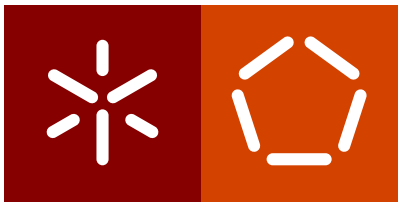
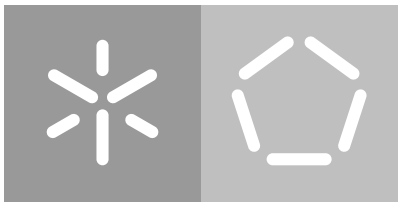


Universidade do Minho
Escola de Engenharia
Departamento de Informática

João Pedro Rodrigues Gomes

OMT, an Ontology Matching System

October 2022



Universidade do Minho
Escola de Engenharia
Departamento de Informática

João Pedro Rodrigues Gomes

OMT, an Ontology Matching System

Master dissertation
Integrated Master's in Informatics Engineering

Dissertation supervised by
Pedro Rangel Henriques
Alda Gancarski

October 2022

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

João Pedro Rodrigues Gomes

_____

Acknowledgements

I would like to express my deepest gratitude to Professor Pedro Rangel Henriques and Professor Alda Gancarski for their supervision, encouragement, contribution and knowledge to this Master's Thesis.

I'd like to acknowledge my mother for her support and endless belief in me.

ABSTRACT

In recent years ontologies have become an integral part of storing information in a structured and formal manner and a way of sharing said information. With this rise in usage, it was only a matter of time before different people tried to use ontologies to represent the same knowledge domain. The area of Ontology Matching was created with the purpose of finding correspondences between different ontologies that represented information in the same domain area.

This document reports a Master's work that started with the study of already existing ontology matching techniques and tools in order to gain knowledge on what techniques exist, as well as understand the advantages and disadvantages of each one. Using the knowledge obtained from the study of the bibliography research, a new web-based tool called OMT was created to automatically merge two given ontologies.

The OMT tool processes ontologies written in different ontology representation languages, such as the OWL family or any language written according to the RDF web standards. The OMT tool provides the user with basic information about the submitted ontologies and after the matching occurs, provides the user with a simplified version of the results focusing on the number of objects that were matched and merged. The user can also download a Log File, if he so chooses. This Log File contains a detailed description of the matching process and the reasoning behind the decisions the OMT tool made. The OMT tool was tested throughout its development phase against various different potential inputs to assess its accuracy. Lastly, a web application was developed to host the OMT tool in order to facilitate the access and use of the tool for the users.

Keywords: Ontology, Ontology Matching Techniques, Ontology Matching Tools, Ontology Alignment

RESUMO

Nos últimos tempos, ontologias têm-se tornado fundamentais quando os objetivos são armazenar informação de forma formal e estruturada bem como a partilha de tal informação. Com o aumento da procura e utilização de ontologias, tornou-se inevitável que indivíduos diferentes criassem ontologias para representar o mesmo domínio de informação. A área de concordância de ontologias foi criada com o intuito de encontrar correspondências entre ontologias que representem informação no mesmo domínio.

Este documento reporta o trabalho de uma tese de Mestrado que começou pelo estudo de técnicas e ferramentas já existentes na área de concordância de ontologias com o objetivo de obter conhecimento nestas mesmas e perceber as suas vantagens e desvantagens. A partir do conhecimento obtido a partir deste estudo, uma nova ferramenta web chamada OMT foi criada para automaticamente alinhar duas ontologias.

A ferramenta OMT processa ontologias escritas em diferentes linguagens de representação, tal como a familia de linguages OWL ou qualquer linguagem que respeite o padrão RDF. A ferramenta OMT fornece ao utilizador informação básica sobre as ontologias e após o alinhamento ocorrer, fornece ao utilizador uma versão simplificada dos resultados obtidos, focando no numero de objetos que foram alinhados. O utilizador pode também descarregar um ficheiro Log. Este ficheiro contém uma descrição destalhada do processo de alinhamento e a justificação para as diferentes decisões tomadas pelo ferramenta OMT. A ferramenta OMT foi testada durante todo o processo de desenvolvimento com diferentes tipos de ontologia de entrada para avaliar a sua capacidade de alinhamento. Por último, foi também desenvolvida uma aplicação web para hospedar a ferramenta OMT de forma a facilitar o acesso e uso da ferramenta aos utilizadores.

Palavras-Chave: Ontologias, Técnicas de Concordância de Ontologias, Ferramentas de Concordância de Ontologias

CONTENTS

LIST OF FIGURES

ACRONYMS

A

AROMA Association Rule Ontology Matching Approach.

D

DSL Domain Specific Language.

K

KIF Knowledge Interchange Format.

N

NLP Natural Language Processing.

O

OAEI Ontology Alignment Evaluation Initiative.

OWL Ontology Web Language.

R

RDF Resource Description Framework.

U

URI Uniform Resource Identifier.

W

W3C World Wide Web Consortium.

1

INTRODUCTION

In computer science, the concept of an Ontology was firstly introduced in (Gruber, 1993) as being a model to store data within a domain that allows the representation and definition of concepts, often referred to as classes, their properties and the existing relations between them. An ontology knowledge base is the result of populating an ontology with data that matches the concepts and properties that have been formally defined.

Ontology Matching methods were created with the purpose of relating information coming from multiple heterogeneous ontology sources into a common ontology model that encapsulates the entire knowledge base from all the sources (Otero-Cerdeira et al., 2015). On a simpler level, these methods have the task of finding correspondences between ontologies. The same concept or property can be defined using different terminology on different ontologies.

An example of a real life application of ontology matching, as shown in (Muhammad and Khan, 2012), is having ontologies that store information about publications where, in one ontology, the publisher of the publication is defined by the term **Publisher** and, in the other, by **Published by**. Another example, shown in (Martinez-Gil et al., 2012), is a group of ontologies that store information about football teams but use differente terminologies for the role of each player on the team, e.g., **forward player** on one ontology corresponds to **striker** on the other.

There has already been a lot of research and work done in the field of Ontology Matching, and currently there are many approaches available that automatically generate matches between Ontologies. However, these techniques are still far from being perfect due to the fact that human input is still needed when the use case requires an accurate matching, making these methods impractical when dealing with big and complex ontologies (Falconer and Noy, 2011).

This thesis focuses, firstly, on the study of the already existing methods, pointing the advantages and disadvantages of each one, and, secondly, on the creation of a new web-based method that relies as less as possible on human input.

1.1 MOTIVATION

The motivation for this thesis stems from the fact that there is still room for a lot of progress to be made in the field of ontology matching. There are a lot of tools developed and available, but all of them require human interaction to achieve high accuracy results. The main objective going forward is to build upon these tools and create new techniques that make the matching tools as much automatic as possible, aiming for completely automatic.

1.2 OBJECTIVES

The main objective of this thesis is the creation of a technique to compare and identify similarities between ontologies of the same domain. In order to accomplish this objective, the following steps are required:

- Study and evaluation of different Ontology Matching techniques in order to compare them, focusing on the advantages and disadvantages of each.
- Implementation of a new novel web-based tool for Ontology Matching.

1.3 RESEARCH HYPOTHESIS

By the end of this Master's Thesis, it will have been proven that is possible to create an ontology matching algorithm that allows for multiple different ontology language inputs and works in an automatic way.

1.4 METHODOLOGY

The methodology used to achieve the objectives is composed of the following phases.

- Bibliographic research.
- Study and analysis of the techniques found in the research.
- Proposal of a new ontology matching tool.
- Test of the implemented tool.
- Creation of the web-based application that implements the newly created ontology matching tool.
- Test of the developed application.

According to the feedback obtained from the testing done in any of the phases described above, some of the previous phases are revisited and the process redone if the results do not match the expectation.

1.5 Document Structure

In Chapter 1, an introduction to the Master's Project and its context is made and to what was the motivation that led to its development.

In Chapter 2, an introduction to the world of ontologies is made and how they came to be, followed by a brief summary of some of the earliest information representation languages that were created. In the following sections of this chapter, the base ontology matching techniques are explored as well as some of the already existing ontology matching tools are presented taking into account what type of algorithms and techniques they use.

In Chapter 3, our proposed solution is presented giving a brief overview of the general architecture of our ontology matching tool as well as all the components that constitute it.

In Chapter 4, the development processed is described, giving particular emphasis on the decisions made to implement the proposed solution presented in Chapter 3.

Lastly in Chapter 5, an overview of the document is given as well as some concluding thoughts on the work done so far and what can be done in the future.

2

STATE OF THE ART

This chapter presents the bibliograhic research that was done in order to obtain the knowledge necessary to then proceed to the implementation step of this Master's Thesis. In the first section, it begins by introducing and explaining what ontologies are, how they came to be and the pioneer knowledge representations languages that existed back then. It then explains the ontology standards that are in place today as well as some of the more used knowledge representation languages. In the final sections, ontology matching techniques are studied and presented as well as already existing ontology matching tools.

2.1 ONTOLOGIES

A conceptualization is an abstract, simplified view of an enclosed domain. It encapsulates all its objects, concepts, entities and the relationships that exist amongst themselves. A body of represented knowledge is fully based and dependant on its conceptualization (Gruber, 1993).

Ontologies are explicit specifications of conceptualizations and were created with the intent of allowing the formal definitions of terms (such as classes, properties, functions), its relations and contraints.

In the 1990 decade, Ontologies were already being used as a way to represent information, despite not existing a standard knowledge representation language, system or model to design them in. It quickly became clear that the ontology portability problem had to be addressed. The portability problem in Ontologies arose as a consequence of the usage of different representation languages and/or systems by different individuals with the intent of designing and implementing their Ontologies, making it impossible to share or use them in a different representation system than the one it was originally designed in (Gruber, 1993).

The following subsections provide a brief overview of some of the different languages/systems that existed and were used to represent and share knowledged bases, such as: the Knowledge Interchange Format (KIF) (Genesereth et al., 1992), CycL (Lenat and Guha, 1990) and Ontolingua (Gruber, 1993).

2.1.1 *KIF*

Knowledge Interchange Format (KIF) was created by Michael Genesereth, Richard Fikes and others (Genesereth et al., 1992). It is a computer language whose intent is to allow the share of that same information among different individuals who utilize different programming languages and systems to represent knowledge. KIF was created and implemented with very specific requirements in mind:

- It had to be able to translate knowledge bases from and to other typical knowledge representation systems.
- Readibility. KIF syntax was constructed in a way to facilitate human reading and undestanding of what's being represented.
- Usability of the language itself to represent knowledge bases, despite not being its main purpose.

A program written in KIF is a group of expressions, composed of rules, variables and operators. Variables are preceded by the character '?'. The operators =>, <= and <=> indicate implications, while the operator = indicates equality in a relation.

The example in Listing 1 is taken from (Gruber, 1995) and represents an example of code written in this language.

```
(defrelation PHYSICAL-QUANTITY
    (<=> (PHYSICAL-QUANTITY ?q)
        (and (defined (quantity.magnitude ?q))
             (double-float (quantity.magnitude ?q))
             (defined (quantity.unit ?q))
             (member (quantity.unit ?q)
                (setof meter second kilogram
                       ampere kelvin mole candela)))
```

Listing 1: KIF syntax example

For a full manual on the language, refer to (Genesereth et al., 1992).

2.1.2 *CycL*

CycL is an ontology language created, designed and developed by Doug Lenat and Ramanathan V. Guha (Lenat and Guha, 1990) and was mainly used in the Cyc project (Lenat et al., 1990).

Cycl is a declarative language based on first-order logic, which means it uses quantified variables and allows the use of sentences that contain those variables. It was designed with some basic ideas in mind:

- Variables are preceded by '?'.
- Constants represent concept names. These concepts can range from simple individual items to collections or functions. Constants are case sensitive and are preceded with '#$'.
- The ability to provide specification and generalization through the use of '#$isa' and '#$genls', respectively. Specification indicates that one item is an instance of a given collection while Generalization indicates that a given collection is a subcollection of another collection.
- Rules that are formed by all of the above.

An example of CycL syntax can be seen in Listing 2.

```
(#$implies
   (#$and
     (#$isa ?OBJ ?SUBSET)
     (#$genls ?SUBSET ?SUPERSET))
   (#$isa ?OBJ ?SUPERSET))
```

Listing 2: CycL syntax example

For a full manual, refer to (Lenat and Guha, 1990).

2.1.3 *Ontolingua*

Aware of the portability problem already discussed in section 2.1, it was introduced in (Gruber, 1993) a system for describing ontologies in such a way that they become compatible with different representation languages.

Ontolingua allows the definition of classes, functions, objects, theories, relations and allows the translation of those definitions to a standard language that is compatible with a great number of knowledge representation systems. Its syntax is very similar to the KIF language syntax.

- Variables are preceded with the '?' prefix.
- Operators that support conjunction, disjunction, implication and negation: forall, exists, and not.

- Operators that indicate material implication such as =>.
- Existence of binary relations that take 2 variables as input. For example: binaryRelation ?a ?b.

An example of Ontolingua's syntax can be seen in Listing 3.

```
(forall ?W
    (=> (writer ?W)
        (exists (?R ?D)
            and (reader ?R)
                (document ?D)
                (writes ?W ?D)
                (reads ?R ?D)
                (not (understands ?R ?D))
        )
)
```

Listing 3: Ontolingua syntax example

For a full manual, refer to (Gruber, 1993).

2.2 ONTOLOGY STANDARDS AND NOTATIONS

The knowledge representation languages presented in the previous sections were the ones that pioneered the designing and sharing of ontologies. As the years went by, all these languages were worked on, perfected and eventually gave their place up to newer languages and standards. This section presents the RDF ontology standard that is in place nowadays as well as some of the most used languages used to design and share ontologies.

2.2.1 *RDF Standard*

Nowadays there are a handful of different languages with which one can design and create an ontology. Most of these languages follow a family of specifications called **Resource Description Framework**, often referred to as **RDF**.

RDF was created by the **World Wide Web Consortium**, often referred as **W3C**, in an attempt to standardize a method for designing and modeling conceptual information with the intent of being implemented or shared via the web.

An integral part of the RDF standard is its data model and the use of a **Uniform Resource Identifier (URI)**. A URI is a unique sequence of characters that can identify anything ranging

from people and places to books or webpages. RDF makes use of them to identify all the concepts and properties represented in the ontology.

The RDF data model is built upon the already existing approaches of classes and entity relationships, and encapsulated it in a structure often referred to as a *triple*. A triple is the way to make statements about anything. It follows the structure:

Subject - Predicate - Object

In this structure, the Subject denotates the resource about which the statement is being made. The Predicate indicates the type of relationship the Subject has with the Object.

For further information on the RDF format and the RDF Schema, refer to RDF Manual.

2.2.2 *OWL*

Web Ontology Language (OWL) is a language first created in 2004 for representing knowledge, namely in the form of ontologies. It was designed and built with the objective to share information over the Internet and with this in mind it follows and respects the **Resource Description Framework (RDF)** standard.

The OWL format and syntax is composed of Annotations, Facts and Axioms, with the latter two being where the most important information is provided.

Annotations are used to indicate authorship of the ontology and to import and include other ontologies.

Facts can be used to state information about a particular individual such as to what class that individual belongs to or what properties and values that individual possesses. Facts can also be used to declare that two individual identifiers represent the same individual or distinct ones.

Axioms are used to provide information about classes and their properties. Properties can be general or sub-properties of others. They can also be transitive, symmetrical and inverse of one another. Restrictions in classes can be made by giving constraints to the related properties, with those constraints being applied to its type or range for example.

A simple example of the OWL syntax can be seen in the Listing 4.

```
1 <owl:Class rdf:ID="Student"/>
2 <owl:Class rdf:ID="Teacher"/>
3
4 <Student rdf:ID="Joao" />
5
6 <owl:ObjectProperty rdf:ID="Teaches">
7   <rdfs:domain rdf:resource="#Teacher"/>
```

```
  <rdfs:range rdf:resource="#Student"/>
</owl:ObjectProperty>
```

Listing 4: OWL syntax example

For further information on the OWL language, refer to OWL Manual.

2.2.3 *Turtle*

Turtle is a language for expressing knowledge in the RDF data model. It was designed by Dave Beckett and was recognized by the World Wide Web Consortium on 25 February 2014.

Much like the RDF data model, Turtle heavily focuses on its triples as a way to convey information. As already explained in section 2.2.1, each triple is composed of a Subject, Predicate and Object and each one has to be expressed in a URI. Turtle allows the grouping of common URIs as a way to abbreviate the same information and therefore easing the construction of the triples. This is done with the creation of prefixes.

An example of the Turtle language syntax and how it allows to abbreviate the URI can be seen in the Listing 5.

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

###  http://prc.di.uminho.pt/2021/Projeto#Artista
:Artista rdf:type owl:Class .

###  http://prc.di.uminho.pt/2021/Projeto#nome
:nome rdf:type owl:DatatypeProperty .

###  http://prc.di.uminho.pt/2021/Projeto#Artist_Adele
:Artist_Adele rdf:type owl:NamedIndividual ,
:Artista ;
:nome "Adele" .
```

Listing 5: Turtle syntax example

For further information on the Turtle language, refer to Turtle Manual.

2.2.4 *OntoDL+*

OntoDL+ is a language created by Alexandre Dias under the supervision of Pedro Rangel Henriques and Cristiana Araújo as part of his Master Thesis in University of Minho. This language allows the representation of knowledge in the form of ontologies as well as the conversion of the ontologies written in this language to other ontology languages such as OWL, Alloy, DOT or Prolog.

It is an improved version of the already existing OntoDL language that allows the pre declaration of attributes, the possibility to structure the ontology in different files or the declaration of Prolog Restrictions in the ontology.

An example of the syntax of OntoDL+, taken from the official OntoDL+ Website, OntoDL+ Website, can be seen in Listing 6.

```
atributos {
  distância:string,
  descrição:string,
  distrito:string,
  nome:string,
  população:string
}

conceitos {
  Ligação[distância],
  Cidade[descrição, distrito, nome, população]
}
```

Listing 6: OntoDL+ syntax example

2.3 ONTOLOGY MATCHING

With the ontology portability issue resolved and the existence of a clear standard for the writing of the ontologies, as well as many different languages that allow it to be done, the interest in ontologies as a way to represent information has never been greater.

As ontologies become more commonplace and their number grows, so does their diversity and heterogeneity (Falconer and Noy, 2011). With this increase in diversity and heterogeneity, it was only a matter of time until different ontologies began trying to describe the same knowledge domains. Therefore a universal interest of trying to match these ontologies arose and the area of investigation known as Ontology Matching was created.

Ontology matching is the process of relating information from heterogeneous sources into a common model that can be queried and reasoned upon. On an abstract level, ontology

matching is the task of finding correspondences between ontologies, for example, the same concept, property or relationship can be defined using different terminologies on different ontologies.

Despite the efforts that have already been made in creating approaches to match ontologies, there is still a lot of room to grow regarding their ability to work in an autonomous way. Usually, these approaches evaluate ontologies given as input and come out with a group of possible correspondences. These correspondences have then to be examined by a person to determine which ones are correct, remove the false positives and create additional correspondences that were missed (Falconer and Noy, 2011), making this whole process impractical when dealing with big and complex ontologies.

2.3.1 *Matching Techniques*

The approaches developed to solve the matching problem all make use of different techniques that each, in its own way, aims to exploit the information present in the ontology.

There are multiple criteria of classification that one can use to group up these approaches (Otero-Cerdeira et al., 2015).

One simple way to divide these approaches is based on whether they analyze the information orthographically (string similarity) or semantically.

On the one hand, analysing information orthographically means that only the strings are important and taken into account. Many string techniques are available for that approach. On the other hand, analysing information semantically means that the meaning and context of the words are also taken into account. Obviously these approaches are not mutually exclusive and are used together in many different tools to achieve better results.

The following subsections take a closer look at some techniques used in these approaches.

String Based Techniques

In this category all the techniques analyse the information orthographically. They all measure the similarity of the Strings used to represent the concepts, properties, labels, comments and relationships of the ontology.

Distance functions techniques map a pair of strings (x, y) to a real number r, where the smaller the value of r, the greater the similarity between the string pair (Cohen et al., 2003). There are many different implementations of these functions.

One of the most important ones is the group of edit distance functions which measure the distance between two strings by calculating the most efficient sequence of pre-established operations that convert one string into the other. The operations allowed to execute are what separates the techniques that belong to this group. The **Levenshtein distance** allows the operations of deletion, insertion and substitution while the **Damerau–Levenshtein distance**

improved upon its predecessor and also allows the transposition of two characters. Last but not least, the **Hamming Distance** only allows the substitution operation which, as a consequence, means this teachnique can only be used when comparing two strings of equal length.

Another type of distance functions techniques are the ones based on tokenization which convert every string into a set composed of all its words (tokens). Afterwards, the similarity of the two strings is calculated by performing operations and comparing the two resulting set of tokens.

A simple technique is the **Jaccard Index**. With (*S*,*T*) being the respective sets of tokens of the string pair (*s*,*t*), the **Jaccard Index** is obtained by the following formula:

$$Jac(s,t) = \frac{|S \cap T|}{|S \cup T|}$$

It calculates the similarity of a pair of strings taking into account the relation between the number of common tokens in regards with the whole set of tokens.

One last technique that makes use of tokenization is the **Overlap coefficient**. As the name suggests, it calculates the similarity of two strings by measuring the amount of overlap that exists between the two resulting sets of tokens. It is calculated by dividing the set intersection by the smaller size of the two sets:

$$OverlapCoefficient(s,t) = \frac{|S \cap T|}{min(|S|,|T|)}$$

Another type of distance functions are the called Hybrid functions which make use of different techniques. For example, the **Monge-Elkan Similarity Function** proposes a recursive approach to this problem, that is best used when comparing two long strings. This technique splits both strings into a set composed of all their sub-strings. It then compares every sub-string of one set with every sub-string of the other set, only saving the maximum value calculated through every iteration. It then computes the average of every maximum value as the value for the distance of two strings. Given a string pair (*s*,*t*), let *K* and *L* be the amount of sub-strings respectively, and **sim** a generic function to calculate the similarity between two strings, the **Monge-Elkan Similarity Function** is given by the following expression:

$$ME(s,t) = \frac{1}{K}\sum_{i=1}^{K} max_{j=1}^{L} sim(S\_i, T\_j)$$

The *Jac* and *ME* are some of the many functions and approaches possible based on strings. This type of techniques is usually the first one to be used in a system in order to find the matching concepts, properties and relationships that are syntactically defined in a similar way.

Language Based Techniques

While, in the previous category, the techniques analyse the information of the ontologies syntactically, the techniques in the language based category analyse the information semantically. This means that the concepts, properties, labels, comments and relationships are not analysed as mere strings but are analysed as words that have a meaning in some language. Therefore, a lot of the techniques used rely on Nature Language Processing (NLP) and on the use of external sources such as dictionaries, lexicons and databases (Otero-Cerdeira et al., 2015).

This approach makes use of different techniques such as **tokenization**, which, as has been explained previously, is the act of decomposing a String into smaller parts that compose it. In the specific case of language based techniques, tokenization is used to identify the words (tokens) that exist in the input string in order to better understand and exploit the possible meaning a word can have in the domain it is being expressed in.

Lemmatisation is another technique that is used. It is the process of finding the **lemma** that corresponds to the word that is being analysed. By definition, a lemma is the canonical form of a word and is considered the base form of a whole set of words that derive from it. For example, considering the set of words `walk`, `walking`, `walked`, `walkabout` all derive from the word `walk` and therefore **walk** is lemma of this set of words. One of the most common ways to find the lemma of a word is by looking it up in a dictionary.

Another technique also used is often referred to as **stop-word elimination**. This process is not as well defined as the other previous two and it depends on who is using it and in what context. It consists on creating a list composed of words that are considered not useful in the context of the problem and are therefore filtered out on the pre-processing stage of the string analysis phase.

These techniques are used to prepare the information being analysed. After they are applied, the resulting terms (`tokens`, `lemmas`) are then compared to check for their similarity. If they are not similar, then dictionaries and thesaurus are used to check if the terms have the same meaning, i.e if they are synonyms.

Language based techniques are some of the most important because they can find matches that, for example, the string based techniques do not. When dealing with ontologies written in different languages, these are the techniques that are mostly used, making use of dictionaries for example. Given that most matches in an ontology matching problem are not going to be orthographically equivalent, these techniques are in great use and hence is why tokenisation and lemmatisation are still being worked on and perfected nowadays.

Constraint and Instance Based Techniques

Constraint based techniques try to exploit the information of the contraints that can be placed on the properties and relationships of an ontology, such as the `Type` of data properties or the `Domain` and `Co-Domain` of relationships. This approach is based on the idea that, if properties and relationships on different ontologies match on the level of the constraints, then they are a potential match that needs to be studied. This technique is harder to be used alone and is mostly used in combination with other techniques to generate potential matches or analyse already generated matches.

Instance based techniques are considered an extension of all the techniques already presented, because they make use of the individuals when dealing with populated ontologies. Despite having the drawback of a much bigger quantity of information to analyze and consequently making the whole process take more time and use more resources, these type of techniques are great at both confirming already potential matches and generating new possible matches. When generating new matches, the general strategy is that if two individuals are alike, then the concepts they belong to are probably alike as well. When confirming potential matches, the general strategy is that if two concepts are alike and to be matched, then their individuals should also be alike (Otero-Cerdeira et al., 2015).

2.4 EXISTING TOOLS

In the following subsections, some of the many already existing ontology matching tools are presented focusing on the techniques each one uses as well as some of the restrictions the system has.

2.4.1 *AROMA*

AROMA (Association Rule Ontology Matching Approach) (David et al., 2006), is a tool for matching web directories, catalogs and ontologies designed in the OWL language.

The main technique used to find and reason matches is the Association Rule Paradigm. This paradigm is focused on the creation of implications (rules) of the type: x —> y that should be interpreted as: "if a term x is found on a given schema of an ontology, then that ontology may be associated with the concept y".

Having this rule paradigm in mind, the first step in AROMA's algorithm is the extraction and selection of relevant terms for each concept followed by association of different rules to form matches in the different schemas. This is a simple overview of the methodology used when dealing with web directories or catalogs.

When the input schemas are those of an OWL ontology, the first step of the process had to be adapted. Rather than only using information contained in the schema of the ontology, the individuals data is also considered.

Although they do not go much in detail on the specific techniques used in each step of the algorithm, it's possible to deduce that language based techniques were probably used, especially tokenisation to perform the extraction of the relevant terms.

This tool isn't one to use in a general problem as it was developed with a specific target in mind, only allowing ontologies written in OWL to be used as input.

For a more in depth explanation of the methodology used in the AROMA tool, refer to (David et al., 2006).

2.4.2 *AUTOMS*

AUTOMS is a tool designed for the automatic matching of domain OWL ontologies (Kotis et al., 2006). As a tool, it integrates multiples methods that must be run in a particular sequence in order for the matching to be sucessful.

Firstly, a lexical matching method is applied. It uses information concerning names, labels and comments of the ontologies concepts and properties and computes their similarity. The similarity between two terms is computed by making use of methods that compare the

typographic similarities of Strings, Substrings, sequences of ASCII characters and calculates how similar they are. It then proceeds to create pairs of possible matchings.

Secondly, WordNet is used to acess how semantically similar the pairs created are. WordNet is a widely available and well-thought of lexical resource for semantic relations.

Following these first two methods, the lexical and semantic mappings are combined into a single structure to determine their similarities in regards to concepts and properties. The individuals of the ontology are then considered to help find matchings for concepts that have not been determined to be similar in the input ontologies.

This tool is a perfect example how multiples of the techniques explained in the previous sections can be used in combination to each other in order to create an algorithm that yields good results. It uses syntactic and semantic based techniques, followed by instance based techniques to generate and confirm possible matches.

For a more in depth explanation of the entire methodology or for any individual methods used, refer to (Kotis et al., 2006).

2.4.3 *Hertuda*

Hertuda is an ontology matching tool (Hertling, 2012) designed to only accept as input ontologies compatible with the Lite or DL versions of the OWL language. It was developed to be a very simple matcher that takes advantages of tokenization and string measure to obtain alignments. It handles classes, object properties and data properties independently, which results in three different results, one for which.

For each concept, all its labels, comments and URIs are extracted, forming a set. To then compare concepts, its respective sets are compared, resulting in a similitary measure value. Before this comparison takes place, all the terms of all the sets have to go through a pre-processing step, where tokenization occurs.

A more detailed look at the algorithm source code used by Hertuda can be seen in the Listing 7, taken from (Hertling, 2012).

```
1  void function hertuda() {
2      for each type in {class, data property, object property}
3          for each concept cOne in ontology one
4              for each concept cTwo in ontology two
5                  if(compareConcepts(cOne, cTwo) > threshould(type)){
6                      add alignment between cOne and cTwo
7                  }
8  }
9
10 float compareConcepts(Concept cOne, Concept cTwo) {
11     for each termOne in {label(cOne), comment(cOne), fragment(cOne)}
12         for each termTwo in {label(cTwo), comment(cTwo), fragment(cTwo)}
```

```
13              conceptsMatrix[termOne, termTwo] = compareTerms(termOne, termTwo)
14
15      return maximumOf(conceptsMatrix)
16  }
17
18  float compareTerms(String tOne, String tTwo) {
19      tokensOne = tokenize(tOne)
20      tokensTwo = tokenize(tTwo)
21
22      tokensOne = removeStopwords(tokensOne)
23      tokensTwo = removeStopwords(tokensTwo)
24
25      for each x in tokensOne
26          for each y in tokensTwo
27              similarityMatrix[x, y] = damerauLevenshtein(x, y)
28
29      return bestAverageScore(similarityMatrix)
30  }
```

Listing 7: Hertuda algorithm source code

This is a much simpler tool compared to Automs and Aroma. Its purpose was to push to its limits the string based techniques and the String similarity functions in order to obtain matches. It can be used for comparison and analysis but it will fail in cases where the terminology used in the matching ontologies is very different.

2.4.4 *AgreementMaker*

AgreementMaker (Cruz et al., 2009) is one of the most developed and matured tools in this area as since its initial publication, it has been enchanced and improved upon multiples times, having been written follow up articles every single time.

One of its advantages is the amount of customization is offers the users in terms of its matching methods used, conceptual or structural methods, the amount of user interaction they require and if only the schema should be considered or the individuals as well. It also has built in metrics such as precision and runtime to present to the user.

The first group of matching algorithms include syntactic and lexical comparison of labels, comments, annotations and instances. The second group of algorithms try to exploit the information present in the structure of the ontologies, such as the concepts, their properties and the relations that exist amongst them. The final group combines all the results obtained in the first two groups with the goal of obtaining a unique final matching.

After the process of matching is done, Agreement Maker also offers automatic methods of evaluating the final matches. It considers the most effect evaluation technique to be the

comparison of the matching found by the tool against a *gold standard* of the domain the ontologies are included in, preferably built by domain experts.

In Figure 1 can be seen the interface of this tool.

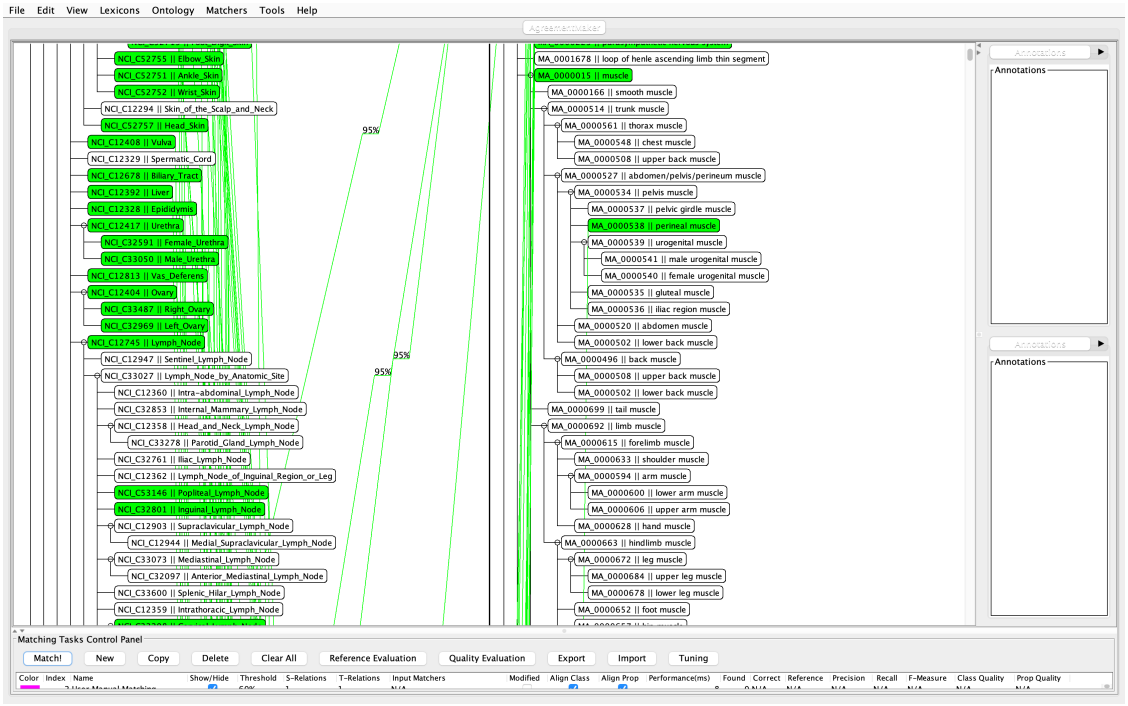


Figure 1: AgreementMaker Tool

For further detail on this tool, refer to its original publishing article (Cruz et al., 2009) and its source code is publicly available in its github repository.

2.4.5 *MEDLEY*

MEDLEY (Hassen, 2012) is an OWL ontology matching system that takes a different approach from the tools that have been covered. It transforms every ontology triple into a graph structure, with the nodes being the ontology's concepts, properties and individuals and links being the OWL primitive that links them to each other.

The techniques used are mostly based on lexical metrics, such as similarity between strings, tokenisation and stemmatisation while also making use of dictionaries to find equivalence between terminology expressed in different languages.

The base logic of this method is that, if a given entity is alike an entity that is already in the graph structure, then the neighbours of that entity must also be neighbours of the given entity, generating possible matches.

For more details on this system, refer to (Hassen, 2012).

2.5 SUMMARY

In this chapter, the world of ontologies was explored. Starting from the very beginning, the initial knowledge representation languages were presented as well as the challenges that were faced and dealt with at the time. It was then followed by a brief overview of some of the ontology languages and standards that are in place today as well as an introduction to the ontology matching problem. A presentation of some of the base ontology matching techniques were given as well as an introduction and analysis of some of the already existing ontology matching tools, as can be seen in Table 1.

	Ontology Matching Tools				
	AROMA	AUTOMS	Hertuda	AM	MEDLEY
multiple input formats	x			x	
syntactic techniques		x	x	x	x
semantic techniques	x	x		x	x
association rules techniques	x				
graphs structure					x
evaluation methods				x	

Table 1: Ontology Matching Tools - Comparison.

By analysing Table 1, it's easy to notice that while most techniques make use of the syntactic and semantic methods presented in Section 2.3.1, only a few of them allow multiple different ontology language inputs, restricting who and what ontologies can be used with those tools. It's also important to notice that only one of the studied tools has built in evaluation methods that allow the user to know how confident on the matchings found the tools is.

While the study of these tools was essential in understanding how to combine multiple techniques into forming an algorithm, the goal of this Master's Thesis is to improve on these tools by creating a tool that allows multiple different ontology languages as input, makes use of said techniques and aims for a fully automatic tool, without the need for human interaction.

3

PROPOSED APPROACH

This section provides an initial architectural design of the ontology matching tool we propose, as well as a more detailed explanation of the different components that make it up. The tool is named **OMT** and will be referenced in the remaining of the document by that.

Figure 2 represents the architecture of the OMT tool.

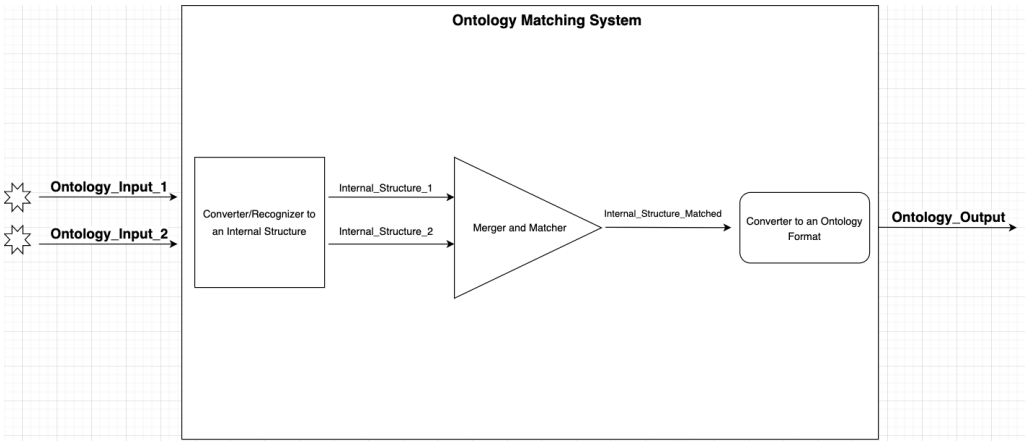


Figure 2: General overview of the system architecture

The two most important components of the OMT tool are the initial converter (Step 2) and the Merger/Matcher (Step 3), as they are the ones that convert the initial ontologies into an homogeneous format to be analysed in order to find matches.

In the following sections, it will be given a more in depth look at each step of the process.

3.1 ONTOLOGY MATCHING TOOL INPUTS

The inputs of an ontology matching tool are a critical part of it, as they determine what and who can use that tool. The majority of the tools already created for this purpose restrict the format of the input ontologies, limiting the group of potential users of the tool. One of the goals for the OMT tool is to be adaptable and easily accessible. Adaptable in the sense that the tool applies different techniques depending on the contents of the ontologies

and easily accessible since it is a web-based tool, making it available to anyone that wishes to use it. With that mind, we intend for the tool not to restrict the format of the input ontologies to a specific language and instead accept different formats such as the OWL language family (Bechhofer et al., 2004) and Turtle (Carothers and Prud'hommeaux, 2014). The input ontologies can also be written in two different natural languages: english or portuguese.

3.2 CONVERTER/RECOGNIZER

The Converter/Recognizer is the first step of the algorithm the OMT tool uses. As mentioned before, the two input ontologies may be written in different ontology languages. Therefore, this step is crucial in homogenizing the information present in the ontologies into an internal structure that is going to hold the information relevant of those ontologies.

3.3 MATCHER AND MERGER

The Matcher and Merger is the most important part of the algorithm. It is where the matching techniques to be used or not used are decided in order to find alignments in the two input ontologies.

This component can be broken into two smaller parts, the Matcher and the Merger. The Matcher finds the correspondences between the two input ontologies. The Merger joins the information present in the ontologies that wasn't matched with the matches found by the Matcher. This way it is possible to create an instance of the internal structure that would represent the matching of the input ontologies. This can be better understood in Figure 3.

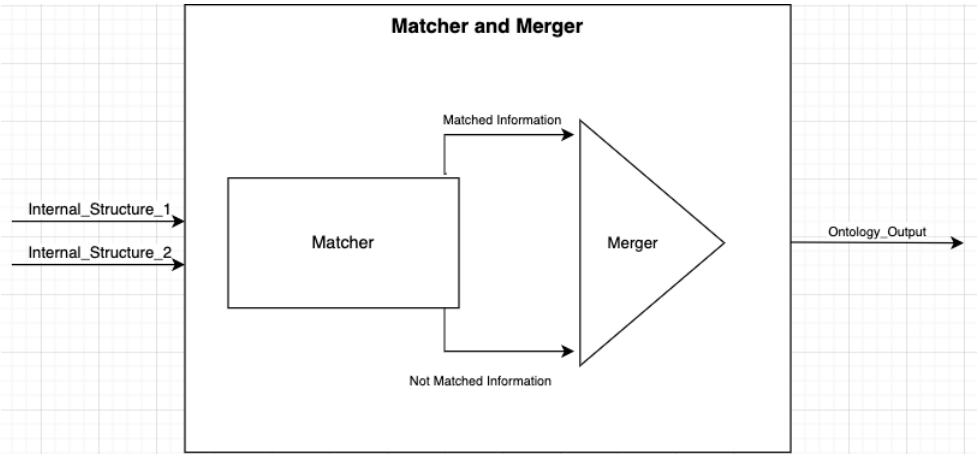


Figure 3: Detailed Matcher/Merger component

3.4 CONVERTER TO AN ONTOLOGY

The Converter to an ontology is the last step of the algorithm. It receives as input the internal structure generated by the Matcher/Merger component and converts the information on that structure into an ontology language. This process is the inverse of the process executed by the converter/recognizer. As it was said regarding that component, one of the goals is to allow diversity, so the goal is to allow the output ontology to be written in different possible languages and it is up to the user to decide in what language he would like their matched ontology to be written as.

4

DEVELOPMENT

This chapter presents the process of implementing the OMT tool, developed in this Master Thesis. Figure 4 and Figure 5 present two small simple ontologies. These two ontologies will be used as an example throughout this chapter to better explain the algorithms the **OMT** tool uses as well as some of its functionalities. The ontology representation used in both Figures is very similar to the internal structure used to save the relevant ontology data and was the representation chosen to show these examples due to being very human readable and easy to understand.

```
CONCEPTS:
    {
    "IRI": #001
    "Label": "Parent"
    "Parent's Concept": None
}

{
    "IRI": #002
    "Label": "Father"
    "Parent's Concept": #001
}

{
    "IRI": #003
    "Label": "Mother"
    "Parent's Concept": #001
}

{
    "IRI": #004
    "Label": "Son"
    "Parent's Concept": None
}

DATA PROPERTIES:
{
    "IRI": #Age
    "Label": None
    "DataType": int
}

{
    "IRI": #Name
    "Label": None
    "DataType": string
}
```

```
OBJECT PROPERTIES:
{
    "IRI": #fatherOf
    "Domain": #002
    "Range": #004
}

{
    "IRI": #motherOf
    "Domain": #003
    "Range": #004
}
```

Figure 4: Input ontology example

```
CONCEPTS:
{
    "IRI": #aa1
    "Label": "father"
    "Parent's Concept": None
}

{
    "IRI": #aa2
    "Label": "mother"
    "Parent's Concept": None
}

{
    "IRI": #aa3
    "Label": "Children"
}
DATA PROPERTIES:
{
    "IRI": #Age
    "Label": None
    "DataType": string
}
```

```
OBJECT PROPERTIES:
{
    "IRI": #fatherOf
    "Domain": #aa1
    "Range": #aa3
}

{
    "IRI": #motherOf
    "Domain": #aa2
    "Range": #aa3
}
```

Figure 5: Input ontology example

Figures 4 and 5 are divided in three sections: Concepts, Object and Data Properties. Every concept has properties associated to it: an **IRI**, a **Label** and a **Parent's Concept**. For example, in Figure 4, the concepts have the following IRIs: 001, 002, 003 and 004. The data properties have a **IRI** and a **Label** like concepts do and also have a **DataType** property. In Figure 4, the data properties have the following IRIs: *Age* and *Name*. The object properties only have the **IRI** property in common with concepts and data properties. Instead they have a **Domain** and **Range** property. In Figure 4, the object properties have the following IRIs: $fatherOf$ and $motherOf$. The $motherOf$ object property can be best thought of as a predicate in the triple:

$$Mother\ motherOf\ Son$$

where Mother and Son represent the Domain and Range concepts, respectively.

Another possible representation of an ontology is based on its graph. Figure 6 displays the concepts and relations of the ontology presented by text in Figure 4 while Figure 8 displays the same information of the ontology presented in Figure 5. The blue circles represent the concepts, identified by their label, while the red circles represent the object properties, identified by their IRI. The links represent the connections between the objects. For example, the circle **fatherOf** is connected to the circle **Son** by the link **range**, meaning that the range property of the fatherOf object property is the concept son. Figure 7 displays the data properties of the same ontology. The circles in red represent the data properties identified by their label.

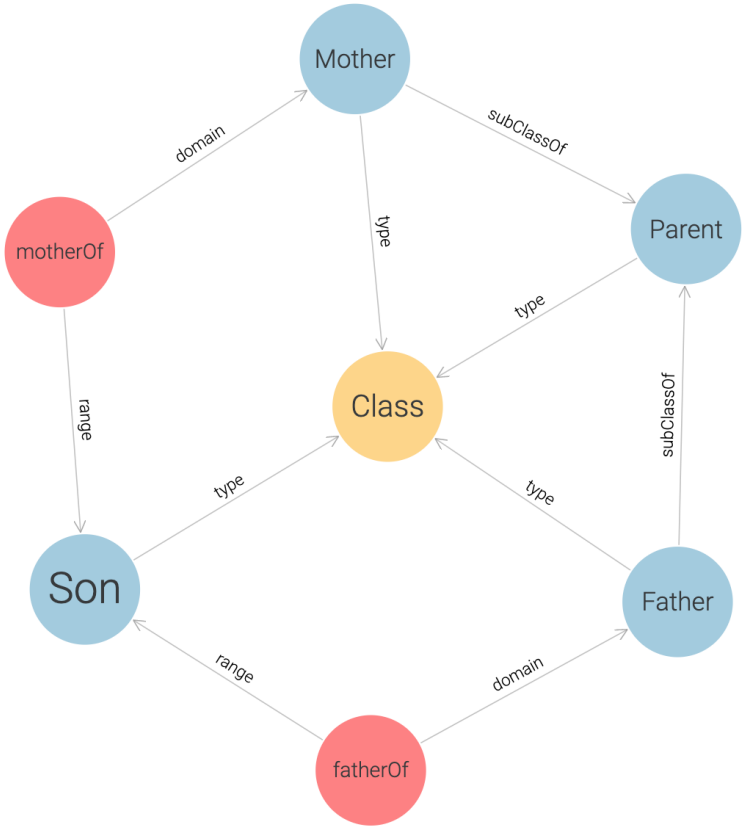


Figure 6: Ontology Graph Representation

4.1 Input Converter/Recognizer Module

One of the OMT tool main objectives is to allow the user to match ontologies written in different languages, such as **OWL** family and **Turtle**. This led to the need of developing a module that would convert the information coming from heterogeneous sources into a homogeneous structure before proceeding to the matching stage. The Input Converter/Recognizer module is the result of that need.

Ontology languages allow the definition of different information regarding the ontology concepts, relations and properties. While all this data is relevant to represent the ontology information, only some of it may be relevant to the matching process. This section presents how this module is built, what technologies are used to achieve its goals and what information is considered relevant to the matching process from the ontologies and why.

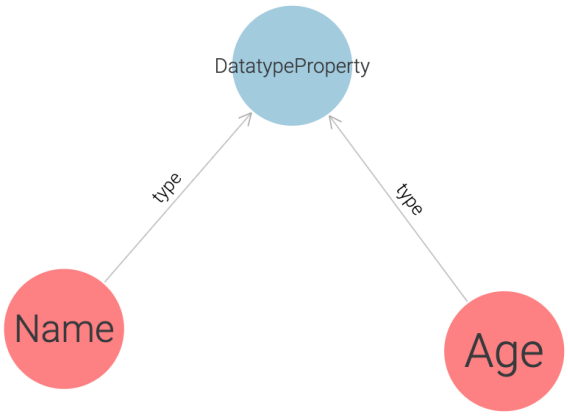


Figure 7: Ontology Graph Representation

4.1.1 *Concepts Processing*

Concepts are of the utmost importance when building ontologies as they are used to represent the entities in the domain being described.

The following list contains the properties that are considered relevant for the matching process and are therefore retrieved.

- **IRI**: Unique identifier of an object.
- **Label**: An optional property used to provide a description of an object in a human readable way.
- **Parent's Concept**: The **IRI** of the parent's class, if applicable.
- **Associated Properties**: The **IRI** of the data properties associated with this concept, if applicable.

Out of all the properties that are retrieved from the concepts, only the **IRI** is guaranteed to exist in every concept. It is, however, expected of the ontologies creators to have, at least, a **Label** associated with the concepts to ensure some level of trustworthiness in the results.

The other two properties are also optional but less common to appear in ontologies as its use is highly dependant on the domain being described. The **Parent's Concept** refers to the possibility to associate some concepts with others in a hierarchical way. The **Associated Properties** refers to the Data Properties that may be associated with a certain concept. When this is the case, an instance of a concept must have set a value for its associated Data Properties as well.

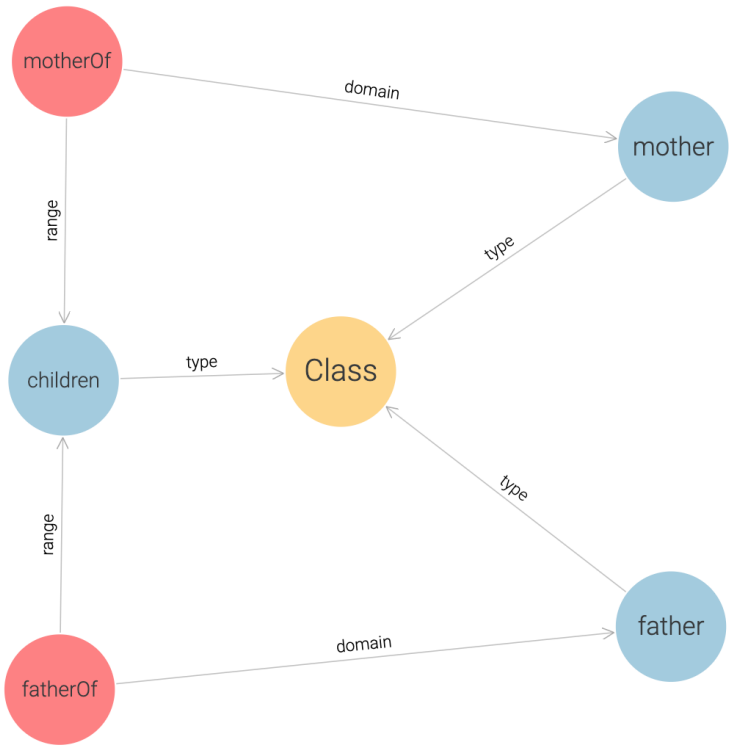


Figure 8: Ontology Graph Representation

4.1.2 *Data Properties Processing*

Data Properties are used to define attributes that can be associated to concepts in the ontology.

The following list contains the properties that are considered relevant for the matching process and are therefore retrieved.

- **IRI**: Unique identifier for an object.
- **Label**: An optional property used to provide a description of an object in a human readable way.
- **DataType**: Used to describe the data type of the Data Property it is related to. Can take many different values such as: String, Integer, Boolean, DateTime and many others.

Besides the **IRI**, the **DataType** property is also mandatory to exist in every Data Property. However, despite being mandatory, it is not guaranteed to contain a value set by the user. This is the case because every ontology language assumes the DataType to be *String* by default whenever it is not specified.

4.1.3 *Object Properties Processing*

Object Properties are used to create and define relations between two concepts.

The following list contains the properties that are considered relevant for the matching process and are therefore retrieved.

- **IRI**: Unique identifier of an object.
- **Domain**: Used to identify the Domain of a relation.
- **Range**: Used to identify the Range of a relation.

The **Domain** and the **Range** properties are used to define what concepts can be used to define the ends of a certain relationship.

It is worth noting that the reason why the **Label** property is not considered in the Object Properties is explained in Section 4.2.4.

4.1.4 *Technologies used*

The processing and parsing of the relevant information from the input ontologies was done using two already existing libraries, **RDFlib** [1] and **Owlready2** [2].

RDFLib is a library built in Python for the purpose of working with the **RDF** data model. This library was used to process ontologies designed in the **Turtle** language and in the **rdf/xml** format. **Owlready2** is a library designed to work with OWL ontologies in Python. This library was used to process ontologies designed in the **owl/xml** format.

Both these libraries were used in conjunction with the **Sparql** [3] query language making it simple to obtain all the desired information from the ontologies.

4.2 MATCHER/MERGER MODULE

The Matcher/Merger module is the most critical one since it is the one responsible for matching the information coming from the input ontologies.

Its algorithm is based on associating a value between 0 and 1 to every pair of objects being compared, where 0 means the two objects definitely do not match and 1 means the two objects definitely match. The value from which the matching algorithm considers that two objects match is defined by the user at the time of uploading the two ontologies to be matched.

1 https://rdflib.readthedocs.io/en/stable/
2 https://owlready2.readthedocs.io/en/v0.37/
3 https://www.w3.org/TR/sparql11-overview/

There are multiple properties considered relevant when matching two objects, but as was already mentioned, most of these properties are optional. This led to the need of developing a matching algorithm able to adapt and use the information available.

This section begins by explaining the properties that were considered relevant in an ontology, their relevance in the matching algorithm and how the matching algorithm works depending on what is being compared and what information is available. Then, it will present the technologies and techniques used to execute the matching.

4.2.1 *Common Properties*

There are a number of properties that can be associated with different type of objects in an ontology and therefore are defined in the Concepts, Data and Object Properties. This subsection presents them and explains their relevance in the matching algorithm.

The **IRI** is the only property that every object has and is mandatory. One would think the assurance that every object would have an **IRI** would make it a crucial property in the matching process. However, this is not the case. The reasoning for this is that the **IRI** acts as an identifier for the object and therefore is up to the user to define it. It can be something meaningful related to the object in question and in this case would be useful to the matching process. It can also be something completely random or even automatically generated. For this reason, the importance of this property is not high. Nevertheless, there are scenarios in which the IRIs have to be compared. This comparison makes use of String Based and Language Based Techniques to determine the IRIs similarity.

The **Label** property is of very high importance. As has been mentioned before, despite being optional, it is expected that most objects would have a label associated to ensure minimum confidence in the matching results. It is considered very relevant due to the fact that it, often, provides a description of the object in a human readable way and be used to match against one another. The comparison of two Labels makes use of String Based and Language Based Techniques to determine their similarity.

4.2.2 *Concepts Matching*

This subsection explains in detail how the OMT tool compares the different properties that a concept can have. Then it moves on to how the algorithm to match two concepts works, focusing on how it adapts and deals with the different scenarios that can arise from the existence, or lack of, certain properties.

Concepts Properties Comparison

The Parent's Concept property provides a hierarchical organization to the concepts. Whenever both concepts being compared have a Parent's Concept associated, these two are compared to check if they match. This process can lead to long times of execution due to the recursive nature of this algorithm, that happens when the two Parent's Concepts being matched also have Parent's Concepts.

The associated Data Properties follow the same logic as the Parent's Concepts. Whenever both the concepts being matched have associated Data Properties, these Data Properties are compared to check if they match.

Concepts Comparison Algorithm

This subsection explains in detail the scenarios the OMT tool considers in order to decide whether two concepts match. Listing 4.1 provides a simpler version of the algorithm that will be referenced in order to help understand the algorithm.

```
boolean matchConcepts(firstConcept, secondConcept, threshold):

    if hasLabel(firstConcept) and hasLabel(secondConcept):
        similarity = compare(firstLabel, secondLabel)

        if similarity close to threshold:
            if hasParentConcepts(firstConcept, secondConcept):
                if matchConcepts(parentConcept1, parentConcept2):
                    similarity++

        if  similarity close to threshold:
            if hasAssociatedProperties(firstConcept, secondConcept):
                if matchProperties(associatedProperties1, associatedProperties2):
                    similarity++

        if similarity > threshold:
            match = True
    else:
        if hasParentConcepts(firstConcept, secondConcept) and
           hasAssociatedProperties(firstConcept, secondConcept):
                if matchConcepts(parentConcept1, parentConcept2) and
                    matchProperties(associatedProperties1, associatedProperties2)
    :
                        match = True
```

Listing 4.1: Concepts Comparison Algorithm

The first action the algorithm makes is checking whether or not both concepts have a label defined (Line 3 of Listing 4.1). Depending on the result of this check, two different scenarios can occur.

The first scenario occurs when both concepts being compared have the **Label** property defined. These are matched to determine their similarity. This similarity value will be the base line value to check against the matching threshold defined by the user. If this similarity value is very close to threshold defined, the concepts are checked for the existence of Parent's Concept and associated Data Properties (Line 7 and 12 respectively of Listing 4.1). If they exist and match, similarity value is slightly increased (Line 8 and 13 respectively of Listing 4.1). If neither the **Parent's Concept** nor the **Data Properties** are defined in both concepts, the value of the labels similarity will be the one to determine whether the two concepts match or not.

The second scenario occurs when the concepts being compared do not have the **Label** property defined but have the **Parent's Concept** and associated **Data Properties** defined (Line 19 of Listing 4.1). Both the **Parent's Concept** and the associatd **Data Properties** are compared to check if they match. The concepts are considered matched if the matching of both the associated **Data Properties** and the **Parent's Concepts** come out true, and not matched if both come out negative. In the case of one coming out positive and one negative, then the **IRIs** are compared. This comparison determines their similarity. If this similarity value is above the threshold value defined by the user, then the two concepts are considered matched.

One last scenario occurs when a concept does not have Labels, Parent's Concept or Associated Data Properties defined, leaving only the IRI information. In this case, the OMT tool does not consider this concept to the matching process as it does not have enough relevant information.

To provide a better explanation, the examples presented in Figures 4 and 5 will be used to demonstrate, in a summed up way, how the algorithm works with a concrete case. Let's take as an example the concepts with the **IRIs** 001 and $aa1$ from the first and second ontology, respectively.

```
"IRI": #001
"Label": "Parent"
"Parent's Concept": None

"IRI": #aa1
"Label": "father"
"Parent's Concept": None
```

Listing 4.2: Example Concepts 001 and aa1

Listing 4.2 presents the information belonging to both concepts. Both concepts have the **Label** property defined which immediately means this case belongs to the first scenario (Line 3 of Listing 4.1). String and Language based techniques are used to determine how similar they are (Line 4 of Listing 4.1). In this case, the two specific labels ("Parent" and "father") will result in a very low similarity score. Since this low similarity score is most likely not close to the threshold value that would be chosen by the user, the **Concept's Parent** and associated **Data Properties** will not be checked (Line 6 and 11 of Listing 4.1). The two concepts will be considered not matched (Line 16 and 17 of Listing 4.1).

```
"IRI": #002
"Label": "Father"
"Parent's Concept": #001

"IRI": #aa1
"Label": "father"
"Parent's Concept": None
```

Listing 4.3: Example Concepts 002 and aa1

Listing 4.3 presents the information of the concepts 002 and $aa1$, from Figures 4 and 5 respectively, that can be compared. Alike the first pair of concepts, these also both have the **Label** property defined which corresponds to the first scenario described. The two labels ("Father", "father") will be compared to determine their similarity (Line 4 of Listing 4.1). The two labels will achieve a perfect similarity score of 1, meaning they are definitely the same in terms of the information they provide. This means this similarity score will be very high and most likely very much above the threshold defined by the user, resulting in the lack of need to check the **Parent's Concept** and associated **Data Properties**. The two concepts will be considered matched (Line 16 and 17 of Listing 4.1).

4.2.3 *Data Properties Matching*

This subsection will explain in detail how the OMT tool compares the different properties that a concept can have. Then it will explain how the algorithm to match two Data Properties works, focusing on how it adapts and deals with the different scenarios that can arise from the existence, or lack of, certain properties.

Data Properties Comparison

The DataType property is specific of the Data Properties. It is used to set the type of values that a Data Property can have assigned to it. It is mandatory in the sense that it is defined in every Data Property. It is not however guaranteed that the creator of the ontology defined it, as it defaults to being a **String** whenever it is not defined. This makes it very tricky to work and evaluate this property as it is not possible to know if a DataType set to String is done so on purpose or just because it is the default.

Data Properties Algorithm

This subsection explains in detail the scenarios the OMT tool considers in order to decide whether two Data Properties match. Listing 4.4 provides a simpler version of the algorithm that will be referenced in order to help understand the algorithm.

```
boolean matchDataProperties(firstDatap, secondDatap, threshold):

    if hasLabel(firstDatap) and hasLabel(secondDatap):
        similarity = compare(firstLabel, secondLabel)

        if similarity close to threshold:
            if matchDataType(DataType1, DataType2):
                similarity++

    else:
        similarity = compare(firstIRI, secondIRI)

        if similarity close to threshold:
            if matchDataType(DataType1, DataType2):
                similarity++

    if similarity > threshold:
        match
```

Listing 4.4: Data Properties Comparison Algorithm

The first action the algorithm makes is checking whether or not both Data Properties have a label defined (Line 3 of Listing 4.4). Depending on the result of this check, two different scenarios can occur.

When both have the Label property defined (Line 4 of Listing 4.4), the algorithm is very similar to the one applied to concept matching. The two labels will be compared and their similarity will be the base line value to check against the threshold. If this similarity is very close to the threshold defined, then the DataTypes are compared to check if they are the same. The result of this comparison will affect whether or not the similarity value goes over the threshold or not (Line 6-8 of Listing 4.4).

When the Label property is not defined in both Data Properties, then the IRI will have to be compared (Line 11 of Listing 4.4). The similarity value resulting of the comparison will determine whether or not the two Data Properties match. Similar to the first scenario, if the similarity result is very close to the threshold, then the DataType will be used (Line 13-15 of Listing 4.4).

The Data Properties presented in Figures 4 and 5 will be used to showcase how the algorithm works with a concrete case.

```
1 "IRI": #Age
2 "Label": None
3 "DataType": int
4
5 "IRI": #Age
6 "Label": "None
7 "DataType": string
```

Listing 4.5: Example Data Properties Age and Age

Listing 4.5 presents the information belonging to both Data Properties. Neither Data Property has the **Label** property defined (Line 3 of Listing 4.4). This leads to the second scenario of the algorithm, where the **IRIs** will be used (Line 11 of Listing 4.4). The two **IRIs** ("Age" and "Age") will be compared to check for their similarity. This similarity will be a perfect score of 1 as the strings are the same. This similarity score will then be much higher than the threshold defined by the user and there will be no need to use the DataTypes (Lines 13-15 of Listing 4.4). The two DataProperties will be considered matched.

4.2.4 *Object Properties Matching*

This subsection will explain in detail how the OMT tool compares the properties that a Object Property has. Then it will explain how the algorithm to match two Object Properties works.

Object Properties Information

The **Domain** and **Range** properties are used to define the two ends of a relationship. Although, in theory, an Object Property can be defined without the Domain and Range being specified, this tool will only consider those that have the Domain and Range defined, as otherwise there would not be enough information to proceed with the matching.

Additionally, the **Label** property is not considered in this matching as its information is considered irrelevant compared to the information obtained via the Domain and Range properties.

Object Properties Algorithm

This subsection explains in detail the scenario the OMT tool considers in order to decide whether two Object Properties match. Listing 4.6 provides a simpler version of the algorithm that will be referenced in order to help understand the algorithm.

```
1 boolean matchObjectProperties(firstObjectp, secondObjectp, threshold):
2
3     if matchConcepts(firstObjectpDomain, secondObjectpDomain):
4         if matchConcepts (firstObjectpRange, secondObjectpRange):
5             match = true
```

Listing 4.6: Object Properties Comparison Algorithm

Given the OMT tool only considers matching two Object Properties if both have the Domain and Range defined, only one scenario can occur. In this scenario, the Domain and Range of one Object Property will be compared against the Domain and Range of the other Object Property. The two Object Properties will only be considered matching if both the Domains and Ranges match (Line 3-5 of Listing 4.6). This is due to the fact that both the Domain and Range are an integral part of an Object Property and if one of them does not match, it renders the result of the other useless in the matching process.

To provide a better explanation, the examples presented in Figures 4 and 5 will be used to demonstrate, in a summed up way, how the algorithm works with a concrete case. Let's take as an example the Object Properties with the **IRIs** $fatherOf$ and $fatherOf$ from the first and second ontology, respectively.

```
1 "IRI": #fatherOf
2 "Domain": #002
3 "Range": #004
4
5 "IRI": #fatherOf
6 "Domain": #aa1
7 "Range": #aa3
```

Listing 4.7: Example Data Properties fatherOf and fatherOf

Listing 4.7 presents the information belonging to both Object Properties. Since there is only one scenario possible, the first thing the algorithm checks for is if both Domains concepts are matched (Line 3 of Listing 4.6). These concepts information can be seen in Listing 4.3. They have already been analysed in this section and they are considered matched. Then, the Ranges concepts are checked (Line 4 of Listing 4.6) and the outcome of this check will determine whether or not these two Object Properties are considered matched or not.

4.3 OUTPUT CONVERTER MODULE

This final module is responsible for joining the information of both ontologies taking into account the results obtained in the matching process and designing the new ontology. Every two objects that are considered matched will be converted into a single object containing the relevant information of both objects. As a default rule of the OMT tool, the resulting new object will have the IRI from the first object of the two that were matched.

4.3.1 *Concepts Matching*

This subsection explains how the tool joins the information of two concepts that are considered matched. Listing 4.8 provides a simpler version of the algorithm.

```
1  conceptsMatching(concept1, concept2):
2      finalIRI = concept1IRI
3      finalLabel = concept1Label + concept2Label
4
5      if hasParentConcept(concept1) and hasParentConcept(concept2):
6          if compareConcepts(concept1Parent, concept2Parent):
7              finalParent = conceptsMatching(concept1Parent, concept2Parent)
8
9          else:
10             finalParentList = [concept1Parent, concept2Parent]
11
12     else if hasParentConcept(concept1) and not hasParentConcept(concept2):
13         finalParent = concept1Parent
14
15     else if hasParentConcept(concept2) and not hasParentConcept(concept1):
16         finalParent = concept2Parent
17
18     else:
```

```
19          finalParent = Empty
20
21      if hasAssociatedDProperties(concept1) and hasAssociatedDProperties(concept2):
22          if compareAssociatedProperties (concept1DProperties, concept2DProperties)
    :
23              finalAssociatedProperties = DataPropertiesMatching(
    concept1DProperties, concept2DProperties)
24
25          else
26              finalListAssociatedDProperties = [concept1DProperties,
    concept2DProperties]
```

Listing 4.8: Concepts Matching

Regarding concepts matching, the Labels of both concepts will be concatenated into a single Label (Line 3 of Listing 4.8). This is a simple solution that retains all the information present in both labels and works whether both concepts have labels or not. Moving onto the Parent's Concept, a couple of different scenarios were taken into consideration. If only one of the concepts has a Parent's Concept, then the resulting object will have that Parent's Concept (Lines 12 and 16 of Listing 4.8). If both concepts have a Parent's Concept and they are considered matched, then the resulting object of the Parent's Concept matching will be the Parent's Concept of the resulting object (Lines 6-7 of Listing 4.8). The same scenario can happen but the Parent's Concept do not match. In this case, then the resulting object will have multiple Parent's Concept (Lines 9-10 of Listing 4.8). Last but not least, the concepts can have associated Data Properties. The resulting object will remain associated with every single Data Property, barring the case when two Data Properties are matched. In this case, the resulting object will be associated with the resulting Data Property (Lines 18-23 of Listing 4.8).

Figure 9 displays the concepts of the ontology that resulted from matching the ontologies presented in Figures 4 and 5. Both ontologies had a concept that could be identified by the **Label** "Father" that got matched and resulted in a single concept also identified in Figure 9 by Father. The same happened with the concepts identified by mother. The Parent's Concept identified by "Parent" from the ontology in Figure 4 did not get matched with any concept and so it got merged into the final ontology. The concepts identified by "Son" and "Children" also got merged as they did not get matched.

4.3.2 *Object Properties Matching*

This subsection explains how the OMT tool joins the information of two Object Properties that are considered matched. The Listing 4.9 provides a simpler version of the algorithm.

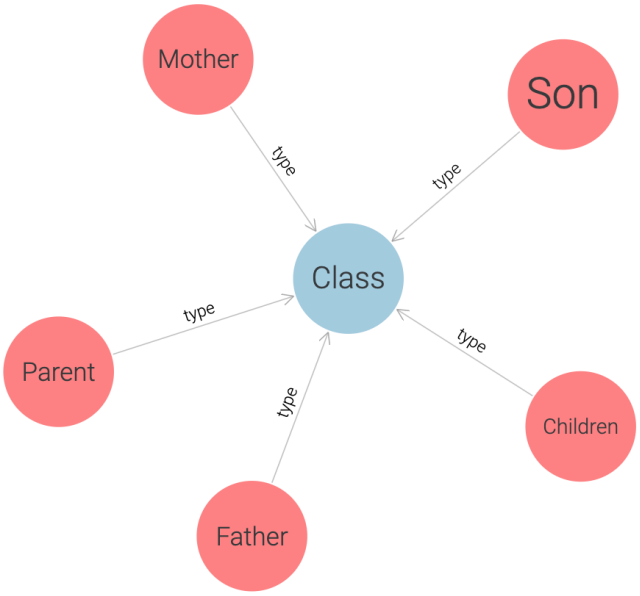


Figure 9: Output Concepts Graph

```
objPropertiesMatching(objectP1, objectP2):
    finalDomain = conceptsMatching (objectP1Domain, objectP2Domain)
    finalRange = conceptsMatching (objectP1Range, objectP2Range)
```

Listing 4.9: Object Properties Matching

Object Properties are much easier to match compared to the concepts. Since two Object Properties are considered matched only when both their Domain and Range match, then the resulting Object Property will have Domain and Range equal to the resulting concept of matching the Domain and Range, respectively.

4.3.3 *Data Properties Matching*

This subsection explains how the OMT tool joins the information of two Object Properties that are considered matched. The Listing 4.9 provides a simpler version of the algorithm.

```
dPropertiesMatching(dataP1, dataP2):
    finalLabel = dataP1Label + dataP2Label

    if dataP1DataType == dataP2DataType:
```

```
5           finalDataType = dataP1DataType
6
7       else:
8           finalDataType = checkPriorities(dataP1DataType, dataP2DataType)
```

Listing 4.10: Data Properties Matching

When it comes to the Data Properties, the Labels are treated the same way as they were in the concepts, being concatenated into a single label (Line 2 of Listing 4.10). The DataTypes were treated differently depending on the scenario. If both Data Properties have the same DataType, then the resulting DataProperty will have that same DataType (Line 4-5 of Listing 4.10). If the DataTypes are different and one of them is **String**, then the resulting DataType will be the Non-String one. This is due to the fact that the default value of a non specified DataType is String. Taking this into account, the non-String DataType will be more meaningful, hence why it is the chosen one. The last scenario happens when the DataTypes are different and neither is String. This is resolved by following a list of priorities that are considered more meaningful than others (Line 8 of Listing 4.10). The list goes as follows:

$$Boolean > Integer$$

$$DateTime > Integer$$

$$Float > Integer$$

$$TimeStamp > Integer$$

Any other combination of DataTypes is converted into a String, as there is not enough information to conclude which DataType is more meaningful.

4.4 LOG FILE

One of the unique functionalities that separates the OMT tool from the already existent ones is the possibility for the user to download a Log File. This file contains the details of every single comparison and decision made by the matching algorithm. It is divided in two different sections, the comparison and the matching.

4.4.1 *Comparison Information*

The comparison information section of the Log File contains the details of every single comparison between two objects that was made throughout the matching process. It has the results of the String Based and Language Based Techniques used to compare the objects'

properties and provides an explanation on the reason why the objects in question were considered matched or not.

The Listing 4.11 presents an example of a section of the Log File that would be generated by using the example ontologies presented in Figure 4 and 5.

```
"Concept1":
{
    "IRI": "002",
    "Label":
    [
        "Father"
    ],
    "Parent's Concept": "None"
},

"Concept2":
{
    "IRI": "aa1",
    "Label":
    [
        "father"
    ],
    "Parent's Concept": "None"
},

"String/Language Based Similarity": 1.0,

"Synonyms": "N/A",

"Analysis Status": "Matched",

"Comments":
    "The concepts were considered matched due to the String/Language Based
    Similarity being much higher than the threshold defined. There was no need to
    check the Parent's Concept."
```

Listing 4.11: Log File Example of a Concept Comparison

Line 1 to 20 of the Listing 4.11 presents the information of the two concepts. Line 21 and 23 present the results of the using String and Language Based similarity as well as dictionaries to compare the two concepts. Line 25 presents whether or not the two concepts were considered matched. Finally, line 27 presents some comments on why some decisions were made and what conclusions were drawn upon by the OMT tool.

This functionality of the Log File is especially useful for the users who may want to decide by themselves if they agree with the decisions the tool has made. It also has the advantage

of providing insight on the algorithm the tool uses and allows the users to further change their ontologies in order to align them with what the tool considers relevant and necessary for more trustworthy matching results.

4.4.2 *Matching Information*

The matching section of the Log File contains the changes that were made when converting two matching objects into a single one. It is presented in a way that allows the user to see how the information in the two matching objects was joined to generate the final object.

The Listing 4.12 presents an example of a section of the Log File that would be generated by using the example ontologies presented in Figure 4 and 5.

```
1  "DataProperty1":
2  {
3      "IRI": "Age",
4      "Label": "None",
5      "DataType": "int"
6  },
7
8  "DataProperty2":
9  {
10     "IRI": "Age",
11     "Label": "None",
12     "DataType": "string"
13 },
14 "Resulting DataProperty":
15 {
16     "IRI": "Age",
17     "Label": "None",
18     "DataType": "int"
19 },
20 "Comments": ["The resulting IRI is the IRI of the first Data Property. The
       resulting Label is the result of concatenating both concepts labels. The
       resulting DataType is the most restrictive DataType of both."]
```

Listing 4.12: Log File Example of a Data Property Matching

Line 1 to 13 of the Listing 5 presents the information of the two Data Properties. Line 16 to 18 present the information of the resulting new Data Property. As was explained in Subsection 4.3.3, the resulting IRI is the IRI of the first object (Line 16). The resulting Label is the result of concatenating both Labels, which were both non existent in this example (Line 17). The resulting DataType is the most restrictive out of the two, which is an int (Line 18). Finally, Line 20 presents additional notes and explanations.

4.5 WEB INTERFACE

One of the main goals for the OMT tool is to be web-based. This allows it to be accessed by anyone from anywhere and also being easier to use as it does not require any programming knowledge to be used. This section covers and present every different page and functionality that exists in the web application.

4.5.1 *File Upload Section*

The file upload page is the main one of the application. It contains a form, as can be seen in Figure 10, that allows the user to fill out some mandatory parameters. The user begins by having to upload two files, the ontologies to be matched(smallhuman.owl and smallmouse.owl, in Figure 10 and its format, chosen between an option of the accepted ontology formats. These formats are, so far, Turtle and the rdf/xml or owl/xml formats of the OWL family language. The formats being assigned to each ontology separately is very important as it allows for the matching of two ontologies written in different languages or formats.

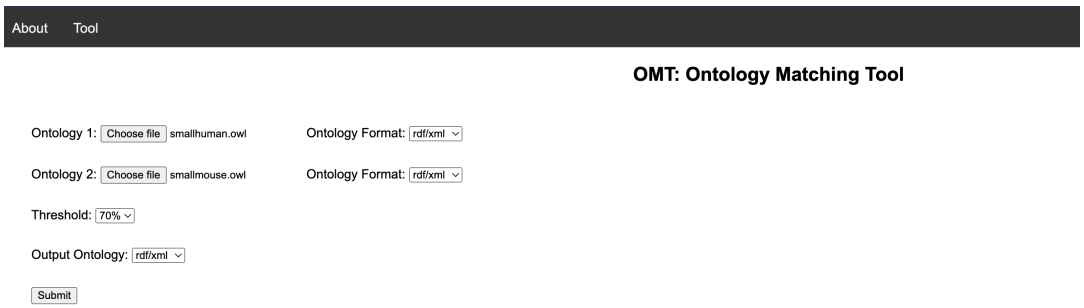


Figure 10: Ontology Upload Interface

Afterwards, the user is also asked to choose different fields all related to the algorithm the tool will use. The first one is the value to assign to the threshold they want the tool to consider when matching two objects. This threshold will be the value of similarity that determines whether two objects match. The second is the output format they want the output ontology to be designed in. This format can be any of the OWL family and Turtle. Last but not least, the user should determine whether or not they want the individuals information of the ontologies to be used.

4.5.2 *Data Section*

Throughout the matching process, the tool OMT makes some metrics available about the ontologies being matched. Firstly, right after the user uploads the ontologies and before the matching occurs, some statistics about the ontologies content is presented, such as the Number of Concepts, Relations and Data Properties, as can be seen in Figure 11, for the example ontologies referred to in Figure 10.

Basic Ontologies Information

Ontology	Number of Concepts	Number of Object Properties	Number of Data Properties	Number of Individuals	Number of Triples
smallhuman.owl	86	0	0	0	206
smallmouse.owl	86	0	0	0	347

Match

Figure 11: Basic Information about the Ontologies Uploaded

Secondly, after the OMT tool executes its matching algorithm, statistics about the number of objects that were matched and merged are presented as well as the possibility for the user to download the output ontology and the corresponding log file, as can be seen in Figure 12.

Ontology	Concepts Matched	Concepts Merged	Data Properties Matched	Data Properties Merged	Object Properties Matched	Object Properties Merged
smallhuman.owl	15	71	0	0	0	2
smallmouse.owl	15	131	0	0	0	3

Download Log File

Download Ontology

Figure 12: Statistics about the Matching.

This statistics allow the user to have a sense of how the matching process went without having to study and check the log file.

4.6 TESTING

Testing is the most important step of the development as it allows for the tool to be tested against different type of inputs to help determine what is working as intended and what is not. Testing has been ongoing throughout the entire development process of this Tool.

This section explains the tests that were conducted, the conclusions that were drawn and the changes they propelled. It is divided in two subsections, one focusing on the

tests conducted on the different techniques used and the other focusing specifically on the matching module of the Tool.

4.6.1 *String and Language Based Techniques Testing*

This subsection focuses on the String and Language Based Techniques that were tried and tested against different type of inputs, showing different examples in which they succeed and fail and how they were ultimately chosen to be used or not.

It is important to mention that these techniques accept two strings as an input and return a value ranging from 0 to 1 representing how similar those two strings are. The output value at which one considers a technique or function to be accurate enough is subjective and there is not one right answer as there is not also a perfect technique for each scenario and multiple can have successful results.

Every technique will be used against different examples that were considered to represent the majority of cases the Tool will face.

One Word Strings

This test makes use of pairs of one word strings. These cases appear often when comparing the IRI's of two ontology objects. The pairs used for testing were as follow:

$$(\text{"car"}, \text{"cars"})$$

$$(\text{"department"}, \text{"departament"})$$

$$(\text{"angel"}, \text{"angle"})$$

The first pair has two strings that represents the same information but one was defined in the singular and the other in the plural tense. The second pair has two strings that also represent the same information but one was misstyped. The thirds pair is the most challenging one as it can both represent a misstype as in the second exame or it can mean two different things. Although this last pair is very difficult to judge, if not impossible, it was still important to try to find a solution that best deal with it, hence why it was included in this testing process.

The Table 2 presents the results of using different String/Language Based Techniques, associating each technique with the number it attributes to each of the pairs. The higher the number, the higher the similarity it considers the two strings to have. The desired result is for the two pairs to have the highest similarity possible and for the third one to have the lowest.

Techniques	1st Pair	2nd Pair	3rd Pair
Editex	0.75	0.91	0.6
Jaro-Wrinkler	0.94	0.98	0.95
Levenshtein Distance	0.75	0.91	0.6
Overlap Coefficient	0.0	0.0	0.0
Generalized Jaccard	0.92	0.97	0.93
Fuzzy Token Sort	1.0	0.9	0.8

Table 2: String/Language Based Techniques - One Word Strings Testing.

The first thing to notice and conclude is that the **Overlap Coefficient** is not well suited for comparing these types of strings. This did not come as a surprised since we know this technique makes use of tokenization to then compare the overlap of the two resulting sets of tokens. In this case, the strings are composed of only word, resulting in the respective set being composed of only one token, the word itself, and therefore the sets do not match. Every other technique produces successful results in all the three pairs, so the choice of which to use will come down to the details. The **Levenshtein Distance** and the **Editex** techniques present the same results. They both achieve a low score in the third pair, which is attractive as that pair is the one we do not intend to consider matching, but the score on the first pair is lower than most of the other techniques. This disencourages of using these techniques as the first pair is the most common case the Tool is expected to encounter while the last one is a corner case that is expected to happen very seldom. The **Fuzzy Token Sort**, the **Generalized Jaccard** and the **Jaro-Wrinkler** techniques all achieve great scores when it comes to the first and second pair but fail identifying that the third pair is not supposed to match. The pros heavily outweigh the cons as identifying the first two pairs as matching is the most important thing as they are the most common cases. Strong arguments can be made to justify choosing any of the three remaining techniques. However since the first pair of strings is the case expected to be more common, the choice fell on the **Fuzzy Token Sort** technique has it achieves a perfect score in this pair.

Multiple Word Strings

This test makes use of pairs of multiple word strings. These cases appear when dealing with Labels or Descriptions of objects in the ontologies. The pairs used for testing were as follow:

$$(\text{"}house\ of\ my\ parents\text{"}, \text{"}my\ parent's\ house\text{"})$$

$$(\text{"}david\ smith\text{"}, \text{"}david\ richard\ smith\text{"})$$

The testing of strings whose length may vary by large quantities and can be composed of several worlds is incredibly difficult given that there are too many practical cases for

one to test. These two pairs of strings cover the two most interesting cases. The first case represents the scenario in which two strings have the same information that can be expressed in different ways. This is incredibly common as both strings are well written and the way they are written just depends on the user that writes them. The second scenario deals with the scenario where some information is omitted in one of the strings. This is also a very common occurrence in texts written by humans in a human readable way.

Techniques	1st Pair	2nd Pair
Editex	0.26	0.60
Jaro-Wrinkler	0.55	0.88
Levenshtein Distance	0.15	0.58
Overlap Coefficient	0.5	1.0
Generalized Jaccard	0.59	0.66
Fuzzy Token Sort	0.74	0.71

Table 3: String/Language Based Techniques - Multiple Words Strings Testing.

The first thing that pops up when analysing Table 3 is the scores produced by the techniques are much lower than the ones in Table 2. This is to be expected as the longer the strings are, the more characters in each string can differ.

Both edit distance techniques, **Editex** and **Levenshtein Distance**, presented very poor scores when dealing with the first pair and by that reason were not in contention to being used by the tool, in spite of showing good results in the second pair. The **Overlap Coefficient** technique got some very interesting results achieving the best result out of every technique regarding the second pair and getting an average result for the first pair. The final decision came down to choosing between the **Jaro-Wrinkler** and the **Fuzzy Token Sort** technique. On the one hand, the **Jaro-Wrinkler** achieved polarizing results, resulting in an average score for the first pair and a very good score for the second pair. On the other hand, the **Fuzzy Token Sort** was the more consistent technique all around, achieving the best score for the first pair out of every technique and an above average score for the second pair. Taking everything into account, consistency ended up mattering more and the **Fuzzy Token Sort** was, once again, the chosen technique.

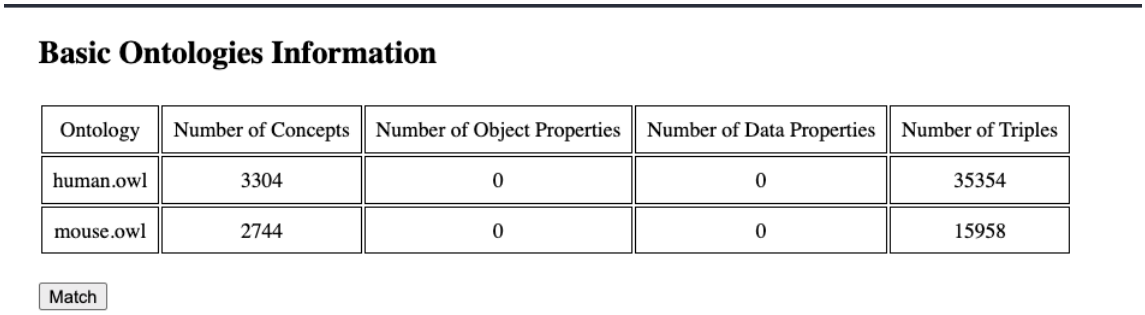
4.6.2 *Matching Module*

Testing and producing results for the OMT tool proved to be challenging due to being very hard to come across input ontologies designed for this purpose. This led to the ongoing testing throughout the development process being made with self made small ontologies

designed with the intent of testing specifics corner cases that may happen but also making use of the **OAEI (Ontology Alignment Evaluation Initiative)** [4]. This initiative every year produces datasets that can be used by Tools that wish to have their work benchmarked. A minority of these datasets are available to the public and one was used to check how this tool does against large input ontologies.

This subsection goes over this available ontology pair and the results the OMT tool produced compared to what was intended.

The ontologies input deal with an Anatomy problem [5] whose purpose is find matches between the anatomy of an Adult Mouse and of an Adult Human. These two ontologies are solely built to test the matching of concepts. The concepts all have the mandatory **IRI**, a **Label** providing a human readable description of the concept and hierarchical information that related concepts to each other by means of Super and Sub Concepts. These ontologies information lined up perfectly with the way the OMT works, since it attributes so much value and priority to the Labels of the Concepts.



Basic Ontologies Information

Ontology	Number of Concepts	Number of Object Properties	Number of Data Properties	Number of Triples
human.owl	3304	0	0	35354
mouse.owl	2744	0	0	15958

Match

Figure 13: Basic Information about Input Ontologies.

Figure 13 presents the Basic Information of the input ontologies. As to be expected, both ontologies do not have any Object or Data Properties and only have a very large quantity of Concepts.

The creators of these example input ontologies consider that the perfect matching results in 1516 concepts being matched while the others are merged into the final output ontology. Before presenting the results the OMT tool achieved, it is worth reminding that this tool matching algorithm is dependant on the matching threshold defined by the user. It is then crucial to test the inputs against different threshold values to obtain a better sense of what threshold might be the most accurate one.

4 http://oaei.ontologymatching.org/
5 http://oaei.ontologymatching.org/2016/anatomy/

Threshold	ConceptsMatched	ObjectPMatched	DataPMatched
50%	2654	0	0
60%	2418	0	0
70%	2173	0	0
80%	1756	0	0
90%	1231	0	0

Table 4: Tool matching results testings.

Table 4 presents the results based on the threshold value that was chosen. As can be seen, the lower thresholds of 50% and 60% are proven to be very lenient in the matching process, achieving scores way outside of what is intended. Meanwhile, the threshold of 90% proves to be too rigid. Therefore the threshold value of 80% appears to be the perfect middle ground between a too lenient and a too rigid threshold. achieving a result of 1756 concepts matched, a result very close to the one the ontologies creators intended.

To conclude, the testing process for the techniques to be used proved to be very useful to determine what techniques to use in the different scenarios that may be presented. The testing process for the tool's algorithm gave insurance that this tool is presenting positive and sucessful results dependending on the threshold value that is chosen but the lack of available example ontologies made it very hard to cover all the scenarios possible.

4.7 TECHNOLOGIES AND TECHNIQUES USED

Multiple different technologies and techniques were used to implement the OMT tool. This section will go over those technologies as well as explain what techniques were used in what scenarios.

4.7.1 *Technologies*

When it comes to building the web application what would host the tool, **Flask** [6] was the chosen one due to being a micro web framework that makes it very easy to build small web applications.

Moving onto the String and Language Based techniques, the library **StringMatching** [7] was used. It is a library built in Python that already implements a wide variety of techniques to compare the similarity of two strings.

6 https://flask.palletsprojects.com/en/2.2.x/
7 https://anhaidgroup.github.io/py_stringmatching/v0.4.x/index.html

Last but not least, **NLTK** [8] and its **Wordnet** [9] interface were used anytime it was required to check if two or more words were synonyms of each other.

4.7.2 *Techniques*

This subsection covers all the techniques that are used to calculate the similarity between two Strings.

To begin with, every String goes through a Language Based Technique called **Stop-world Elimination**. This is a very important step to homogenize the inputs before comparing them. It is done by eliminating or changing every character that is not considered relevant for the comparison. The inputs are converted to lower case and every character that is not a letter or a number is erased as well as consecutive whitespaces. Lastly, the specific words *and*, *the* and *'s* are also erased.

As shown in the Section 4.6, the technique that presented the best results was **Fuzzy Token Sort** and therefore this technique is used in every scenario that required two strings being compared for similarity.

8 https://www.nltk.org/
9 https://www.nltk.org/howto/wordnet.html

5

CONCLUSION

In this document, the area of ontology matching is explored. It is done so by, firstly, exploring the world of ontologies, how they came to be and what led to their development. It is explained how, without any ontology standards, the portability issue became overwhelming, leading to the first attempts at creating languages with the intent to express information in an ontology format, such as KIF, CycL and Ontolingua. Afterwards, the ontology matching problem is presented, alongside its motivation and its importance in today's world. In Section 2.3.1, some of the techniques used to find matches are presented, such as string, languages, constraints and instance based techniques. In section 2.4, some of already existing tools are analysed and presented, giving special importance to the type of inputs they process as well as the methods they use to reach the matching goal. Moving on from the ontology matching problem, our proposed solution to this problem alongside its architecture was presented, providing an explanation of all the components that make up our tool. After the proposed solution is presented, the development processed is explained. Firstly, focusing on the reasoning and logic behind the tool's algorithm for comparing and matching ontology objects and the techniques that were used to do so. Secondly focusing on an additional feature that was not planned from the start, the Log File. The Log File is a functionality of the OMT tool that allows the user to download detailed information about the matching process and all the decisions that led to the final merged ontology. Last but not least, an explanation of the Web Application that was developed to host the OMT tool is given providing insight on the different interfaces that it provides and how to use them.

This Master Thesis's two main objectives are considered achieved. The first objective led to the study of some of the already existing Ontology Matching Tools which allowed to compare the advantages and disadvantages of each and helped us set up the specific goals for our tool. These specific goals became to build a full autonomous tool that does not require the need for human interaction and allows multiple input ontology formats. In the context of this Master Thesis, a Paper was written and published presenting an early look at the prototype that would later become this tool as well as the study and research process of the state of the art. (Gomes et al., 2022)

Despite being pleased by the accomplishment of the objectives that were set as well as some extra features that were not planned from the beginning, there are still some features that can be worked on. For future work, every different module of the tool can be improved upon. Support can be added to more ontology languages in order for them to be acceptable by this tool. In the algorithms the tool uses, a more robust solution can be built that would make better use of the hierarchical information present in the ontologies as well as the individuals.

BIBLIOGRAPHY

Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah McGuinness, Peter Patel-Schneijder, and Lynn Andrea Stein. OWL Web Ontology Language Reference. Recommendation, World Wide Web Consortium (W3C), February10 2004. See http://www.w3.org/TR/owl-ref/.

Gavin Carothers and Eric Prud'hommeaux. RDF 1.1 turtle. February 2014. URL http://www.w3.org/TR/2014/REC-turtle-20140225/.

William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, 2003.

Isabel F. Cruz, Flavio Palandri Antonelli, and Cosmin Stroe. Agreementmaker: Efficient matching for large real-world schemas and ontologies. *Proc. VLDB Endow.*, 2:1586–1589, 2009.

Jérôme David, Fabrice Guillet, and Henri Briand. Matching directories and owl ontologies with aroma. pages 830–831, 01 2006.

Hong-Hai Do and Erhard Rahm. Chapter 53 - coma — a system for flexible combination of schema matching approaches. In Philip A. Bernstein, Yannis E. Ioannidis, Raghu Ramakrishnan, and Dimitris Papadias, editors, *VLDB '02: Proceedings of the 28th International Conference on Very Large Databases*, pages 610–621. Morgan Kaufmann, San Francisco, 2002.

Jérôme Euzenat, Pavel Shvaiko, et al. *Ontology matching*, volume 18. Springer, 2007.

Sean M. Falconer and Natalya F. Noy. *Interactive Techniques to Support Ontology Matching*, pages 29–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

Richard Fikes and Tom Kehler. The role of frame-based representation in reasoning. *Commun. ACM*, 28, 1985.

Michael Genesereth, Richard Fikes, Ronald Brachman, Thomas Gruber, Patrick Hayes, Reed Letsinger, Vladimir Lifschitz, Robert Macgregor, John Mccarthy, Peter Norvig, Ramangouda Patil, and Len Schubert. Knowledge interchange format version 3.0 reference manual. 09 1992.

João Rodrigues Gomes, Alda Lopes Gançarski, and Pedro Rangel Henriques. OMT, a Web-Based Tool for Ontology Matching. In João Cordeiro, Maria João Pereira, Nuno F. Rodrigues, and Sebastião Pais, editors, *11th Symposium on Languages, Applications and Technologies (SLATE 2022)*, volume 104 of *Open Access Series in Informatics (OASIcs)*, pages 8:1–8:12, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-245-7. doi: 10.4230/OASIcs.SLATE.2022.8. URL https://drops.dagstuhl.de/opus/volltexte/2022/16754.

Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *Int. J. Hum. Comput. Stud.*, 43:907–928, 1995.

Walid Hassen. Medley results for oaei 2012. In *Proceedings of the 7th International Conference on Ontology Matching - Volume 946*, OM'12, page 168–172, Aachen, DEU, 2012. CEUR-WS.org.

Sven Hertling. Hertuda results for oaei 2012. In *OM*, 2012.

Fumiko Kano Glückstad. Terminological ontology and cognitive processes in translation. pages 629–636, 01 2010.

Prodromos Kolyvakis, Alexandros Kalousis, and Dimitris Kiritsis. Deepalignment: Unsupervised ontology matching with refined word vectors. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 787–798, 2018.

Konstantinos Kotis, Alexandros G Valarakos, and George A Vouros. Automs: Automated ontology mapping through synthesis of methods. In *Ontology Matching*, page 96, 2006.

André Lara, Pedro Rangel Henriques, and Alda Lopes Gançarski. Visualization of ontology evolution using ontodi graph. 2017.

Douglas Lenat, Ramanathan Guha, Karen Pittman, Dexter Pratt, and Mary Shepherd. Cyc: Toward programs with common sense. *Commun. ACM*, 33:30–49, 08 1990. doi: 10.1145/79173.79176.

Douglas B. Lenat and Ramanathan V. Guha. Building large knowledge-based systems: Representation and inference in the cyc project. 1990.

Jorge Martinez-Gil, Ismael Navas Delgado, and Jose Aldana Montes. Maf: An ontology matching framework. *Journal of Universal Computer Science*, 18:194–217, 01 2012. doi: 10.3217/jucs-018-02-0194.

Usman Muhammad and Imran Khan. Similarity measures and their aggregation in ontology matching. *International Journal of Computer Science and Telecommunications*, 3:52–57, 05 2012.

Lorena Otero-Cerdeira, Francisco Javier Rodríguez-Martínez, and Alma María Gómez-Rodríguez. Ontology matching: A literature review. *Expert Syst. Appl.*, 42:949–971, 2015.

Willem van Hage, Antoine Isaac, and Zharko Aleksovski. Sample evaluation of ontology-matching systems. pages 41–50, 01 2007.

Peng Wang. Lily results on seals platform for oaei 2011. In *OM*, 2011.

Lu Zhou, Michelle Cheatham, and Pascal Hitzler. *Towards Association Rule-Based Complex Ontology Alignment*, pages 287–303. 02 2020. ISBN 978-3-030-41406-1. doi: 10.1007/978-3-030-41407-8_19.