A Recommendation System Involving Human-in-the-loop to Improve the Quality of Ontologies

Student Name: Pramit Bhattacharyya

IIIT-D-MTech-CS-20-MT18142
July, 2020

Indraprastha Institute of Information Technology
New Delhi

Thesis Advisors
Dr. V. Raghava Mutharaju

Submitted in partial fulfillment of the requirements
for the Degree of M.Tech. in Computer Science & Engineering,
without Specialization

Certificate

This is to certify that the thesis titled **"A Recommendation System Involving Human-in-the-loop to Improve the Quality of Ontologies"** submitted by **Pramit Bhattacharyya** for the partial fulfillment of the requirements for the degree of *Master of Technology* in *Computer Science & Engineering* is a record of the bonafide work carried out by him under my guidance and supervision at Indraprastha Institute of Information Technology, Delhi. This work has not been submitted anywhere else for the reward of any other degree.

Dr. V. Raghava Mutharaju
Indraprastha Institute of Information Technology, New Delhi

Abstract

Building an ontology is not only a time-consuming process, but it is also confusing, especially for beginners and the inexperienced. Although ontology developers can take the help of domain experts in building an ontology, they are not readily available in several cases for a variety of reasons. Ontology developers have to grapple with several questions related to the choice of classes, properties, and the axioms that should be included. Apart from this, there are aspects such as modularity and reusability that should be taken care of. From among the thousands of publicly available ontologies and vocabularies such as Linked Open Vocabularies (LOV), it is hard to know the terms (classes and properties) that can be reused in the development of an ontology. A similar problem exists in implementing the right set of ontology design patterns (ODPs) from among the several available. Generally, ontology developers make use of their experience in handling these issues, and the inexperienced ones have a hard time. In order to bridge this gap, we propose a tool named OntoSeer, that monitors the ontology development process and provides suggestions in real-time by interacting with the ontology developers to improve the quality of the ontology under development. It can provide suggestions on the naming conventions to follow, vocabulary to reuse, ODPs to implement, and axioms to be added to the ontology. OntoSeer has been implemented as a Protégé plug-in and is available at github link `https://github.com/kracr/ontoseer`.

Keywords: Ontology Engineering, Ontology Quality, Ontology Design Pattern, LOV, and Protégé plug-in

Acknowledgments

I would like to express my deepest gratitude to my advisor **Dr. V. Raghava Mutharaju** for his guidance and support. I would like to thank him for his mentorship at every stage of this thesis work. I would not have been able to complete my thesis work this smoothly in IIIT Delhi without his consistent support.

I would also like to thank the thesis committee members for evaluating my work. I want to thank my friends and college mates for their immense support. Most importantly, none of this would have happened without my family's love and patience - my parents **Late Amalendu Bhattacharyya**, and **Tuhina Bhattacharyya**, to whom this thesis is dedicated. I would also like to thank my elder sister **Mrs. Ishita Bhattacharyya**, my brother-in-law **Mr. Santanu Debnath**, and my nephew **Master Amrut Debnath**, and my cousin **Dr. Anup Bhattacharya** for their continuous support throughout the process.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ontology is a machine-interpretable shared vocabulary of basic concepts of a domain and relationships among them. Ontology finds its primary usage in knowledge-based applications where they are used for sharing knowledge among the researchers.

Ontology development is generally a group activity, where domain experts, ontology developers, and other stakeholders meet to discuss and develop an ontology. This makes ontology development time consuming and expensive process, especially if the scope of the ontology is broad. It is not always possible to take the help of domain experts either because they are not available for the entire duration of the ontology development process or are not available at all. Another issue is that the ontology developer could be inexperienced. Even experienced developers face issues while building ontologies, and this problem only magnifies in the case of inexperienced developers. They will have to deal with several questions, such as the following.

- What should be the classes, properties, and instances in this domain?
- Which classes should be related to each other?
- Are there any classes, properties, or instances that can be reused?
- How should the ontology be made modular?
- What naming conventions should be followed?
- If something has to be a property, should it be an object property or a data property?
- Should something be a class, or a property, or an instance?
- What should be the class hierarchy? Is it correct?
- Does the ontology answer the competency questions set forth initially by the development group?

- Is the ontology too big? Should it be divided into multiple smaller modules?
- What should be some of the axioms in the ontology?

Experienced ontology developers may be able to answer some of these questions and make better design decisions. But even for them, it will be hard to keep track of all the new vocabularies, ontologies, and the ontology design patterns [10] that get published in the repositories. Our proposed tool, named *OntoSeer*, is a Protégé plug-in that works with the ontology developer during the ontology development process and gives suggestions in real-time that can lead to better quality ontologies. The contributions of this work are as follows.

- A tool that recommends classes, properties, axioms, and ontology design patterns to (re)use based on the description, competency questions, and the ontology under development.
- A mechanism to provide suggestions of names to use during the ontology development. These suggestions follow the naming conventions.
- Integration with Protégé (available as a plug-in) to improve the ontology development experience.

Chapter 2

Background

2.0.1 Basic Terminologies

Ontology

According to Gruber, "An ontology is a formal, explicit specification of a shared conceptualization." [8]

- formal: ontology should be defined in a formal language.
- explicit specification: concepts, relations between them, and the constraints on them should be explicitly defined.
- shared: ontology should be a shared view between several parties, a consensus rather than an individual view.
- Conceptualization: ontology should be an abstract, simplified view of the world that we wish to represent for some purpose.

Some of the reasons for building an ontology are:

- To act as a medium for sharing the understanding of the structure of information among researchers.
- For enabling reuse of domain knowledge.
- For making domain assumptions explicit.
- For separating domain knowledge from the operational knowledge.
- For analyzing domain knowledge.

Figure 2.1: Representation of book ontology in schematic form.

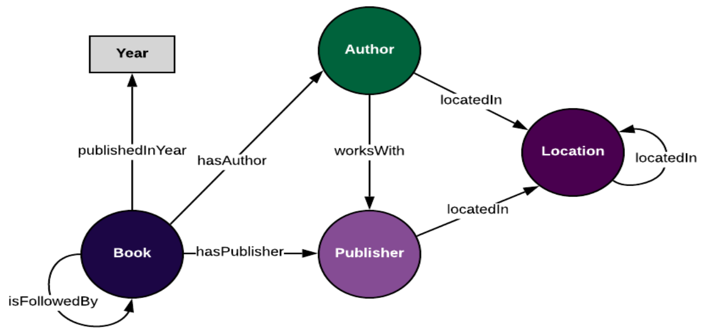


Figure 2.1 shows a book ontology where "Book" class is related to the "Author" class by "hasAuthor" property. Again, "Book" class is related to the "Publisher" class by "hasPublisher" property.

SNOMED-CT[1] and GeneOntology[2] are some of the widely used ontology in the medical domain with more than 28000 axioms.

OWL (Web Ontology Language)

OWL is a knowledge representation language used to build ontologies. There are three types of entities in OWL. They are as follows.

1. **Classes:** Classes are the focus of most ontologies. Classes describe concepts in the domain. For example, a class of wines represents all wines. Specific wines are instances of this class. A class can have subclasses that represent concepts that are more specific than the superclass. For example, we can divide the class of all wines into red, and white wines [5].

2. **Roles or Properties:** Also defined as roles, properties connect one class with another or one individual with another individual or a class with an individual. There are two types of properties.

   - **Object properties or Abstract roles:** They connect individual to individual. For example, a role defined to connect the person's name to the organization he is affiliated with.
   - **Concrete roles or Data property:** They connect an individual to data values. For example, a role connecting a person to their first name.

[1]http://www.snomed.org/
[2]http://geneontology.org/

3. **Instances or Individuals:** They are the instances or constants in the domain. For example, mary (is a Woman), julie (is a Parent). Here mary is the instance of a class woman, while julie is the instance of a class parent.

Axioms

We will define axiom with the help of an example. Let A be an atomic class, i.e., a class name, and let R be an (abstract) role. Statements or Axioms in OWL are divided into two groups, namely, into TBox statements and ABox statements. The TBox contains terminological (or schema) knowledge, while the ABox contains assertional knowledge about instances (i.e., individuals). Formally, a TBox consists of statements of the form $C \sqsubseteq D$, where C and D are class expressions. An ABox, on the other hand, consists of statements of form C(a) and R(a, b), where C is a class expression, R is a role, and a, b are concepts.

Competency Questions

Michael Grüninger and Mark Fox defined competency questions as the questions an ontology should be competent to answer [7]. Let us consider a family ontology consisting of the names of all the family members. This family ontology should be able to answer some of the questions relating to kin relationships:

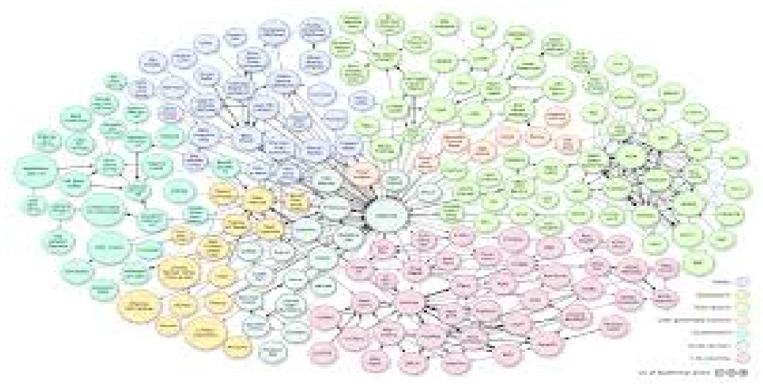
- Parentage
- Grandparents
- Great-grandparents
- Ancestors
- Aunts, uncles, and cousins to the second degree

Other than these the family ontology should be able to answer the

- Marital relationships between individuals.
- Should be able to represent in-law relationships – parents, siblings, etc.
- Should represent birth, death, and marriage years.

Competency Questions (CQs) must cover the scope of the ontology. CQs are most commonly used to test ontology's quality, as the ontology is competent enough to answer the basic questions.

Figure 2.2: Linked Open Data cloud diagram giving an overview of published data sets and their interlinkage relationships.



Linked Data

Linked Data refers to a set of best practices for publishing and interlinking structured data on the Web. Linked Data are machine-readable, are linked to other external data sets, and can also be linked to from external data sets [2]. Linked Data principles are:

- Use URIs as names for things.
- Use HTTP URIs, so that people can look up those names (make URIs dereferenceable).
- When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
- Include links to other URIs, so that they can discover more things.

Fig 2.2 shows the linked open data cloud of published data sets and relationships between them.

Vocabularies

Vocabulary consists of classes, properties, and datatypes that define the meaning of data. RDF vocabularies are themselves expressed and published following the Linked Data principles; this gives humans and machines access to the definitions of the terms used to qualify the data. Some commonly used vocabularies are:

1. **SKOS:** It is a vocabulary for knowledge organization systems (KOS) such as thesauri, classification schemes, subject heading systems and taxonomies of Semantic Web.

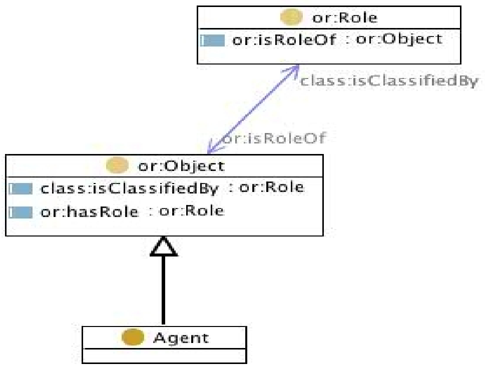
2. **FOAF:** It integrates a social network of human collaboration friendship and association with representational networks describing cartoon networks in factual terms.

ODP

ODP or ontology design patterns can be considered analogous to software design patterns known from object-oriented modellings. Patterns on the Semantic Web typically emerge from (linked) data, ontologies, and queries, as well as from procedural aspects of design at either the modelling or implementation stage. The main innovation of design patterns lies in the observation that most datasets and ontologies share similar publishing challenges, which can be approached by a common strategy. ODPs help in this aspect. An ODP can be described using the following fields:

- Name: It signifies the title name of the ODP like Trajectory pattern.
- Intent: It signifies the domain on which this ODP can be used.
- Competency questions: It consists of the questions ODP is competent to answer.
- Scenarios: Examples of requirements in natural language that can be modelled using this pattern.
- Diagram: It describes the UML diagram representing the pattern.
- Elements: It provides with the list of classes and relations in the pattern.
- Consequences: It describes the advantages and disadvantages of using the said ODP.
- Known uses: Examples of realistic ontologies where the pattern can be used.
- Reengineered from: It refers to the ontology or conceptual schema from which the pattern has been extracted.
- Related patterns: Other patterns that are either a specialization, generalization, composition, or component of this pattern. It also lists patterns that are generally used along with the described pattern.
- Building block: It provides with a reference implementation (URI or OWL file).

Figure 2.3: Figure showing AgentRole ODP, a commonly used ODP describing its usage.



2.0.2 Ontology Modelling

There is no one "correct" way or methodology for developing ontologies. An iterative approach to ontology development is the most commonly used procedure, which starts with a rough first pass at the ontology. We then revise and refine the ontology. Some fundamental rules followed in ontology design:

1. There is no one correct way to model a domain— there are always viable alternatives. The best solution almost always depends on the application that developer have in mind.

2. Ontology development is an iterative process.

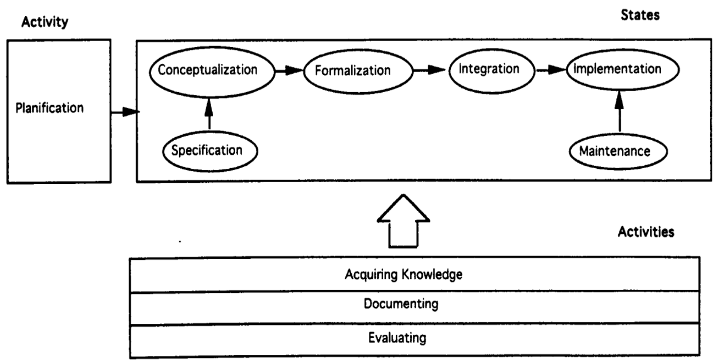
3. Concepts in the ontology should be close to objects (physical or logical) and relationships in the domain of interest. These are most likely to be nouns (objects) or verbs (relationships) in sentences that describe the domain.

We will primarily focus on MethOntology [5], the most commonly used procedure to build ontology from scratch. Some of the steps involved are:

Specification

The goal of the specification phase is to produce an ontology specification document written in natural language, using a set of competency questions.

Figure 2.4: Steps involved in ontology modelling according to MethOntology Approach.



Knowledge Acquisition

Developers can acquire knowledge from books, handbooks, figures, tables, and even from human experts. Knowledge acquisition is an iterative process and thus collides with other stages of the ontology development life cycle.

Conceptualization

In the conceptualization stage, the domain knowledge is structured into a conceptual model that describes the problem. For this, a domain vocabulary consisting of terms that include concepts, instances, verbs, and properties are built.

Integration

In the integration stage, the developer try to reuse the definitions already built in other ontologies instead of building from scratch and thus increasing the speed of development. OntoSeer, our work focuses significantly on the integration stage of the ontology life cycle.

Evaluation

Evaluation subsumes the terms Verification and Validation. Verification refers to the process that guarantees the correctness of an ontology, with respect to CQs. Validation guarantees that the ontologies correspond to the system that they are supposed to represent based on subclass hierarchies and naming conventions.

Our tool OntoSeer is compatible with any of the modelling methods undertaken by the developer. The developer, at any point, can change the design, and OntoSeer will update itself accordingly. For a new developer, it is not always possible to determine the scope of the model at first or to have the competency questions. OntoSeer also takes this unfavourable situation into account and provides a recommendation, though better and more refined recommendations will require more prior knowledge of the domain.

Chapter 3

Related Work

Tools that are similar to OntoSeer are OntoClean [9], OOPS! [14] [13], and OntoCheck [16]. We briefly discuss the functionality of these tools and how they differ from OntoSeer.

OOPS! stands for Ontology Pitfall Scanner. It is an online tool that detects pitfalls or common modelling errors that ontology developers make. The pitfalls are divided into three categories - structural, functional, and usability-profiling. There are around 40 pitfalls, but here we discuss only some of them. Having cycles in the class hierarchy is a pitfall that comes under the structural dimension. Generally, the ontology should not contain more than one semantically equivalent class. Missing disjointedness axioms is a common pitfall. Unless explicitly stated, classes are not disjoint. A functional pitfall is connecting disconnected components. Sometimes two unrelated entities are joined by a relationship. This is a pitfall that should be avoided. A usability pitfall is using different naming conventions in the ontology. For example, a super-class will have a name following one naming convention, and a subclass name follows a different naming convention. Missing domain and range for properties is another common pitfall.

One of the main differences between OOPS! and OntoSeer is that OntoSeer works along with the ontology developer in creating better quality ontologies. It does not evaluate ontologies post creation. Another major difference is OntoSeer recommends terms (classes, properties, instances) that can be reused. It does this by checking the similarity of the terms from the ontology being built with existing ontologies. OntoSeer also recommends ODPs that can be used in making the ontology modular.

OntoCheck [16] is a plugin for the Protégé to allow for easy checks on compliance towards ontology naming conventions. In particular, OntoCheck plugin helps to clean up an ontology by enforcing naming conventions meeting most of the requirements outlined for such a tool. Found test violations can be corrected to foster consistency in entity naming.

One of the main differences between OntoCheck and OntoSeer is OntoCheck only indicates the class names that violate the naming convention. It does not provide any recommendation for alternate names that can be used, which OntoSeer does. For example, if someone builds classes like HumanBeing, names, nitrogenoxide, then OntoCheck outputs that 67 percent of the classes do not follow naming conventions and class names and nitrogenoxide do not follow naming conventions. In contrast, OntoSeer not only indicates the violation but also recommends class names such as NitrogenOxide to the user.

OntoSeer uses the OntoClean paper to validate and verify the class hierarchy. A property p subsumes q if and only if, for every possible state of affairs, all instances of q are also instances of p [9]. The three characteristics mentioned in OntoClean are used by OntoSeer for class hierarchy validation. They are

1. Rigidity
2. Identity
3. Unity

The important distinction between OntoClean and OntoSeer is OntoSeer also takes into account the cases where the developer is not aware of the objects he/she is developing. Other than that, OntoSeer also recommends axioms, naming conventions, and ODPs.

The Linked Open Vocabularies (LOV) [17] is an initiative that harnesses information about the relationships between vocabularies. The number of vocabularies indexed by LOV is 716 (as of July 2020). The primary purpose of LOV is to encourage the reusability of well-documented vocabularies. It is the only open-source catalog that accepts all types of search criteria, including ontology search, metadata search, and a SPARQL endpoint access.
Ontology Search: LOV allows searching for terms (classes, properties) according to the domain they address.
Ontology Assessment: LOV retrieves the terms according to the ranking metrics they have implemented, which have been addressed in later sections.

LOV Ranking

LOV devised a ranking mechanism adapting term-frequency inverse document frequency (tf-idf), which they applied to the vocabularies' inherent graphical structure. Term t of a Vocabulary V has been considered as the basic unit instead of a word. Tf-Idf takes into account the relevance and importance of a resource to the query.

One of the biggest challenges in ontology is that the same terms can be used as class names or descriptions. To do away with this LOV defined a normalized mechanism:

- **Primary Label:** The highest scores have been assigned to this class, which includes rdfs:label , dce:title, dcterms:title, skos:prefLabel.

- **Secondary Label:** A medium score has bees assigned to the properties that includes: rdfs:comment, dce:description, dcterms:description, skos:altLabel.

- **Tertiary Label:** Finally, all properties not falling in the previous categories are considered as tertiary labels for which a low score is assigned. An example of tertiary label matching the term “person” is rdarel2:name “Person”.

The equation for normalized ranking is, therefore,
$norm(t, V)=lengthNorm(field) * \prod_{p \epsilon V} boost(p(t))$
The final score of t for a query Q is a combination of the tf-idf, the importance of label properties of t on which query terms matched, and the popularity of that term in the LOV dataset. LOV added a fourth parameter in their ranking - the popularity - as it is of fundamental importance in the Semantic Web. The popularity metric indicates how widely a term is already used (in frequency and the number of datasets using it). The equation for LOD is, therefore,
$score(t, q)=tf(t, V) * idf(t, V) * norm(t, V) * pop(t, D)$
OntoSeer uses the LOV ranking model for recommending vocabularies, terms, and axioms.

Chapter 4

OntoSeer Description

In this section, we discuss the various features of OntoSeer, along with the implementation details for each of those features in subsequent sections. OntoSeer features are as follows.

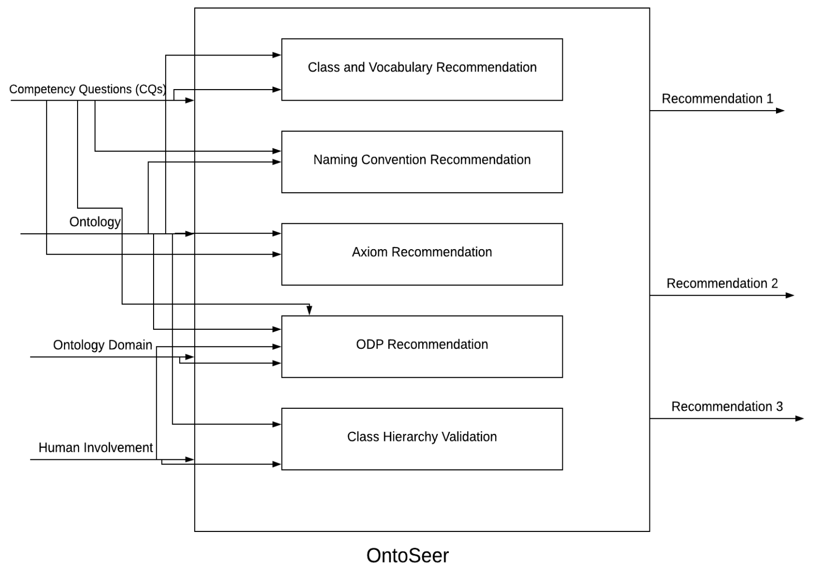
1. Class, Property, and Vocabulary Recommendation.
2. Naming Convention Suggestion.
3. Axiom Recommendation.
4. ODP Recommendation.
5. Class Hierarchy Verification.

4.0.1 Class, Property, and Vocabulary Recommendation

It is challenging for ontology developers, especially the inexperienced ones, to quickly come up with suitable class and property names. OntoSeer's class and vocabulary recommendation feature help in this regard. OntoSeer's class and vocabulary recommendation feature take the developed class and property terms in the ontology as input. Figure 4.1 shows a diagram of possible inputs to OntoSeer's class and vocabulary recommendation feature. Competency questions (CQs) and ontology domain descriptions are optional requirements, but the development of an ontology is mandatory to use the vocabulary recommendation feature.

In most cases, an ontology developer will not have access to CQs. To handle that averse situation, OntoSeer can recommend classes and properties without CQs also. After the ontology developer starts putting in classes and properties, OntoSeer can suggest additional classes and 33 properties based on the ontology corpus. It will look for classes and properties in the corpus that are similar to the ones made and will make suggestions.

Figure 4.1: A schematic representation of input and output for OntoSeer's features.



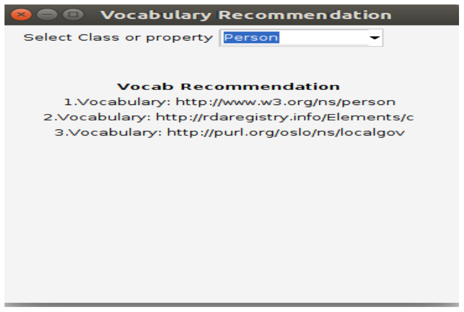
The corpus OntoSeer uses is the Manchester OWL Corpus[1], which consists of Bioportal[2], and Oxford OWL[3] corpora. This corpus is indexed using an inverted index, and OntoSeer searches through that index for the recommendation. The procedure for building an inverted index and searching through has been described in section 4.0.3 .

Ontoseer will use the classes and properties that have already been built in the ontology for suggesting class names and property names. OntoSeer not only suggests names but also provides the user with the vocabulary names where the said class or property is present. For each of the class and property, Ontoseer will first query LOV[4]. It also searches the bioportal repository[5] using Bioportal's REST API. For searching in Bioportal, the user must have a Bioportal account as the API key is necessary for authentication of the developer, or else OntoSeer will throw an exception and will require restarting of the plugin for further usage. Data from URLs returned in JSON format is parsed and converted to a string and is shown in the plugin. Ontoseer only shows the top three vocabularies based on the scoring matrix of LOV and Bioportal.

Thus on querying "Person", Ontoseer recommends "Person" from www.w3.org where "Person" is present in the title and rdaregistry.info where "Person" is present in elements section. Along with that, OntoSeer also recommends alternate names such as persoon.

[1]`http://mowlrepo.cs.manchester.ac.uk/datasets/mowlcorp/`
[2]`https://bioportal.bioontology.org/ontologies`
[3]`https://www.cs.ox.ac.uk/isg/ontologies/`
[4]`https://lov.linkeddata.es/dataset/lov/api/v2/term/suggest?q=`
[5]`http://data.bioontology.org/recommender?input=`

Figure 4.2: Vocabulary Recommendation from OntoSeer on querying class Person. The output is based on the result returned by LOV search query.



Vocab Recommendation
1.Vocabulary: http://www.w3.org/ns/person
2.Vocabulary: http://rdaregistry.info/Elements/c
3.Vocabulary: http://purl.org/oslo/ns/localgov

Figure 4.2 shows OntoSeer's recommendation on querying class "Person". The vocabulary recommendation outputs the three vocabularies where person term is present.

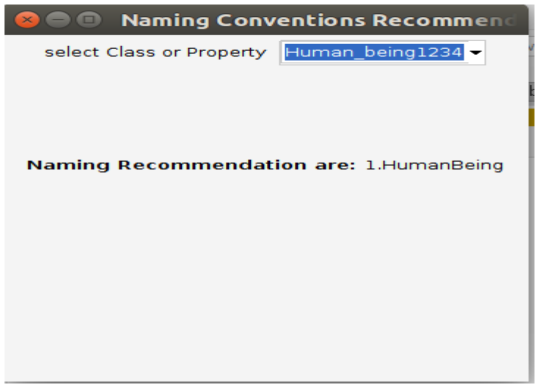
4.0.2 Naming Convention and Suggestion

For a beginner, it is a challenge to abide by the naming conventions. To cater to that need, we have incorporated the naming conventions systems for classes and properties in OntoSeer. OntoSeer's naming convention recommendation feature takes the developed class and property terms in the ontology as input. Figure 4.1 shows a diagram of possible inputs to OntoSeer's naming convention recommendation feature. The development of an ontology is mandatory to use the naming convention recommendation feature, while CQs and ontology domain descriptions are optional requirements.

While suggesting OntoSeer looks for the following discrepancies in the name.

- Use of numbers in class and property names are discouraged. So OntoSeer suggests alternatives that do not involve numbers. Thus if a user builds a class "Human1234", it will recommend using "Human" instead.
- It is generally considered as a bad practice to use any special characters other than an underscore while naming terms. So, OntoSeer recommends name only with an underscore as the special character.
- Usage of camel-case while naming is considered as a good practice. Thus if the user builds a class named "Human_being", OntoSeer will recommend using "HumanBeing".
- OntoSeer focuses on use of verb sense terms while naming properties. This is implemented using Stanford-core-NLP part-of-speech taggers, where each property name is checked whether it is started with a verb form or not.

Figure 4.3: The naming recommendation for the selected concept is shown. The recommended name follows the naming convention for classes.



Naming Recommendation are: 1.HumanBeing

OntoSeer, along with naming conventions, also suggests alternate names for the classes built. For this, OntoSeer refers to the already downloaded and indexed WordNet-3.0[6] and use its nouns as the baseline dictionary. A string matching algorithm named Jaro-Winkler is used, and a class name is suggested.

Jaro-Winkler distance is a string metric for measuring the edit distance between two sequences, or in other words, Jaro distance between two words is defined as the minimum number of single-character transpositions required to change one word into the other [3].

The Jaro-Winkler distance uses a prefix scale, which gives more favorable ratings to strings that match from the beginning for a set prefix length. As it gives more importance to the words with an identical prefix, hence the Jaro-Winkler distance seems very interesting to our use case of syntactic matching. The Jaro Distance between two sequences s1 and s2 is defined by:

d_j=1/3*$((m/|s1|)+(m/|s2|)+((m-t)/m))$

d_j is the Jaro distance, m is the number of matching characters (characters that appear in s1 and s2), t is half the number of transpositions (compare the i-th character of s1 and the i-th character of s2 divided by 2), |s1| is the length of the first string, |s2| is the length of the second string.

The similarity coefficient is above the threshold of 0.7 which has been set after observing the scores across multiple iterations. OntoSeer, in this aspect, is different from LOV or BIO-Portal name suggestion as they only suggest names after matching with the terms that are present in their vocabularies. OntoSeer spellchecker, on the other hand, suggests names that are present in the dictionary of WordNet. Thus if a user builds a class named “acicul”, OntoSeer will recommend alternate names such as “acicula”, “aciculae”, “auriculae”. Naming Suggestion for OntoSeer is implemented using our own

[6]`https://wordnet.princeton.edu/`

built-in inverted index. We indexed the base file, which is noun.txt of WordNet, and used the indexed file for our string comparison algorithm.

4.0.3 Axiom Recommendation

OntoSeer can retrieve axioms from the ontology corpus that are similar to the ones in the current ontology. It will also remind the user about axioms such as disjointedness among class siblings, and property characteristics (inverse, symmetric, transitive, etc.), that were missed or overlooked. OntoSeer's axiom recommendation feature takes the developed class and property terms in the ontology as input. Figure 4.1 shows a diagram of possible inputs to OntoSeer's axiom recommendation feature. The development of an ontology is mandatory to use the naming convention recommendation feature, while CQs and ontology domain descriptions are optional requirements.

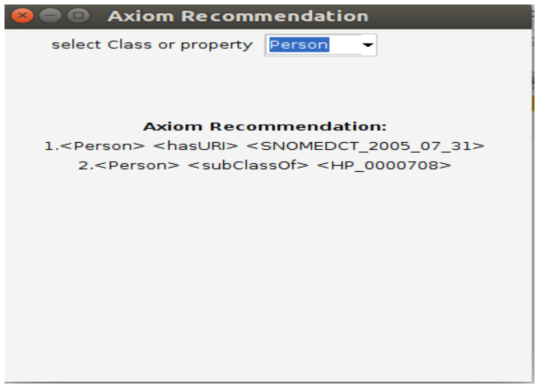
In order to recommend axioms, we make use of the Manchester OWL corpus, Bioportal corpus, MOWL corpus, and Oxford corpus. Since Lucene [1], a well-known java library that provides powerful indexing and searching features, is not compatible with the Protégé tool, we have built an inverted index structure. We have indexed the ontology files across Manchester OWL corpus, Bioportal corpus, MOWL corpus, and Oxford corpus for axiom recommendation.

An inverted index is an index data structure storing a mapping from content, such as words or numbers, to its locations in a document or a set of documents [12]. We have built a record-level inverted index that contains a list of references to documents for each term (class-names, instance-names, property-names). This way, we have been able to store the occurrences of the term (class names, property names) in each OWL files. While indexing, we separate IRI's into domain names and class or property names. We use only the local part of the IRI that contains the class or property name for indexing. For example, for an IRI of the form <http://www.geneontology.org/formats/oboInOwl/hasDate> we index only the local part hasDate.

It then allows us to perform queries on this index, returning results ranked by the relevance to the query as no scoring methodology is implemented.

In our indexing implementation for vocabulary suggestion, each OWL file is the unit of search and index. While for axiom suggestion, we have taken each tuple as the index. An index consists of one OWL file or a tuple of a document. A tuple is defined as a data entity composed of subject, object, and predicate. For example, in the statement " Bob knows Alice", "Bob" is the subject, "knows" is the predicate, and "Alice" is the object. Indexing involves adding documents or tuples to an index, and searching involves retrieving documents or tuples from an index based on functionality. Searching requires an index to have already been built. On querying class "Person", OntoSeer will recommend
⟨Person⟩⟨hasURI⟩⟨SNOMEDCT_2005_07_31⟩
⟨Person⟩⟨subClassOf⟩⟨HP_0000708⟩

Figure 4.4: Axiom recommendations on Person class.We have ignored the IRI's for simplicity of the image. The recommendation is based on ranking implemented by OntoSeer.



```
Axiom Recommendation:
1.<Person> <hasURI> <SNOMEDCT_2005_07_31>
2.<Person> <subClassOf> <HP_0000708>
```

where hasURI is a property and HP_0000708, SNOMEDCT_2005_071 are classes of MOWL repository.

4.0.4 ODP Recommendation

OntoSeer recommends ODPs that can be used in making the ontology modular. The attributes that are used for ODP recommendation are the description of the ontology (optional), domain of the ontology (mandatory), some sample classes and properties related to the domain (optional), and competency questions (optional). OntoSeer's ODP recommendation feature takes input from the user about the domain of the ontology, which is mandatory. All other attributes are optional. Figure 4.1 shows a diagram of possible inputs to OntoSeer's ODP recommendation feature. Ontology domain description and human involvement are necessary for ODP recommendation, while all the other are optional requirements.

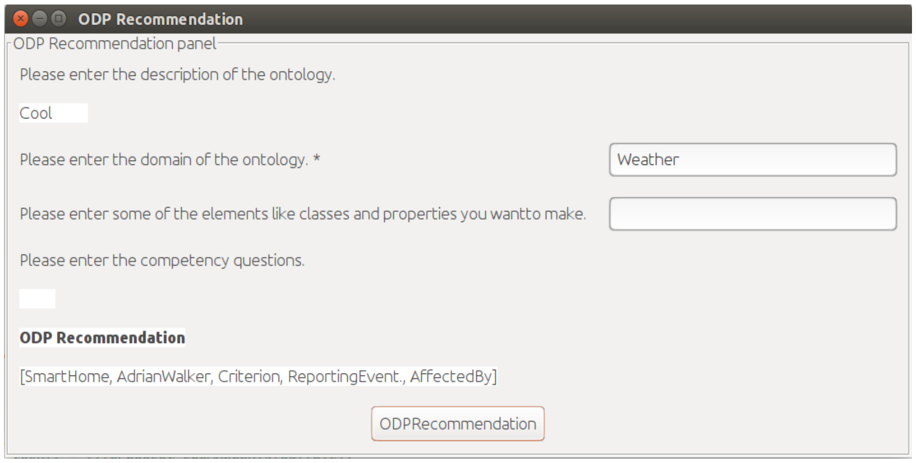
Users may not have answers to all the questions; nevertheless, in most cases, the user will be able to answer at least one of them. It is necessary to provide at least the domain as input, or else OntoSeer will throw an exception.

For recommendations, we have scraped through all the ODPs listed in ODP site[7]. We have collected each of the intent, domain, competency questions, elements, and additional information that is listed for each ODP in individual ODPs.

OntoSeer makes use of string similarity for ODP recommendation. We have taken each of the ODP documents as a string and, the inputs provided by the user will be matched against them. Ontoseer then recommends ODP based on the string similarity between them using the Jaro-Winkler metric of string matching.

[7]http://ontologydesignpatterns.org/wiki/Main_Page

Figure 4.5: OntoSeer's ODP recommendations on providing "College" as intent and "University" as domain.


ODP Recommendation

ODP Recommendation panel

Please enter the description of the ontology.

Cool

Please enter the domain of the ontology. * Weather

Please enter some of the elements like classes and properties you wantto make.

Please enter the competency questions.

ODP Recommendation

[SmartHome, AdrianWalker, Criterion, ReportingEvent., AffectedBy]

For user input of description of ontology being "Cool", and domain being "Weather", this technique recommends the following ODPs SmartHome, AdrianWalker, Criterion, ReportingEvent, AffectedBy. OntoSeer is using syntactic similarity. Hence SmartHome having score 0.63 are recommended. On analysis, we will find class names and additional information of the ODP having similar terms as "Cool" and "Weather" as smart homes have facilities to control room temperature. The other ODPs like AdrianWalker and Criterion, had scores of 0.62 and is having information of "Cool", and "Weather" as its concepts. The recommendation can be more refined if Doc2Vec methodology [11] is used because it considers semantic matching too. The incompatibility of deeplearning4j [6] with Protégé compelled us to use the Jaro-Winkler method, which gives comparable results.

4.0.5 Class Hierarchy Verification

Determining the class hierarchy is confusing, especially to the inexperienced ontology developers. This often leads to improper hierarchy. Ontology developers need to carefully answer a few questions while building the class hierarchy.

1. Can a particular class act as a super-class or a subclass?
2. Should a particular subclass have more than one parent?
3. Is a particular class, truly a subclass of the super-class?

It seems that a social entity can be a subclass of groups of people because groups of people form a social entity. However, the social entity is positively unity class, while groups of people are negatively unity class. A negative unity class cannot subsume a positive unity class. Hence this hierarchy is wrong.

OntoSeer makes use of rigidity (R), identity (I), and unity (U) characteristics of the classes and takes the user input based on these characteristics to validate the class hierarchy.

1. **Rigidity:** A property is rigid if it is essential to all its possible instances; an instance of a rigid property cannot cease to exist in the future or fail to be its instance in a different domain. For example, having a brain is a rigid property for a human being but is not a rigid property while developing an ontology on the Wizard of Oz.

2. **Identity:** Identity constraint is the criteria we use to answer questions like, “Is that my dog?”. Two objects are identical if they are the same. For two different objects, we usually check whether they have the same essential properties or not. If they have, then they are identical else not.

3. **Unity:** Unity constraint checks whether properties have wholes as instances or not. For example, “Ocean” is a subclass of “Water”, since all oceans are water. However, we will be having a contradiction if we claim that instances of the “Water” must not be wholes, and instances of “Ocean” always are.

Ontoclean defined some broad classes and assigned values to these three characterisitics or constraints which are shown in 4.1.

Table 4.1: Object described in OntoClean

Type	Identity	Unity	Rigidity
Entity	-I	-U	+R
Country	+I	+U	-R
Organization	+I	+U	+R
Legal Agent	+I	-U	-R
Agent	+I	-U	-R
Social Entity	-I	+U	+R
Vertebrate	+I	+U	+R
Invertebrate	+I	+U	+R
Fruit	+I	+U	+R
Person	+I	+U	+R
Food	+I	-U	-R
Living being	+I	+U	+R
Group	-I	-U	+R
Amount of matter	-I	-U	+R

OntoSeer interacts with the ontology developer and based on the inputs, determines the characteristics a particular class has. The questions asked by OntoSeer are as follows.

1. Do the instances of the class cease to exist in the future? For example, “Person” will always be person but “Student” can cease to exist to be a student in future.

2. Are the super-class and subclass identical? For example, two one hour duration of time interval are identical but an hour interval on Wednesday are not identical to an hour interval on Friday.

3. Is the subclass part of the whole class? For example, a lump of clay is part of amount of matter but amount of matter is not part of a lump of clay.

OntoSeer validates the class hierarchy based on these three characteristics. It recommends that “Person” cannot be a subclass of “Student” as it violates Rigidity criteria. Similarly, “Water” cannot be a subclass of “Ocean” for violating the unity criteria, whereas “Groups of people” cannot be a subclass of “Social entity” for violating the identity criteria. Table 4.2 shows the super-class and its possible subclass characteristics for validating the class hierarchy.

Table 4.2: Table showing values of rigidity, identity, unity for class hierarchy validation

Rule	Super-Class Value	Posible SubClass Value
Identity	Positive	Negative,Positive
Identity	Negative	Negative
Rigidity	Positive	Negative,Positive
Rigidity	Negative	Negative
Unity	Positive	Negative,Positive
Unity	Negative	Negative

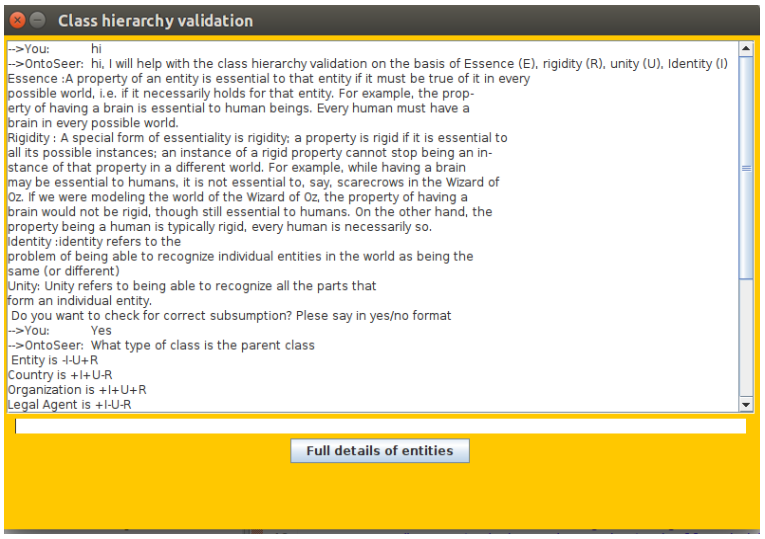
Figure 4.6 shows how OntoSeer shows the characteristic values of the entities defined in OntoClean. Figure 4.7 shows the user providing inputs that cause no violation of any of the three constraints. Hence, Figure 4.8 shows the output as all the constraints and subclass hierarchy are correctly maintained. On the other hand, Figure 4.9 shows the user providing inputs that cause a violation of all three constraints. Hence, Figure 4.10 shows the output as all the constraints are violated, and the subclass hierarchy is incorrect.

4.0.6 Challenges Faced and Efforts Required in building OntoSeer

One of the challenging aspects of building OntoSeer is the incompatibility of Protégé with the various open-source libraries. Hence it took a significant amount of time to build OntoSeer.

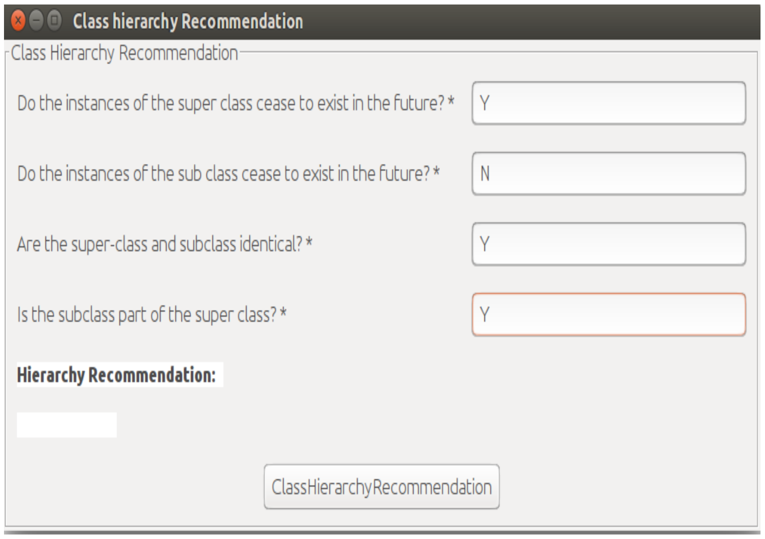
- LOV and Bioportal have a limited number of ontologies. To make the recommendation better, we have indexed the MOWL corpus. The inverted index has been

Figure 4.6: OntoSeer showing the rigidity, identity, unity values of objects mentioned in Ontoclean.



```
-->You:        hi
-->OntoSeer:  hi, I will help with the class hierarchy validation on the basis of Essence (E), rigidity (R), unity (U), Identity (I)
Essence :A property of an entity is essential to that entity if it must be true of it in every
possible world, i.e. if it necessarily holds for that entity. For example, the prop-
erty of having a brain is essential to human beings. Every human must have a
brain in every possible world.
Rigidity : A special form of essentiality is rigidity; a property is rigid if it is essential to
all its possible instances; an instance of a rigid property cannot stop being an in-
stance of that property in a different world. For example, while having a brain
may be essential to humans, it is not essential to, say, scarecrows in the Wizard of
Oz. If we were modeling the world of the Wizard of Oz, the property of having a
brain would not be rigid, though still essential to humans. On the other hand, the
property being a human is typically rigid, every human is necessarily so.
Identity :identity refers to the
problem of being able to recognize individual entities in the world as being the
same (or different)
Unity: Unity refers to being able to recognize all the parts that
form an individual entity.
 Do you want to check for correct subsumption? Plese say in yes/no format
-->You:        Yes
-->OntoSeer:  What type of class is the parent class
 Entity is -I-U+R
Country is +I+U-R
Organization is +I+U+R
Legal Agent is +I-U-R
```

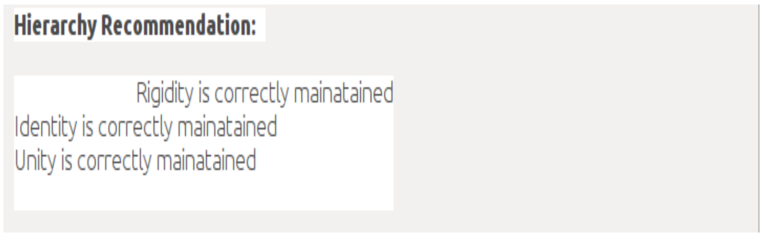
Figure 4.7: OntoSeer's user inputs to question for class hierarchy valiadation. In this figure no violation to Table 4.2 mentioned standards has been done.


Class hierarchy Recommendation
Class Hierarchy Recommendation
ClassHierarchyRecommendation

Do the instances of the super class cease to exist in the future? *	Y
Do the instances of the sub class cease to exist in the future? *	N
Are the super-class and subclass identical? *	Y
Is the subclass part of the super class? *	Y

Hierarchy Recommendation:

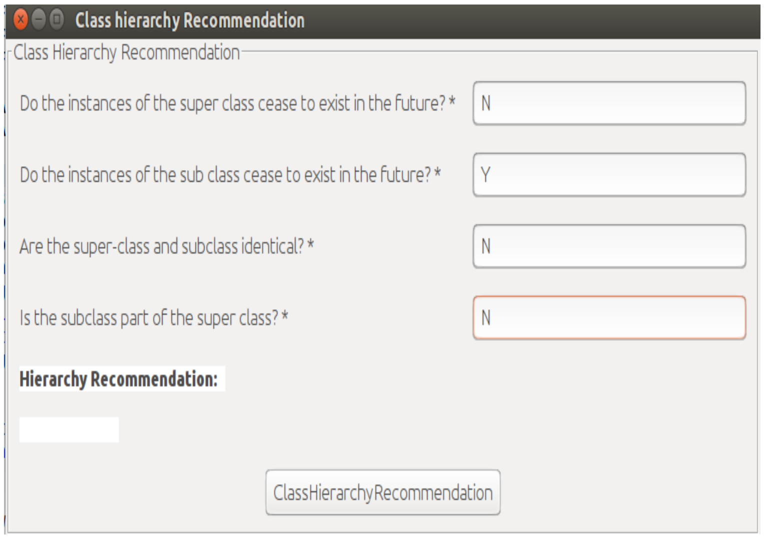
Figure 4.8: OntoSeer verifying the class hierarchy subsumption based on rigidity, identity, unity mentioned in Table 4.2. Here no violations to rigidity, identity and unity constraints has been done which is shown in output.



Hierarchy Recommendation:

Rigidity is correctly mainatained
Identity is correctly mainatained
Unity is correctly mainatained

Figure 4.9: OntoSeer's user inputs to question for class hierarchy valiadation. In this figure violations to Table 4.2 mentioned standards has been done.



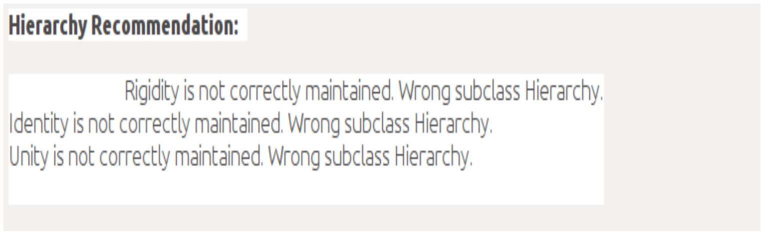
Class hierarchy Recommendation
Class Hierarchy Recommendation

Do the instances of the super class cease to exist in the future? *	N
Do the instances of the sub class cease to exist in the future? *	Y
Are the super-class and subclass identical? *	N
Is the subclass part of the super class? *	N

Hierarchy Recommendation:

ClassHierarchyRecommendation

Figure 4.10: OntoSeer verifying the class hierarchy subsumption based on rigidity, identity, unity mentioned in Table 4.2. Here violations to rigidity, identity and unity constraints has been done which is shown in output.



Hierarchy Recommendation:

Rigidity is not correctly maintained. Wrong subclass Hierarchy.
Identity is not correctly maintained. Wrong subclass Hierarchy.
Unity is not correctly maintained. Wrong subclass Hierarchy.

built from scratch due to the incompatibility of Protégé with Lucene. It took a considerable amount of effort to build the inverted index with individual terms.

- Protégé is not compatible with Deeplearning4j. Hence, instead of Doc2Vec, we have to use the Jaro-Winkler string matching algorithm for ODP recommendation. We have implemented the Jaro-Winkler algorithm from scratch, thus taking more time to build the ODP recommendation feature.
- OntoSeer was taking significantly more time to recommend axioms with the inverted index built during vocabulary recommendation. To reduce the retrieval time, we have indexed tuples instead of individual terms. Building an inverted index in the form of tuples from scratch again took a significant amount of effort.
- A significant amount of time and effort has been spent making the UI user-friendly and compatible with Protégé.

Chapter 5

Evaluation

In our evaluation of OntoSeer, we would like to get answers to the following questions.

1. Are the inexperienced ontology developers benefit from the recommendations of OntoSeer?
2. Is OntoSeer saving the time of ontology developers?
3. Are ontology developers able to make richer ontologies due to the axiom recommendations of OntoSeer?
4. Are ontology developers reusing existing ontologies more than they generally used to do?
5. How easy is it to use OntoSeer? Is there any feedback for the improvement of the tool?
6. Are the ontology developers able to create more modular ontologies by incorporating ODPs?
7. Are the ontology developers able to build ontologies that follow the naming conventions?

5.0.1 Dataset

OntoSeer makes use of the existing ontologies and vocabularies in making the recommendations. We, therefore, here describe the datasets that OntoSeer uses. OntoSeer has been implemented in java, jdk version being 1.8.0_252, and Protégé version is 5.5.0 and is available at github link `https://github.com/kracr/ontoseer`. OntoSeer uses the following datasets.

a) **Competency questions (CQs)**. 92 CQs and their corresponding ontologies are available at Software Ontology[1]. 52 CQs and ontologies are available at ArCo[2]. Several CQs and their associated ontologies are available from CORAL [4] and [15] .

b) **Ontologies**. We have collected ontologies from several repositories such as NCBO BioPortal[3], Manchester OWL Corpus[4], Oxford OWL Repository[5], and Protégé Ontology Library[6].

c) **Ontology Design Patterns**. ODPs are available at the ODP repository[7]. Since there is no bulk download option, we used a web scraper to collect all the ODPs. There are six categories of ODPs, and their URLs are used as the seed for the web scraper.

d) **Vocabularies**. Several vocabularies are indexed at the Linked Open Vocabularies (LOV)[8]. These vocabularies can be accessed using either the SPARQL endpoint or the LOV API.

5.0.2 User Study

We have conducted a survey among fifteen people from various educational institutes and of different proficiency levels in ontology development, and in using Protégé. Five of these fifteen are from our research group, while the remaining ten people are from various institutions. For the evaluation, the users were provided with two different scenarios. One scenario was based on Chess-playing, and the other was on the History of Human Evolution. The users were asked to build ontologies on them and along with that to validate the recommendations from OntoSeer. The survey has been conducted through an anonymous Google form. The complete list of questions used for user study are as follows:

1. How was the installation process?
2. How experienced are you with ontology modelling?
3. How experienced are you with using Protege?
4. Do you know about LOV and ontology repositories such as BioPortal?

[1]`https://softwareontology.wordpress.com/2011/04/01/user-sourced-competency-questions-for-software`

[2]`https://github.com/ICCD-MiBACT/ArCo`

[3]`https://bioportal.bioontology.org/ontologies`

[4]`http://mowlrepo.cs.manchester.ac.uk/datasets/mowlcorp/`

[5]`https://www.cs.ox.ac.uk/isg/ontologies/`

[6]`https://protegewiki.stanford.edu/wiki/Protege_Ontology_Library`

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

[8]`https://lov.linkeddata.es/dataset/lov/`

5. If Yes, please specify how often do you reuse classes and properties from LOV and ontology repositories?

6. Do you use Ontology Design Patterns (ODPs) while modelling?

7. If Yes, please specify how often do you use ODPs?

8. This section should be answered if competency questions have been provided to OntoSeer.

   (a) How useful are the class and property recommendations?
   (b) How useful are the ontology vocabulary recommendations?
   (c) How useful are the ODP recommendations?
   (d) How useful are the axiom recommendations?
   (e) How useful are the naming convention checker?

9. This section should be answered if competency questions are not provided to OntoSeer.

   (a) How useful are the ontology vocabulary recommendations?
   (b) How useful are the ODP recommendations?
   (c) How useful are the axiom recommendations?
   (d) How useful are the naming convention checker?

10. How was your experience of modelling an ontology without OntoSeer?

11. How was your experience of modelling an ontology with OntoSeer?

12. Does Ontoseer help in saving modelling time?

13. Do you have any suggestions for improving the user experience?

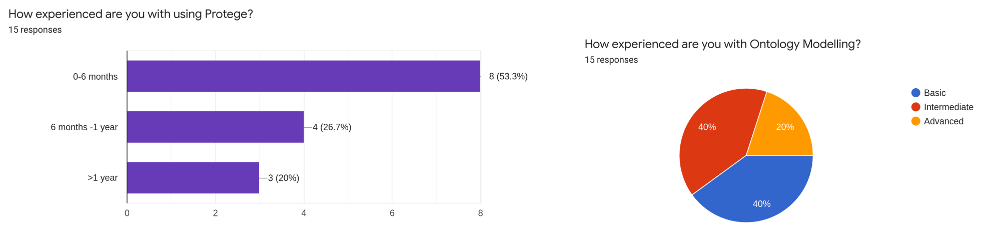


Figure 5.1: Figure shows users expertise with Protégé and in ontology modelling.

The first few questions on the evaluation focussed on the ease of the installation procedure. Thirteen users said that the plugin's installation is easy, and the instructions were clear, whereas two users found it moderately difficult to install the plugin. Other than that, the users were also asked about their expertise in building ontology and using Protégé. Figure 5.1 shows the users' expertise in using Protégé.

Seven people taking the survey mentioned they were unaware of terms like LOV and Bioportal, as shown in Figure 5.2, while six people said they were not well-versed with ODPs, as shown in Figure 5.3. In Figure 5.2 and Figure 5.3, we have included only those responses who said that they are aware of vocabularies and ODP, respectively.

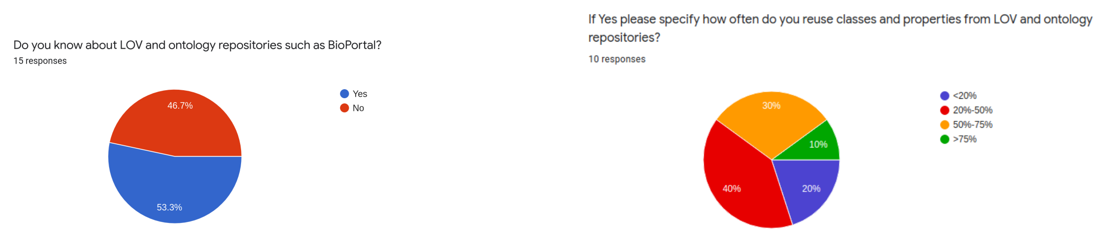


Figure 5.2: Figure shows the percentage of users being aware of LOV and the frequency of using LOV by knowledgeable users.

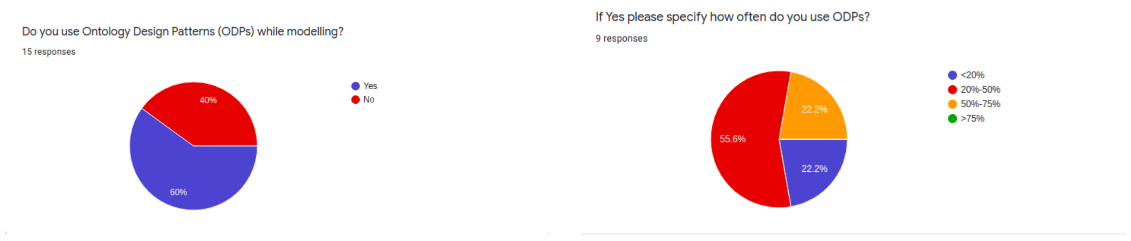


Figure 5.3: Figure shows the percentage of users being aware of ODP and the frequency of using ODP by knowledgeable users.

Table 5.1: OntoSeer Feature Result without CQs

Type(Score Range)	Class	Vocab	Naming	Axiom	ODP
Poor(1-2)	0	0	0	3	0
Neutral(3)	3	2	2	3	1
Good(4-5)	12	13	13	9	14

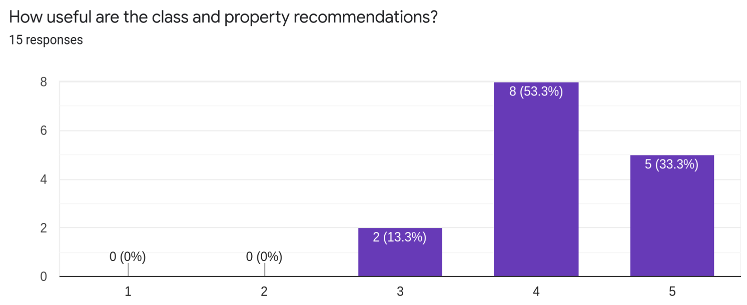
Table 5.1 represents the cumulative user responses for various features when the user is not having CQs, whereas Table 5.2 represents the cumulative user responses for various features when the user has CQs. Figure 5.4 to Figure 5.8 are the graphical representations of the user responses showed in Table 5.1 and Table 5.2.

On analyzing the figures Figure 5.4 to Figure 5.8 and Table 5.1 and Table 5.2, it is observed that OntoSeer performs better when CQs are present. This is because CQs give a more

Table 5.2: OntoSeer Feature Result with CQs

Type(Score Range)	Class	Vocab	Naming	Axiom	ODP
Poor(1-2)	0	0	0	2	0
Neutral(3)	2	1	1	3	0
Good(4-5)	13	14	14	10	15

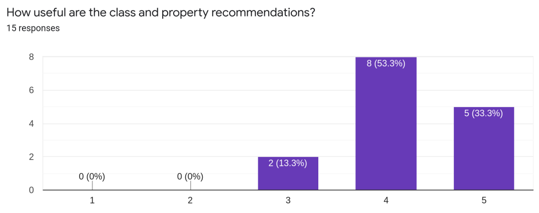
Figure 5.4: Figure represent user evaluation of class and property recommendations with CQs.



clear idea about the minimum requirements that the ontology should satisfy. As a result, OntoSeer gets more prominent class names and property names, and the recommendation gets more refined than the recommendations when no CQs are present.

Our analyses also show that the 85% of users are satisfied with Ontoseer recommendations, barring axiom recommendation. One of the reasons for low satisfaction for Axiom recommendation is that we have indexed only a few OWL corpora leading to failure in capturing all the recommendations. One of the suggestions that we have received from user study is to index more ontology corpus to get better axioms recommendation.

One important aspect of the evaluation of OntoSeer is the quality of the ontology that



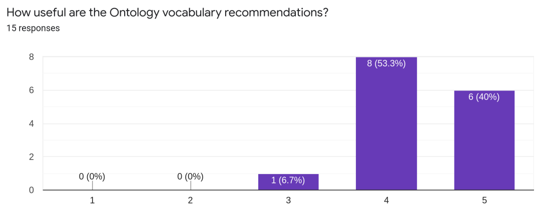


Figure 5.5: Figures represent user evaluation of various Vocabulary Suggestion without and with CQs.

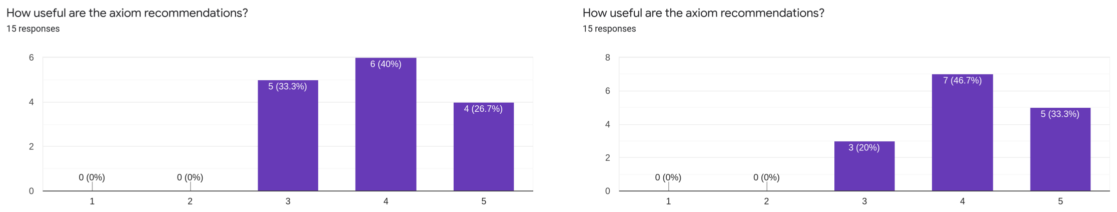


Figure 5.6: Figures represent user evaluation of various Axiom Recommendation without and with CQs.

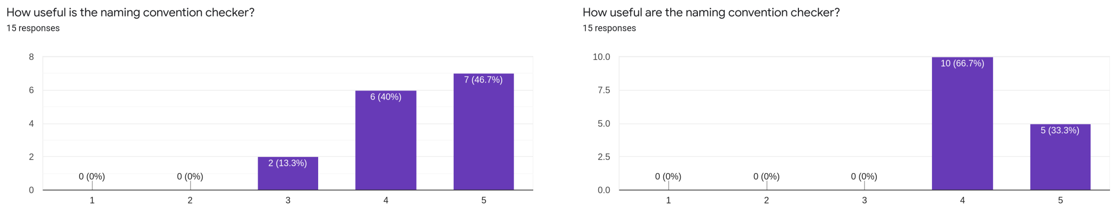


Figure 5.7: Figures represent user evaluation of various Naming Conventions without and with CQs.

gets developed and the amount of time it consumes to develop it. Eleven of the users taking the survey have said that OntoSeer helped in improving their ontologies, whereas thirteen users were of the opinion that OntoSeer saves modeling time as well. Figure 5.9 shows the performance of Protégé with Ontoseer over Protégé without Ontoseer.

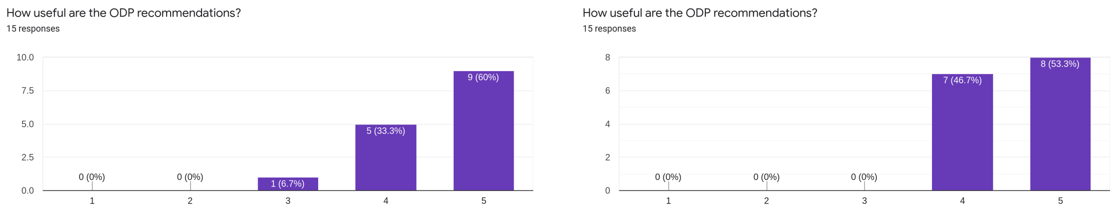
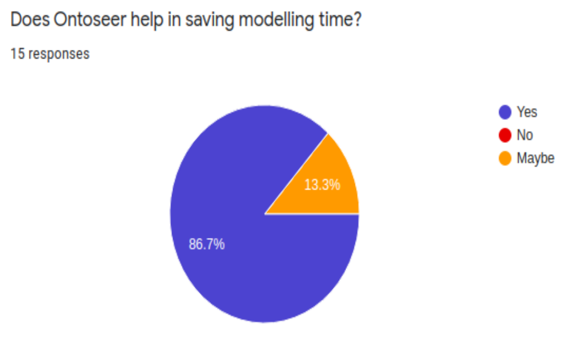


Figure 5.8: Figures represent user evaluation of various ODP Recommendation without and with CQs.

Figure 5.9: Figure shows user response on consumption of modelling time by OntoSeer.



Chapter 6

Sustainability Plan

With an increase in ontology modelling research, more vocabularies are getting added to LOV, Bioportal, while new and improved ODPs are regularly published. OntoSeer at present has indexed a limited number of vocabularies and ODPs, so with time, the recommendations will lose its prominence. Thus sustainability or maintenance of the tool is of significant concern. In building OntoSeer, we have been able to collect only a few OWL corpora and index them. The end-user can add other corpora according to their suitability by downloading them. OntoSeer will index them when it is first invoked. For LOV and BioPortal, the vocabularies will get updated as they get added on the website as OntoSeer is directly querying the respective websites. Users can download newly added ODPs by scraping through the main page of the ODP category as text files. OntoSeer will take those newly added text files into account and will update its recommendation accordingly.

Chapter 7

Conclusion

Ontology developers, especially the inexperienced developers, face many problems regarding the reuse of vocabularies or ODPs. Similarly, they may miss out on axioms, or the naming of the terms may not be according to conventions. It is not easy even for experienced developers to know whether the subclass hierarchy is correct or not.

To address the issues mentioned above, we have developed a tool named OntoSeer that recommends classes, properties, axioms, ontology design patterns and validates the class hierarchy in real-time while developing the ontology. According to 73.33 percent of users, OntoSeer has helped them in building better ontology, and 86.7 percent of users agree that it saves their modeling time. OntoSeer has been implemented as a Protégé plug-in and is available at github link `https://github.com/kracr/ontoseer`.

In the future, we plan to engage in a dialogue with the ontology developer to resolve confusing issues, such as whether a particular term should be a class or an instance or a property. This confusion will be resolved by asking the ontology developer a series of questions such as, does the term interact with other concepts, are the terms very specific or general, if a term is made into a class, what will be its instances? We are also planning to help the ontology developer to check whether the CQs would be answered by the ontology. This can be accomplished by having a dialogue with the ontology developer. At intermittent points of ontology development, OntoSeer can identify the subset of CQs (given by the ontology developer) that are related to the ontology developed so far. Ontology developers can then indicate whether these CQs can be answered using the ontology built so far. If they cannot be answered, changes need to be made to the ontology.

We are planning to make embeddings out of the ontologies that we have indexed, which are few of the popular ontologies. As a result, we can predict the axiom recommendations, closest to the one already present in the ontology. We are also planning to take the indexed ontologies as ground truth and want to evaluate the current ontology concerning that. This eliminates the need for a user study.

Bibliography

[1] Mamatha Balipa and Balasubramani Ramasamy. Search Engine using Apache Lucene. *International Journal of Computer Applications*, 127:27–30, 2015.

[2] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data: The Story so Far. *International Journal on Semantic Web and Information Systems*, 5:1–22, 2009.

[3] Kevin Dreßler and Axel-Cyrille Ngonga Ngomo. On the efficient execution of bounded jaro-winkler distances. 2015.

[4] Alba Fernández-Izquierdo, María Poveda-Villalón, and Raúl García-Castro. CORAL: A corpus of Ontological Requirements Annotated with Lexico-Syntactic Patterns. In *The Semantic Web*, pages 443–458. Springer International Publishing, 2019.

[5] Mariano Fernández-López, Asunción Gómez-Pérez, and Natalia Juristo. Methontology: from ontological art towards ontological engineering. *Engineering Workshop on Ontological Engineering (AAAI97)*, 1997.

[6] Adam Gibson, Chris Nicholson, Josh Patterson, Melanie Warrick, Alex Black, Vyacheslav Kokorin, Samuel Audet, and Susan Eraly. Deeplearning4j: Distributed, open-source deep learning for Java and Scala on Hadoop and Spark. 2016.

[7] Michael Grüninger and Mark Fox. Methodology for the Design and Evaluation of Ontologies. 1995.

[8] Nicola Guarino, Daniel Oberle, and Steffen Staab. What Is an Ontology? In *Handbook on Ontologies*, International Handbooks on Information Systems, pages 1–17. Springer, 2009.

[9] Nicola Guarino and Christopher A. Welty. An Overview of OntoClean. In *Handbook on Ontologies*, International Handbooks on Information Systems, pages 201–220. Springer Berlin Heidelberg, 2009.

[10] Krzysztof Janowicz. Introduction: Ontology Design Patterns in a Nutshell. 2016.

[11] Quoc V. Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. *CoRR*, 2014.

[12] Ajit Mahapatra and Sitanath Biswas. Inverted indexes: Types and techniques. *International Journal of Computer Science Issues*, 8, 2011.

[13] María Poveda-Villalón, Mari Carmen Suárez-Figueroa, and Asunción Gómez-Pérez. Validating Ontologies with OOPS! In *Knowledge Engineering and Knowledge Management*, pages 267–281. Springer Berlin Heidelberg.

[14] 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)*, (2):7–34, 2014.

[15] Yuan Ren, Artemis Parvizi, Chris Mellish, Jeff Z. Pan, Kees van Deemter, and Robert Stevens. Towards Competency Question-Driven Ontology Authoring. In *The Semantic Web: Trends and Challenges*, Lecture Notes in Computer Science, pages 752–767. Springer International Publishing, 2014.

[16] Daniel Schober, Ilinca Tudose, Vojtěch Svátek, and Martin Boeker. Ontocheck: Verifying ontology naming conventions and metadata completeness in Protégé 4. *Journal of biomedical semantics*, page S4, 2012.

[17] Pierre-Yves Vandenbussche, Ghislain A. Atemezing, and María and Vatant, Bernard Poveda-Villalón. Linked Open Vocabularies (LOV): A gateway to reusable semantic vocabularies on the web. 8(3):437–452, 2016.