UNIVERSITY OF PIRAEUS

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGIES

DEPARTMENT OF DIGITAL SYSTEMS

POSTGRADUATE PROGRAMME

“INFORMATION SYSTEMS & SERVICES”

OntoLink: A System for Improved Ontology Alignment

by

Constantinos Ginos

Submitted
in partial fulfilment of the requirements for the degree of
Master of Science
in the specialization of “Big Data & Analytics”
at the
UNIVERSITY OF PIRAEUS
September 2024

Thesis Supervisor: Christos Doulkeridis
Title: Prof.

Author: Constantinos Ginos

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Prof. Christos Doulkeridis, for his invaluable guidance and support throughout my graduate thesis. His expertise, encouragement, have greatly enriched my research and contributed to its success. Without his contributions, this thesis would not have been fulfilled.

I would also like to extend my heartfelt thanks to Mr. Giorgos Santipantakis for his invaluable advice and assistance. He has always been there to share his knowledge and provide guidance. Without the time and effort, he put into discussing my research, this thesis would not have been completed.

Abstract

Ontology Alignment is a crucial task in the field of knowledge management and semantic web, enabling the integration and interoperability of data from heterogeneous sources by matching concepts across different ontologies. Despite notable advances in this field, aligning accurately ontologies continues to present significant challenges. Many state-of-the-art systems often fall short in terms of efficiency and effectiveness, often consuming of a lot of computational power, while struggling to provide precise and reliable mapping.

This thesis introduces the OntoLink system designed to address these challenges. Specifically, OntoLink uses the anonymous classes found in the input ontologies, with the help of the Manchester syntax, to discover new mappings. In addition, it employs a specific reasoning task, utilizing the HermiT reasoner, to uncover further mappings. OntoLink's architecture has been designed to produce a set of precise output mappings, even when working with rich ontologies, that contain numerous classes and properties.

Our system was rigorously tested on ontologies from the repository of BioPortal. The results of OntoLink have been compared to the mappings that BioPortal reports between ontologies. In this way, we ensured that the final results were reliable and thoroughly validated.

The primary objective of this work is to introduce a newly developed system aimed at improving the efficiency and accuracy of ontology alignment tasks. OntoLink's performance has been tested on experiments with the ontologies PSDO, STATO, NGBO, confirming that the system performs and discover mappings with high efficiency and accuracy. Overall, OntoLink shows promising results in improving ontology alignment task.

Περίληψη

Η Ενοποίηση Οντολογιών (Ontology Alignment) είναι ένα σημαντικό κομμάτι στον τομέα της διαχείρισης γνώσης και του σημασιολογικού ιστού, επιτρέποντας την ενσωμάτωση και διαλειτουργικότητα των δεδομένων από ετερογενείς πηγές με την αντιστοίχιση εννοιών (mapping) μεταξύ διαφορετικών οντολογιών. Παρά την πρόοδο σε αυτό το πεδίο, το ontology alignment, εξακολουθεί να παρουσιάζει προκλήσεις. Πολλά σύγχρονα εργαλεία συχνά αποτυγχάνουν ως προς την αποδοτικότητα και αποτελεσματικότητα, παρέχοντας έτσι ανακριβή mappings αλλά και δαπανώντας αρκετή υπολογιστική ισχύ.

Η διατριβή αυτή παρουσιάζει το σύστημα OntoLink που σχεδιάστηκε για να αντιμετωπίσει αυτές τις προκλήσεις. Συγκεκριμένα το OntoLink, αξιοποιεί τις ανώνυμες κλάσεις που υπάρχουν στις οντολογίες, με τη βοήθεια του συντακτικού Μάντσεστερ, για την ανακάλυψη νέων mapping. Επιπλέον, γίνεται χρήση της συλλογιστικής (reasoning), εφαρμόζοντας τον HermiT reasoner, για τον εντοπισμό επιπλέον mapping. Η αρχιτεκτονική του OntoLink έχει σχεδιαστεί έτσι ώστε να παράγει ένα σύνολο από ακριβή mappings, ακόμα και όταν χρησιμοποιηθώ οντολογίες με μεγάλο πλήθος κλάσεων και ιδιοτήτων.

Το σύστημα μας εφαρμόστηκε σύνολα οντολογίες από το αποθετήριο του BioPortal. Τα αποτελέσματα του OntoLink έχουν συγκριθεί με τα mappings που προσφέρονται από το BioPortal, μεταξύ διαφόρων οντολογιών. Με τον τρόπο αυτό, εξασφαλίσαμε πως τα τελικά αποτελέσματα ήταν αξιόπιστα.

Ο κύριος σκοπός της εργασίας αυτής είναι η παρουσίαση ενός νέου συστήματος που έχει ως στόχο το αποτελεσματικό ontology alignment. Το OntoLink έχει δοκιμαστεί σε πειράματα με τις οντολογίες PSDO, STATO και NGBO επιβεβαιώνοντας ότι το εργαλείο λειτουργεί και ανακαλύπτει mappings με αποτελεσματικό τρόπο. Συνολικά, το OntoLink παρουσιάζει υποσχόμενα αποτελέσματα για τη βελτίωση των ontology alignment διεργασιών.

1. Introduction

1.1. Problem Statement

Ontology alignment is a challenging and time-consuming computational task, essential for knowledge management and the semantic web. It enables the integration and interoperability of data from heterogeneous sources by aligning concepts across different ontologies. Despite advances in this field, the diversity and the complexity of the ontologies make the accurate alignment a persistent challenge. The time and computational resources needed to discover mappings often make the process inefficient and unreliable.

Existing state-of-the-art systems and systems designed for ontology alignment often lack efficiency and effectiveness. These systems frequently miss easily computed mappings or discover incorrect mappings, leading to an inaccurate and incomplete alignment. Such errors reduce the quality of data integration and make applications that depend on these mappings less reliable. These ongoing issues highlight the need for improved tools and systems capable of improving the accuracy and completeness of ontology alignment tasks.

Furthermore, many current approaches rely on inefficient methods to calculate mappings. These inefficiencies make the alignment process a challenging task, resulting in inaccuracies and low precision in mappings among ontologies. Therefore, there is a need for innovative methodologies and techniques that can optimize the process of calculating mappings. Enhancing the efficiency and effectiveness of the methods is essential for advancing the field of ontology alignment.

1.2. Motivation

With the rapid spread of information around the world, ontologies have become popular way to represent and extract knowledge. This has led to the development of many and large ontologies, making their integration a challenging task, sometimes demanding a lot of computational resources. Therefore, in order to align ontologies efficiently and accurately, systems and techniques need to consider that this process requires time and computational resources.

In today's world, data come from many different sources, each having its own structure and vocabulary. For example, one set containing weather data that can be in CSV format, while another is in JSON, both representing the same information, using different labels. In the CSV file, the columns about data about clouds might as "Clouds" whereas in the JSON file, it might be "cloud_cover". Even though both of the datasets cover the same topic, these differences can make their integration difficult. Ontology Alignment addresses this issue by making easier the integration and combination of data from different sources.

Moreover, ontology alignment plays a key role for the Semantic Web, where data from various sources are linked and need to be processed by the machines. The Semantic Web provides

information that is represented in machine interpretable format, so that machines can process an understand it. Ontology alignment, helps in this situation, by combining data and schemas from diverse sources, ensuring that the similar concepts are recognized across the different machines.

Systems often need to communicate to understand each other's data. In some cases, systems need to follow commonly agreed schema (international standard -ISO). Alternatively, they implement ad-hoc translators between systems (which often takes a lot of effort). However, these approaches are not always feasible, since no standard can cover every possible application domain, standards usually need some time for updates (e.g. system cannot hold their messages and wait for some update of the standard used, that adapts to new technologies or system features), systems may have different domains (or perspectives, goals, features, etc). Ontology alignment can bridge the gap between systems, by discovering mappings between the schemas used by the systems, without affecting the systems, and more importantly without constraining the systems on which schemas to use. Ontology alignment makes sure that the meaning of terms and relationships stays consistent across systems, allowing them to communicate easily. In addition, it helps organizations to share and reuse knowledge without having to reinvent the wheel, making is simpler to build and expand their knowledge base.

Nevertheless, ontology alignment helps in solving the semantic heterogeneity, where various domains use different terms for the same concept or the same term for different concepts. For example, the word "*balloon"* can refer to a party balloon or a hot air balloon: they are the same word, but have a different meaning. Ontology alignment clarifies these differences, making the context easier to understand.

In summary, ontology alignment plays a key role for sharing and understanding concepts across different domains and systems. It supports the Semantic Web, towards expanding Linked Open Data.

1.3. Proposed Solution

This thesis introduces OntoLink, a system designed for ontology alignment that leverages unique methods to discover accurate mappings between ontologies. OntoLink's architecture is composed of components, each of which aims to uncover mappings effectively. It produces a precise set of mappings that helps the integration and interoperability of various ontologies.

OntoLink employs a novel approach to extract mappings. Initially, OntoLink's architecture integrates the indexing component of LogMap [1], extending it to store not only classes but also object properties. It also employs a similar iterative process to discover mappings. OntoLink then makes good use of the definition of classes provided in ontologies anonymous classes found in ontologies, with the help of the Manchester Syntax [2]. This method ensures that the set containing the mappings is expanded as the process progresses. In addition, OntoLink employs a reasoning task, applying the HermiT reasoner [3], to infer additional mappings if possible and detect any inconsistency that might occur.

OntoLink's architecture is designed to discover high-quality mappings that can align different ontologies accurately. It aims not only to uncover matches between named classes but also to identify matches between object properties within the ontologies. Overall, OntoLink enhances the data integration and improves the interoperability.

Preliminary experimental results indicate that OntoLink outperforms state-of-the-art systems. Ontologies from the BioPortal repository showcase that OntoLink achieves higher precision and recall metrics compared to other systems, indicating its effectiveness in discovering mapping. It's a system that accurately matches classes and object properties between diverse ontologies, producing a high-quality set of mappings.

In conclusion, with this thesis, we aim to present OntoLink, a newly developed system that discovers mappings across diverse ontologies. Its architecture features new ways of finding that enhance the mapping discovery process. With the development of OntoLink, we aim to make a significant contribution to the field of ontology alignment.

1.4. Structure of Thesis

This thesis is organized into five chapters. Chapter 2 (*The Theoretical Background*) lays the groundwork by discussing fundamental concepts, syntax and semantics of ontologies, reasoning tasks, and types of mappings used in ontology alignment. Chapter 3 (*OntoLink*) focuses on the structure and the components of the OntoLink system, detailing how each part functions. Chapter 4 (*Experimental Evaluation*) outlines the setup and results of the experiments conducted to evaluate OntoLink's performance, summarizing key findings and their significance. Finally, in Chapter 5 (*Concluding Remarks*) we discuss OntoLink's future and potential improvements.

2. Theoretical Background

In this chapter, we lay out the theoretical foundations, essential for understanding the concept of ontology alignment. We begin by diving into the ontologies, exploring their syntax and semantics. We also discuss the importance of consistency and satisfiability within an ontology. The chapter then, transitions into the specifics of ontology alignment, covering the different types of mappings and the challenges encountered in the alignment process. Additionally, we provide an overview of existing ontology alignment systems, comparing and analysing their strengths and weakness. With the presentation of this theoretical background, we provide a clearer understanding of the ontology alignment.

2.1. Ontologies: Syntax and Semantics

2.1.1. Definition of Ontology

In recent years, ontologies have become a popular way to represent and share knowledge within a specific field. They provide a clearer understanding of concepts, which are groups of entities with common characteristics. For example, ontologies can help us grasp simple concepts like the structure of a book or more complex ones like a vessel's trajectory in an area of the sea.

An ontology is the specific representation of a conceptualization from a domain of interest, describing the existing entities and how they are related. In simpler terms, it is a strictly description of a domain of knowledge, consisting of a set of terms and their semantic relationships [4]. An ontology provides a vocabulary as well as the definitions to capture the meaning of a concept, in a formal way that can be processed by the machines.

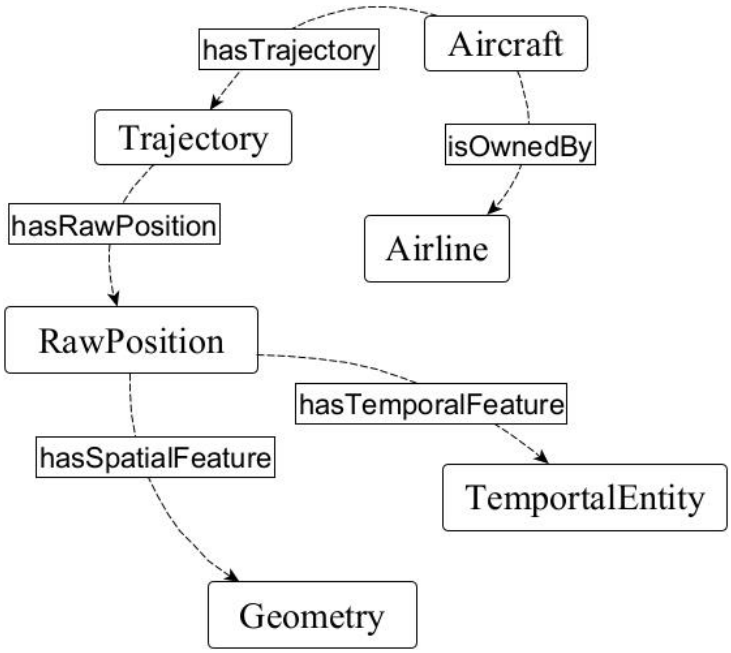

Figure 1. Example of an ontology.

Figure 1 showcases the concept of an aircraft's trajectory. In this example, the terms *Airline, Aircraft, Trajectory, RawPosition, Geometry* and *TemporalEntity,* represent the entities of the ontologies, while *isOwnedBy, hasTrajectory, hasRawPosition, hasSpatialFeature* and *hasTemporalFeature* show the relationships among these entities.

Ontologies are used in many fields to help organize and manage information. They are important for technologies such as peer-to-peer information sharing or query answering. Moreover, they are commonly used in areas like healthcare, where they help doctors and systems share patient information accurately or in e-commerce, where they organize products and services.

2.1.2. The Syntax

Ontologies play a key role in the Semantic Web by helping to use and represent data, even from heterogeneous data sources. In the Semantic Web, the entities are treated as resources, each having a unique identifier for their distinction, called a *Uniform Resource Identifier (URI)* For example, a URI could like *http://example.org/car* could refer to a resource labeled as *"car"*. Additionally, they can have a literal, such as number, string, datetime.

It's important to have a way to represent information about resources or entities. This is where the *Resource Description Framework (RDF)* is useful, as it provides a framework for describing the semantics of information about entities. It allows us to link data from different formats, like XML, JSON, and more. RDF offers a fundamental vocabulary to describe the resources.

With the use of the RDF, we can represent information as a directed graph, where the nodes represent the resources and the edges represent the relationships between them. This creates what is called an RDF triple, which has the format of (*subject – predicate – object).* The subject and predicate, are nodes on the graph, while the predicate is the edge that connects them. The subject and predicate can be donated by a URI, while the object can have either a URI or a literal.

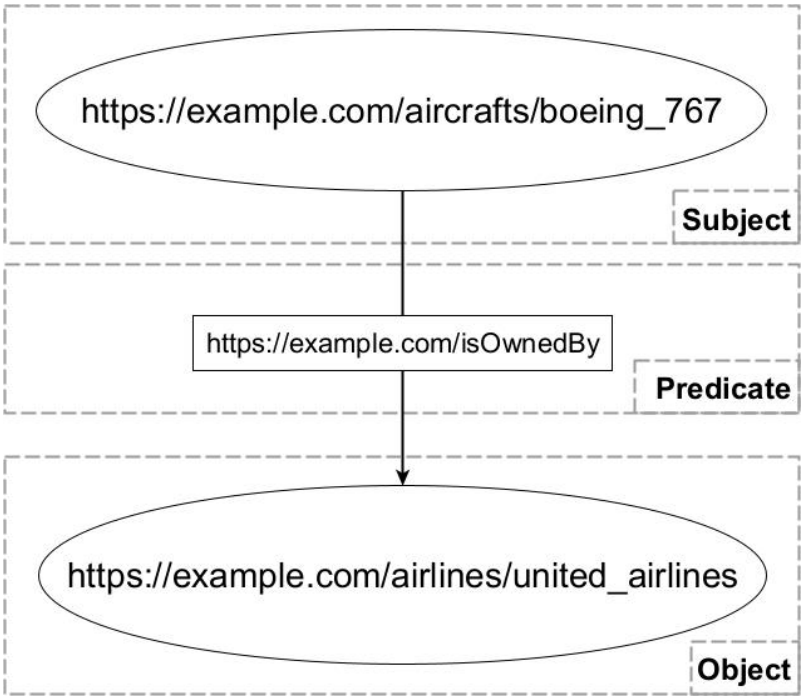

Figure 2. Example of an RDF triple.

Figure 2, depicts an example of an RDF triple. This triple is composed from the components:

i. **Subject** - *https://example.com/aircrafts/boeing_767*
In this example, the resource with the URI https://example.com/aircrafts/boeing_767 is the subject, which is an instance of the class Aircraft.

ii. **Predicate** - *https://example.com/isOwnedBy*
The predicate specifies the relationship between the subject and the object. In this example, https://example.com/isOwnedBy is the predicate.

iii. **Object** - *https://example.com/airlines/united_airlines*
In the triple, the resource with the URI https://example.com/airlines/united_airlines is the object, which is an instance of the class Airline.

In addition, the *Resource Description Framework Schema (RDFS)*, it's an extension of the RDF, that offers a richer vocabulary to describe resources and their relationships. RDFS offers terms that helps to define and model RDF data. Key terms provided by RDFS are:

1. **rdfs:Class**
Resources divided into groups are called classes. Classes represent sets resources within the ontology, that share common attributes or characteristics. For example, in Figure 1, *Aircraft* could be class, representing all the types of aircrafts.

2. **rdfs:Property**
   Properties describe relationships between resources, specifically between a subject resource and an object resource. Properties themselves are also resources. For example, in Figure 1, *isOwnedBy* is a property that links the resources *Aircraft* and *Airline*, showing the relation between them.

To create an ontology that describes precisely a specific domain of interest, we need a language capable of representing complex knowledge. For example, in the aviation domain, we might need to specify that an airplane has exactly two wings or that it can carry a maximum of 300 passengers. Therefore, the W3C OWL 2 Web Ontology Language (OWL) was designed to represent rich and complex knowledge. OWL2 is built upon RDF/S and is a more expressive language and enables the ontology engineer to build more detailed models of a domain. OWL2 provide additional features to implement an ontology. Some key features are:

1. **Complex Class**
   In order to express more complex knowledge, OWL provides logical constructors such as *and, or* and *not.* These allows to define more complex classes based on certain constraints or conditions. Using these constructors, we can combine classes with names into complex classes, creating more detailed classes. Below, is an example of complex class in OWL, that states the class Father consists of exactly those objects which are instances of both Man and Parent.

   Class: *Father*
   EquivalentTo:
   Man **and** *Parent*

2. **Type of Property Restrictions**
   A key aspect of defining complex classes in OWL is the use of property restrictions. These restrictions use constructors involving properties and are used to create more detailed, logic-based class definitions. OWL2 provides an extended set of these restrictions, including:

   i. *owl:someValuesFrom*
   This restriction defines a class as a set of all individuals that are connected by a specific property to another individual that belongs to a certain class. For example, we can define that a class *Parent* is a *Person* who has some *child Person.*

   ii. *owl:allValuesFrom*
   This restriction specifies a class of individuals for which all related individuals must be instances of a given class. For example, a class *Airplane* can have only instances from class *Wing* as its wings.

   iii. *owl:minQualifiedCardinality*
   This operator sets the minimum number of values that a property can have. For instance, a class *Child* must have at least two instances from the *Parent* class.

iv. *owl:maxQualifiedCardinality*
This operator sets the maximum number of values that a property can have. For example, a class *Child* can have at most two instances from the *Parent* class.

v. *owl:qualifiedCardinality*
This operator specifies an exact number of values that a property must have. For example, we can specify that the class *ParentOfTwins* consists of all individuals who have exactly two instances of the class *Child*

2.1.3. The user-friendly Manchester Syntax

The ontology engineering is a challenging task which becomes even harder defining concepts and relations using the RDF/S format. To address this, a simpler and more user-friendly syntax was needed. This led to the creation of the Manchester OWL Syntax [2].

The Manchester OWL Syntax is a user-friendly and compact syntax used to write OWL ontologies. It provides a more human-readable syntax to represent and manage knowledge expressed in OWL. The goal of creating it, was to provide a simple and compact syntax, avoiding complex symbols and terms. With its use, we can precisely define classes and properties within an ontology. Table 1 shows terms used in Manchester Syntax compared to those in OWL, with examples provided. We use C and F to represent classes, R to represent a property, and n to represent a numerical value.

Table 1. The Manchester OWL Syntax.

OWL Constructor	Manchester OWL Syntax	Example
intersectionOf	C **AND** F	Animal **AND** Mammal
unionOf	C **OR** F	Mammal **OR** Bird
complementOf	**NOT** C	**NOT** Mammal
someValuesFrom	R **SOME** C	hasFriend **SOME** Student
allValuesFrom	R **ONLY** C	hasFriend **ONLY** Student
minQualifiedCardinality	R **MIN** n C	hasFriend **MIN** 2 Student
maxQualifiedCardinality	R **MAX** n C	hasFriend **MAX** 5 Student
qualifiedCardinality	R **EXACTLY** n C	hasFriend **EXACTLY** 3 Student
minCardinality	R **MIN** n	hasFriend **MIN** 2
maxCardinality	R **MAX** n	hasFriend **MAX** 5
cardinality	R **EXACTLY** n	hasFriend **EXACTLY** 3

Based on Table 1, we can see that defining classes and properties for an ontology, is a simple and straightforward way using the Manchester Syntax. It allows us to create more complex ontologies easily. Figure 3 provides a more detailed example using Manchester Syntax.

```
Class: Vessel
    EquivalentTo:
        hasPart only (Engine or Hull or Deck or Sail)
        and (hasPart min 1 Engine)
        and (hasPart max 5 (Engine or Deck))
        and (hasPart exactly 1 Hull)
        and (hasPart some Sail)
        and
            not (hasPart some Fuselage)
```

Figure 3. Example of Manchester syntax.

This example defines the class *Vessel* with its characteristics and restrictions using Manchester Syntax. In details:

- *hasPart only (Engine or Hull or Deck or Sail)*
  This restriction defines the set of individuals that are related with only Engine, Hull, Deck, or Sail instances, linked by the hasPart property.

- *hasPart min 1 Engine*
  This restriction defines the set of instances that have at least one instance of the class Engine, linked by the hasPart property.

- *hasPart max 5 (Engine or Deck)*
  This restriction defines the set of instances that have at maximum 5 instance either of the class Engine or Deck, linked by the hasPart property.

- *hasPart exactly 1 Hull*
  This restriction defines the set of instances that have exactly one instance of the class Hull, linked by the hasPart property.

- *hasPart some Sail*
  This restriction defines the set of instances that are linked to Sail by the hasPart property.

- *not (hasPart some Fuselage)*
  This restriction defines the set of instances that are not linked to class *Fuselage* by the hasPart property.

As shown in the example, the class *Vessel* is the *Named Class or Atomic Class*, while the constraints we described are the *Complex Classes or Anonymous Classes*. With these restrictions, we provided a more detailed description for the class Vessel. Manchester Syntax allows us to define the components of a vessel using simple and compact terms. Instead of using complex syntax, we can utilize a simple vocabulary that is easy for both humans and machines to understand. Moreover, it is widely supported by various ontology processing tools, such as Protégé and libraries like OWLAPI in Java. Figure 4, showcases how Manchester syntax is represented in Protégé [5].

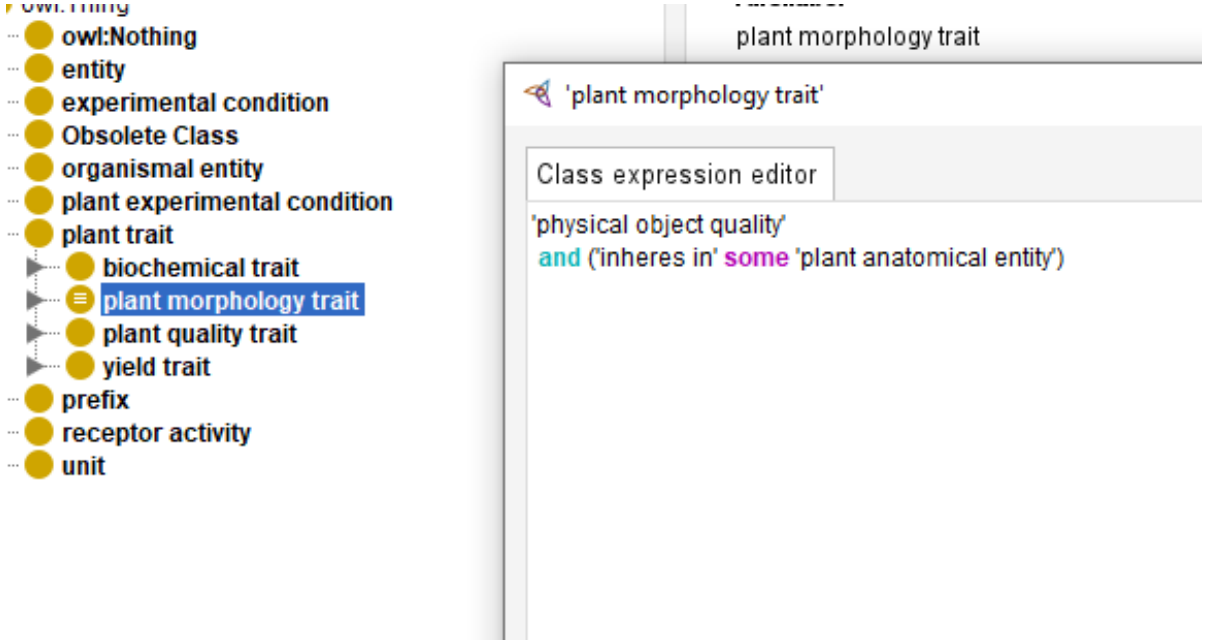

Figure 4. Manchester syntax in Protégé.

In summary, the Manchester Syntax, offers a user-friendly approach to construct ontologies. It simplifies their representation, by using terms like “some”, “values", and many more instead of complex symbols, which makes them easier to read. Manchester Syntax is widely used in ontology processing tools and libraries, helping users develop and design ontologies efficiently.

2.1.4. The Semantics

In our everyday life, we often encounter the same word used in different ways. For example, the word *rock* can refer to a stone you find in a forest or a music band you enjoy. *I picked up a rock from the mountain* is different from *I like this rock band*. This difference is based on the meaning each word has in a sentence, which is what *semantics* is about; explaining the meaning of the words.

The main concern of the semantics is to clarify what the expressions of a language means. For instance, in an ontology, there might be a resource labelled “*earth”*. On its own, “*earth”* is just

a string. But if the statement "*earth isA Planet*" is made, semantics add meaning by showing that "*earth*" is an instance, "*Planet"* is a class, and "*earth*" is an instance of the class "*Planet*". In this way, semantics adds meaning to the symbols, ensuring that machines can understand and interpret them.

Mathematical logic is applied to deduct inferences on the statements within an ontology. To do so, a function that maps RDF/S and OWL statements into set theory must be applied. The interpretation function I maps each atomic class C_i – i.e. class with name - to a set of elements, and each atomic property R_j into pairs of elements. Every triple of the format $(subject, predicate, object)$ or (s, p, o) is a statement. A triple can be represented with a vocabulary of URIs, blank nodes (bnodes) and literals. Formally they can be written as: $(s, p, o) \in (\text{URI} \cup \text{bnode}) \times \text{URI} \times (\text{URI} \cup \text{bnode} \cup \text{literal})$. An RDF graph is defined as a finite set of statements. Then, inference rules must be applied on the statements to give them a specific meaning.

The inference rules are valid due to the notion of entailment. In simple terms, entailment is a semantic relation between RDF graphs - i.e. set of statements. Given a graph E, we say that I satisfy E, if $I(E)\ is\ true$. The graph E is satisfiable when there is an interpretation that satisfies it. Now given another graph G, we say that G entails a graph E if every interpretation that satisfies G also satisfies E. Those entailments, can be viewed as rules that add the semantics to a graph, and therefore, to an ontology. Below, we show some key entailment rules, where $x, y, z, w, and\ v$ represent unique URIs.

Table 2. Entailment Rules.

If G contains:	Then G entails recognizing E	Interpretation
$x\ w\ y$.	x rdf:type rdfs:Resource .	x is an instance of class Resource in RDFS
$x\ w\ y$.	y rdf:type rdfs:Resource .	y is an instance of class Resource in RDFS
$x\ w\ y$.	w rdf:type rdf:Property .	w is an instance of class Property in RDF
x rdf:type rdfs:Class .	x rdfs:subClassOf rdfs:Resource .	x is a subclass of Resource - all instances of x are also instances of Resources.
x rdfs:subClassOf y . y rdfs:subClassOf z .	x rdfs:subClassOf z .	x is a subclass of z - all instances of x are also instances of z.
x rdfs:subClassOf y . z rdf:type x .	z rdf:type y .	z is an instance of class y.
x rdfs:subPropertyOf y . y rdfs:subPropertyOf z .	x rdfs:subPropertyOf z .	x is a subproperty of z - all resources related to x are also related to z.
x owl:equivalentClass y .	y owl:equivalentClass x .	class y is equivalent to class x - x and y contain the same individuals.
x owl:equivalentClass y . y owl:equivalentClass z .	x owl:equivalentClass z .	class x is equivalent to z - x and z contain the same individuals.
w owl:equivalentProperty v .	v owl:equivalentProperty w .	property v is equivalent to property w - w and v are the exact same properties.

Entailment rules are essential for defining an ontology. For example, in Figure 5, we have two ontologies that represent the same concept: a teacher assigning homework. The first ontology can be expressed as the triple (Teacher, gives, Homework), while the second can be expressed as (Professor, assigns, Schoolwork). To establish that both ontologies represent the same concepts, specific entailment rules must be defined. These rules are:

- :Teacher owl:equivalentClass :Professor .
- :Homework owl:equivalentClass :Schoolwork .
- :gives owl:equivalentProperty :assigns .

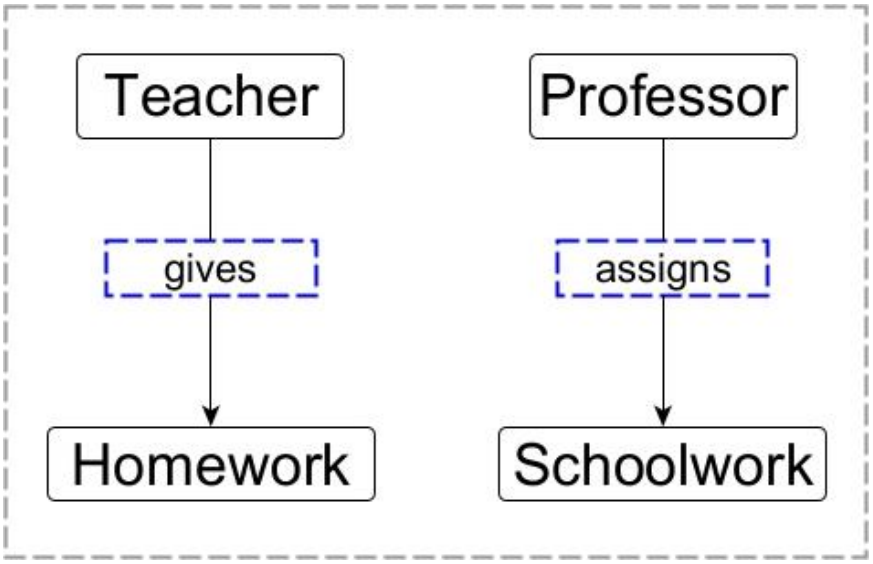

Figure 5. Example of semantically same ontologies.

By defining these rules, we assert that the class *Teacher* is equivalent to *Professor*, *Homework* is equivalent to *Schoolwork*, and the property *gives* is equivalent to *assigns*. This application of entailment rules ensures that the two ontologies have same terms, therefore are semantically same. Although they use different terms for the same classes and properties, they convey the same meaning. Without these definitions, we would struggle to define the resources and relations among them.

s. Syntax and semantics are two distinct yet complementary aspects of an ontology. The sentence "*Flying ship plays on the road"* is syntactically correct but lacks its semantics. Similarly, a computer can correctly structure a sentence but must also understand its meaning to interpret it correctly. Therefore, both proper syntax and semantics must be defined to make an ontology machine-interpretable. Otherwise, it can lead to inconsistencies within the ontology. Many tools and systems have been developed to detect inconsistencies. These systems are known as *reasoners,* and by applying entailment rules they can derive new statements. We will explore them further in the following chapter.

2.1.5. Reasoning Tasks

As shown previous sections, it is crucial for an ontology to use both correct syntax and semantics. If these are not used properly, it can lead to problems and contradictions. This process of ensuring that knowledge is represented accurately is known as *reasoning* in ontology. Reasoning uses rules and constraints from Description Logic to infer new knowledge and check if the ontology is consistent.

Reasoning tasks are essential for keeping an ontology consistent. They help ensure that knowledge is represented correctly within the ontology. Some key reasoning tasks are the following:

1. **Class Subsumption:** Determines if one class is subclass of another class. It completes the class hierarchy with inferred subsumptions (including those involving Top or Bottom concepts

Figure 6. Example of class subsumption.

Figure 6 shows an example of class subsumption in Manchester syntax. Here, the class *Vehicle* subsumes the class *Car* because a car is a more specific concept within the broader concept of a *Vehicle.* In this way, we can organize the structure of the ontology.

2. **Satisfiability:** Satisfiability refers to the ability of a class to have at least one instance without causing any logical contradictions in the ontology. An *unsatisfiable* class, on the other hand, refers to the class that cannot have any instances because of conflicting constraints. Figure 7 shows an example of an unsatisfiable class in Manchester syntax. The class *HybridCar* is unsatisfiable because it is defined as the intersection of *GasCar* and *ElectricCar*, which are disjoint classes. This makes *HybridCar* an empty class, so we cannot create any instances for it.

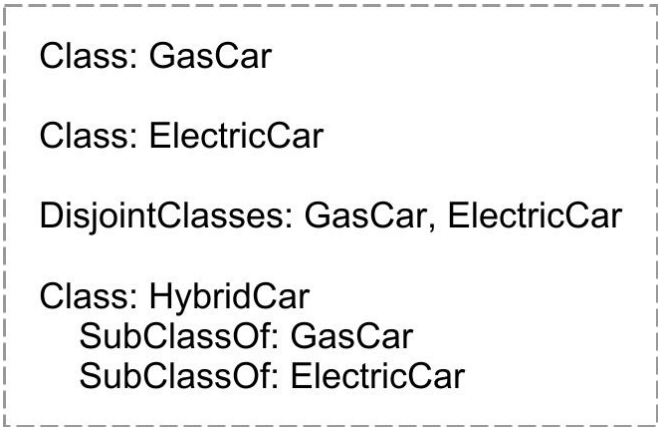

Figure 7. Example of unsatisfiable class.

3. **Consistency Check:** We say that an ontology is *consistent* if it doesn't have any instances that create logical conflicts. Thus, if there is an instance that causes logical conflicts, it makes what is known as an *inconsistent* ontology. It's important to ensure there is no conflicting information that would make it impossible to represent the knowledge correctly within the ontology.

```
Class: GasCar

Class: ElectricCar

DisjointClasses: GasCar, ElectricCar

Class: HybridCar
   SubClassOf: GasCar
   SubClassOf: ElectricCar

Individual: Ford
   Types: HybridCar
```

Figure 8. Example of inconsistency.

Figure 8 shows an example of inconsistency in Manchester Syntax. In this example, the disjointness constraint states that *GasCar* and *ElectricCar* cannot overlap. Since *HybridCar* is defined as a subclass of both *GasCar* and *ElectricCar*, it should be an empty class. If we specify an individual, like Ford, as a *HybridCar*, it must be an instance of both *GasCar* and *ElectricCar*. However, because *GasCar* and *ElectricCar* are disjoint, this is impossible. Therefore, Ford cannot be both *GasCar* and *ElectricCar* at the same time. This leads to a contradiction, making the ontology inconsistent.

Class subsumption, satisfiability, and consistency checks might be different tasks, however they are related to one another. Class subsumption sets up the hierarchy of classes, which impacts satisfiability by determining class relationships. Satisfiability checks if there are no instances that can cause conflicts. Consistency checks ensure that the entire ontology has no logical contradictions and that all instances come from classes that are not unsatisfiable.

Reasoning tasks are essential for maintaining the coherence of ontologies. By performing task such as, class subsumption, satisfiability and consistency check, we ensure that ontologies manage and represent the knowledge correctly. To perform these tasks, tools have been developed called *reasoners.* Reasoners apply algorithms based on Description Logics [6] to handle various sizes and complexities of ontologies. Many reasoners, such as HermiT, Pellet and ELK [7, 8], have been developed to perform these tasks accurately and efficiently.

HermiT is one of the most popular and highly-regarded reasoners due to its efficiency. Since it's available to the public for free, it's been widely used across various systems. Previously, reasoning with large and complex ontologies was time-consuming. However, HermiT handles these tasks efficiently and in a short period of time. Moreover, it can manage reasoning for ontologies that require significant computational power. This is why HermiT is used in this work.

2.2. Ontology Alignment

2.2.1. Ontology Mappings

Ontology alignment plays a crucial role in ontologies and knowledge representation. However, it's important to understand some key concepts in more detail. In this section, we will define the terms relevant to our work for clearer understanding.

Correspondence. Given two ontologies O_1 and O_2 , a correspondence between O_1 and O_2 is defined as the triple $< e_1, r, e_2 >$, where

- $e_1 \epsilon O_1, e_2 \epsilon O_2$ represent entities from each ontology
- r is the relationship between these entities e_1 and e_2

In our work, we focus mainly on correspondences between classes or between object properties in ontologies. For classes, we use the format $< c_1, r, c_2 >$, where $c_1 \epsilon O_1$ and $c_2 \epsilon O_2$ and r is the relationship between c_1 and c_2. For object properties, we use the format $< p_1, r, p_2 >$, where $p_1 \epsilon O_1$ and $p_2 \epsilon O_2$ and r is the relationship between p_1 and p_2. For example, a correspondence from Figure 12 could be defined as $< Car, \equiv, Automobile >$. In some case, it can be formatted as $< p_1, r, p_2, d >$, where $d \in [0,1]$ is the similarity (or condidence) degree.

Mapping. Mapping refers to the set of correspondences found between two ontologies [34]. In our work, we focus on mappings between classes or between object properties. For example, in Figure 5, we can define the set of correspondences $M = \{< Teacher, \equiv Professor >, < Homework, \equiv Schoolwork >, < gives, \equiv assigns >\}$, which is the mapping between the two ontologies. Essentially, a mapping represents an instance of a graph, and therefore, an instance of an ontology.

Ontology Alignment. Ontology alignment, or ontology matchings, refers to the process of discovering correspondences between terms from different ontologies and generating a final mapping set. It takes as input a set of ontologies O and produces a final set of correspondences M.

2.2.2. Types of Mappings

Mappings are crucial for ensuring integration and interoperability of data from heterogeneous sources. Each type of mapping has a specific purpose and shows how classes and properties relate to each other. In this section, we will explore the different types of mappings used in ontology alignment, explaining their relation. The types of mappings are:

1. **Equivalence** $(\equiv)$
   Equivalence mappings between two classes C_1 and C_2 mean that the matched classes are the same or equivalent. This means both classes contain exactly the same set of

individuals. Similarly, an equivalence mapping between two object properties, P_1 and P_2 means that a class X can be connected to class Y by both properties P_1 and P_2. In an RDF alignment file is denoted as $=$.

2. **Subsumption** ($\sqsubseteq/\sqsupseteq$)
   Subsumption mappings indicate that one class or property is a more specific case of another. A class C_1 is said to be a *subclass* of another class C_2 if every instance of C_1 is also an instance of C_2. In this case, C_1 is subsumed by C_2 ($C_1 \subseteq C_2$) . Similarly, a property P_1 is said to be a *sub-property* of another property P_2 if, whenever P_1 holds between two classes, P_2 also holds between those classes ($P_1 \subseteq P_2$). In an RDF alignment file is denoted as $<$ or $>$.

3. **Disjointness** ($\perp$)
   Disjointness indicates that two classes or properties are defined as disjoint, meaning that they do not share any common instances or values. A disjoint mapping refers to two classes and properties that are defined as disjoint from each other. Two classes C_1 and C_2 are said to be disjoint if no individual can simultaneously be an instance of both classes. Similarly, two properties, P_1 and P_2 are said to be *disjoint* if there are no instances that can have both properties at the same time. In an RDF alignment file is denoted as $\%$.

There are different types of mappings, each representing a specific semantic relation. Many tools and systems have been created to identify these various types mappings. In our work, we focus only on equivalence mappings – i.e. the triples $< c_1, \equiv, c_2 >$ between classes and the triples $< p_1, \equiv, p_2 >$ between object properties.

2.2.3. Definition of Ontology Alignment

In ontology engineering, it's a common approach to reuse or extend existing ontologies rather than creating new ones. Different domains may have overlapping concepts. For instance, an ontology about vehicle maintenance and an ontology about trajectories of moving object are likely to have common conceptualization. That is why, ontologies such are these can be combined to create a single and comprehensive ontology. This process is known as ontology alignment. The goal of ontology alignment is to detect terms that have the same meaning in a context – i.e. the interpretation function maps the corresponding symbols to the same elements of the domain.

As previously mentioned, we can align two ontologies by finding correspondences between their entities, producing a final mapping. This process helps achieve semantic heterogeneity. In this work, we mainly focus on equivalence mappings between classes or between object properties. For example, consider the two ontologies, O_1 and O_2, shown in Figures 9 and 10. Both represent the same concept: *a car that is owned by someone and fixed by a mechanic*.

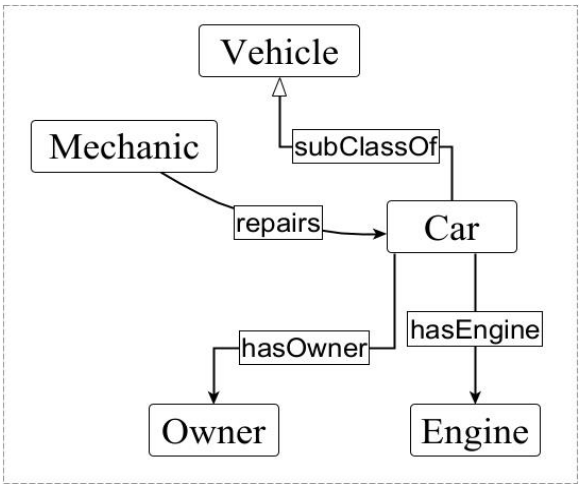

Figure 9. Ontology O_1.

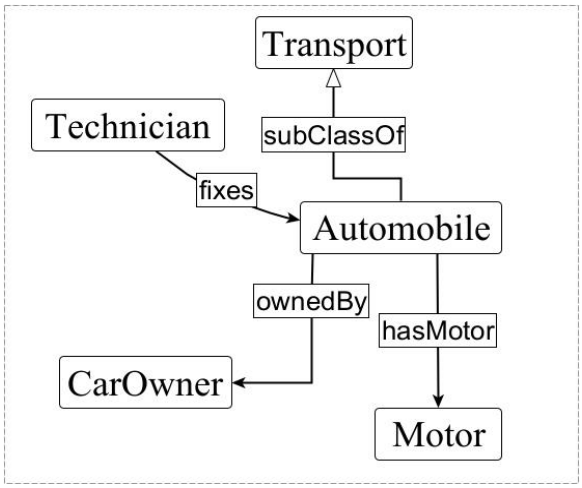

Figure 10. Ontology O_2.

These two ontologies might use different names for classes and properties, yet they represent the same concept. Classes are shown in rectangles with round corner, and the properties are rectangles on the edges. Assume, that we want to create a database that stores data about this concept. In order to achieve that we must integrate O_1 and O_2, and create a single comprehensive ontology. This requires identifying the correspondences between their terms. In this example, and in our work, we focus on equivalence correspondences, where terms can be merged into one. For instance, the classes *Car* and *Automobile* have different names but refer to the same set of physical objects, so they can be merged into a single class. Figure 11 shows all possible equivalences between the two ontologies, highlighted with blue edges.

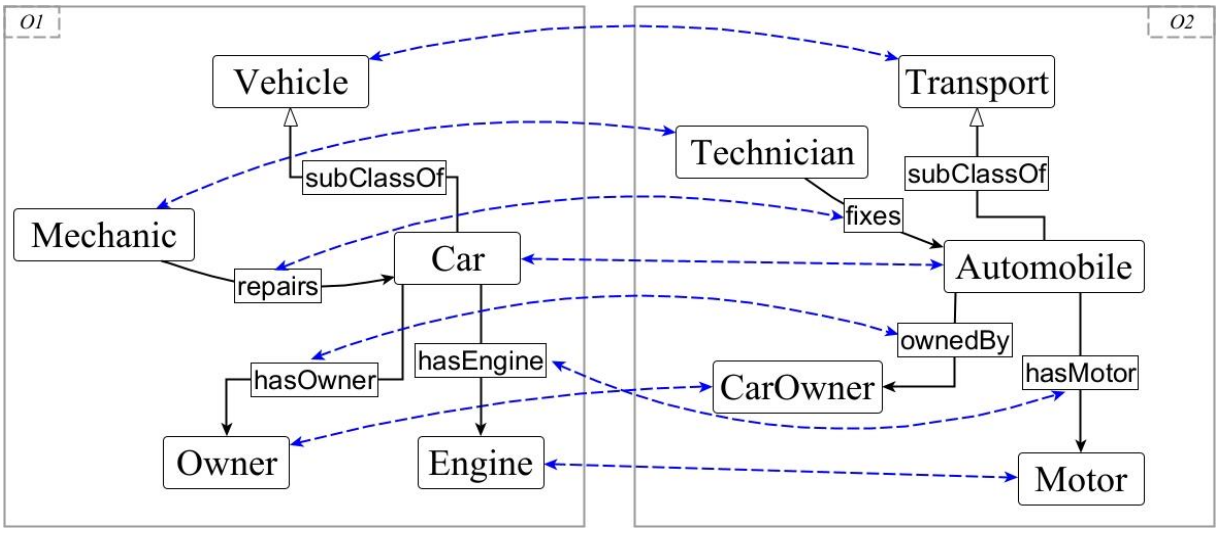

Figure 11. Mappings between ontologies O_1 and O_2

As shown in Figure 11, there are equivalences between classes and between object properties. The final mapping that includes all the possible correspondences is the following:

- $< Vehicle, \equiv, Transport >$
- $< Car, \equiv, Automobile >$

- $< Mechanic, \equiv, Technician >$
- $< Owner, \equiv, CarOwner >$
- $< Engine, \equiv, Motor >$
- $< repairs, \equiv, fixes >$
- $< hasOwner, \equiv, isOwnedBy >$
- $< hasEngine, \equiv, hasMotor >$

Discovering mappings between ontologies is crucial for managing ontologies effectively. Mappings establish semantic relationships between entities in different ontologies, finding a solution in the heterogeneity problem. They are also essential for the Semantic Web because they enable different systems to understand and use information, achieving semantic interoperability. In the following section, we will provide formal definitions of key concepts to help us understand ontology alignment in depth

2.2.4. Challenges in Ontology Alignment

While ontology alignment helps solve the problem of data heterogeneity, it does not come without challenges. These include managing and processing large ontologies, the time and computational resources required, and more. Understanding these challenges is important for improving the process and developing effective ontology alignment systems. Some key challenges that the ontology alignment process poses are:

Complexity in the Alignment Process. Ontologies often vary in structure making it hard to find exact or even approximate structural matches between them. This challenge is becoming harder when dealing with large-scale ontologies that contain numerous classes, properties, and instances. Aligning them demands a lot of time and computational resources. Therefore, the process must handle these difficulties while keeping the mappings accurate and reliable [30].

High Memory Requirements. One major challenge in ontology alignment is the high memory demand. As ontologies become larger and more complex, more memory is needed to process and store them. A simple approach is to compare each entity from one ontology with every entity from a second ontology. However, this becomes a problem when working with large ontologies, as it can cause out-of-memory errors, leading to failure. Efficient memory management is crucial to avoid these issues [31].

Execution Time of the Mapping Process. Increased execution time is a crucial challenge in the ontology alignment process. The algorithms used to find correspondences can be computationally intensive, especially with large ontologies. As ontologies grow in size and expressiveness the time needed for these algorithms can increase significantly, slowing down the process. To address this, strategies like parallel processing, efficient indexing, and heuristic-based approaches can help reduce execution time and improve performance [32].

Evaluation of the Alignment. Evaluating the final alignment process is an important task. In other words, it's important to check if the discovered mappings are correct. When aligning two ontologies from different areas of interest, it's essential to have a reference to verify if the mappings are correct. The OAEI campaigns [9] provide reference files created by experts

for users to evaluate their results. Additionally, repositories like BioPortal offer potential mappings for their ontologies to assist users. Having a reliable reference is crucial to ensure that the mappings are accurate and reliable [33].

These challenges are major obstacles that alignment systems need to overcome for accurate and reliable ontology alignments. When developing a system for ontology alignment, it's important to keep these challenges in mind to ensure accurate mappings. In the next chapter, we will explore and compare state-of-the-art systems and systems for ontology alignment.

2.3. Ontology Alignment Systems

2.3.1. Overview of State-of-the-Art Systems

With technological advancements over the years, many systems have been developed to handle ontology alignment tasks. These systems are used in various fields like healthcare, agriculture, finance, and many more, making data integration and interoperability possible. There are many different ontology alignment systems available.

Many systems use different methods to achieve ontology alignment. A common approach is using dictionaries such as UMLS or WordNet. The dictionaries along with indexing structures and matching algorithms aim to discover mappings accurately. Systems like LogMap and AML [10] are examples. On the other hand, many systems use machine learning models and algorithm, such as Neural Networks or Large Language Models (LLMs), to add intelligence to the process. Examples include LogMap-ML and Matcha [11]. Each system has its own way of finding mappings and achieving alignment.

In this section, we give an overview of the latest ontology alignment systems. By looking at these systems, we aim to provide a better understanding of the current technologies in this field. The systems mentioned were also used in the 2022 OAEI, demonstrating their capabilities.

1. ***LogMap***

   Developed at the University of Oxford, LogMap is a highly scalable ontology matching system known for its advanced reasoning and diagnosis capabilities. It excels in detecting and repairing unsatisfiable classes., ensuring reliable ontology alignment.

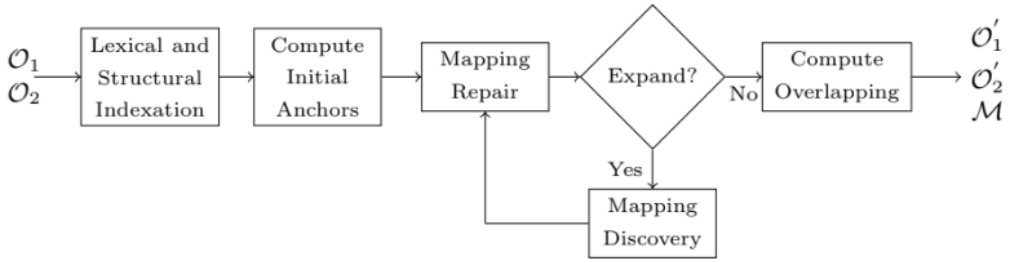

Figure 12. Architecture of LogMap [1].

LogMap has an architecture that consists of a multi-step process to achieve efficient and reliable mappings between ontologies. The first step involves lexical indexation, where the labels of classes and their lexical variations are indexed, using external lexicons like WordNet or UMLS. This is followed by structural indexation, where LogMap uses an interval labelling schema to represent the class hierarchies of the input ontologies. The system then computes initial mappings, called *anchor,* by intersecting the lexical indexes of the ontologies. The core process of LogMap involves an iterative cycle of mapping repair and discovery: the repair step uses a reasoning algorithm to detect and repair unsatisfiable classes, while the discovery step expands contexts for each anchor and matches classes using the ISUB tool [12]. This iterative process continues until no further context expansion occurs, resulting in a set of clean mappings that avoid logical errors. Finally, LogMap estimates the overlap between the input ontologies, providing a fragment that represents the shared content. This helps curators identify any additional mappings that might have been missed.

LogMap has two additional variations: *LogMapLt*, which uses simple string matching, and *LogMapBio*, which uses BioPortal to find ontologies instead of using preselected ones.

2. ***AgreementMakerLight (AML)***

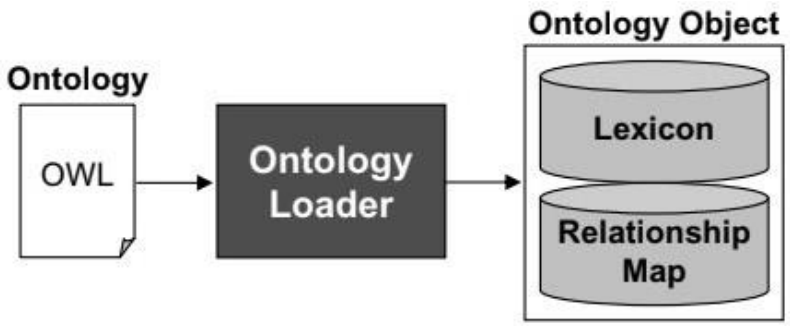

Figure 13. Ontology Loading Module of AML.

The AgreementMakerLight, developed in Java, consists of two core modules: the *ontology loading module* and the *ontology matching module*. The loading module handles input ontologies and external knowledge sources, while the matching module aligns ontology objects using various matching algorithms. Key data structures include the *Lexicon*, which stores lexical information, the *RelationshipMap*, which stores structural information, and the Alignment, which stores mappings between ontologies. AML is designed for flexibility and extensibility, supporting the inclusion of different matching algorithms.

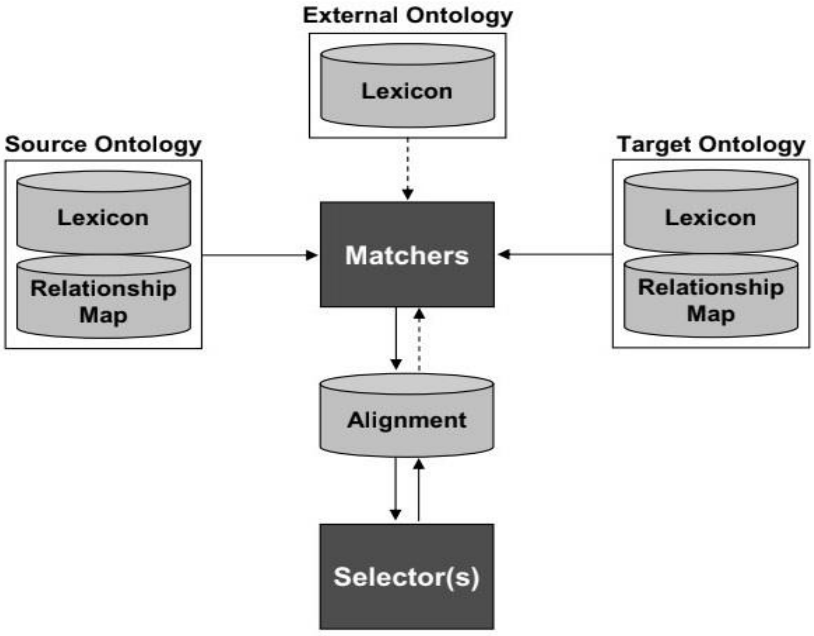

Figure 14.Ontology Matching Module of AML [10].

3. ***Matcha***

Matcha is an underdevelopment ontology matching system designed to address challenges like complex ontology matching, as well as machine-learning-based matching. It builds on the AML system, enhancing its modularity and flexibility. Matcha includes several matching methods; the *Instance-based Class Matcher* uses overlapping individuals to match classes, the *Lexical Matcher* finds full name matches, the *LLM Matcher* calculates similarity between embeddings, the *Mediating XRef Matcher* relies on cross-references, the *String Matcher* measures string similarity, and the *Word Matcher* checks word similarity using a weighted index. For instance matching, it has the *Attribute Matcher* for literal matches, the *Attribute String Matcher* using the ISUB metric, and the *Attribute to Lexicon Matcher* for comparing lexicon entries. Additionally, Matcha introduces a neural-based multilingual translation module that employs an Encoder-Decoder LSTM [13] architecture to translate ontology labels, adding them to the original lexicons. Some features, such as alignment repair and interactive matching, are still under development.

4. ***LSMatch*** **-** Large Scale Ontology Matching System

LSMatch (Large Scale Ontology Matching System) [14] is designed to find correspondences between ontologies based on their lexical properties. The latest version can handle both monolingual and multilingual alignments. It accepts input in any format and loads schemas as RDF graphs. After extracting classes, properties, and instances, it processes the data by stemming, removing stopwords, and normalizing characters.

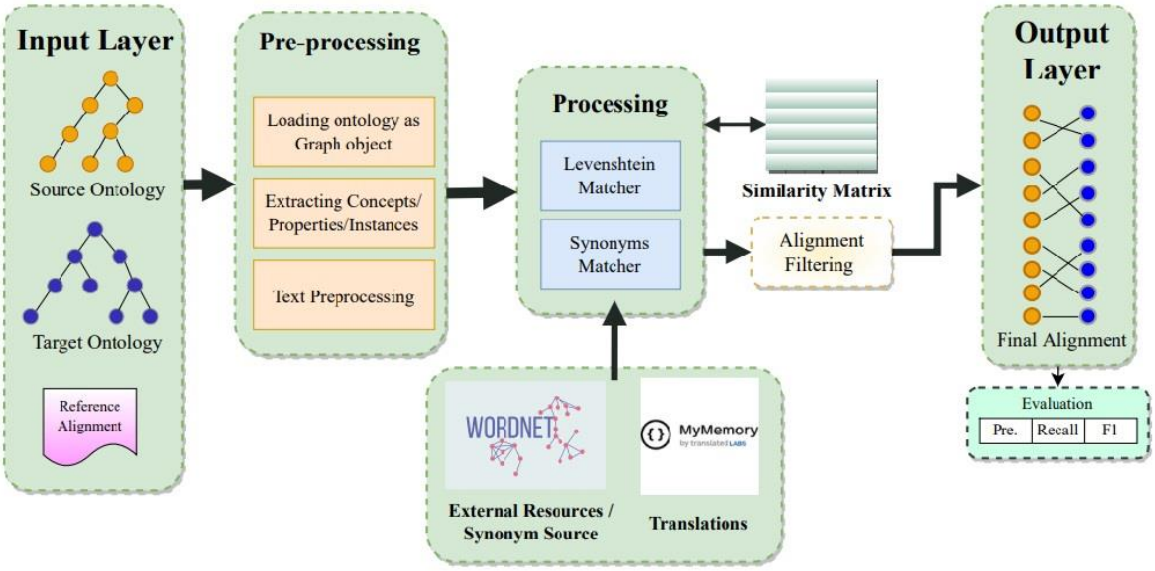

Figure 15. Architecture of LSMatch [14].

LSMatch has several key modules:

- **Levenshtein Matcher:** Measures the minimum changes needed to transform one string into another.
- **Background Knowledge**: Uses a synonym matcher that fetches synonyms from WordNet via Python's NLTK library [15].
- **Synonym Matcher:** Checks for synonyms during execution and retrieves them on the spot if not pre-fetched.
- **Translations:** Utilizes MyMemory's translation memory [16] for reliable and free translations.

For storing alignments, LSMatch uses a dictionary of hashed <key, value> pairs for efficient updates. It keeps alignments with a combined score (from Levenshtein and Synonym matching) of 0.5 or higher, with a final selection threshold of 0.95.

5. ***ATMatcher***

ATMatcher, also known as ATBox, is a system for matching knowledge graphs and aligning ontologies and instances. It includes the alignment repair filter from LogMap, but the matching component isn't fully integrated yet due to time limitations. The system has different processing pipelines for Tbox and Abox matching, which are combined for the final alignment. In the Tbox matching process, the system retrieves classes and properties using Jena methods and removes unnecessary stopwords from labels. It also extracts synonyms from the English Wiktionary through DBnary [17]. A new feature called bounded path matching finds classes within a hierarchy that are already matched, helping to identify potential new correspondences with an average confidence score. The instance matching component remains unchanged, and all correspondences are finalized with a cardinality filter to ensure one-to-one alignment based on the confidence score.

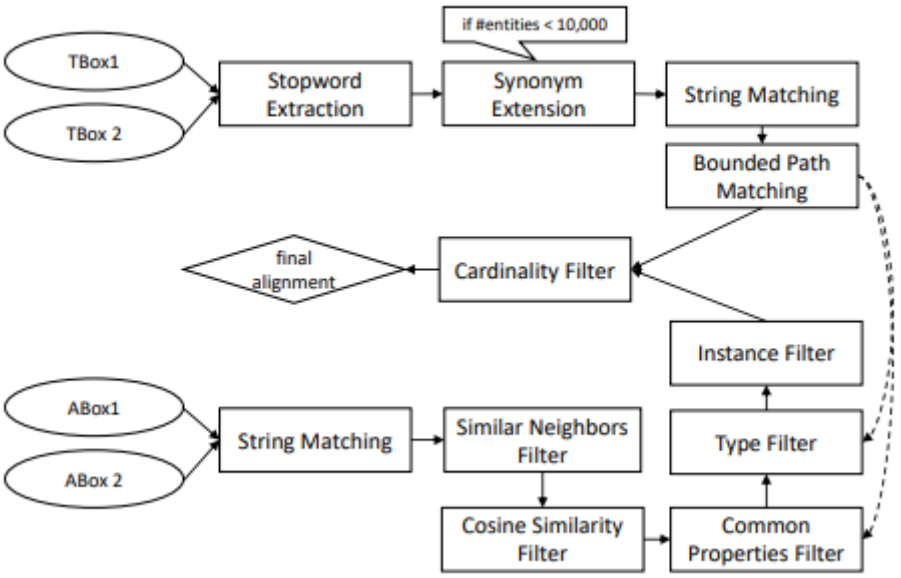

Figure 16. Overview of the ATMatcher [18].

2.3.2. Evaluation Metrics

Each state-of-the-art system has its own way of finding mappings, ranging from using lexicons to applying machine learning techniques. However, it's important to evaluate how efficient each system is. To do this, we need to define key metrics to assess their performance. These metrics are:

Precision. A measure of the ratio of correctly identified correspondences to the total number of found correspondences, as shown in Equation 1. Precision is used to check the degree of correctness of the ontology matching algorithm:

$$Precision = \frac{correct\ correspondences}{total\ found\ correspondences} \qquad (1)$$

For example, based on Figure 12, if our system identified the mappings $\{Vehicle, \equiv, Transport >, \quad Car, \equiv, Automobile > \quad , \quad Engine, \equiv, Motor >, \quad repairs, \equiv, fixes > Mechanic, \equiv, CarOwner >\}$ and the mapping $Mechanic, \equiv, CarOwner >$ is incorrect, then the precision would be:

$$Precision = \frac{4}{6} = 0.667$$

Recall. It measures the number of accurately identified correspondences relative to the total number of expected correspondences, as depicted in Equation 2. Recall serves as a metric to assess how effectively an ontology matching algorithm identifies relevant mappings.

$$Recall = \frac{correct\ correspondences}{expected\ correspondences} \qquad (2)$$

For example, based on Figure 11, if our system identified the mappings $\{< Vehicle, \equiv, Transport >, \quad < Car, \equiv, Automobile > \quad , \quad < Engine, \equiv, Motor >, \quad < Mechanic, \equiv, Technician >, \quad < Owner, \equiv, CarOwner >, \quad < repairs, \equiv, fixes >, \quad , hasEngine, \equiv, hasMotor >\}$, and the expected mappings are eight(8) , then the recall would be:

$$Recall = \frac{7}{8} = 0.875$$

F-Measure. It is a metric used to evaluate the accuracy. It combines both precision and recall into a single measure, providing a balance between the two. The F-measure, as shown in Equation 3, reaches its best value at 1 (perfect precision and recall) and worst at 0 (either precision or recall is zero).

$$F - measure = 2 \times \frac{precision \times recall}{precion + recall} \qquad (3)$$

Using the recall and precision from the previous examples, the F-measure would be:

$$F - measure = 2 \times \frac{0.667 \times 0.875}{0.667 + 0.875} \approx 0.76$$

Total Execution Time. Total execution time is the amount of time a system takes to complete the entire mapping discovery process.

2.3.3. Comparison and Analysis of Alignment Systems

Now that we've explained the metrics, we can analyze and compare the systems: LogMap (including its variations LogMap-Lite and LogMap-Bio), AgreementMakerLight, LSMatch, and

Matcha. For this comparison, we used the anatomy set from OAEI 2022, which includes two ontologies: the Adult Mouse Anatomy ontology and the NCI Thesaurus ontology [19]. The results will focus specifically on the Mouse-NCI pair. The tests were run on a machine with an AMD Ryzen 5 2600 Six-Core Processor (3.40 GHz), 16 GB of RAM, and Ubuntu 24.04.1 LTS. Table 2 presents these results.

Table 3. Performance Metrics of Ontology Alignment Systems.

	Runtime (s)	Size	Precision	Recall	F-measure
LogMap	6	1402	0.915	0.848	0.880
LogMap-Lite	2	1147	0.962	0.728	0.828
LogMap-Bio	1194	1596	0.873	0.919	0.895
AML	29	1471	0.956	0.927	0.941
Matcha	22	1482	0.951	0.930	0.941
LSMatch	17	1009	0.952	0.634	0.761
ATMatcher	151	1037	0.978	0.669	0.794

Among the various ontology alignment systems evaluated, LogMap-Lite stands out for its runtime efficiency, completing its task in just 2 seconds, making it the fastest among the systems. LogMap also demonstrates impressive speed with a runtime of 6 seconds. Matcha and LSMatch offer relatively quick runtimes of 22 and 17 seconds respectively. AML is slower at 29 seconds but offers high accuracy and recall. ATMatcher follows with a runtime of 151 seconds. In contrast, LogMap-Bio is the slowest among the systems, taking a total of 1194 seconds.

In terms of mappings found, LogMap-Bio has the highest number of mapping found, with 1,596. However, while LogMap-Bio finds the most, it takes significantly more time compared to the rest systems. Matcha follows with 1,482 correspondences found, while AML has 1,471. LogMap identifies 1,404 correspondences, performing well with its short execution time LogMap-Lite and ATMatcher find 1,147 and 1,037 mappings, respectively. LSMatch has the smallest number with 1009 found.

The results indicate that ATMatcher achieves the highest precision at 0.978, followed by LogMap-Lite at 0.962. AML achieves precision at 0.956 and LSMatch at 0.952. Matcha demonstrates a strong precision of 0.951, while LogMap and LogMap-Bio exhibit slightly lower values at 0.915 and 0.873, respectively. Overall, all systems demonstrate high precision, indicating their effectiveness.

Recall measures how well a system finds relevant correspondences. AML and Matcha have high recall scores of 0.927 and 0.93, respectively. LogMapio-Bio is next with 0.919, followed by LogMap at 0.848. LogMap-Lite has a lower score of 0.728, while ATMatcher has a score of 0.669. LSMatch has the lowest recall score at 0.634

The F-measure combines precision and recall. AML and Matcha both have the highest F-measure of 0.941. LogMap-Bio follows with 0.895, while LogMap has 0.880. LogMap-Lite also

performs well with a score of 0.828. ATMatcher has F-measure at 0.794, while LSMatch has the lowest F-measure at 0.761.

All the systems mentioned have good results. Some systems, like LogMap-Bio, trade speed for higher accuracy, while others, like Matcha, balance speed and performance. All systems have good accuracy, scoring above 82%. In terms of recall, some systems show high recall, meaning they identify mappings correctly, while others have lower recall, indicating difficulties in discovering them. High precision is important, but having a good recall value is also crucial. Most systems achieve a high F-measure, with scores above 75%. Overall, each system performs well, using its own methods to uncover mappings.

3. OntoLink

Ontology alignment remains a challenging and demanding task that many systems aim to optimize it by providing different approaches. In this chapter, we introduce OntoLink, a new system designed for ontology alignment. OntoLink can handle both small and large ontologies to produce accurate mappings. Its architecture is designed to not only find reliable mappings but also resolve issues such as inconsistency and unsatisfiability. It has been developed using Java.

In its architecture, OntoLink has employed some components from LogMap, that help in the alignment process. Specifically, it utilized LogMap's string matching that is used to discover mappings. Additionally, the structures where classes, properties and mappings are stored are put in use. Moreover, OntoLink adopts LogMap's iterative process where new mappings are found, and the respective structures are updated. These features from LogMap help shape OntoLink's architecture.

OntoLink uses Manchester syntax to find additional mappings, by evaluating the definitions of complex classes, under the scope of previously detected correspondences. Furthermore, OntoLink, integrates the HermiT reasoner in its architecture, to infer new correspondences and detect inconsistencies triggered by false correspondences. HermiT helps find equivalent classes and resolve inconsistencies or unsatisfiable classes. These processes will be explained in more detail in the following sections

In the next sections, we will first explain the algorithm used in OntoLink, showing the steps it takes to discover mappings. Then, we will look at the architecture of OntoLink and how its components work together to achieve its goals.

3.1 The Algorithm

OntoLink's algorithm is designed to enhance the ontology alignment process. It follows a series of structured steps to identify mappings. Figure 17 shows the algorithm

Algorithm 1 OntoLink's algorithm

```
1:  Input: Ontology O1, Ontology O2
2:  Output: Set of Correspondences M
3:  M ← detectMappingFromImportedTerms(O1, O2)
4:  while new mappings are detected do
5:      if M does not contain (mappings) then
6:          M.add(mappings)
7:      end if
8:      for ET_O1, ET_O2 in O1 and O2 do
9:          if terms of ET_O1, ET_O2 are contained in M then
10:             M.add(ET_O1.namedClassO1 , ET_O2.namedClassO2)
11:         end if
12:     end for
13:     mergedOntology ← Initialize an empty ontology
14:     for axiom_O1, axiom_O2 in O1 and O2 do
15:         A ← applySubstitutionRule(axiom_O1, axiom_O2)
16:         mergedOntology.addAxioms(A)
17:     end for
18:     apply HermitT reasoner on mergedOntology
19:     while inconsistencies are detected do
20:         Get the explanations from the reasoning
21:         for CC, RR in explanations do
22:             Freq ← calculateFrequencies(CC, RR)
23:             mergedOntology.remove(CC, RR, Freq)
24:         end for
25:     end while
26: end while
```

Figure 17. OntoLink's algorithm

OntoLink begins its process by having as input two ontologies, O_1 and O_2, and aims to produce the set of mappings M. The initial step of the algorithm, involves applying the function ***detectMappingFromImportedTerms(O1,O2)***. This function is straightforward, and is presented in Figure 18.

Algorithm 2 detectMappingsFromImportedTerms()

```
Input: Ontology O1, Ontology O2
Output: Set of Mappings M
for class_O1, class_O2 in O1 and O2 do
    if class_O1.URI == class_O2.URI then
        M.add(class_O1.URI, class_O2.URI)
    end if
end for
for objProp_O1, objProp_O2 in O1 and O2 do
    if objProp_O1.URI == objProp_O2.URI then
        M.add(objProp_O1.URI, objProp_O2.URI)
    end if
end for
```

Figure 18. detectMappingFromImportedTerms() function.

Many ontologies reuse classes and properties from other ontologies. This means that two ontologies might share the same imported classes or properties from a third ontology. That is why, this function works by comparing the URIs of classes and object properties in the input ontologies. If the URIs match, are the same, it considers this an equivalence correspondence. The set M then is updated with the found mappings. Although this may seem simple, it plays a significant role in discovering additional mappings later on. Many existing systems struggle to identify these mappings because they rely on label comparison. This approach ensures that the same resources are recognized across ontologies. It is also efficient, since it does not require a lot of time and computational power.

Next, OntoLink uses features from LogMap. Specifically, LogMap contains an iterative process in its architecture, where new mappings are discovered through string matching using the UMLS lexicon. During each iteration, OntoLink checks if the discovered correspondences are already in the set M. If not, it adds them to M. Moreover, OntoLink also uses LogMap's structures that store mappings, classes, and properties.

Following, our system uses Manchester syntax to find more mappings. With the use of it, we can represent ontology axioms and restrictions in simple terms. By utilizing the *EquivalentTo* axiom, we can uncover additional mappings. Specifically, if we have two classes to compare, and their corresponding anonymous classes are equivalent, then that means that those two named classes are also equivalent. In this case, we identify them as an equivalent correspondence. For example, consider the two *EquivalentTo* axioms below:

- ***Ontology*** O_1**:** 'molecular label' *EquivalentTo* 'molecular entity' and ('has role' some 'molecular label role')
- ***Ontology*** O_2**:** 'individual molecular label' *EquivalentTo* 'molecular body' and ('has a role' some 'molecular label role')

If we only keep the anonymous classes (the ones that follow *EquivalentTo*) we have:

- ***Ontology*** O_1**:** 'molecular entity' and ('has role' some 'molecular label role')
- ***Ontology*** O_2**:** 'molecular body' and ('has a role' some 'molecular label role')

As mentioned earlier, the set M holds all discovered correspondence, meaning it represents the mappings. OntoLink then, with the use of the set M, compares each term from the anonymous classes in O_1 with those in O_2. If the two comparing terms are contained in M, meaning they are a mapped, the comparison is valid. Else if they are not contained in M, meaning they are not a mapped, the comparison is invalid. Only when all terms in the two anonymous classes are found in M then both the anonymous classes are considered equivalent. If the anonymous classes are equivalent, their named classes are also considered equivalent. Therefore, we have a mapping.

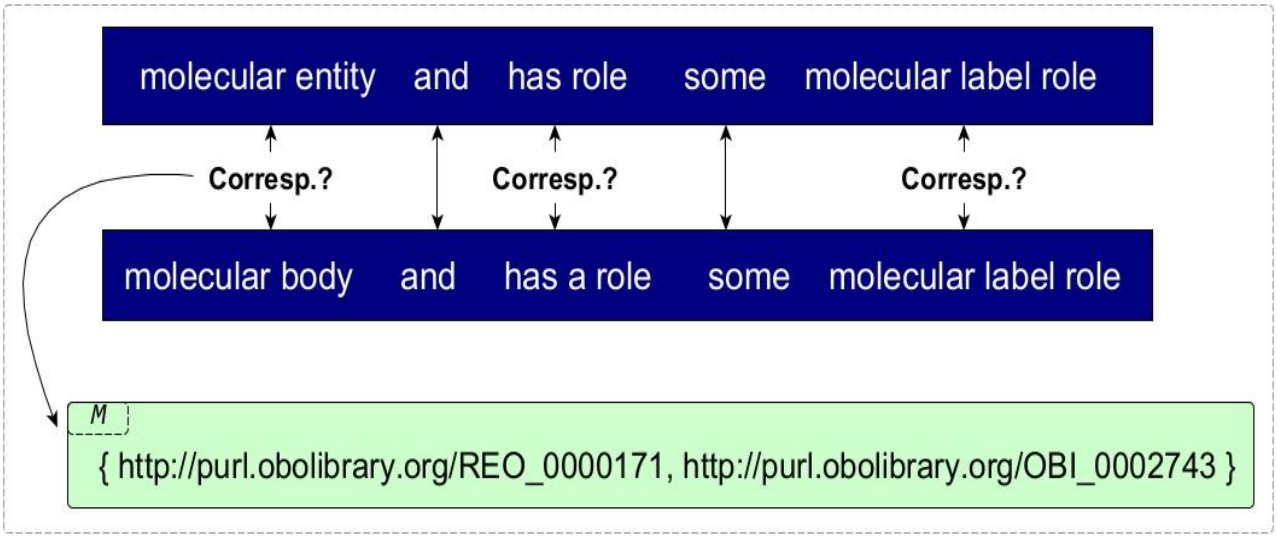

Figure 19. Comparison of Anonymous Classes.

In our example, OntoLink compares two anonymous classes, as shown in Figure 19, to check if each term is a match. It looks at whether *molecular entity* and *molecular body* are a correspondence, as well as *has role* and *molecular label role*, using a set M. Therefore, *molecular entity* and *molecular body* are considered correspondence since they are in M. Logical operators like *and* and *some*, which are fundamental terms for ontologies, are treated with a string matching. This process continues until all terms are checked. In the end, OntoLink identifies that the two anonymous classes as equivalent. The named classes *molecular label* and *individual molecular label* are then considered equivalent, resulting in a mapping. This method systematically helps us to discover additional mappings.

In the next step, our system initializes an empty ontology referred to as *mergedOntology*. It processes the axioms from O_1 and O_2 by applying a substitution rule, which creates new axioms. Figure 20 shows how this substitution rule works.

Algorithm 3 The substitution rule

1: **Input:** Axioms $axiom_{O1}, axiom_{O2}$, mappings $M = \{m_1, \ldots, m_n\}$
2: **Output:** Axioms $axiom'_{O1}, axiom'_{O2}$
3: **for** $axiom_{O1}, axiom_{O2}$ in $O1$ and $O2$ **do**
4: **if** $axiom_{O1}$ and $axiom_{O2}$ contain classes from M **then**
5: Rename classes into CC_i
6: **end if**
7: **if** $axiom_{O1}$ and $axiom_{O2}$ contain properties from M **then**
8: Rename properties into RR_i
9: **end if**
10: $axiom'_{O1} \leftarrow$ Update $axiom_{O1}$
11: $axiom'_{O2} \leftarrow$ Update $axiom_{O2}$
12: **end for**

Figure 20. Substitution rule.

This algorithm for the substitution rule is straightforward. For every known class mapping, such as $O_1{:}\,C_1 = O_2{:}\,C_2$, the system stores them in *mergedOntology* as CC_i, where i presents a unique identification. Similarly, for every know object property mapping, such as $O_1{:}\,R_1 = O_2{:}\,R_2$, it stores them in *mergedOntology* as RR_j where j presents a unique identification as well. In this way, we treat a mapping between two classes or properties as new, single class or property. For example, the class mapping from Figure 19 would be renamed and stored with the label *CC_224* and a URI like *http://purl.obolibrary.org/CC_224.* Any axiom involving *molecular entity* and *molecular body* will have those terms replaced with *CC_224*. It's clear that the original URIs will also be replaced. In this way, we have created a new ontology that we can use and process.

We can use HermiT with the *mergedOntology* to find any inconsistencies and unsatisfiable classes. HermiT helps us identify and fix these issues, and it can also discover new equivalent classes, leading to additional mappings. We take advantage of the explanations provided by HermiT. Here's how this approach works: for each *inconsistency* in the *mergedOntology*, if either CC_i or RR_j is involved, it removes them and the related axioms from the ontology. Therefore, it has discarded a mapping. OntoLink then stores the removed mapping in a structure, called *Freq* as shown in the algorithm in line 22, having entries that are formatted as $< Mapping, Frequency >$. The mapping is stored as the key, and the number of times it occurred is stored as the value, representing its frequency. After removing all the involved, we run HermiT again. If new inconsistencies arise, we remove those as well and update the frequency set. This process continues until there are no more inconsistencies in the *mergedOntology*. With this approach, we aim not only to resolve inconsistencies and unsatisfiable classes but also to help HermiT find new equivalences and mappings.

In summary, this approach aims to find additional mappings and resolve the inconsistencies in the ontologies. For an alignment task, it is important to first check for inconsistencies and then look for unsatisfiable classes. Additionally, we effectively use the discovered object property mappings. We have implemented a sound approach that uncovers more mappings

After completing the reasoning task, OntoLink starts over to find new mappings. It's important to note that any mappings that caused problems in the *mergedOntology* are removed from

the set M in the next round. This helps minimize the inconsistencies. The algorithm ends when no new mappings are found during the iterative process, meaning that all correspondences between the input ontologies have been examined. Finally, a set of mappings is produced.

3.2. The Architecture of OntoLink

Now that we've explained the OntoLink algorithm, it's important to describe its structure. In this chapter, we will break down OntoLink's architecture, explaining its components, their roles, and how they work together to find mappings for ontology alignment. This analysis is essential for a better understanding of our system. Figure 21 shows an overview of OntoLink.

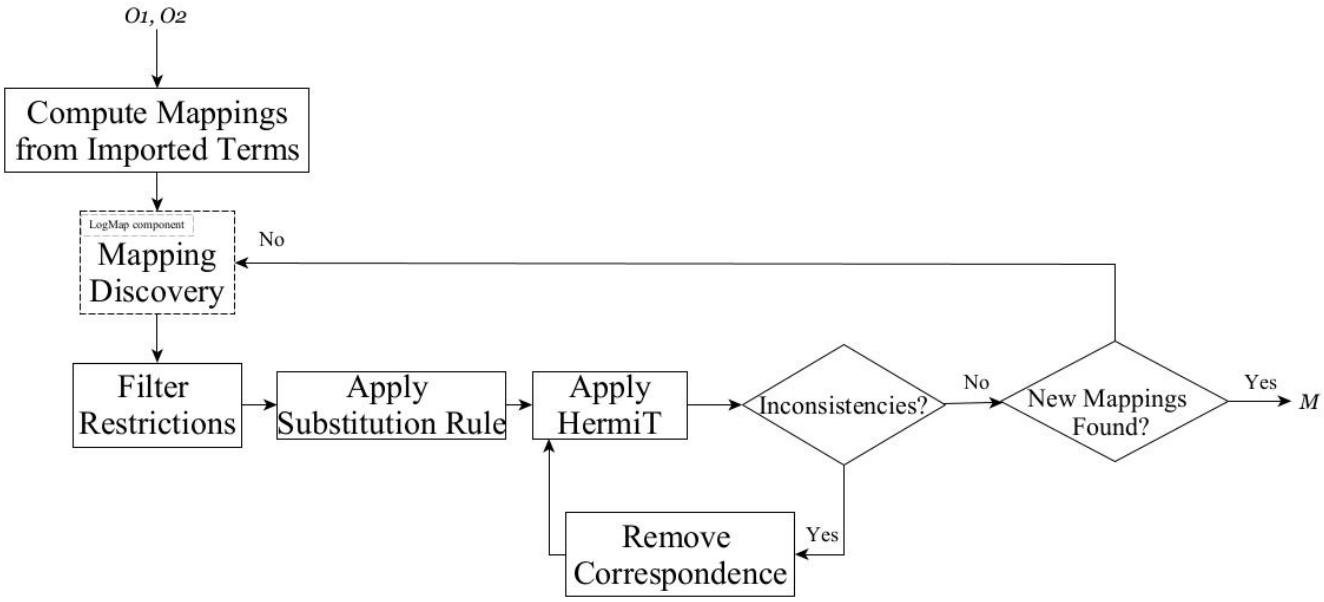

Figure 21. Overview of OntoLink.

Following, we will give an overview of the main steps OntoLink performs, as shown in Figure 21.

1. *Compute Mappings from Imported Terms*. After parsing the two input ontologies, O_1 and O_2, the first step is to identify mappings from imported terms. Many ontologies reuse terms from other ontologies, so there is a possibility that O_1 and O_2 share the same terms from a third ontology. Identifying these mappings is simple: we just check if the URIs of classes and object properties from both ontologies are the same, meaning they refer to same resource. This is the *detectMappingFromImportedTerms(),* as shown in Figure 18. The mappings are then stored in a set structure used from LogMap's architecture.

   Although this step seems simple, it's important for our system. Many existing tools miss these types of mappings because they rely only on string matching between labels. This process doesn't slow down our system, as it takes a short period of time to compare URIs. These mappings are the first ones discovered by OntoLink and will be used as input for the next steps of the system.

2. *Mapping Discovery.* For the next step, we utilized some parts of LogMap. LogMap has an iterative process to discover mappings. It begins by finding *anchor mappings*, which are the first mappings it has identified. We boost this process by giving as input the mappings we found in the earlier step, making the discovery of anchor mappings more effective. This helps us find more mappings. Once the discovery has finished, we check if new mappings are found. If there are, we store them in the structure that holds the mappings, and will be used in the next step of OntoLink.

3. *Filter Restrictions.* In this step, we compare the terms in anonymous class to find more mappings, as mentioned earlier. It's important to use the mappings we've already found. We convert the *EquivalentTo* axioms in simpler format using the Manchester syntax. The named class appears on the left side and the anonymous class on the right. In more technical aspect, we store the axioms in an array. In the first position we store the named classes and in the second the anonymous classes. We used an array for an easier access and performance. Then, we compare the terms of the anonymous classes using our mapping structure, as shown in lines 8-12 in OntoLink's algorithm. This helps us discover more mappings or mappings that need several iterations to be found. In the end, we get an enriched set of mappings for the next step of OntoLink.

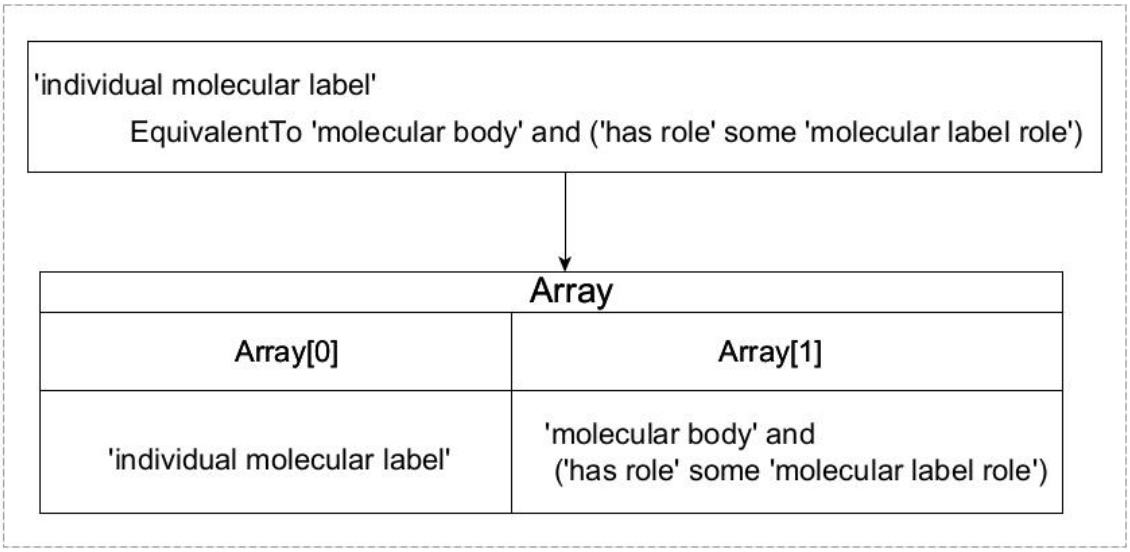

Figure 22. The structure of the array.

4. *Apply Substitution Rule.* Having a rich and larger set of mappings helps in this step. As shown in the algorithm, we use substitution rules in our process. All the discovered mappings between classes and properties are given new unique URIs and labels. As shown in the algorithm of Figure 17, we now treat these mappings as a single class or property. v This allows us to create a new ontology, called *mergedOntology*. Figure 23 shows part of the *mergedOntology* in Protege.

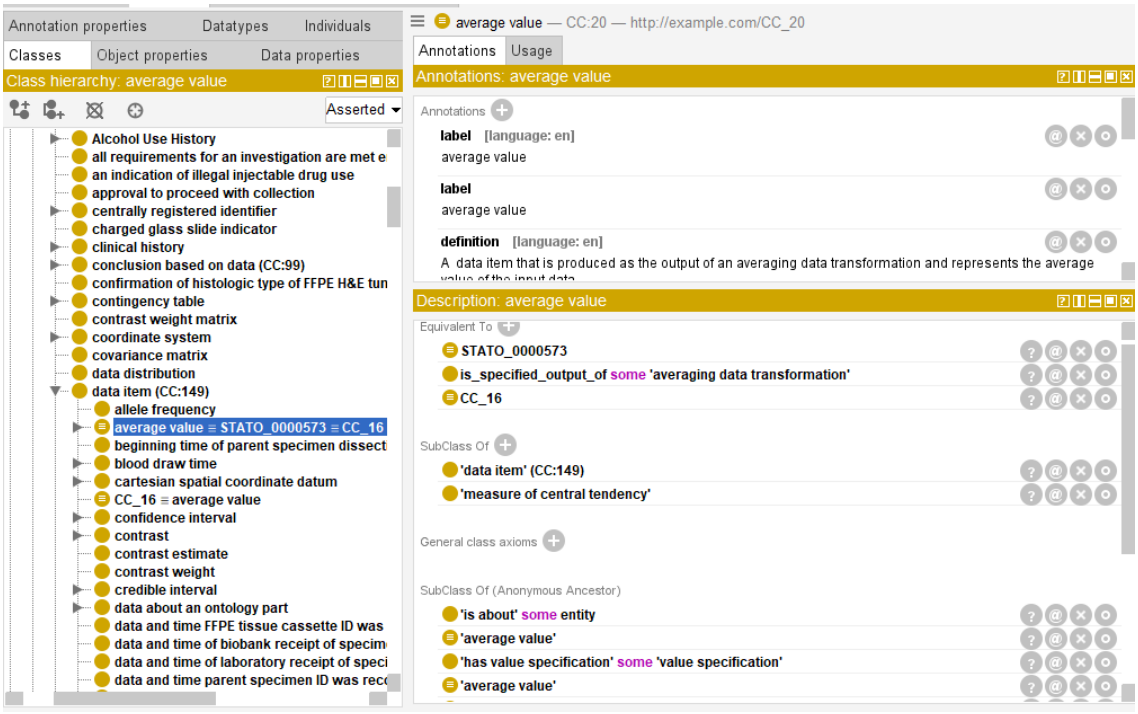

Figure 23. Part of the mergedOntology.

In this ontology we can apply reasoning tasks to uncover more information. We can find out more equivalences among classes and object properties, or identify inconsistencies that may occur.

5. *Apply HermiT & Remove Axioms.* Once we create a new ontology, we can apply a reasoner to it. We chose the *HermiT* reasoner because it is widely used and can handle ontologies of any size efficiently. When we run *HermiT* on the *mergedOntology*, we aim to find any inconsistencies, unsatisfiable classes, or new equivalences between classes and object properties. If *HermiT* finds equivalences, we treat them as new mappings. We check if these mappings are already in our set. If they're not, we add them and we identify them as new mappings

   If *HermiT* finds inconsistencies or unsatisfiable classes, it provides explanations showing in which axioms the issue has occurred. We then check if any CC or RR , meaning our mappings, is involved. If there are, we make the assumption that they might be causing the issue. The axiom, in which CC or RR are involved, are removed from the mergedOntology, and the mappings are stored in a *map* structure. The key is the mapping itself and the value is how many times it has appeared causing an issue (*frequency*). For example, in Figure 24 we can see correspondences removed because they create inconsistency problems. We can see the class *CC_137* has a frequency of 3, meaning it participates in three (3) different conflicts. This process continues until no more inconsistencies or unsatisfiable classes are found.

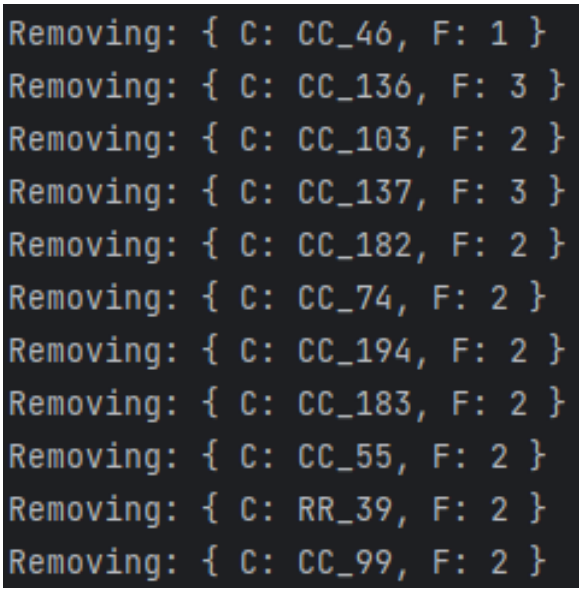

Figure 24. Removed mapping.

Finally, OntoLink checks if new mappings are discovered. If it finds any, it goes through another iteration, repeating all the previous steps. If no new mappings are found, OntoLink finishes its process and produces a final mapping.

This is the architecture of OntoLink. While it may seem straightforward, OntoLink is designed to improve the ontology alignment process. Its components work together effectively, allowing it to handle both large and small ontologies and generate a comprehensive set of mappings.

3.3. Application of OntoLink

After explaining the algorithm and architecture of OntoLink, it's important to demonstrate its application. This will help make its usage clearer for users. In this section, we will showcase how OntoLink works using a pair of ontologies.

We chose to demonstrate OntoLink's application using two ontologies. The first ontology is the Ontology for Biobanking (OBIB) [20], designed to represent and model information about biobank repositories and biobanking administration. The second is the Ontology of Minimum Information About Biobank data Sharing (OMIABIS) [21], which is used to describe biobanks, research on samples and associated data. We will show how OntoLink works using the OBIB-OMIABIS pair. Therefore, the first step in OntoLink's process is to gain information about the characteristics of the input ontologies.

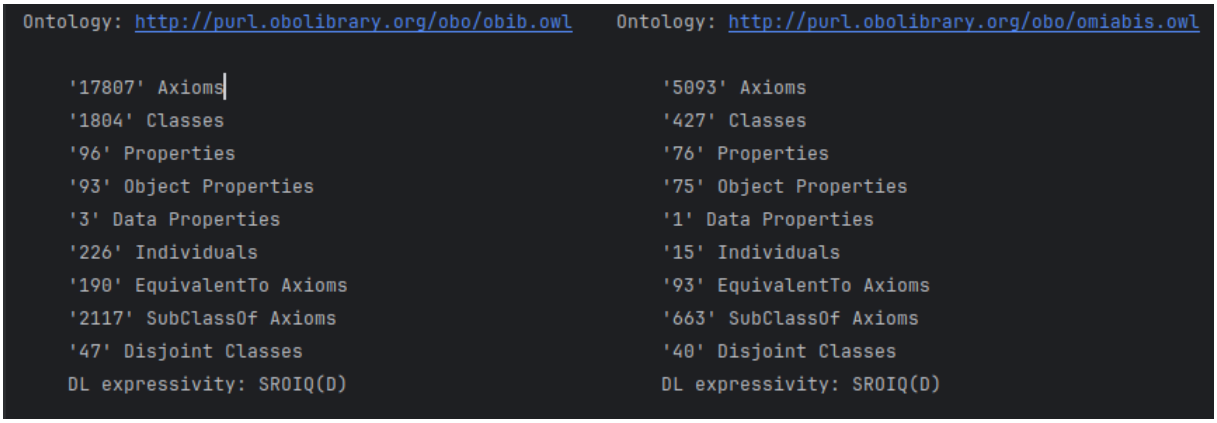

Figure 25. Attributes of the input ontologies.

As shown in Figure 25, OntoLink has extracted key information from both input ontologies. It found that OBIB contains 17,807 axioms and 1,804 classes, while OMIABIS has 5,093 axioms and 427 classes. Additionally, OntoLink identified the number of *EquivalentTo* axioms in each ontology, with OBIB having 190 and SLSO having 93. This step is important for OntoLink's process, as it gives users a better understanding for the mapping discover process. For instance, if both ontologies have many *EquivalentTo* axioms, OntoLink can identify a large number of mappings by utilizing these axioms.

Next, OntoLink checks if any mappings between imported terms are missing. In our case, no mappings of this type were missing, so OntoLink spent almost no time on this step. Afterward, OntoLink attempts to find mappings using the EquivalentTo axioms and with the help of Manchester syntax, as explained earlier. By leveraging these axioms, OntoLink identified 124 correspondences. Some were between the same terms, while others involved different terms. Out of these, 54 were missing from its storage structure, so it added 54 new mappings. Figure 26 shows the output from this process as displayed in the console.

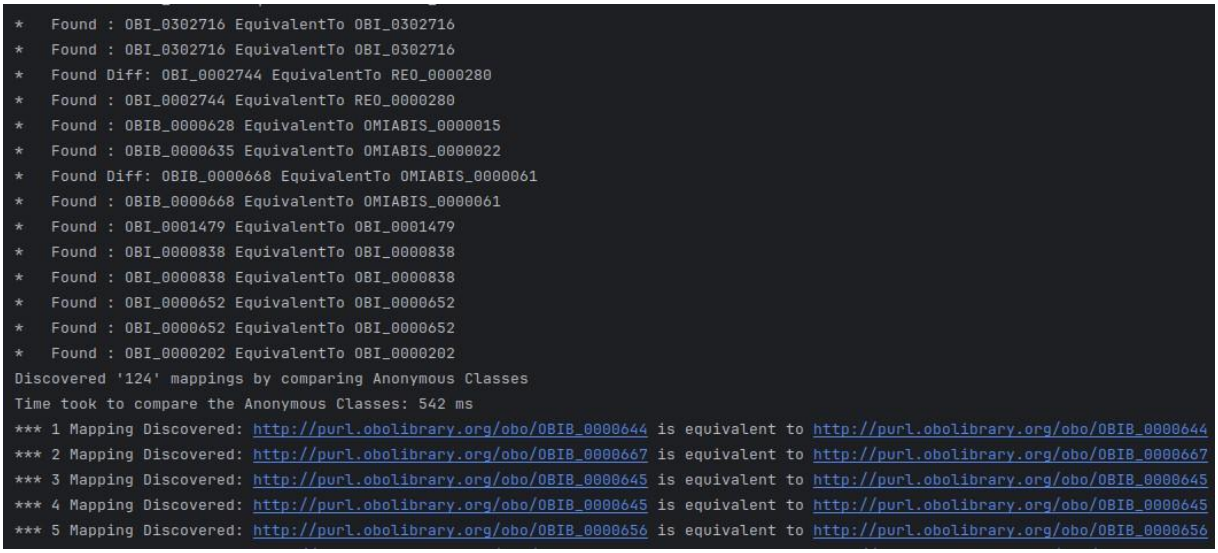

Figure 26. Mappings found.

Next, OntoLink uses the substitution rule (as shown in Figure 20) to create the mergedOntology. It stores the discovered mappings in the mergedOntology. In Figure 29, we can see the creation of the mergedOntology as displayed in the console. Since it is formed by merging the two input ontologies, the mergedOntology has attributes like classes and axioms from both ontologies. At this stage, the user can access both the mappings and the mergedOntology, which are stored locally.

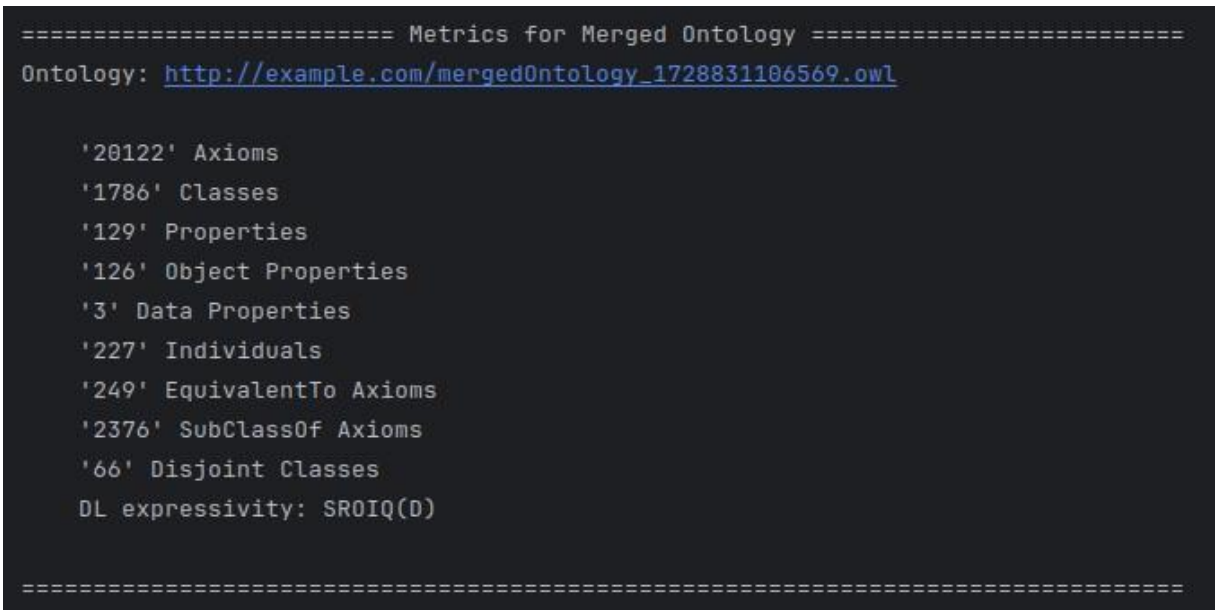

Figure 27. The mergedOntology.

After creating the mergedOntology, we can discover additional mappings and check the consistency using the HermiT reasoner. By applying HermiT to the mergedOntology, we identify equivalences between classes and object properties. OntoLink found a total of 1385 equivalences, which are considered equivalent correspondences. Some of these may have been found earlier, however all new correspondences are displayed in the console. In Figure 28, we found new equivalences.

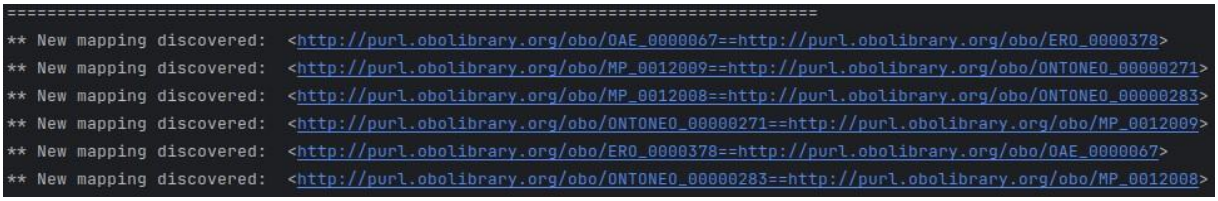

Figure 28. Discovery of new mappings.

At the same time, we check the consistency of the mergedOntology. If it is inconsistent, HermiT will alert us, and a warning message will appear in the console. In our case example, HermiT detected inconsistencies. Using the explanation provided by HermiT, we can find the axioms the inconsistency occurred and check if a mapping (CC_i and RR_j) is involved. If there are mappings involved, we discard them and the axioms that are involved and run HermiT again. An example of this process is shown in Figure 26, where mappings are removed. This process continues until no more inconsistencies are found.

Once the reasoning task is complete, OntoLink starts over to identify more mappings. It's important to note that any mappings that caused issues are removed from the structure where all the mappings are stored. Therefore, OntoLink will try to find additional mappings by repeating the same steps. If no new mappings are found, the process terminates.

4. Experimental Evaluation

In this chapter, we will showcase OntoLink by aligning two pairs of ontologies: NGBO-STATO and PSDO-STATO. First, we will present the setup used for its application and then we will compare and evaluate OntoLink's performance against other existing systems. Through this evaluation, we aim to show OntoLink's efficiency and effectiveness.

4.1. System Evaluation Setup

Evaluating the performance of an ontology alignment system is crucial. Campaigns like OAEI provide reference alignments, and there are also repositories with that offer mappings among ontologies that can be used for evaluation. In this section, we will explain the setup used to evaluate OntoLink's performance, which includes using specific pair of ontologies and a repository for testing the results.

To evaluate OntoLink's performance, we used publicly available ontologies. Specifically, we worked with the *Performance Summary Display Ontology (PSDO),* the *Statistics Ontology (STATO)*, and *Next Generation Biobanking Ontology (NGBO)* [22, 23, 24]. The pairs we tested in our system were *NGBO-PSDO* and *STATO-PSDO*. Below are the details of the ontologies we used:

1. *Performance Summary Display Ontology - PSDO*
The PSDO is a lightweight ontology commonly used in healthcare domain. Its purpose is to manage and explain visualizations of clinical performance and their outcomes in healthcare quality improvement settings. The PSDO has the following characteristics:

- 424 Axioms
- 96 Classes
- 10 Properties
  10 Object Properties
- 0 Data Properties
- 0 Individuals
- 6 EquivalentTo Axioms
- 0 Disjoint Classes

Additionally, PSDO uses ALC expressivity [25], a fundamental logic that supports conjunction, disjunction and negation, along with quantifiers such as *some* and *only*. ALC provides the foundational means to express basic relationships and constraints within an ontology

2. *Statistics Ontology - STATO*

STATO is a general-purpose ontology that provides a vocabulary with statistical terms. It includes classes like mean, average value, and many more. Its goal is to help users describe situations where statistics are needed. STATO has the following features:

- 11,678 Axioms
- 889 Classes
- 71 Properties
- 64 Object Properties
- 7 Data Properties
- 14 Individuals
- 104 *EquivalentTo* Axioms
- 9 Disjoint Classes

STATO has SROIQ(D) expressivity [26], meaning it combines features from the SROIQ language with the ability to include data values, like integers and strings. This allows to represent complex concepts in ontologies, including class hierarchies, relationships between properties, and rules that involve both individuals and data values.

3. *Next Generation Biobanking Ontology – NGBO*

The NGBO is commonly used in the biomedical field. Its purpose is to represent and manage information about omics digital assets in biobanks. The NGBO consists of the following characteristics:

- 11,814 Axioms
- 1,516 Classes
- 133 Properties
- 129 Object Properties
- 4 Data Properties
- 0 Individuals
- 6 *EquivalentTo* Axioms
- 25 Disjoint Classes

Moreover, NGBO uses SRIQ expressivity [27], which means it can represent complex relationships in ontologies. It includes features like inverse properties, cardinality restrictions, and transitive roles, allowing for more detailed and structured knowledge representation.

To provide a strong basis for our evaluations, we used BioPortal, one of the largest and most used ontology repositories available today. BioPortal provides mappings between different ontologies, including those we used in our study. By using these established mappings as our reference point, we could accurately present and evaluate OntoLink's performance. In this way, we can calculate important metrics like precision, recall, and F-score, giving us a solid standard for comparison.

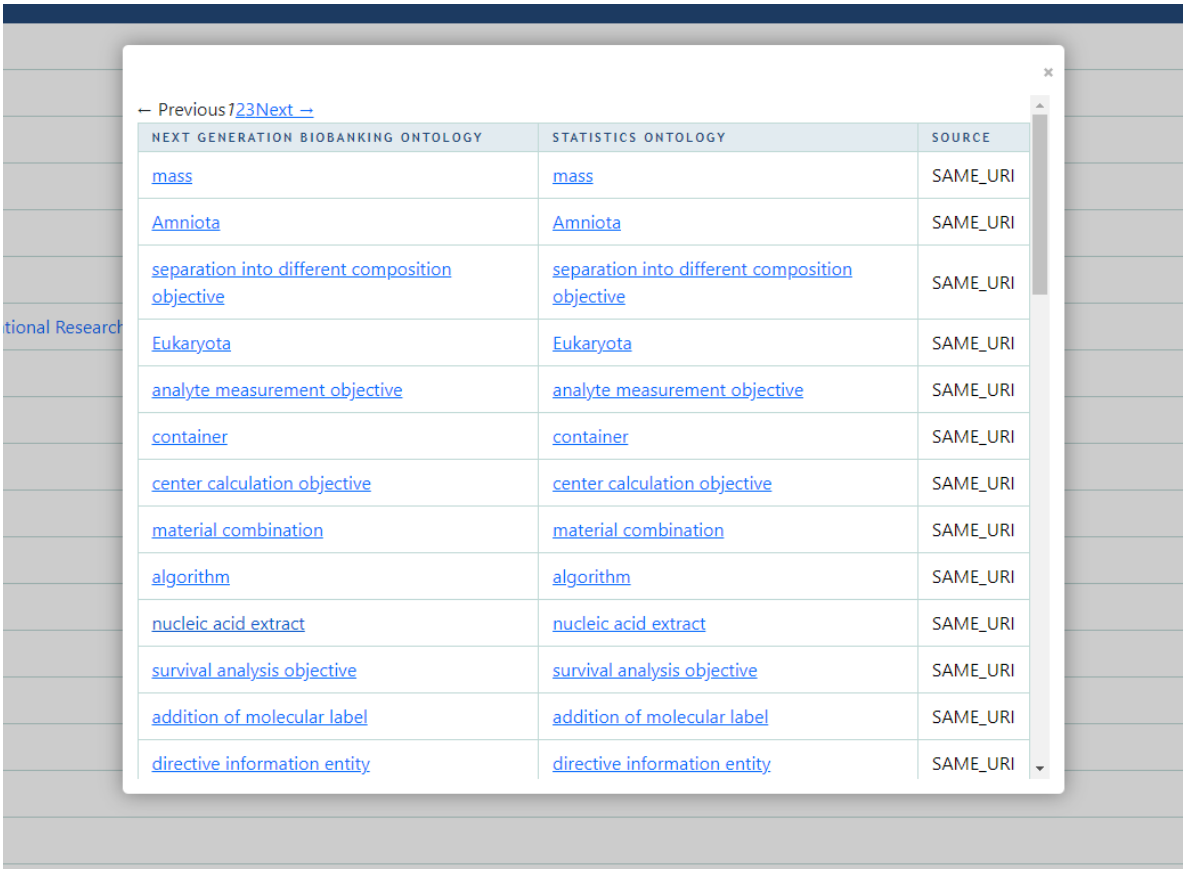

Figure 29. Mappings from BioPortal.

With this setup, we will demonstrate OntoLink's capabilities. We will showcase its performance in identifying mappings and compare it to current ontology alignment systems

4.2. Experimental Results

In this section, we will demonstrate how OntoLink performs compared to other state-of-the-art systems. We will evaluate its performance against LogMap, LogMap-Lite, LogMap-Bio, LSMatch, Match, and ATMatcher, which were introduced in section 2.2.1. For this evaluation, we will use the pairs of ontologies *PSDO-STATO* and *NGBO-STATO* and assess the results based on the mappings provided by the BioPortal repository.

We will focus on key metrics such as precision, recall, and F-measure to analyze our system in detail. These metrics are described in section 2.3.2. We will also focus in size of alignment produced by the systems. We hope that this analysis will provide valuable insights into OntoLink's effectiveness and reliability in uncovering mappings.

Table 4. Results PSDO - STATO.

	Execution Time (s)	Size	Precision	Recall	F-measure
LogMap	6.56	25	1.000	0.142	0.250
LogMap-Lite	3.5	18	1.000	0.125	0.223
LogMap-Bio	58	0	0.000	0.000	0.000
AML	7	17	1.000	0.125	0.223
LSMatch	5	7	0.000	0.000	0.000
Matcha	31	2	0.000	0.000	0.000
ATMatcher	21	9	0.110	0.125	0.118
OntoLink	61.5	38	1.000	0.714	0.834

Table 4 shows the results of the state-of-the-art systems using the pair of ontologies *PSDO-STATO*. LogMap-Lite had the fastest execution time at 3.5 seconds, followed by LSMatch at 5 seconds. LogMap completed in 6.56 seconds, and AML was close behind at 7 seconds. ATMatcher and Matcha took longer, at 21 seconds and 31 seconds, respectively. LogMap-Bio finished in 58 seconds, while OntoLink was slightly slower at 61.5 seconds.

In terms of the number of mappings found, OntoLink discovered the most, with 38 in total. LogMap followed with a set of 25 found, and LogMap-Lite found 18. AML identified 17 correspondences and ATMatcher produced. Matcha found only 2 correspondences between the two ontologies, and LogMap-Bio produced an empty set of mappings.

Precision measures the correct mappings found compared to the total mappings retrieved by the systems. Most systems, including OntoLink, LogMap, LogMap-Lite, and AML, achieved a perfect precision of 1.000, meaning all their retrieved mappings were correct. ATMatcher had a lower precision of 0.110. LogMap-Bio, LSMatch, and Matcha had a precision of 0.000.

Recall measures the correctly found mappings over the total number of expected mappings. OntoLink had the highest recall at 0.714. Next was LogMap with 0.142. LogMap-Lite, AML, and ATMatcher followed with 0.125. LogMap-Bio, LSMatch, and Matcha had the lowest recall, with a value of 0.000.

In terms of F-measure, OntoLink achieved the highest F-measure of 0.834, followed by LogMap at 0.250. LogMap-Lite, AML, and ATMatcher all had an F-measure of 0.223. LogMap-Bio, LSMatch, and Matcha had the lowest F-measure, each with a score of 0.000.

In the pair of ontologies *PSDO-STATO*, OntoLink took the longest time to complete, taking nearly one minute. However, this extra time allowed it to discover the mappings among existing ontology alignment systems, and achieve the highest evaluation metrics. The additional processing time helped the system perform better.

Table 5. Results NGBO - STATO.

	Execution Time (s)	Size	Precision	Recall	F-measure
LogMap	59	183	0.110	0.250	0.150
LogMap-Lite	7	19	0.157	1.000	0.270
LogMap-Bio	392	0	0.000	0.000	0.000
AML	12	5	0.200	0.200	0.250
LSMatch	9	203	0.375	1.000	0.545
Matcha	35	13	0.000	0.000	0.000
ATMatcher	56	147	0.125	0.334	0.182
OntoLink	252	195	0.538	1.000	0.700

Table 5 shows the evaluating results of existing ontology alignment systems using the pair of ontologies NGBO - STATO. LogMap-Lite had the fastest time at 7 seconds, followed by LSMatch at 9 seconds. AML finished in 12 seconds. Matcha took 35 seconds, while ATMatcher took 56 seconds. LogMap needed 59 seconds. OntoLink took 252 seconds to complete, and LogMap-Bio had the longest time at 392 seconds.

Among the systems, LSMatch produced the largest size of mapping, with 203 correspondences in total. OntoLink followed closely with 195 correspondences found, and LogMap found 183. ATMatcher produced a set of 147. LogMap-Lite and Matcha found 19 and 13, respectively. AML discovered 5 correspondences between the two ontologies, while LogMap-Bio produced an empty set of mappings.

In terms of precision, OntoLink had the highest precision valued at 0.538. Following was LSMatch at 0.375. AML had a precision of 0.200, and LogMap-Lite had 0.157. ATMatcher followed with 0.125, while LogMap had 0.110. Both LogMap-Bio and Matcha had a precision of 0.

Several systems achieved a perfect recall of 1.000. OntoLink, LSMatch and LogMap-Lite were the systems that achieved a recall of 1.000. ATMatcher followed with a recall of 0.334. Next, LogMap had a value of 0.250, while AML had a recall of 0.200. Both LogMap-Bio and Matcha had a recall of 0.000.

The F-measure gives us a better understanding of the evaluation results. OntoLink had the highest F-measure at 0.700. Next was LSMatch at 0.545. LogMap-Lite achieved 0.270, with AML coming close at 0.250. ATMatcher had a score of 0.182, and LogMap had 0.150. Both LogMap-Bio and Matcha had an F-measure of 0.000.

In the pair of ontologies *NGBO-STATO,* OntoLink took over 200 seconds to complete the alignment. However, this additional time allowed it to achieve the highest precision, recall, and F-measure among all the systems. The longer processing time helped OntoLink to achieve a better overall performance.

5. Concluding Remarks

5.1. Findings

In this work, we introduced OntoLink, a newly developed system for ontology alignment. OntoLink was designed to improve the discovery of mappings between ontologies. s application was demonstrated on the PSDO-STATO and NGBO-STATO ontology pairs and compared against other state-of-the-art systems.

It was necessary for us to understand why the development of OntoLink was important how it could contribute to the research field. After examining existing ontology alignment systems and methods used to discover mappings, we found some limitations. For example, many systems overlooked mappings between imported terms from external ontologies, even though these terms refer to the same resource. This simple step is crucial, as discussed in Chapter 3. Additionally, we found out that many systems do not check for ontology consistency, leading to mappings between inconsistent ontologies. We also wanted OntoLink to handle mappings between object properties, which is often overlooked. Addressing these gaps was key in creating OntoLink.

To address these challenges, we developed OntoLink. However, to fully build the system, we incorporated key components from LogMap's architecture to assist us. After properly integrating these elements, we enhanced the system's structure. Using Manchester syntax, we worked with anonymous classes in the ontologies to uncover additional mappings. As explained in Chapter 3, we compared the terms of anonymous classes to find mappings between named classes.

Next, we used reasoning tasks to discover more mappings. By applying a substitution rule and the HermiT reasoner, we were able to find additional mappings. HermiT was chosen among existing reasoners because it is the most efficient and fastest at checking for inconsistencies and unsatisfiable classes in ontologies. HermiT helped us check if the mappings found are correct and provided us a sound approach for identifying them.

After developing OntoLink, it was important to demonstrate its application. We tested it on the PSDO-STATO and NGBO-STATO ontology pairs. While OntoLink took some extra time to produce the final set of mappings, this additional time helped it identify more mappings and create a richer, larger alignment set. This led to higher precision, recall, and F-measure, which are key metrics for evaluating a system's performance.

Overall, OntoLink is a newly developed system designed to perform ontology alignment. Its architecture consists of components that work together to discover and identify mappings. OntoLink uncovers mappings in an effective and reliable way.

5.2. Future Work

The creation of OntoLink, presented in this work, is a new contribution in the field of ontologies. However, there are several potential avenues for future research and improvement on the system. Key areas for improvement could be the following:

- Since OntoLink uses HermiT as its reasoner, it would be interesting to explore the use of other reasoners in its architecture. There are many available options, such as FaCT++ or Openllet [28, 29], which could be tested for their effectiveness and compatibility with our system.

- Using a different lexicon in our system could be an interesting task. Currently, OntoLink uses UMLS or WordNet, depending on user needs. Replacing the existing lexicon with an alternative could provide insights about the performance of OntoLink.

- Additionally, developing a simple user interface could significantly enhance the user experience. This would make OntoLink more accessible and easier to use without complicating the process for users.

6. References

[1] E. Jiménez-Ruiz and B. Cuenca Grau, "LogMap: Logic-based and Scalable Ontology Matching," Department of Computer Science, University of Oxford. [Online]. Available: https://www.cs.ox.ac.uk/isg/projects/LogMap/papers/paper_ISWC2011.pdf. [Accessed Nov. 2023]

[2] M. Horridge, N. Drummond, J. Goodwin, A. Rector, R. Stevens, and H. H. Wang, "The Manchester OWL Syntax," The University of Manchester, and The Ordnance Survey. [Online]. Available: https://ceur-ws.org/Vol-216/submission_9.pdf. [Accessed Dec. 2023].

[3] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang, "HermiT: An OWL 2 Reasoner," [Online]. Available: https://www.cs.ox.ac.uk/people/boris.motik/pubs/ghmsw14HermiT.pdf. [Accessed Mar. 2024].

[4] Ι. Βλαχάβας, Π. Κεφαλάς, Ν. Βασιλειάδης, Φ. Κόκκορας, and Η. Σακελλαρίου, "Τεχνητή Νοημοσύνη", 4th ed. Thessaloniki, Greece: Εκδόσεις Πανεπιστημίου Μακεδονίας, pp. 896-897, 2023. [Accessed Jun. 2024].

[5] Stanford University, "Protégé," [Online]. Available: https://protege.stanford.edu/. [Accessed Dec. 2023].

[6] M. Krötzsch, F. Simancík, and I. Horrocks, "Description Logics," Department of Computer Science, University of Oxford, UK, The University of Manchester, and The Ordnance Survey, 2014. [Online]. Available: https://www.cs.ox.ac.uk/people/ian.horrocks/Publications/download/2014/KrSH14.pdf. [Accessed Jun. 2024].

[7] N. Purohit and A. Joshi, "Pellet: A Practical OWL-DL Reasoner," Department of Computer Science, Georgia State University, Atlanta, GA 30303. [Online]. Available: https://tinman.cs.gsu.edu/~raj/8711/sp11/presentations/pelletReport.pdf. [Accessed Jun. 2024].

[8] Y. Kazakov, M. Krötzsch, and F. Simančík, "ELK Reasoner: Architecture and Evaluation," Institute of Artificial Intelligence, Ulm University, Germany, and Department of Computer Science, University of Oxford, UK, 2012. [Online]. Available: https://ceur-ws.org/Vol-858/ore2012_paper10.pdf. [Accessed Jun. 2024].

[9] Ontology Alignment Evaluation Initiative. [Online]. Available: https://oaei.ontologymatching.org/. [Accessed Sep. 2023].

[10] D. Faria, C. Pesquita, E. Santos, M. Palmonari, I. F. Cruz, and F. M. Couto, "The AgreementMakerLight Ontology Matching System," LASIGE, Department of Informatics, Faculty of Sciences of the University of Lisbon, Portugal; Department of Informatics, Systems and Communication, University of Milan Bicocca, Italy; ADVIS Lab, Department of Computer Science, University of Illinois at Chicago, USA. [Online]. Available: https://cake.fiu.edu/Publications/Faria+al-13-

TA.The_Agreement_Maker_Light_Ontology_Springer_downloaded.pdf. [Accessed Sep. 2023].

[11] D. Faria, M. C. Silva, P. Cotovio, P. Eugénio, and C. Pesquita, "Matcha and Matcha-DL results for OAEI 2022," LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal; INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal. [Online]. Available: https://ceur-ws.org/Vol-3324/oaei22_paper11.pdf. [Accessed Sep. 2023].

[12] G. Stoilos, G. B. Stamou, and S. D. Kollias, "A string metric for ontology alignment," in "Proc. of the International Semantic Web Conference (ISWC)", 2005, pp. 624–637. [Online]. Available: http://dx.doi.org/10.1007/11574620_45. [Accessed Sep. 2023].

[13] T. Wang, P. Chen, K. Amaral, and J. Qiang, "An Experimental Study of LSTM Encoder-Decoder Model for Text Simplification," Department of Computer Science, University of Massachusetts Boston, and Department of Computer Science, Hefei University of Technology. [Online]. Available: https://arxiv.org/pdf/1609.03663. [Accessed May 2024].

[14] A. Sharma, A. Patel, and S. Jain, "LSMatch and LSMatch-Multilingual Results for OAEI 2022," National Institute of Technology Kurukshetra, India, and Eastern International University, Vietnam. [Online]. Available: https://ceur-ws.org/Vol-332. [Accessed May 2024].

[15] "Natural Language Toolkit (NLTK)," [Online]. Available: https://www.nltk.org/. [Accessed May 2024].

[16] "MyMemory by Translated Labs," [Online]. Available: https://mymemory.translated.net/. [Accessed May 2024].

[17] G. Sérasset, "DBnary: Wiktionary as a Lemon-Based Multilingual Lexical Resource in RDF," Semantic Web Journal, special issue on Multilingual Linked Open Data, 2014. [Online]. Available: https://www.semantic-web-journal.net/system/files/swj648.pdf. [Accessed May 2024].

[18] S. Hertling and H. Paulheim, "ATBox Results for OAEI 2022," Data and Web Science Group, University of Mannheim, Germany, 2022. [Online]. Available: https://ceur-ws.org/Vol-3324/oaei22_paper4.pdf. [Accessed May 2024].

[19] Ontology Alignment Evaluation Initiative, "OAEI-2022 Campaign: Anatomy," 2022. [Online]. Available: https://oaei.ontologymatching.org/2022/anatomy/index.html. [Accessed Nov. 2023].

[20] M. Brochhausen et al., "OBIB-a novel ontology for biobanking," Journal Name, vol. X, no. X, pp. X-X, [Online]. Available: https://www.researchgate.net/publication/301827222_OBIB-a_novel_ontology_for_biobanking. [Accessed May 2024].

[21] M. Brochhausen *et al.*, "Developing a semantically rich ontology for the biobank-administration domain," *J. Biomed. Semant.*, vol. 3, no. S-2, pp. 1-14, 2012. [Online]. Available: https://core.ac.uk/download/pdf/81178577.pdf. [Accessed May 2024].

[22] Z. Landis-Lewis, C. Stansbury, J. Rincón, and C. Gross, "Performance Summary Display Ontology: Feedback intervention content, delivery, and interpreted information," University of Michigan Medical School, Ann Arbor, Michigan, USA and Harvard Medical School, Cambridge, Massachusetts, USA, 2022. [Online]. Available: https://icbo-conference.github.io/icbo2022/papers/ICBO-2022_paper_2172.pdf. [Accessed May 2024].

[23] Statistics Ontology. [Online]. Available: https://stato-ontology.org/. [Accessed May. 2024].

[24] D. Alghamdi, D. M. Dooley, G. Gosal, E. J. Griffiths, and W. W. L. Hsiao, "NGBO: Introducing -omics data into biobanking ontology," University of British Columbia, Vancouver, BC V6T 1Z4, Canada; BC Centre for Disease Control Public Health Laboratory, Vancouver, BC V5Z 4R4, Canada; King Fahad Medical City, Riyadh 59046, Saudi Arabia. [Online]. Available: https://ceur-ws.org/Vol-2931/ICBO_2019_paper_56.pdf. [Accessed May. 2024].

[25] M. Horridge, M. Egaña Aranguren, J. Mortensen, M. Musen, and N. F. Noy, "Ontology Design Pattern Language Expressivity Requirements," Stanford Center for Bioinformatics Research (BMIR), Stanford University, California, USA, Biological Informatics, Centre for Plant Biotechnology and Genomics (CBGP), Technical University of Madrid (UPM), Spain, 2024. [Online]. Available: https://ceur-ws.org/Vol-929/paper3.pdf. [Accessed Apr. 2024].

[26] I. Horrocks, O. Kutz, and U. Sattler, "The Even More Irresistible SROIQ," School of Computer Science, The University of Manchester, Kilburn Building, Oxford Road, Manchester M13 9PL, UK. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=8cff77d373d84844e4084bbda77a6f797f25ede9. [Accessed Apr. 2024].

[27] I. Horrocks, O. Kutz, and U. Sattler, "The Irresistible SRIQ," School of Computer Science, The University of Manchester, Kilburn Building, Oxford Road, Manchester, M13 9PL, UK. [Online]. Available: https://ceur-ws.org/Vol-188/sub20.pdf. [Accessed Apr. 2024].

[28] D. Tsarkov and I. Horrocks, "FaCT++ Description Logic Reasoner: System Description," in "Automated Reasoning", U. Furbach and N. Shankar, Eds., vol. 4130. Berlin, Heidelberg: Springer, 2006. [Online]. Available: https://doi.org/10.1007/11814771_26. [Accessed: Aug. 2024].

[29] "Openllet: An Open Source OWL DL Reasoner for Java," GitHub. [Online]. Available: https://github.com/Galigator/openllet. [Accessed: Aug. 2024].

[30] P. Shvaiko and J. Euzenat, "Ontology matching: state of the art and future challenges," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 1, pp. 158-176, Jan. 2013. [Online]. Available: https://doi.org/10.1109/TKDE.2011.253. [Accessed Aug. 2024].

[31] P. Ochieng and S. Kyanda, "Large-Scale Ontology Matching: State-of-the-Art Analysis," *ACM Comput. Surv.*, vol. 51, no. 4, Art. 75, pp. 1-35, Jul. 2018. [Online]. Available: https://doi.org/10.1145/3211871. [Accessed Oct. 2023].

[32] P. Shvaiko and J. Euzenat, "Ten challenges for ontology matching," in *Proc. 7th Int. Conf. Ontologies, DataBases, and Applications of Semantics (ODBASE)*, Monterey, Mexico, Nov. 2008, pp. 1163-1181. [Online]. Available: https://hal.science/hal-00825956. [Accessed Aug. 2024].

[33] J. Euzenat, C. Meilicke, P. Shvaiko, H. Stuckenschmidt, and C. Trojahn dos Santos, "Ontology Alignment Evaluation Initiative: six years of experience," *J. Data Semant.*, vol. XV, no. 6720, pp. 158-192, 2011. [Online]. Available: https://doi.org/10.1007/978-3-642-22630-4_6. [Accessed: Aug. 2024].

[34] W3C, "OWL 2 Web Ontology Language Mapping to RDF Graphs (Second Edition)," 11-Dec-2012. [Online]. Available: https://www.w3.org/TR/owl2-mapping-to-rdf/. [Accessed Sep. 2024].1