeE` to have a pet that is neither a Dog nor a Cat.

Consider models A and B describing `PetLoverTypeD` and `PerLoverTypeF` and answer question 7 below.

Model A: `PetLoverTypeD`

```
PetLoverTypeD SubClassOf hasPet some (not Dog)
```

Model B: `PerLoverTypeF`

```
PetLoverTypeF SubClassOf hasPet only Dog
```

7. Is it true that `PetLoverTypeD` is disjoint to `PetLoverTypeF`? That is, there can be no instance that is at the same time of type `PetLoverTypeD` and `PetLoverTypeF`.

- Yes
- No

Consider models A and B both describing `PetLoverTypeE` and answer question 8 below.

Model A: `PetLoverTypeE`

```
PetLoverTypeE SubClassOf hasPet only (Cat or Dog)
```

Model B: `PetLoverTypeE`

```
PetLoverTypeE SubClassOf hasPet some Cat
PetLoverTypeE SubClassOf hasPet some Dog
PetLoverTypeE SubClassOf hasPet only (Cat or Dog)
```

8. Select the correct statement about models A and B describing `PetLoverTypeE`.

- Model A allows for instances of `PetLoverTypeE` to not have any pet at all.
- Model B allows for instances of `PetLoverTypeE` to not have any pet at all.
- None of the models allow for instances of `PetLoverTypeE` to not have any pet at all.
- Both models allow for instances of `PetLoverTypeE` to not have any pet at all.

Consider models A and B both describing `PetLoverTypeE` and answer question 9 below.

Model A: `PetLoverTypeE`

```
PetLoverTypeE SubClassOf hasPet some Cat
PetLoverTypeE SubClassOf hasPet some Dog
PetLoverTypeE SubClassOf hasPet only (Cat and Dog)
```

Model B: `PetLoverTypeE`

```
PetLoverTypeE SubClassOf hasPet some Cat
PetLoverTypeE SubClassOf hasPet some Dog
PetLoverTypeE SubClassOf hasPet only (Cat or Dog)
```

9. Select the correct statement about models A and B describing `PetLoverTypeE`.

- In both models `PetLoverTypeE` cannot have any instances as it is unsatisfiable.
- Both models require instances of `PetLoverTypeE` to have at least one Cat and at least one Dog and these instances cannot have a pet that is neither a Dog nor a Cat.
- Model A requires instances of `PetLoverTypeE` to have both at least one Dog and at least one Cat, while for Model B `PetLoverTypeE` cannot have any instances.
- `PetLoverTypeE` in Model A cannot have any instances, while instances of `PetLoverTypeE` in Model B cannot have a pet that is neither a Dog nor a Cat.

Submit

Appendix D: Instructions for Tutorial and Quiz

Instructions

Summary

Detailed Instructions

Examples

Identify Modeling Errors in Ontology Classes

You will be presented one or two ontology classes. For each class you will be shown the class name, a description, possibly one or more synonyms of the class name and the axioms defining the class.

The task is to check if the class(es) contain(s) a modeling error and to select the correct type of modeling error if applicable. Some tasks do not contain a modeling error, or a modeling error of a type which is not a given option. In these cases, please select the option "*The given class(es) do(es) not contain any of the given modeling errors, or is/are modelled correctly.*".

Types of Modeling Errors

Below are descriptions of the types of modeling errors that can be chosen for each task. Also see the tab *Examples*.

- Missing Disjointness Axiom**
 In OWL, classes are overlapping by default and disjointness between classes needs to be explicitly asserted using disjointness axioms. If there is no disjointness axiom between two classes, an individual can be a member of both of them simultaneously. If, on the other hand, there is a disjointness axiom between two classes, no individual can be a member of both of them at the same time. The omission of disjointness axioms is a common modeling error, for example because disjointness is assumed to be by default.
- Confusion between logical and linguistic "and"**
 In common linguistic usage, "and" and "or" do not correspond consistently to logical conjunction and disjunction, respectively. Modeling errors can be made when a concept formulated in natural language is modelled too "literally" in an ontology.
- Trivially satisfiable universal (allValuesFrom) Restrictions**
 Modeling errors arising from the incorrect assumption that a universal (allValuesFrom) restriction also implies an existential (someValuesFrom) restriction. A universal restriction is satisfied in case either all asserted property values conform to it, or no property values are asserted at all. The second option may get overlooked, leading to a modeling error.
- Missing Closure Axiom (missing allValuesFrom Restriction)**
 Another common modeling error is to model a concept using existential (someValuesFrom) restrictions to state that the concept has certain property values, but not stating that these are all the property values allowed via universal (allValuesFrom) restrictions. These modeling errors may often come from disregarding the Open World Assumption.

Notation and Presentation

For each class the class name, possible synonyms, a description of the concept the class represents and the class axioms are given. The class axioms are given in Manchester syntax.

Manchester Syntax Overview

The following table shows common OWL concepts and their representation in Manchester syntax.

OWL Concept	Manchester Syntax	Example
Equivalent Classes	Class EquivalentTo ClassExpression	CheesyPizza EquivalentTo Pizza and (hasTopping some CheeseTopping)
Subclass	Class SubClassOf ClassExpression	OnionTopping SubClassOf VegetableTopping
Conjunction/Intersection	ClassExpression and ClassExpression	VegetableTopping and CheeseTopping
Disjunction/Union	ClassExpression or ClassExpression	VegetableTopping or CheeseTopping
someValuesFrom	ObjectProperty some ClassExpression	CheesyPizza hasTopping some CheeseTopping
allValuesFrom	ObjectProperty only ClassExpression	VegetarianPizza hasTopping only VegetarianTopping
Disjointness	Class DisjointWith Class	VegetableTopping DisjointWith CheeseTopping

NOTE: As some classes have many Disjointness axioms, these are combined for brevity.

Example: ClassA DisjointWith ClassB and ClassA DisjointWith ClassC are combined to ClassA DisjointWith (each of) ClassB, ClassC.

Instructions



Summary

Detailed Instructions

Examples

Good examples

The two classes *VegetableTopping* and *CheeseTopping* represent concepts that are different from each other. Nothing is both a vegetable and a cheese. This can be modelled as:

- *VegetableTopping SubClassOf PizzaTopping*
- *CheeseTopping SubClassOf PizzaTopping*
- *VegetableTopping DisjointWith CheeseTopping*

A *CardinalePizza* is a pizza with only tomatoes, mozzarella and ham.

- *CardinalePizza SubClassOf Pizza*
- *CardinalePizza hasTopping some TomatoTopping*
- *CardinalePizza hasTopping some MozzarellaTopping*
- *CardinalePizza hasTopping some HamTopping*
- *CardinalePizza hasTopping only (TomatoTopping or MozzarellaTopping or HamTopping)*

Class Name: *HawaiianPizza*

- *HawaiianPizza SubClassOf Pizza*
- *HawaiianPizza hasTopping some TomatoTopping*
- *HawaiianPizza hasTopping some MozzarellaTopping*
- *HawaiianPizza hasTopping some HamTopping*
- *HawaiianPizza hasTopping some PineappleTopping*
- *HawaiianPizza hasTopping only (TomatoTopping or MozzarellaTopping or HamTopping or PineappleTopping)*

A *CardinalePizza* is a pizza with only tomatoes, mozzarella and ham.

- *CardinalePizza SubClassOf Pizza*
- *CardinalePizza hasTopping some TomatoTopping*
- *CardinalePizza hasTopping some MozzarellaTopping*
- *CardinalePizza hasTopping some HamTopping*
- *CardinalePizza hasTopping only (TomatoTopping or MozzarellaTopping or HamTopping)*

Bad examples

Missing Disjointness Axiom

Omitting the disjointness axiom is therefore a modeling error, as with this model an individual could be both a cheese and a vegetable simultaneously:

- *VegetableTopping SubClassOf PizzaTopping*
- *CheeseTopping SubClassOf PizzaTopping*

Confusion between logical and linguistic "and"

Modeling the class very literally after its description could lead to the axiom

CardinalePizza SubClassOf Pizza and (hasTopping only (TomatoTopping and MozzarellaTopping and HamTopping)),

which is a modeling error, as the class would become unsatisfiable due to *TomatoTopping*, *MozzarellaTopping* and *HamTopping* being disjoint from each other.

Trivially satisfiable allValuesFrom Restrictions

A common modeling error is to assume that an *allValuesFrom (only)* restriction implies existential (*some*) restrictions, e.g. modeling the class as:

- *HawaiianPizza SubClassOf Pizza*
- *HawaiianPizza hasTopping only (TomatoTopping or MozzarellaTopping or HamTopping or PineappleTopping)*

Using this model, a pizza without any toppings would be a valid *HawaiianPizza*, as there are no axioms that state that there **MUST** be any toppings.

Missing Closure Axiom

A common modeling error is to model the class as:

- *CardinalePizza SubClassOf Pizza*
- *CardinalePizza hasTopping some TomatoTopping*
- *CardinalePizza hasTopping some MozzarellaTopping*
- *CardinalePizza hasTopping some HamTopping*

This states that any pizza with at least one *TomatoTopping*, at least one *MozzarellaTopping*, at least one *HamTopping* and *any other toppings* is a valid *CardinalePizza*.

Adding a closure axiom fixes this modeling error and avoids that pizzas with *TunaToppings* or *SpinachToppings* are valid *CardinalePizzas*:

- *CardinalePizza hasTopping only (TomatoTopping or MozzarellaTopping or HamTopping)*

Appendix E: Feedback Questionnaire

Feedback Questionnaire

This feedback questionnaire is part of an experiment at TU Wien.

Your answers are submitted anonymously.

* Erforderlich

Q1: Did you enjoy taking this quiz? *

1 2 3 4

No, not at all. Yes, very much.

Q2: Would you like to take a similar quiz again, e.g. in a distance learning course? *

1 2 3 4

No, not at all. Yes, very much.

Q3: Do you think that the quiz helped you better understand ontology verification? *

1 2 3 4

No, not at all. Yes, very much.

Q4: Which of the modeling errors did you find the easiest to identify? *

- Missing Disjointness Axioms
- Confusions between the Logical and Linguistic Meaning of "and"
- Trivially Satisfiable Universal (allValuesFrom) Restrictions
- Missing Closure Axioms (Missing allValuesFrom Restrictions)

Q5: Which of the modeling errors did you find the hardest to identify? *

- Missing Disjointness Axioms
- Confusions between the Logical and Linguistic Meaning of "and"
- Trivially Satisfiable Universal (allValuesFrom) Restrictions
- Missing Closure Axioms (Missing allValuesFrom Restrictions)

Q6: Do you have any other remarks? (optional)

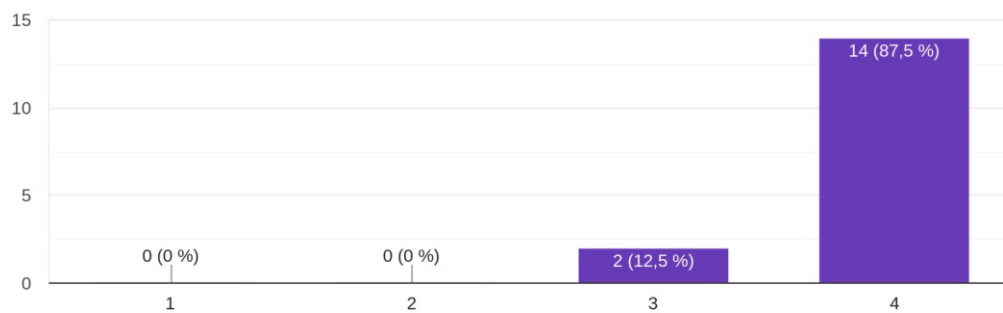
Meine Antwort

Thank you very much for participating in this experiment!

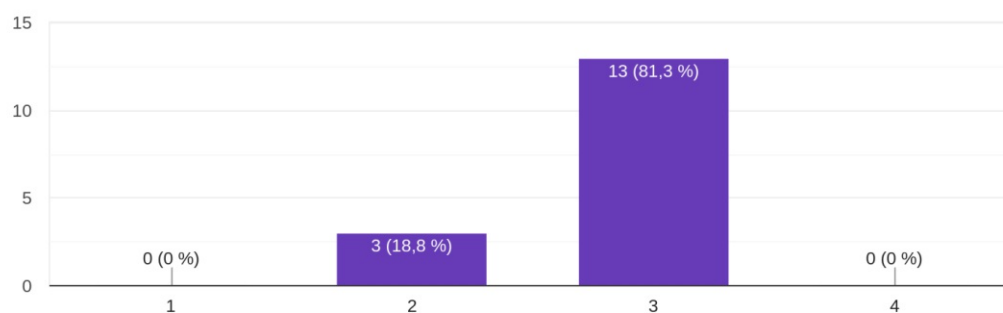
Senden

Appendix F: Results of the Self-Assessment Form

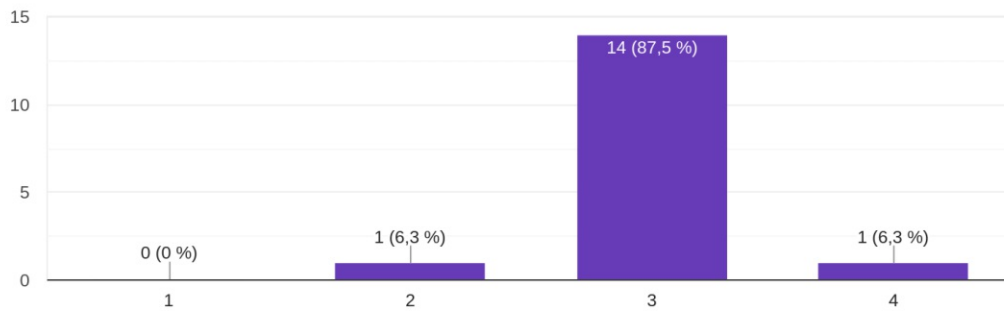
Q1: How would you rate your understanding of English documents?



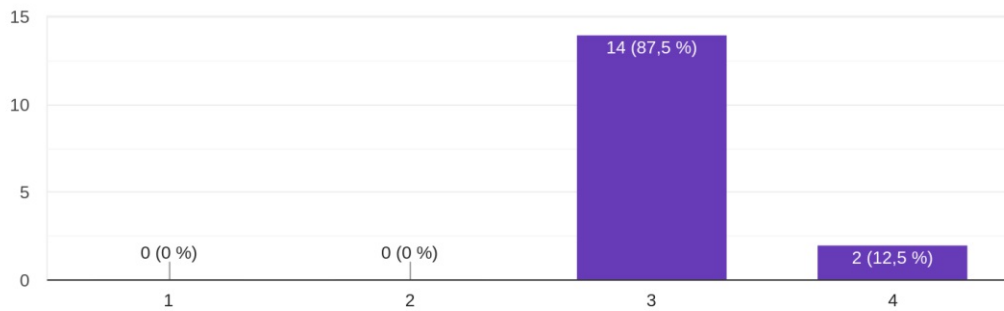
Q2: How would you rate your knowledge of formal logic?



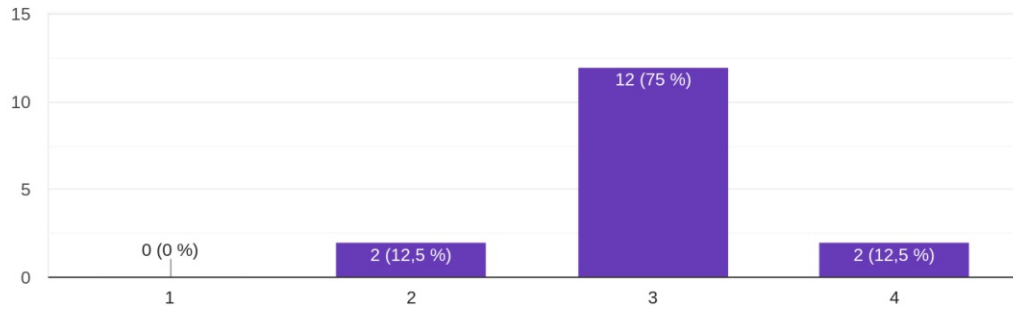
Q3: How would you rate your knowledge in Model-Driven Engineering?



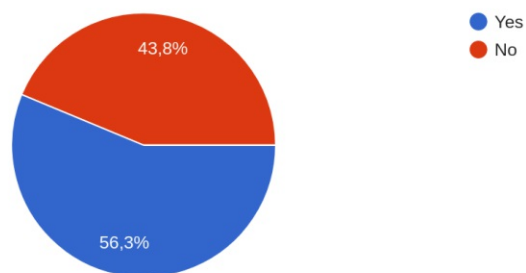
Q4: How would you rate your knowledge in ontology modeling?



Q5: How would you rate your knowledge of web-based knowledge representation languages (e.g., RDF(S), OWL)?

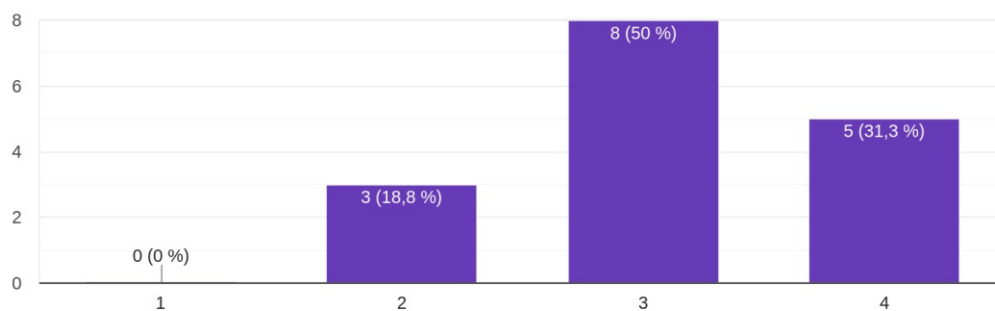


Q6: Do you have any prior experience working with Crowdsourcing platforms?

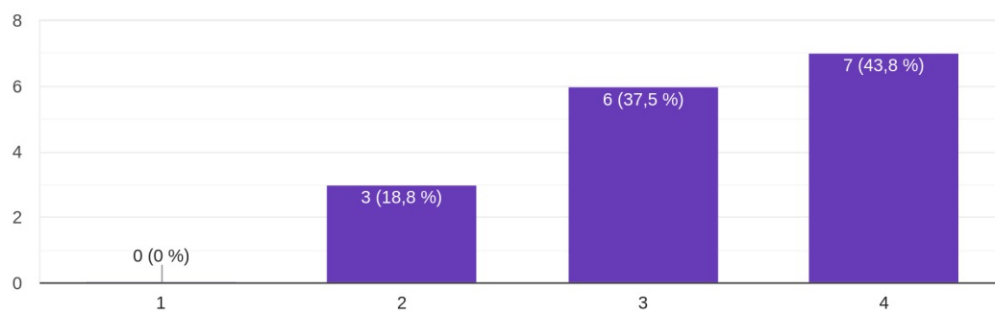


Appendix G: Results of the Feedback Questionnaire

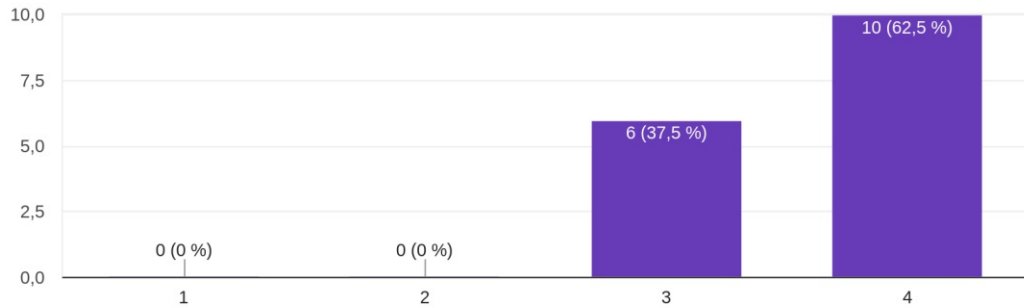
Q1: Did you enjoy taking this quiz?



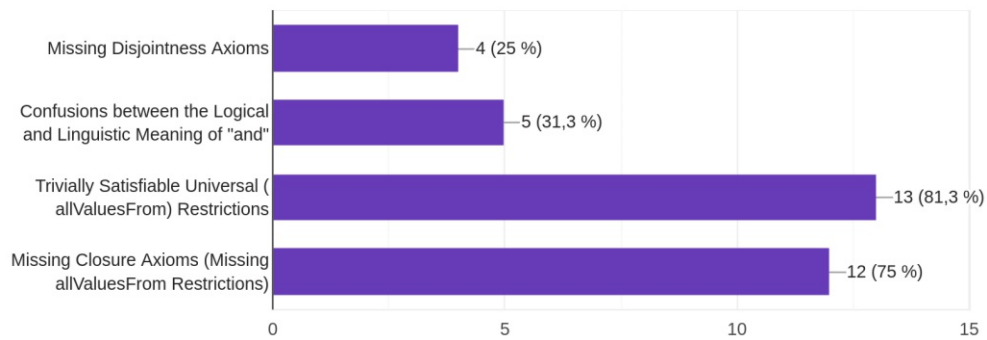
Q2: Would you like to take a similar quiz again, e.g. in a distance learning course?



Q3: Do you think that the quiz helped you better understand ontology verification?



Q4: Which of the modeling errors did you find the easiest to identify?



Q5: Which of the modeling errors did you find the hardest to identify?

