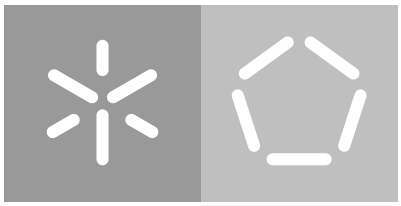


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Simão Freitas Monteiro

Multilingual ontologies creation

January 2024



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Simão Freitas Monteiro

Multilingual ontologies creation

Master dissertation
Master Degree in Informatics Engineering

Dissertation supervised by
Pedro Rangel Henriques
Alda Gancarski

January 2024

AUTHOR COPYRIGHTS AND TERMS OF USAGE BY THIRD PARTIES

This is an academic work which can be utilized by third parties given that the rules and good practices internationally accepted, regarding author copyrights and related copyrights.

Therefore, the present work can be utilized according to the terms provided in the license bellow.

If the user needs permission to use the work in conditions not foreseen by the licensing indicated, the user should contact the author, through the RepositóriUM of University of Minho.

License provided to the users of this work



Attribution-NonCommercial
CC BY-NC
https://creativecommons.org/licenses/by-nc/4.0/

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.

Simão Freitas Monteiro

_____

ACKNOWLEDGEMENTS

I would like to express my gratitude to Professor Pedro Rangel Henriques and Professora Alda Gancarski for all their sympathy, availability, encouragement and knowledge, which were essential for the development of this Master's Thesis.

I would also like to thank all my friends and family who have always supported me and believed in me, making this whole journey a lot more easier.

ABSTRACT

Semantic Web multilingual information has been growing in last years. Consequently we assist to the need of creating multilingual ontologies from monolingual ones in order to allow for the semantic interoperability between applications in different natural languages. Cross-lingual enrichment is however often based in human interaction, which is a cumbersome and error-prone task.

This document presents a study of already existing multilingual ontology creation methods in order to gain knowledge on what techniques exist, pointing the advantages and disadvantages of each one. From the knowledge obtained with the bibliographic research, a new web-based method was proposed and implemented. The detailed description of this development process is the main content of the present Master's dissertation.

The Multilingual Ontology Creation Tool produces multilingual ontologies in two different ways. The first is based on translating certain elements of an input ontology and the other is done through a triangulation process between two existing multilingual ontologies. In the first approach, that involves translation, a LogFile is also generated offering to the user the possibility to download it for future examination. This LogFile contains detailed information concerning translation choices the system has made that can affect the quality of the translations. The tool introduced and discussed in this dissertation was tested throughout its the development. Lastly, a web application was developed to host the tool in order to facilitate its access and use for the users.

Key-Words: Ontology, Multilingual Ontology, Multilingual Ontology Creation Tools

RESUMO

A informação multilingue da Web semântica tem vindo a aumentar nos últimos anos. Consequentemente, assistimos à necessidade de criar ontologias multilingues a partir de ontologias monolingues, a fim de permitir a interoperabilidade semântica entre aplicações em diferentes línguas naturais. No entanto, o enriquecimento multilingue baseia-se frequentemente na interação humana, o que é uma tarefa complicada e propensa a erros.

Este documento apresenta um estudo dos métodos de criação de ontologias multilingues já existentes, com o objetivo de conhecer as técnicas existentes, apontando as vantagens e desvantagens de cada uma. A partir do conhecimento obtido com a pesquisa bibliográfica, foi proposto e implementado um novo método baseado na Web. A descrição detalhada deste processo de desenvolvimento é o conteúdo principal da presente dissertação de mestrado.

A Ferramenta de Criação de Ontologias Multilingues produz ontologias multilingues de duas formas diferentes. A primeira baseia-se na tradução de certos elementos de uma ontologia de entrada e a outra é feita através de um processo de triangulação entre duas ontologias multilingues existentes. Na primeira abordagem, que envolve a tradução, é também gerado um LogFile que oferece ao utilizador a possibilidade de o descarregar para análise futura. Este LogFile contém informação detalhada sobre as escolhas de tradução que o sistema fez e que podem afetar a qualidade das traduções. A ferramenta apresentada e discutida nesta dissertação foi testada ao longo do seu desenvolvimento. Por fim, foi desenvolvida uma aplicação web para alojar a ferramenta, de modo a facilitar o seu acesso e utilização por parte dos utilizadores.

Palavras-Chave: Ontologia, Ontologia Multilingue, Ferramentas de criação de ontologias multilingues

CONTENTS

1 Introduction 1
- 1.1 Motivation 2
- 1.2 Objectives 2
- 1.3 Research Hypothesis 2
- 1.4 Methodology 2
- 1.5 Document Structure 3

2 State of the Art 4
- 2.1 Ontologies 4
  - 2.1.1 KIF (Knowledge Interchange Format) 6
- 2.2 Ontology Standards 7
  - 2.2.1 RDF 7
  - 2.2.2 OWL 8
  - 2.2.3 Turtle 9
- 2.3 Development of Ontologies 9
- 2.4 Multilingual Ontologies 11
  - 2.4.1 Cross-lingual ontology enrichment 12
  - 2.4.2 Ontology Localization 13
- 2.5 Existing tools 16
  - 2.5.1 LabelTranslator 16
  - 2.5.2 SOMMO 18
- 2.6 Summary 18

3 Proposed Approach 20
- 3.1 System Architecture 20
  - 3.1.1 System inputs 20
  - 3.1.2 Translation service 21
  - 3.1.3 Translation Ranking 21
  - 3.1.4 Translation Inference 21
  - 3.1.5 Output generator 21

4 Development 23
- 4.1 Translator 23
  - 4.1.1 Extraction of information 23
  - 4.1.2 Translate the extracted elements 26
  - 4.1.3 Technologies used 27

4.2 Ranking 28
4.3 Output generator 31
4.4 Logfile 33
4.5 Inference/Triangulation 34
4.5.1 Inference Algorithm 34
4.6 Web Interface 39
4.6.1 Home Page 39
4.6.2 Triangulation Process Page 41
4.7 Case study 42
4.8 Technologies used 47
5 Conclusion 48
A Case Study: Ontology about basketball 50
B Visualisation of the Basketball Ontology 68

LIST OF FIGURES

Figure 1	Sample of the Basketball Ontology OWL document	5
Figure 2	Graphic view sample of the example Basketball Ontology	6
Figure 3	Proposed tasks for the Ontology localization activity by (Espinoza et al., 2009)	15
Figure 4	Example tuples of some translations of the ontology label "chair" into Spanish taken from (Espinoza et al., 2008)	17
Figure 5	General overview of the system architecture	20
Figure 6	Two lines of an English-Portuguese ontology	38
Figure 7	Two lines of an English-Italian ontology	38
Figure 8	Two lines of an Italian-Portuguese ontology resulting from an inference between the other two	38
Figure 9	Home page	39
Figure 10	ToolTip Text from Ontology Context	40
Figure 11	Example Output box with buttons to download	41
Figure 12	Triangulation process page	42

ACRONYMS

K

KIF Knowledge Interchange Format.

O

OL Ontology Learning.

OWL Web Ontology Language.

R

RDF Resource Description Framework.

U

URI Uniform Resource Identifier.

W

W3C World Wide Web Consortium.

1

INTRODUCTION

Unlike the meaning in philosophy, in computer science an ontology is a way of showing the properties of a subject and how they are related, by defining a set of concepts and categories that represent the subject. Structurally an ontology is mainly composed of individuals, classes, attributes and relations.

Due to a wide proliferation of multilingual data on the Semantic Web, there are many ontologies on the web in different natural languages. These monolingual ontologies sometimes have labels or local names presented in a certain language that are not easily understandable to speakers of other languages. One of the main goals of the creation of multilingual ontologies is to allow interoperability between the monolingual ones. Looking from a boarder perspective, these multilingual ontologies can also allow interoperability between different systems as the usage of ontologies is increasing in several different fields, for example in web based applications.

One of the most popular ways to create multilingual ontologies is through the application of cross-lingual ontology enrichment techniques (Ibrahim et al., 2019a) which expand the target ontology with additional concepts and semantic relations extracted from external resources in other natural languages. However, most of this is done manually and it becomes a very resource demanding and time-consuming task. Therefore, fully automated cross-lingual ontology enrichment approaches are highly desired.

Most of the literature have focused on enriching monolingual ontologies with multilingual information in order to translate those ontologies to make more effective the search for concepts or relations. In addition, in the cross-lingual ontology matching task, there is a lack of exact one-to-one translation between terms across different natural languages which negatively affects the matching results (Keet, 2020). For instance, in French, there are two words for the English '**river**': one for a river that ends in the sea and another word for a river that doesn't (**fleuve** and **rivière**). Small problems like these mean there is still plenty of room for improving existing solutions.

With this in mind, this project 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 more accurate solution with less margin of error.

1.1 MOTIVATION

The motivation for this thesis arises from the fact that there is still room for improvement in the field of multilingual ontology creation. There are a lot of tools developed and available, but most of them require manual intervention to achieve high accuracy results and most of the times is a very time consuming task. The main objective going forward is to build upon these tools and create new techniques that make the multilingual ontologies creation tools as much automatic as possible.

1.2 OBJECTIVES

The main objective of this thesis is the design of a methodological and technological approach to create multilingual ontologies. In order to accomplish this objective, the following steps are required:

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

1.3 RESEARCH HYPOTHESIS

By the end of this Master's Thesis, it will have been proven that is possible to create a multilingual ontology creation algorithm that allows accurate semantic translation of words/expressions and works in the most automatic way possible.

1.4 METHODOLOGY

The methodology used to achieve the objectives was composed of the following steps:

- Bibliographic search;
- Study and analysis of the techniques found in the research;
- Proposal of a new multilingual ontology creation technique and tool;
- Creation of the web-based application that implements the newly created multilingual ontology creation approach;
- Test of the developed application.

As this is an incremental method, problems detected in any phase will imply to go backand repeat previous tasks untial a proper solution is reached.

1.5 DOCUMENT STRUCTURE

In Chapter 1, an introduction to the domain of this Master's work is made and what was the motivation that led to its development.

In Chapter 2, an introduction to the world of ontologies is made, followed by a brief summary of some of the earliest information representation languages that were created. In the following sections of that chapter, some multilingual ontologies creation techniques such as cross-lingual enrichment are explored as well as some of the already existing multilingual ontology creation tools are presented.

In Chapter 3, a solution is proposed, giving a brief overview of the general architecture of the multilingual ontology creation tool. The components that constitute it are also discussed.

In chapter 4, the development process is described, giving particular emphasis on the decisions made to implement the proposed solution.

In chapter 5, an overview of the document is given as well as directions for future work.

2

STATE OF THE ART

This chapter is a representation of the bibliographic research work done in order to obtain useful information necessary to propose a new solution and then implement it. The first section begins with a brief introduction about ontologies, what they are, how they appeared and why they became so important these days. Then it presents some of the ontology standards that are used nowadays as well as some of the more used knowledge representation languages. In the final section of this chapter, multilingual ontology creation techniques are studied and presented as well as already existing multilingual ontology creation tools.

2.1 ONTOLOGIES

The term "ontology" first appeared in the field of philosophy that studies the being, as well as related concepts such as existence, becoming, and reality. In computer and information science, ontology is a technical term used to refer to an object that is created with the intention of facilitating the modelling of knowledge about a certain domain, whether it be real or hypothetical (Gruber, 2009). Ontologies are explicit specifications of conceptualizations and their representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members).

The term **ontology** was adopted by the AI community in the 1980s to describe both a theory of a modelled world and a component of knowledge systems. 10 years later, in the early 1990s, an initiative to develop interoperability standards identified a technological stack that pointed out the ontology layer as a required element of knowledge systems.

Ontologies play a crucial role in defining a data modelling representation at a level of abstraction above particular database designs (either logical or physical), allowing for the export, translation, querying, and unification of data across several independently created systems and services. Database interoperability, cross-database searching, and the incorporation of web services are examples of successful implementations to date.

Although ontologies were already being used as a way to represent information, there was no standard language or model to design them thus preventing them from being shared

or used in systems other than those in which they were developed. Seeing the importance of ontologies as a way to represent information and realising how useful they are, the first languages to develop them quickly emerged. Typically, ontologies are expressed in languages that are more expressive than logical formalisms. This enables the ontology designer to express semantic constraints without being forced to choose a specific encoding method.

In order to serve as an example throughout this document, the ontology depicted as a graph in Figure 2 about basketball is defined in the listing shown in Figure 1 using the OWL language, a standard language to represent ontologies that will be discussed further on.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <Ontology xmlns="http://www.w3.org/2002/07/owl#"
4      xml:base="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol"
5      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
6      xmlns:xml="http://www.w3.org/XML/1998/namespace"
7      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
8      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"  ontologyIRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol">
9      <Prefix name="" IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol"/>
10     <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
11     <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
12     <Prefix name="xml" IRI="http://www.w3.org/XML/1998/namespace"/>
13     <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
14     <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
15     <ClassAssertion>
16         <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/Player"/>
17         <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/Schroder"/>
18     </ClassAssertion>
19     <SubClassOf>
20         <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/PointGuard"/>
21         <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/Position"/>
22     </SubClassOf>
23     <ObjectPropertyDomain>
24         <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/playsAs"/>
25         <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/Schroder"/>
26         </ObjectPropertyDomain>
27     <ObjectPropertyRange>
28         <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/playsAs"/>
29         <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/PointGuard"/>
30     </ObjectPropertyRange>
31     <ObjectPropertyAssertion>
32         <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/playsFor"/>
33         <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/Schroder"/>
34         <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/uminho"/>
35     </ObjectPropertyAssertion>
```

Figure 1: Sample of the Basketball Ontology OWL document

The example exhibited in Figure 1, on lines 16 and 17, it can be seen a "Player" called "Schroder". It is also possible to see, from line 23 to 30, that "Schroder" "playsAs" "PointGuard" being "PointGuard" a subclass of "Position" as can be seen on lines 20 and 21. Finally, from line 31 to 35, it can be seen that "Schroder" plays for the team "uminho". In the rest of the ontology, present in Appendix A, it is possible to observe more relations of this kind as well as other concepts about basketball. An entire graphic view of the ontology is also displayed in Appendix B.

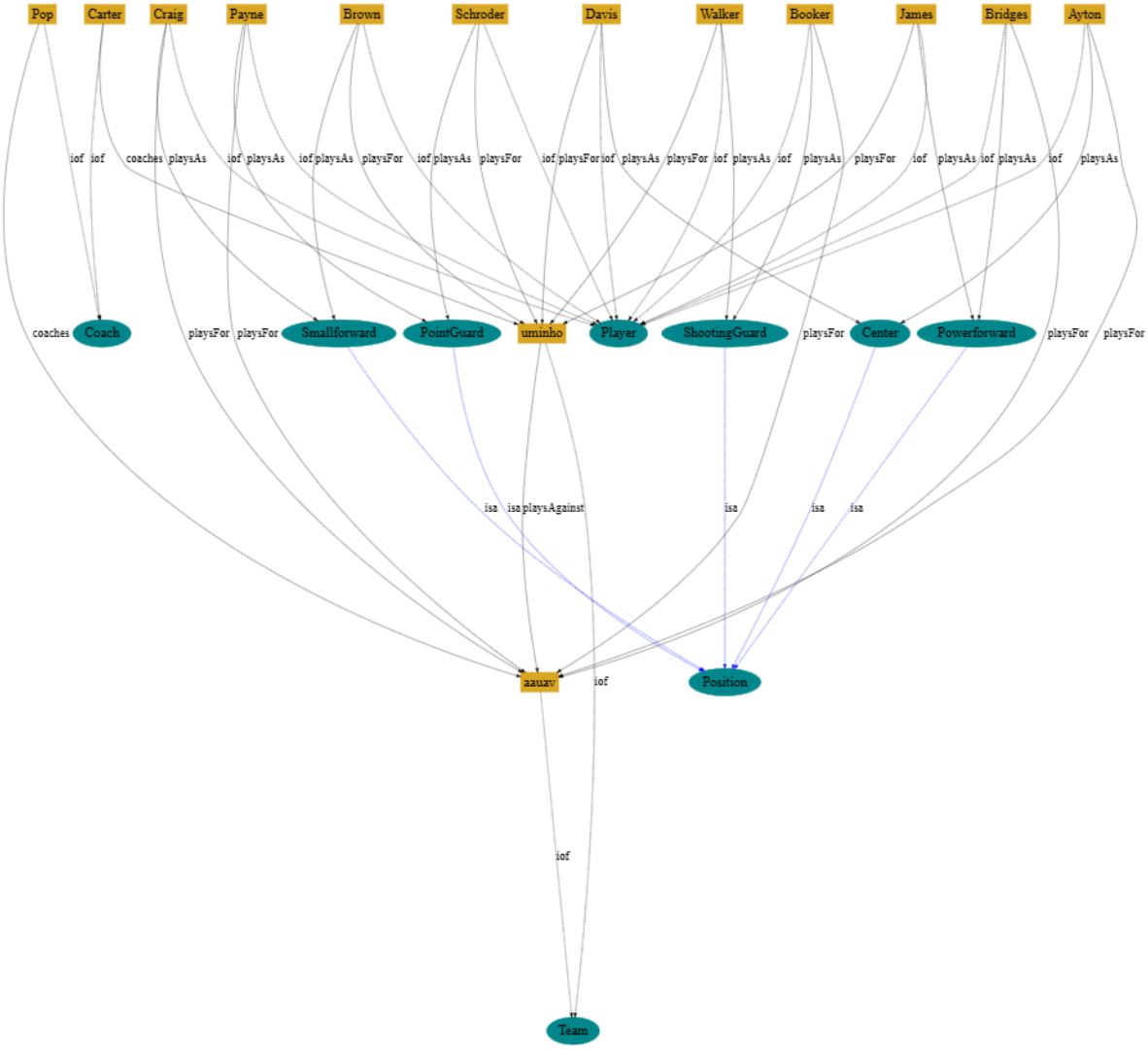


Figure 2: Graphic view sample of the example Basketball Ontology

In the next sections it will be presented some of the first system/languages that have appeared to represent and share knowledge bases.

2.1.1 *KIF (Knowledge Interchange Format)*

Knowledge Interchange Format (KIF) was created by Michael Genesereth, Richard Fikes and others in 1992 and it is a computer language created to make it possible for systems to exchange and reuse knowledge from knowledge-based systems (Genesereth et al., 1992).

There are three general essential features in KIF language:

- Translatability - provide effective ways to translate declarative knowledge bases between common knowledge representation languages and vice versa.
- Readability - designed to make it easier for people to read and understand what is being represented.
- Usability as a representation language.

A KIF program is a collection of expressions made up of rules, variables, and operators. Variables are preceded by the character "?" or "@" and are called individual variable and sequence variable respectively. Some operators are for example "and", "or" and "=>", "<=" and "<=>" which indicate implications. To see the full manual about KIF language, refer to (Genesereth et al., 1992).

The example in Figure 2.1 of KIF code is taken from (Gruber, 1995).

```
(deffunction QUANTITY.MAGNITUDE
    (=> (and (defined (QUANTITY.MAGNITUDE ?q ?u))
             (= (QUANTITY.MAGNITUDE ?q ?u) ?m))
        (and (physical-quantity ?q)
             (unit-of-measure ?u)
             (magnitude ?m))))
```

Listing 2.1: KIF syntax example

2.2 ONTOLOGY STANDARDS

Languages like KIF were the basis for the development of standards that are used nowadays. This section will look at different models and languages that are currently being used as standards for ontology representation.

2.2.1 *RDF*

The World Wide Web Consortium (W3C) is promoting the semantic web as a way to produce more online machine-processable information. The key to make Semantic Web possible is the Resource Description Framework (RDF). For those who maybe asking what Semantic Web means, in a few words it is the idea that there should be meaning in the web. In other words, the information we see on the web is easily interpreted by humans but does not have any meaning for machine communication.

RDF offers a logic for organising data that, when applied to particular situations, would always lead to the usage of methods for describing the information that would make it easier for machines to process it.

In an RDF model there are three main type of things we can look at:

- **Resources** are identified in an RDF model by an Uniform Resource Identifier (URI) that is a unique sequence of characters and can identify everything from web pages to pictures to people.
- **Properties** are specific features of a resource and each property has its own meaning.
- **Values** that a property takes and it can either be a literal, i.e. a string, or another new resource.

The RDF data model is based on the already established concepts of classes and entity relationships, and it has condensed these into a structure known as a triple. These triple allows the creation of statements and follows the following structure:

Subject-Predicate-Object

The resource about which the assertion is being made is designated as the Subject in this structure. The Predicate describes the kind of relationship the Subject and Object have.

Resources and property types cannot be specified using RDF. You can associate a property with a resource after you have an instance of it. However, you cannot indicate what kind of resource it is or what kind of property it is. RDF Schema is responsible for doing that.

To see more about RDF and RDF Schema, see (W3C, 2014a), (W3C, 2014b) and (Krichel, 2002).

2.2.2 *OWL*

Web Ontology Language, usually known as OWL, is an ontology language that follows the RDF standard and was created to represent and share knowledge over the Internet. The current version of OWL, also referred to as "OWL 2", was published in 2009 and it is a revision and extension of the previous one. Like RDF, OWL is also part of the W3C Semantic Web technology stack.

There are 3 sublanguages of OWL: **OWL Lite**, **OWL DL** and **OWL Full**. The decision between using OWL Lite or OWL DL is based on how much the users need the more expressive constructs that OWL DL offers. The decision between using OWL DL or OWL Full primarily depends on how much a user needs RDF Schema's meta-modeling capabilities.

A series of annotations, axioms, and facts make up an OWL ontology in the abstract syntax. OWL annotations allow users to record information such as ontology authorship or references to other ontologies.

The axioms and facts of an OWL ontology carry the most of the ontology's content, where it is provided details about the classes, attributes, and individuals in the ontology.

Facts can be used for two different purposes. The first type of fact provides details about a certain individual, including the classes to which they belong as well as their properties and values. The second type of fact can be used to distinguish whether two individuals identifiers represent the same person or two different ones.

Axioms are used to give information about classes and properties and their abstract syntax varies depending on the OWL sublanguage is being used. Properties can have global domains and ranges, be functional, inverse functional, symmetric, or transitive, and be identical to subproperties of other properties.

To see more about OWL refer to (W3C).

2.2.3 *Turtle*

Turtle is a language developed to give RDF a textual syntax, allowing an RDF graph to be completely written in a compact and natural text form. Turtle, which does not rely on XML like RDF/XML, is widely regarded as being more readable and simpler to change manually than XML. Turtle offers a mechanism to combine three URIs into a triple (Listing 2.2) and strategies to compress such information, such as by factoring out common portions of URIs.

```
1 <http://example.org/player/James>
2   <http://example.org/relation/playsFor>
3   <http://example.org/team/uminho> .
```

Listing 2.2: Triple represented in Turtle

To see more about Turtle refer to (W3C, 2014c).

2.3 DEVELOPMENT OF ONTOLOGIES

An ontology defines a common vocabulary for people that need to share knowledge within a domain. It provides relations between basic concepts in the domain and machine-interpretable definitions of those concepts. The fact that this definitions can be machine-interpretable make ontologies play a big role on the Semantic Web (Gruber, 2006).

In the following subsections, several processes that contribute to creation of ontologies will be shown and explained.

As manual construction of ontologies is extremely laborious and time consuming, there is a great motivation to automate the process, and so ontology learning emerges. Ontology learning is a wide research area that can be divided in three different works: ontology

enrichment, inconsistency resolution and ontology population. Ontology enrichment is known as the process of adding new concepts and semantic relations to an existing ontology and positioning them in the right places within the ontology. Inconsistency resolution is the task of resolving inconsistencies that appear in an ontology. Ontology population is adding new instances of concepts to the ontology. To see some more complete work about ontology learning refer to (Shamsfard and Barforoush, 2003), (Gómez-Pérez, 2003) and (Cimiano, 2006).

To (semi-)automatically extract complete ontologies from natural language literature, ontology learning (OL) is utilized (Cimiano et al., 2006). The procedure is typically broken down into the following eight activities, though not every ontology learning system will necessarily use them all.

Domain terminology extraction

During the domain terminology extraction step, domain-specific terms are extracted, for example by calculation of the TF/IDF, and after, the resulting list of terms needs to be filtered by a domain expert. The next phase in the OL system's process is to identify synonyms because they have the same meaning and hence describe the same concept. Therefore, clustering and the use of statistical similarity metrics are the most widely used techniques.

Concept discovery

In the concept discovery phase terms are grouped into meaning-bearing units which translates to concepts. These domain-specific terms and their synonyms, which were found during the domain terminology extraction stage, form the grouped terms.

Concept hierarchy derivation

In the concept hierarchy derivation step, the OL system tries to arrange the extracted concepts in a taxonomic structure with unsupervised hierarchical clustering methods. Such algorithms frequently produce noisy results, hence an additional supervisory step—such as user evaluation—is included. The use of numerous patterns that should denote a sub- or supersumption relationship is another way to derive a concept hierarchy. Patterns like "X is a Y" indicate that X is a subclass of Y. Although such patterns can be easily analysed, they typically occur too rarely to be able to derive sufficient sub- or supersumption relationships. Instead, bootstrapping techniques are created, which provide greater coverage by automatically learning these patterns.

Learning of non-taxonomic relations

In this step, relationships that don't express any sub- or supersumption, e.g. plays-for, are extracted. There are two typical methods for resolving this subtask. First, anonymous relationships are extracted and then, in a subsequent phase, correctly labelled. The second method pulls out verbs that show a connection between the entities represented by the words around them. An ontologist must examine the outcomes of both methods to assure accuracy.

Rule discovery

Axioms, which are formal descriptions of concepts, are created for the retrieved concepts during rule discovery. This method yields a set of axioms, which is later understood to be a concept description. An ontologist then examines this output.

Ontology population

The ontology is now enriched by examples of concepts and properties. Methods based on the matching of lexico-syntactic patterns are used for the addition of instances of concepts. Applying bootstrapping techniques, which gather relation tuples, adds instances of properties.

Concept hierarchy extension

The OL system attempts to add more concepts to an established ontology's taxonomic structure in this step. This can be done in one of two ways: supervised (using a trained classifier) or unsupervised (using the use of similarity metrics).

Frame and Event detection

The OL system attempts to extract complex associations from text during frame/event detection, such as who left where and when from what location.

2.4 MULTILINGUAL ONTOLOGIES

Nowadays there is a large amount of people using the internet that speak native languages other than English making the Web a multilingual platform where people from different countries and with different backgrounds can exchange and produce lots of information. A key objective is to create a level playing field where people from all nations have equal access to information and opportunity to contribute to that information. However, most of the factual data on the Semantic Web is in English, so there is a considerable bias in favour

of this language and cultures with an English-based foundation. Multilingual ontologies are useful to solve this problem because they allow the information to be available in several languages, thus enabling greater portability and interoperability between different programs that may use these ontologies.

2.4.1 *Cross-lingual ontology enrichment*

One of the most used techniques for the construction of a multilingual ontology is through the use of cross-lingual enrichment applied to a monolingual ontology. Cross-lingual ontology enrichment facilitates the semantic compatibility of various ontologies in various natural languages. Cross-lingual ontology enrichment techniques applied to a monolingual ontology aims to expand it with additional concepts and semantic relations extracted from resources in other natural languages. Imagining a user wants to extract information from two ontologies in two different languages, only the usage of cross-lingual enrichment techniques allow the creation of a more complete ontology, for example, enriching one of the ontologies with information present in the other ontology. However, this is time-consuming work and to solve this problem there are already some works that attempt to automate this task (Ibrahim et al., 2019a).

To achieve best and more reliable results, manual enrichment approaches are preferable. To enrich an ontology it is necessary to have a notion of the field you are dealing with and therefore the presence of qualified people who know the domain will produce better results both in the translation of documents and in the creation of new specific terms to add to the ontology. Therefore, although automated solutions save a lot of work, it is essential that there is always some manual intervention of a domain expert, in order to review and accept or reject the system's proposals.

Imagine it is desired to enrich an ontology S with information present in an ontology T that is in a different language of S.

Pre-matching

Given these two ontologies, the first step is to prepare the ontologies. These preparement can be divided in two phases: **pre-processing** and **translation**.

The goal of the pre-processement is to prepare the local names and labels of the concepts of T and S by applying several natural language processing techniques(NLP), such as tokenization, normalization, stop words removal and POS-tagging (part-of-speech tagging). After having two sets of pre-processed concepts, *C't* and *C's*, the translation phase can start.

In the translation phase, each class in *C's* is translated, using tools like Google translator, to the language of T. A list of translations is associated with each concept, for example, the concept label "treinador" in Portuguese, has a list of two possible English translations:

"coach" and "trainer". The best translation will be chosen further on through the usage of matching techniques.

Matching

In order to identify which, and where the new information will be added to T, potential matches between S and T should be identified. The two most used matching techniques are **terminological matching** and **structural matching**:

- **Terminological matching:** In this task it is chosen which information can be added to T. To choose the best translation for each concept that matches the corresponding one in T, it is performed a pairwise string matching between them. One of the most popular metrics chosen is the Jaccard similarity metric that measures the similarity between two strings. According to the tests done for the MultiFarm benchmark's ontology alignment task in (Cheatham and Hitzler, 2013), Jaccard similarity has achieved the best score in terms of precision.

  Being S and T the respective set of tokens of the strings X and Y, the Jaccard Index is obtained by the following formula:

  $$Jac(x, y) = \frac{|X \cap Y|}{|X \cup Y|}$$

  After running several experiments, the best matches are found, however it is necessary the approval of an expert to confirm the best translation was selected. If no match is found that means that a new concept was found and it will be added to T.

- **Structural matching:** Previously the goal was to find which information could be added to T and now the goal is to find where the information can be added. In order to get a translated ontology S_{trans}, each concept in S is replaced by the best translation found in the previous phase. Then it is made a comparison between the triples in S_{trans} and T, and then it is decided whether a triple needs to be enriched or needs to be added as q a new concept to T.

OECM is a cross lingual ontology enrichment tool and it is an implementation of this techniques. To read more about this work please refer to (Ibrahim et al., 2019b).

2.4.2 *Ontology Localization*

According to (Espinoza et al., 2009), ontology localization is defined as "the adaptation of an ontology to a particular language and culture". This adaptation can be in a political,

social or cultural context or simply focused on language. The focus of this section is mainly on the latter aspect, the language.

Ontology localization is a process that encourages the reuse of ontologies that are already developed for the target domain rather than creating new ones from scratch. As a result, ontology localization also has an economic component because it has the potential to be less expensive than creating a whole new ontology for the target community.

Ontology localization can have an impact on either the conception itself or the ontology's surface, or lexical layer. The lexical layer of an ontology is defined in (Cimiano et al., 2010), as all the labels, definitions, and supporting documentation written in natural language that enable human comprehension of the ontology. Since language is the tool we use to understand and experience reality, any disparities at the conceptualization layer caused by differences in the cultural or geopolitical background would have an effect on the lexical layer. On the other hand, changes in the lexical layer are very unlikely to cause changes on other layers.

Even though changes on the lexical layer usually do not affect other layers, this can happen and in that scenario some decisions need to be made. Imagine an example taken from (Cimiano et al., 2010), where we have a geographical ontology in French. French speakers may be more likely to distinguish between rivers that flow into the sea (fleuve) and rivers that flow into other rivers (rivière) in their ontologies. A French ontology designer is more likely to include this distinction in the ontology because it is directly lexicalized in French (in contrast to languages like English, Spanish, or German). An ontology engineer has two fundamental options when localising this ontology into another language:

- Preserving the notion of rivers flowing into other rivers versus rivers flowing into the sea. This means that, rather than using a single designation as the label for each concept, there will be a paraphrase in the target languages instead of direct lexicalization.
- Remove this distinction and merely retain the idea of a river, without further distinguishing rivers that run into the sea from rivers that flow into other rivers.

The choice will be made by taking into account whether the distinction between rivers in actual life is an important one when taking into account the applications that the target ontology is expected to support.

According to (Espinoza et al., 2009), the goal of the ontology localization activity is to translate an ontology expressed in a source natural language into a target natural language. The input should be an ontology whose labels are expressed in one or several natural languages, from which one is selected as source natural language. The expected output is an ontology whose labels have been correctly translated to the target natural language. The resulting translations are added to available labels of the original ontology. This ontology

localization activity should be done by software developers and ontology practitioners in collaboration with domain and linguistic experts.

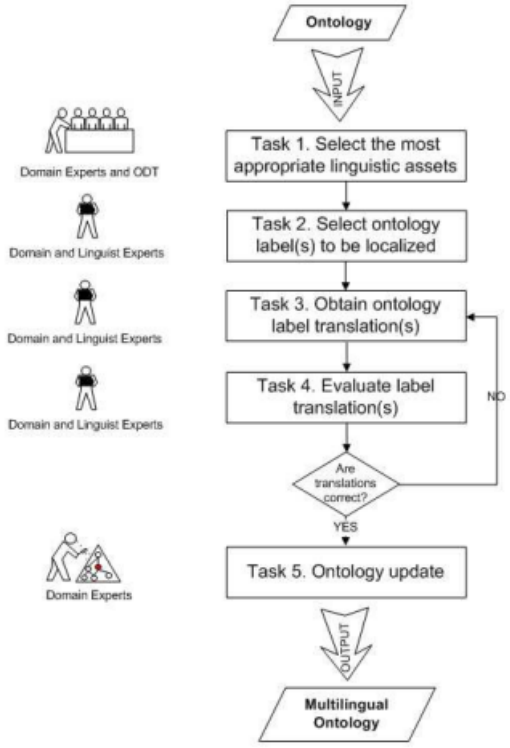


Figure 3: Proposed tasks for the Ontology localization activity by (Espinoza et al., 2009)

In Figure 3, taken from (Espinoza et al., 2009), a proposed set of tasks for the ontology localization activity is shown. This is a very general system and serves as a good example for what ontology localization activity is.

The goal of the task number 1 is to choose linguistic resources that will be most helpful in adapting a product or service for a specific locale or market. These resources are selected based on their level of consensus among the community, their coverage of different languages and features, and their ability to accurately identify lexical differences between languages. The output is a set of linguistic assets that can improve the efficiency and consistency of the Localization Activity.

Then, task Number 2 involves selecting labels in an ontology that need to be translated from a source language to a target language. The task is performed by domain experts and the ontology development team, and the output is a set of labels and their context within the ontology. The context is a small excerpt of surrounding labels that describes the meaning of a specific label.

Task Number 3 aims to find the best translation in a target language for each label in an ontology. It is performed by domain and linguist experts, who use various techniques such

as cross-language term extraction, word sense discovery, and word sense disambiguation. The output is a ranked set of translated labels.

Task Number 4 evaluates the translations of labels in a target language, using linguistic criteria and tests that can be automated when possible. It is carried out by domain and linguist experts, and the output is a set of labeled translations with their corresponding evaluations.

Finally, the goal of task number 5 is to update an ontology with translated labels in a target language. It is performed by domain experts using selected translations as input, and the output is an updated ontology with labels in the target language.

An implementation of this model is shown in Sec.2.5.1.

2.5 EXISTING TOOLS

The presentation of some of the currently existing multilingual ontology creation tools follows, with an emphasis on the methodologies employed by each tool and some of the system's limitations.

2.5.1 *LabelTranslator*

When given an ontology with labels that are described in a source natural language, LabelTranslator (Espinoza et al., 2008) determines the most likely translation of each label into the target natural language. This system aims to automate the process of ontology localization which reduces the human efforts to do this task manually.

The system essentially consists of the following stages:

- the user chooses the label of the ontology term(s) to be translated;
- the translation service accesses bilingual or multilingual linguistic resources to look for the possible translations of the selected label(s);
- the ranking method compares the obtained translation possibilities against the label(s) in the original language and its ontological context;
- the user confirms the translation proposed by the ranking method, or chooses the translation that better fits in the ontological context;
- the system updates the ontology model with the selected translation(s) and additional linguistic information;
- finally, the linguistic information associated with ontology term(s) is shown to the user.

A set of potential translations $T = [t1, t2, ..., tn]$ in a target language are returned by the translation service given an ontology label l specified in a source language as input. Currently the system supports translations among English, Spanish and German. If the translation service does not find any translation, the user can type his own translation.

An ontology label represented by the symbol t is a tuple $< trs, senses >$, as shown in Fig.4, where *trs* is the translated label and *senses* a list of semantic senses extracted from different knowledge pools. Each sense is represented as a tuple $sk =< s, grph, descr >$, where s is the list of synonym names[1] of word k, *grph* describes the sense *sk* by means of the hierarchical graph of hypernyms and hyponyms of synonym terms found in one or more ontologies, and *descr* is a description in natural language of such a sense. To discover the senses of each translated label they consider an approach proposed in (Trillo et al., 2007).

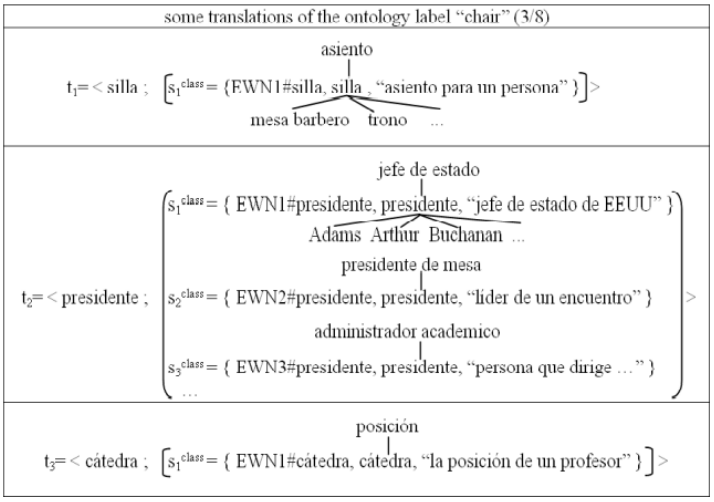


Figure 4: Example tuples of some translations of the ontology label "chair" into Spanish taken from (Espinoza et al., 2008)

The system takes a list of words as input, determines their meanings in real time, and then extracts a list of senses from several ontology pools, taking into account any potential semantic overlap between senses. An overview of this process can be consulted in Espinoza et al. (2008). For compound labels the system either treats them as single words if they have an entry in linguistic ontologies like EuroWordNet (Vossen, 1997), or translates them using a compositional method. This method splits the label into tokens, translates this tokens individually and then combines them again, taking into account the word order of the target language, to form the compound label in the target language.

After the system finds all the candidate translations, a ranking method is applied. This method uses a disambiguation algorithm (Pedersen et al., 2005) to sort the translations. Once the translations are ranked, the user can decide either to choose the best translation from a

1 The system extracts the synonym names of a term by consulting the synonym relationships defined in the ontology of such a term

list of the best possible translations given by the system or to accept the automatic choice of the system.

Despite this work already developed, the tool has not yet been developed or is difficult to access and to date, no update on this work has been found either, so it is thought that it has not been continued. Another negative point about this work is also having a limit of only 3 languages available for translation.

2.5.2 *SOMMO*

By analysing the semantics encoded in both the source and target ontologies, the Semantic oriented mapping for multilingual ontologies (SOMMO) framework seeks to improve the multilingual ontology mapping results produced by current monolingual ontology matching techniques. The ontology translation step and the monolingual ontology matching phase are the most difficult steps in such a multilingual ontology mapping approach.

The first step is the translation of the source language's ontology into a different language. Several different techniques are applied to find the best translation possible for each concept with the help of other tools like *GoogleTranslate* API and (The Apache Software Foundation, 2023).

Then the system matches both target and source ontologies, now in the same language, using monolingual ontology mapping technologies. Finally, the metadata gathered during the ontology translation phase are of use in the final monolingual matching process. When used in conjunction with existing monolingual ontology matching technologies such metadata can help the rendering of more precise and confident matching results between the translated source ontology and the target ontology. Therefore, regardless of the original natural languages that were employed, accurate matching relationships are created between concepts from the original source ontology and the target ontology.

2.6 SUMMARY

Throughout this chapter, the world of multilingual ontologies was explored. Initially, for those who were not familiar with the concept of ontology, there is an introduction to the topic where the most fundamental concepts about what an ontology is are explained. Next, the current standard models and languages used for ontology representation were presented. The most common techniques to create ontologies were also discussed, followed by the presentation of what multilingual ontologies are and what their purpose is. Finally, some existing tools for creating multilingual ontologies were presented.

As it was possible to observe, although this concept of multilingual ontologies is relatively new, there is already a lot of information and work done around this subject. However,

mainly during the exploration of existing tools for the creation of multilingual ontologies, the lack of tools that create ontologies in a more generalized way is perceptible. There are several works that involve the creation of multilingual ontologies but most of them are works oriented to a certain theme or to a specific ontology.

With this in mind, the work of this Master's project focuses on the creation of a more automatic and generic tool that seeks to obtain reliable results for several distinct languages.

3

PROPOSED APPROACH

This chapter describes the architectural design of MOC, the multilingual ontology creation tool, that must be build as well as a more detailed explanation of the different components that make up the system.

3.1 SYSTEM ARCHITECTURE

Figure 5 represents the architecture of the multilingual ontology creation proposed tool.

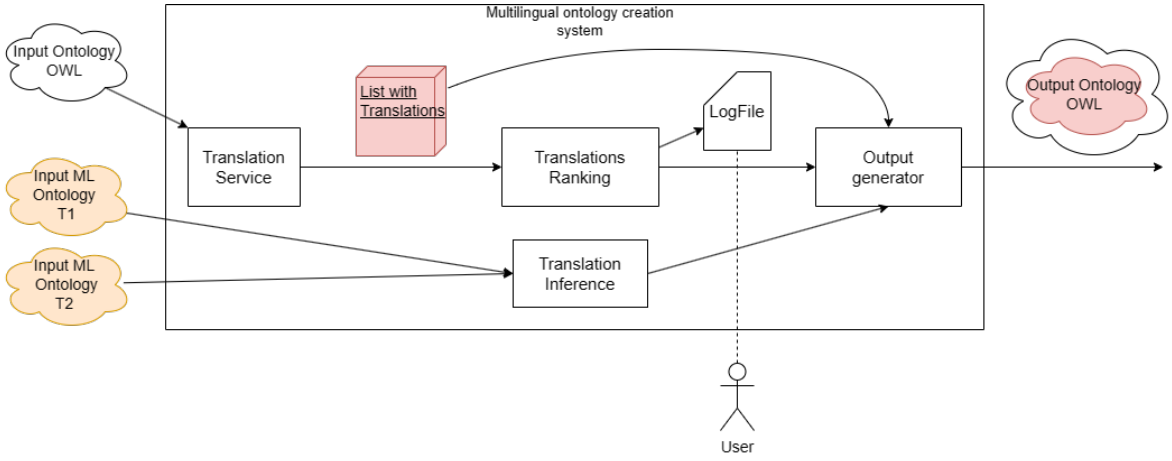


Figure 5: General overview of the system architecture

The next sections explain the purpose of each component to better understand how the whole system works.

3.1.1 *System inputs*

Since at this moment the most used language for ontology representation is OWL, the system only accepts ontologies in this language. The system accepts two types of input. The first is a single ontology, where the goal is to create or localize a multilingual ontology for

the languages the user chooses. The second option is 2 multilingual ontologies on the same domain, for example, one in English-Spanish and another in Spanish-French, so that it is possible, through triangulation, to obtain an English-French ontology.

3.1.2 *Translation service*

The aim of this service is to extract from the ontology all the labels that are to be translated and then translate them and save the translations in a new structure, such as a list. This service uses existing external tools such as **DEEPL API**[1] and **LingueeTranslator** [2] to help in this task.

3.1.3 *Translation Ranking*

After obtaining the possible translations for each label, it is necessary to choose the translation that best fits the ontology domain. To help in this task, there is a translation ranking service. Through this service it is intended to choose the "best" translation out of the list of possible translations. This decision is obtained by providing the translator with the context of the ontology. Even so, the translation chosen may not be the one the user prefers, so a LogFile is also created with information about the system's decision and the possible alternatives.

3.1.4 *Translation Inference*

The Translation Inference component is used to obtain a multilingual ontology through triangulation between two other ontologies. It should be noted that the two multilingual ontologies given as input must already exist, and must be the same, differing only in the language they are in. It should be noted that this process only serves to make it quicker to obtain an ontology if the user already has two ontologies with the characteristics previously described. For instance, one of the ontologies should be in English-Spanish and the other one in Spanish-German in order to obtain the final ontology in English-German.

3.1.5 *Output generator*

The Output generator component is intended to gather the final translations, obtained from the Translation Choice component, and create the final ontology. As it can be seen in Figure 5, the arrow connecting the "List of Translations" and the "Output generator" is

1 https://deep-translator.readthedocs.io/en/latest/
2 https://www.linguee.pt/

just to show that some information from the list is used on the output generator. After the translations are chosen, this service will construct back the new OWL ontology with the source information and the new translated one.

4

DEVELOPMENT

In this chapter, the process of implementing the Multilingual Ontology Creation tool (named MOC tool), developed in this Master Thesis is presented. The different components and structures used are explained in detail in the following sections.

4.1 TRANSLATOR

The first component of this system (Figure 5) is the translator component. Here, the OWL ontology is received as input and then passes through several functions to process it and then work with the information extracted.

4.1.1 *Extraction of information*

First of all, it is necessary to extract from the ontology the elements that will be translated. After some consideration and some visualization of OWL ontologies, it was decided that extraction should be done based on 3 properties. The first thing to extract are the labels. In the labels it can be found human-readable information about classes, properties and individuals and therefore their translation is very important. In an OWL/XML ontology, the labels are typically represented using the **'<rdfs:label>'** element.

Another element to be extracted are the comments. In an OWL/XML ontology, comments can be added to provide additional information or documentation about classes, properties, individuals, or any other elements in the ontology. The comments are typically represented using the **'<rdfs:comment>'** element.

Finally, in order to translate as much information as possible, any content that has the property "**xml:lang=""** " is extracted. This property indicates that the content of that XML tag is in a particular language and is therefore translatable content.

Once the objects to be translated have been chosen, the file is parsed. To do the parsing of the ontology a more manual semantic approach is done. The code starts by initializing two variables: **line_number** and **line_info**.

line_number keeps track of the line number being processed (useful information to reconstruct the final ontology later), while **line_info** is an empty list that will store the extracted information from each line in the ontology. Next, the code enters a loop that iterates over each line of the ontology. The code then determines if the current line has the **rdfs:label** tag, as illustrated in Listing 4.1, which indicates the presence of an ontology label.

```
1  if "<rdfs:label" in line:
2              id_end = 10
3              id = "rdfs:label"
4
5              lang_match = re.search(r'xml:lang="([^"]*)"', line)
6              lang = lang_match.group(1) if lang_match else orgn_lang
7
8              content_start = line.find(">", id_end) + 1
9              content_end = line.find("</rdfs:label>")
10             content = line[content_start:content_end].strip() or None
11
12             line_info.append((line_number, id, lang, content))
13 }
```

Listing 4.1: Extraction of labels from the ontology

The following actions are executed if the line contains **rdfs:label** tag:

- As can be seen on line 2, the end index of the tag name is set to a constant value of 10 as "rdfs:label" is 10 characters long, and to extract the next informations it is needed to skip these characters. On the next line, the **id** variable is also set to "rdfs:label" in order to use it when reconstructing the final ontology.
- On line 5, a regular expression is used to search for the value of the **xml:lang** attribute in the line. On line 6, it captures the value in a group if found and then assigns that value to a variable called **lang**. Otherwise, it assigns a default value stored in the **orgn_lang** variable.
- On the next 3 lines, the code finds the closing tag ">", and adds one position in order to start catching the content of the tag. The code then finds the closing tag "</rdfs:label>". Knowing where the content starts, and where it ends, it is then assigned to a variable named **content**. If the content is empty, it assigns **'None'** instead.
- Finally, on line 12, the tuple (**line_number**, **id**, **lang**, **content**) is appended to the **line_info** list, representing the information extracted from the line.

If the tag **'<rdfs:label'** is not found, the code searches for the tag **'<rdfs:comment'** and the extraction of the information is done in a similar way as the label tags, as shown in Listing 4.2.

```
elif "<rdfs:comment" in line:
            id_end = 12
            id = "rdfs:comment"

            lang_match = re.search(r'xml:lang="([^"]*)"', line)
            lang = lang_match.group(1) if lang_match else orgn_lang

            content_start = line.find(">", id_end) + 1
            content_end = line.find("</rdfs:comment>")
            content = line[content_start:content_end].strip() or None

            line_info.append((line_number, id, lang, content))
```

Listing 4.2: Extraction of comments from the ontology

If a line lacks a **rdfs:label** or **rdfs:comment**, a generic tag with the **xml:lang** attribute is searched, as demonstrated in Listing 4.3.

```
    elif "xml:lang=" in line:
            id_start = line.find("<") + 1
            id_end = line.find(" ", id_start)
            id = line[id_start:id_end]

            lang_match = re.search(r'xml:lang="([^"]*)"', line)
            lang = lang_match.group(1) if lang_match else None

            content_start = line.find(">", id_end) + 1
            content_end = line.find("</")
            content = line[content_start:content_end].strip() or None

            line_info.append((line_number, id, lang, content))
```

Listing 4.3: Extraction of any tag that contains the lang property

Similar procedures are used in this instance for the code to extract the tag name, language value, and content. In this case, the main difference is in the assignment of a value to the **id** variable. As, in this case, the **id** can be any variable value, the code looks for what comes before the lang property and assigns that to the **id** variable.

Structure to store the extracted elements

The extracted elements are stored in a simple list of tuples. Each tuple has the following items: line number, id, lang, content.

The **line_number** is useful to know where in the original file the information was removed so that later, when constructing the output, the new information can be added in the correct place.

The **id** is where the information about the tag being worked on is stored, so that in the output generator it is possible to construct a new XML line with the correct information.

The **lang** variable is where the language in which the content is written is stored.

The **content** variable stores the content of the XML tag.

4.1.2 *Translate the extracted elements*

Once the elements have been extracted, they need to be translated. To translate these elements 2 python packages are used: **deepl** and **LingueeTranslator** from deep_translator. Due to the different output types of each package, it is necessary to use 2 different translators.

After a comprehensive assessment of the strengths and weaknesses associated with the **Deepl API** and the **Google Translate API** in the context of a multilingual ontology project, the inclination is towards selecting the **Deepl API** as the preferable translation tool. This inclination is rooted in the paramount importance of precision and context preservation when undertaking the construction of a multilingual ontology.

Deepl is known for its ability to provide translations of high quality that effectively retain the nuances and contextual intricacies present within the source material. The utilization of neural machine translation by **Deepl** aligns cohesively with the requirements of developing a multilingual ontology, which mandates meticulous and nuanced translations to accurately capture the underlying meanings of terms and sentences.

In contrast, the **Google Translate API** provides an expansive selection of supported languages and harnesses the robust infrastructure of Google. However, the particular emphasis that the **Deepl API** places on context-awareness and generating translations that sound natural renders it a more suitable choice for the project's objectives. Given that accuracy is of greatest importance in ontology-based efforts, even minor misinterpretations can lead to discrepancies within the ontology structure.

Considering the potential impact of translation quality on the final output of the multilingual ontology, the strengths of the **Deepl API** in context preservation and accuracy outweigh any associated limitations, such as potential usage costs or restricted free usage. The conviction lies in the precision and dependability offered by the **Deepl API**, which is expected to contribute significantly to the overall accomplishment and integrity of the multilingual ontology project.

Nevertheless, it is essential to conduct a comprehensive series of test translations using both APIs, incorporating sample sentences from the project. This empirical evaluation will

yield the confidence and assurance required to proceed with the **Deepl API** as the primary translation solution.

To sum up, the **Deepl API** emphasis on translation quality and context preservation positions it as the optimal choice for the multilingual ontology project. Its ability to capture the essence of terms and sentences accurately across diverse languages is in line with the project's core objectives, promising a robust and accurate foundational structure for the ontology.

Once the translation API to be used has been chosen, it is time to move on to its integration and use. For strings composed of more than 1 word, the **deepl** package is used because it can usually make good interpretations about the sentence context and generally gives good results.

In the case of strings of only one word, there may be cases of uncertainty as to which is the best translation to choose as can be seen on Listing 4.4. For example, if the word "drums" is being translated from Portuguese to English, the question arises whether the correct translation is "battery" (of a car, for example), or "drums" (musical instrument). In such cases, it is used the **LingueeTranslator** which allows different output options.

```
['battery', 'drums', 'battery', 'beat', 'knock', 'whip', 'whisk', 'slam', 'flap',
    'pound', 'spank', 'bump', 'smite', 'clash', 'ram']
```

Listing 4.4: Output of LingueeTranslator for the Portuguese-English translation of the word "bateria".

These different options are stored in a dictionary, associated with **item_uri**. The decision of which translation is best is made at a later stage after studying a bit about the context of the ontology in question. This way, more reliable results are obtained.

Joining these two translators it is possible to obtain translations for 23 different languages making this tool very general and comprehensive.

4.1.3 *Technologies used*

As mentioned earlier, one API and one library are used, **Deepl API** [1] and **LingueeTranslator** from the **deep_translator** [2] python library. **Deepl API** is a powerful language translation tool that allows developers to integrate high-quality, human-like translations into their own applications. The API is designed to be user-friendly making it easy to implement although it does have some limitations when it comes to the free plan, as it has a monthly limit on the number of characters translated.

1 https://www.deepl.com/docs-api
2 https://deep-translator.readthedocs.io/en/latest/

The **deep_translator** Python library is a convenient and efficient tool for language translation and language detection tasks. It acts as a wrapper for various translation APIs, one of them being **LingueeTranslator**.

LingueeTranslator provides translations sourced from a vast database of bilingual texts, ensuring contextually accurate and reliable results. All these tools combined help to obtain reliable translations, thus improving the quality of the final ontology.

4.2 RANKING

The ranking module serves as the locus for the analysis of instances of ambiguity as previously mentioned. Within this module, a comprehensive review of the ontology's context is undertaken, aiming to discover the optimal translation from the array of potential translations derived from the Translator module.

In information technology, an ontology denotes a structured representation of concepts specific to a particular domain. That said, the context of the ontology referred to above is nothing more than the domain of the ontology. In the ranking module, the context assumes an important role, assisting in the determination of the most fitting translation aligning with the ontology. To effectively "extract" the ontology's context, users are prompted to provide a brief account of the domain/context alongside their inputs, supporting the efficacy of the applied methodology.

The chosen approach for resolving instances of ambiguity revolves around the translation of "context, original_word". While assurance cannot be absolute regarding the translation alignment, incorporating the original word within the context of the ontology significantly heightens the probability of achieving the desired translation. The chosen translation service, specifically **Deepl**, adjusts its translations in harmony with the sentence embedding the word. Following this translation, only the translated original_word is isolated for subsequent comparison against the roster of potential translations obtained in the prior module. If a match between any of the probable translations and the translation obtained from this module occurs, that translation is chosen. If no such correspondence appears, the first translation on the list of translations is chosen because of its popularity.

Understanding the context of the ontology holds paramount significance in navigating instances of ambiguity. To underscore this, consider a brief example: the Portuguese term "bateria" can assume diverse translations in English, as can be seen in Listing 4.4. The selection among these possibilities is rendered more manageable through the provision of ontology context to the translator.

Algorithm

Listing 4.5 presents the algorithm that resolves cases of ambiguity when deciding which is the best translation to use.

```
def ranker(translated_lines, ontology_context, target_lang):
    #(...)
    updated_lines = []
        for line in translated_lines:
            line_number, id, lang, content, translated = line
            if isinstance(translated, list):
                new_content = ontology_context + ', ' + content.lower()
                translation = translator.translate_text(new_content, source_lang=
    lang, target_lang=deepl_langs[target_lang]).text
                best_translation = translation.split(", ")[1]
                if best_translation in translated:
                    logger.info(f"Multiple translations found for line {
    line_number} for {content}. Choosing '{best_translation}' from options {
    translated}.")
                    translated = best_translation
                else:
                    logger.info(f"Multiple translations found for line {
    line_number} for {content}. Choosing '{translated[0]}' from options {
    translated}.")
                    translated = translated[0]

            updated_lines.append((line_number, id, lang, content, translated))
        return updated_lines
```

Listing 4.5: Best Translation Algorithm

This algorithm receives as input a list of tuples from the previous module, the context of the ontology, provided by the user, and the language being added to the ontology. Each tuple contains the following information: (line_number, id, lang, content, translated_line). The "line_number" corresponds to the number of the line in the ontology where the content is located; the "id" identifies the type of resource, e.g. label, comment, etc.; lang represents the language in which the content is; and translated_line is the translation or possible translations of the content.

Initially, it is necessary to analyse the contents of the list of tuples with the translations to identify which lines have more than one possible translation. To do this, the list is iterated and it is tested if the "translated" variable is a list, meaning that there is more than one possible translation for the content. In this case, a string is created containing the context and the original content to be translated, separated by an hashtag, so that after translation it is easier to remove the translated word. Deepl is then used to translate the new string. Once

the translation is complete, a split is made through the hashtag, and the second element of the split, which corresponds to the new translation, is taken. A comparison is then made between the translation obtained and the list of possible translations. If there is a match, the translation obtained is chosen as the best possible, otherwise the first one on the list is chosen as it is the most popular in most cases.

Still, after this algorithm is done, if the translation chosen is not the one the user thinks it is the most suitable, a simple Logfile is created where all the cases of ambiguity that arise in the translation of the ontology can be found.

To better understand how this algorithm works, let's recall the example of the portuguese word "bateria". Two different contexts will be applied in order to prove the impact of the context on the result of the translations. First, let's see the Listing 4.6 where the context "musica" is applied to the word "bateria".

```
1    original_word = "bateria"
2    context = "musica"
3    translated_label = translator.translate_text(context + '#' + original_word,
     source_lang="pt", target_lang="en-gb").text
4    print(translated_label)
```

Listing 4.6: Applying context "musica" to the translation of the word "bateria"

Giving this context, the translation obtained, as desired, is "drums", as can be seen in Listing 4.7.

```
1    $ python3 test.py
2    music#drums
```

Listing 4.7: Result of running the program above

Now, on Listing 4.8, lets apply the context "carro" to the translation service and then see the result.

```
1    original_word = "bateria"
2    context = "carro"
3    translated_label = translator.translate_text(context + ', ' + original_word,
     source_lang="pt", target_lang="en-gb").text
4    print(translated_label)
```

Listing 4.8: Applying context "carro" to the translation of the word "bateria"

```
1    $ python3 test.py
```

```
2     car , battery
```

Listing 4.9: Result of running the program with context "carro"

Looking at Listing 4.9 , depending on the context in which the word is translated, the result obtained is different. Although there are cases in which the most desirable translation may not be chosen, with this method it is possible to greatly improve the final results.

4.3 OUTPUT GENERATOR

The Output Generator module is responsible for joining the new information obtained to the original ontology. As a reminder, this new information comes from the "ranking" module, in the form of a list of tuples, which has already been talked about above in the document (Sec.4.1.1). Having this new information and the original ontology, it is necessary to add this information to the ontology.

Initially, to do this, some parsers of XML files were tested, but none of them worked as desired. The most common problem was that these parsers built a tree with the different parent and child tags, and then when reconstructing the document, the tags became disorganised in comparison to the original document. That said, a more hard coded method, shown in Listing 4.10, is used, in which the contents of the original file are first copied, the information is added, and then a file is created where the "join" function is applied with all the complete information.

```
1 def generator(translated_lines, xml_file, target_lang):
2     modified_content = xml_file[:]
3     shift = 0
4     for line in translated_lines:
5         line_number, id, lang, content, translated = line
6         if translated != "None":
7             xml_line = "<"+ id + " xml:lang=\"" + target_lang + "\">" +
    translated + "</" + id + ">"
8             indentation = get_indentation(xml_file[line_number - 1])
9             indented_line = indentation + xml_line
10            modified_content.insert(line_number + shift, indented_line + '\n')
11            shift += 1
12    modified_xml = ''.join(modified_content)
13    return modified_xml
```

Listing 4.10: Generator function from output generator

Initially, on line 2, a copy of the input **xml_file** is assigned to the **modified_content** variable. It is in this variable that the new information is inserted.

Then, a **shift** variable is initialised to 0 on line 3. The **shift** variable is used to keep track of the adjustments made to the line numbers when inserting the translated XML lines into the **modified_content**. During the iteration through the **translated_lines**, if a translated line is to be inserted, the **indented_line** is inserted at a specific position in **modified_content**. Since an insertion may cause subsequent insertions to shift down in the **modified_content**, the shift variable is incremented to reflect this shift. Summarising this variable ensures that subsequent insertions are made at the correct position, accounting for previous insertions and maintaining the order of insertion according to the line numbers.

Then, on line4, the program iterates over the **translated_lines** and, on line 5, for each tuple it extracts some relevant information: **line_number**, **id**, **lang**, **content**, and **translated**. Sometimes, when dealing with different ontologies, there are cases where the content of a tag is empty, and in these cases a "None" is allocated to the "translated" variable. If translated is not "None", it constructs a new XML line using the extracted information and the **target_lang** as can be seen on line 7.

Then, the program determines, on line 8, the indentation of the line before the insertion point using the **get_indentation** function. Then, on line 10, it inserts the indented XML line at the appropriate position in **modified_content**.

Finally, on line 12 and 13, it concatenates the modified content into a single string and returns it.

get_indentation function

The purpose of the **get_indentation** function, shown in Listing 4.11, is to determine the indentation of a particular line of text in an XML file. In XML and many other programming or markup languages, indentation is used for readability and to visually represent the structure or hierarchy of the content.

```
1 def get_indentation(line):
2     indentation = ""
3     for char in line:
4         if char.isspace():
5             indentation += char
6         else:
7             break
8     return indentation
```

Listing 4.11: **get_identation** function from output generator

This function takes a single parameter, **line** (a string). The function iterates through each character in **line** and it checks if a character is a white-space character (e.g., space or tab). The iteration continues until a non-white-space character is encountered, at which point the

loop breaks. Then the indentation string, which contains the leading white-space characters (indentation) from the input line, is returned.

For example, if a line begins with eight spaces, the **get_indentation** function will return those eight spaces, indicating the indentation level for that line.

The purpose of extracting indentation is to maintain consistent formatting and preserving the structure of the content when generating new content, especially in a structured format like XML.

4.4 LOGFILE

One of the features that our system also offers is the option of downloading a Logfile once the final ontology has been obtained. This option is used to complete the program and inform the user of certain decisions made by the system. This file will contain information on decisions made in the case of several possible translations for a given line.

As mentioned above, this file is built in the ranker module because this is where the decisions about which translation to use are made.

When a decision is made as to which translation goes into the final ontology, it is added to the Logfile that several possible translations were found for a certain line for a given content, as can be seen in Listing 4.12.

```
1    logger.info(f"Multiple translations found for line {line_number} for {content
     }. Choosing '{best_translation}' from options {translated}.")
```

Listing 4.12: Line that adds information to Logfile

It is also shown which translation has been chosen and what the other possible translations are.

Having this information in the Logfile allows the user to look at the different possible translations and eventually, if he feels that the chosen translation is not the best, he can edit the ontology manually to make it as accurate as possible.

Listing 4.13 is a small example of a Logfile for a simple ontology.

```
1    Multiple translations found for line 24 for Person. Choosing 'pessoa' from
     options ['pessoa'].
2    Multiple translations found for line 30 for Book. Choosing 'livro' from
     options ['livro', 'reservar', 'agendar']
```

Listing 4.13: Logfile example

For example, for the word "Book", 3 different possible Portuguese translations were found, but only "livro" was chosen. If the user is not happy with the choice, he can consult the Logfile and see which line the translation is on and change it if he wishes.

4.5 INFERENCE/TRIANGULATION

In addition to the most common functionality of the proposed system, the creation of a multilingual ontology through translations, there is also a feature that makes it possible to obtain a new multilingual ontology by triangulating two others. This process is very useful because there is no need for translations or any other elaborate algorithm, making it simple and quick in terms of execution time.

One restriction for this feature is that both ontologies have to be "the same" in terms of structure and tags, but must be in different language pairs. This means that, for example, one of the ontologies must be in English-Portuguese, and the other in Portuguese-French, so that it is possible to obtain an ontology in English-French.

4.5.1 *Inference Algorithm*

This algorithm receives two xml files, which correspond to the two ontologies, and for each ontology the two languages in which each is written. These languages are requested as input from the user and if one of the languages is not the same in the two ontologies, the user is told that triangulation is not possible. Let´s look at Listing 4.14 and analyse the languages comparison from both ontologies.

```
def inference(xml_file1, xml_file2, lang_1A, lang_2A, lang_1B, lang_2B):
    result_ontology = xml_file1[:]
    line_number = 0
    if lang_1A == lang_1B:
        common_lang = lang_1A
    elif lang_1A == lang_2B:
        common_lang = lang_1A
    elif lang_2A == lang_1B:
        common_lang = lang_2A
    elif lang_2A == lang_2B:
        common_lang == lang_2A
    else:
        print("Can't triangulate with the selected languages. One of the
    languages must match in both ontologies.")
```

Listing 4.14: Languages comparison between both xml files

Firstly, the lines from one of the ontologies are copied to the **result_ontology** variable, because it is in this variable that changes will be made to produce the final ontology. Once again, in order not to lose the order of the tags, it's easier to iterate over the lines in the file instead of using an xml parser.

After this, a comparison is made between the 4 languages that the user has provided in order to find out which language is common to both ontologies.

Then the iteration over the lines of the first ontology begins. It is only necessary to iterate over one of the ontologies because, as the structure of both is expected to be the same, following one line on one of the ontologies, it will be elementary to go and get information from the other ontology. In a nutshell, what this algorithm does is find the line in the second ontology that contains the information corresponding to the line being iterated over, but in the missing language. To help with this process, two variables are created, as can be seen in Listing 4.15: **line_number**, **skip_next_line**.

```
    skip_next_line = False

    for line in xml_file1:
        line_number += 1
        if skip_next_line:
            skip_next_line = False
            continue
```

Listing 4.15: Initialization of skip_next_line and line_number variables

The purpose of the **line_number** variable is to store the number of the line being iterated. As the algorithm loops through the XML file, **line_number** serves as a reference to the current line being processed. It's crucial for correctly extracting data from both **xml_file1** and **xml_file2** at the appropriate points throughout the iteration. For instance, based on the current **line_number** in **xml_file1**, the algorithm needs to access corresponding lines in **xml_file2** in order to compare content and language information between **xml_file1** and **xml_file2**. **line_number** ensures that the algorithm accesses the proper lines in **xml_file2** for comparison.

The **skip_next_line** variable is a boolean flag that controls when the XML lines are processed. When this variable is set to True, it indicates that the next line in the XML should be skipped during processing. When a case of **label**, **comment** or tag with the **lang** attribute is found, a comparative analysis is made between the contents of one ontology and another. When one of these cases is found, its content is immediately analysed and comparisons are made to find out whether the content is common to the other ontology or not. As there is the variable **line_number**, it is possible to access the content of the next line, which will probably be in another language, and therefore after all the comparisons have been made, it

will no longer be necessary to access the next line in the iteration, which is why there is the variable **skip_next_line**.

Extraction of information

As in the "translator" module, the extraction of information from an xml line is carried out through semantic analysis with the help of python's re library. Looking to Listing 4.16, it is noticeable that the extraction process is exactly the same, but in this module information is extracted from at least 3 XML lines, 1 from the ontolo gy to be iterated and 2 from the other ontology.

```
    if "<rdfs:label" in line:
        id_end = 10
        id = "rdfs:label"
        content_startA1 = line.find(">", id_end) + 1
        content_endA1 = line.find("</rdfs:label>")
        contentA1 = line[content_startA1:content_endA1].strip() or None

        lineB1 = xml_file2[line_number-1]
        lang_matchB1 = re.search(r'xml:lang="([^"]*)"', lineB1)
        langB1 = lang_matchB1.group(1) if lang_matchB1 else None
        content_startB1 = lineB1.find(">", id_end) + 1
        content_endB1 = lineB1.find("</rdfs:label>")
        contentB1 = lineB1[content_startB1:content_endB1].strip() or None

        lineB2 = xml_file2[line_number]
        lang_matchB2 = re.search(r'xml:lang="([^"]*)"', lineB2)
        langB2 = lang_matchB2.group(1) if lang_matchB2 else None
        content_startB2 = lineB2.find(">", id_end) + 1
        content_endB2 = lineB2.find("</rdfs:label>")
        contentB2 = lineB2[content_startB2:content_endB2].strip() or None
```

Listing 4.16: Piece of code for extracting information from lines part 1

It is necessary to extract information from 3 lines because it is necessary to realise, firstly, which line is common to both ontologies and, secondly, which line from the second ontology should be added to the result ontology.

Content Comparison

To find the line common to both ontologies, it is necessary to make a comparison between them. Since each ontology is in two different languages, ideally it would be enough to compare the **lang** attribute of each line in each ontology, but sometimes this attribute is missing from some lines. It's more effective to compare the content itself. So, let´s analyse

Listing 4.17, the first piece of code that compares the content from the lines

```
1    if contentA1 == contentB1:
2        skip_next_line = True
3        if langB2 == None:
4            result_ontology[line_number-1] = identation + "<"+ id + ">" +
     contentB2 + "</" + id + ">\n"
5        else:
6            result_ontology[line_number-1] = identation + "<"+ id + " xml:lang=\"
     " + langB2 + "\">" + contentB2 + "</" + id + ">\n"
7
8    elif contentA1 == contentB2:
9        skip_next_line = True
10       if langB1 == None:
11           result_ontology[line_number-1] = identation + "<"+ id + ">" +
     contentB1 + "</" + id + ">\n"
12       else:
13           result_ontology[line_number-1] = identation + "<"+ id + " xml:lang=\"
     " + langB1 + "\">" + contentB1 + "</" + id + ">\n"
```

Listing 4.17: Piece of code for comparing content from lines part 2

First, in line 1, the content extracted from the line of the ontology being iterated is compared with the corresponding line in the second ontology. If the content is the same, it means that the line common to both ontologies has been found and, therefore, it is necessary to add the content of the next line, already fetched in Listing 4.16, of the second ontology to add to the final ontology. As the **result_ontology** variable contains a copy of the content of the ontology being iterated, this addition of information is done by replacing the common line of the iterated ontology with the non-common line of the other ontology. For example, if there are two ontologies, one in English-Portuguese and the other in English-French, if a common line in English is found in both, the algorithm replaces the common line from the ontology being iterated, with the one in French from the other ontology. If there is no match between the content of the iterated line and the corresponding line in the second ontology, then the same comparison is made with the next line in the second ontology as illustrated in Listing 4.18.

```
1    else:
2        lineA2 = xml_file1[line_number]
3        content_startA2 = lineA2.find(">", id_end) + 1
4        content_endA2 = lineA2.find("</")
5        contentA2 = lineA2[content_startA2:content_endA2].strip() or None
6
7        if contentA2 == contentB1:
```

```
            skip_next_line = True
            if langB2 == None:
                result_ontology[line_number-1] = identation + "<"+ id + ">" + contentB2 + "</" + id + ">\n"
            else:
                result_ontology[line_number-1] = identation + "<"+ id + " xml:lang=\"" + langB2 + "\">" + contentB2 + "</" + id + ">\n"

        elif contentA2 == contentB2:
            skip_next_line = True
            if langB1 == None:
                result_ontology[line_number-1] = identation + "<"+ id + ">" + contentB1 + "</" + id + ">\n"
            else:
                result_ontology[line_number-1] = identation + "<"+ id + " xml:lang=\"" + langB1 + "\">" + contentB1 + "</" + id + ">\n"

        else:
            print("Something went wrong")
```

Listing 4.18: Piece of code for comparing content from lines

If there is a match, this means that the first compared line of the second ontology is the information to be added to the final ontology. If there is no match, then it is because the line being iterated is not the one containing the shared information and, therefore, it is necessary to extract the content of the second line in order to make new comparisons and realise which line needs to be introduced into the final ontology.

Figure 6 and Figure 7 show two small example lines from an example ontology. Figure 8 shows the result of giving such input to the triangulation process.

```
<rdfs:comment xml:lang="en">The BSD license on the BFO project site refers to code used to build BFO.</rdfs:comment>
<rdfs:comment xml:lang="pt">A licença BSD no site do projeto BFO refere-se ao código utilizado para construir o BFO.</rdfs:comment>
```

Figure 6: Two lines of an English-Portuguese ontology

```
<rdfs:comment xml:lang="en">The BSD license on the BFO project site refers to code used to build BFO.</rdfs:comment>
<rdfs:comment xml:lang="it">La licenza BSD sul sito del progetto BFO si riferisce al codice usato per costruire BFO.</rdfs:comment>
```

Figure 7: Two lines of an English-Italian ontology

```
<rdfs:comment xml:lang="it">La licenza BSD sul sito del progetto BFO si riferisce al codice usato per costruire BFO.</rdfs:comment>
<rdfs:comment xml:lang="pt">A licença BSD no site do projeto BFO refere-se ao código utilizado para construir o BFO.</rdfs:comment>
```

Figure 8: Two lines of an Italian-Portuguese ontology resulting from an inference between the other two

For the lines containing comments or the lang attribute, exactly the same process is carried out, with only a slight variation in the way the information is extracted due to the ids of each line.

4.6 WEB INTERFACE

The primary objective of the Multilingual Ontology Creation tool is to operate through a web interface. This enables universal accessibility, allowing anyone to use it from any location without needing any programming expertise. In this section, it is detailed comprehensively and demonstrated each webpage and feature found within the web application.

4.6.1 *Home Page*

In the Homepage of MOC tool it can be found the main functionality, to create a multilingual ontology by translating different ontology objects. Looking at Figure 9, it is possible to see various fields for the user to fill in, which are very important for the programme to work correctly.

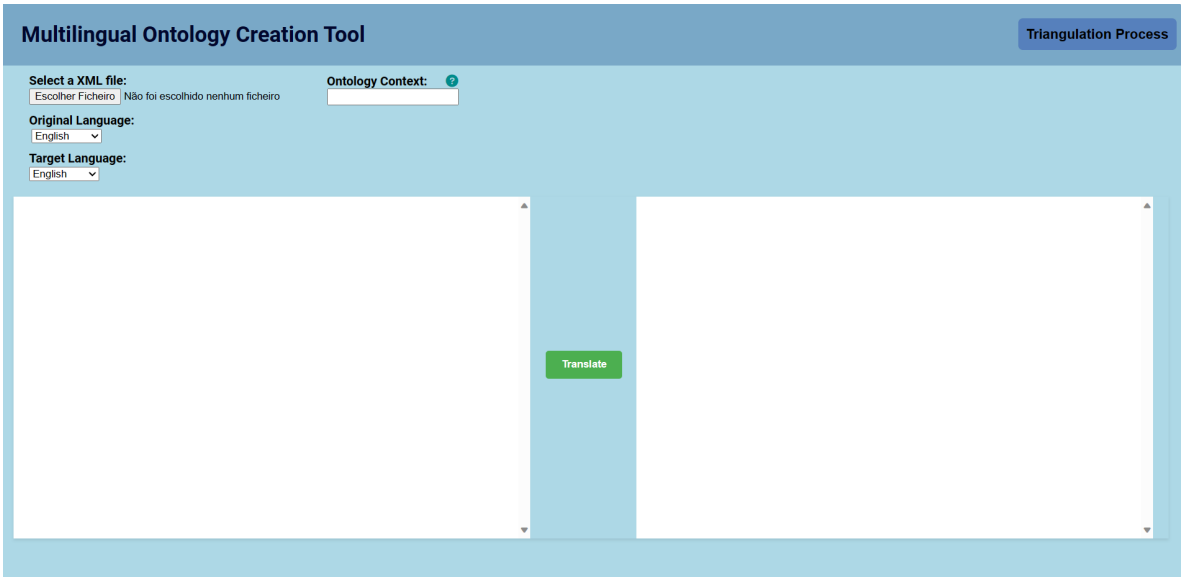


Figure 9: Home page

In the first field, named "Select a XML file:", it can be found the place where the user has to upload his ontology. The format of the input ontology, to be translated, shall be OWL since the programme is only prepared for XML ontologies. Once uploaded, the file is

displayed in the white box on the left, giving the user a preview of the document he has uploaded to the programme.

Under the upload box, there are two dropdowns for the user to choose two languages. The first, "Original Language", is for the user to enter the original language of the ontology because sometimes the "lang" attribute in the tags doesn't exist and it is necessary to provide the language of the content in order to do the translations. The use of a language detection mechanism was discarded because it sometimes fails when the sample of content is small, and can wrongly identify a language other than the one intended.

The second, "Target Language", allows the user to choose which language he wants to add to the ontology. There are a total of 23 different languages to choose from in these dropdowns.

There is also a text box, "Ontology Context", which is used to receive a short string from the user to help understand the context of the ontology. Filling in the Ontology Context field is extremely important as it can be crucial in improving the quality of the translations by adapting them to the specific domain of the ontology. To help the user understand what is required in this field, there is a tooltip text, that can be seen in Figure 10 with a short message to help to understand.

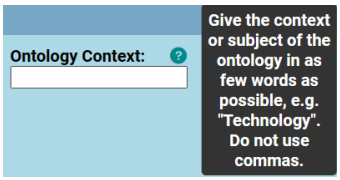


Figure 10: ToolTip Text from Ontology Context

Once all the fields are in line with the user's preferences, he can then use the green "Translate" button to start the programme. The white box on the right, which has not yet been mentioned, is used to display the final ontology like in Figure 11.

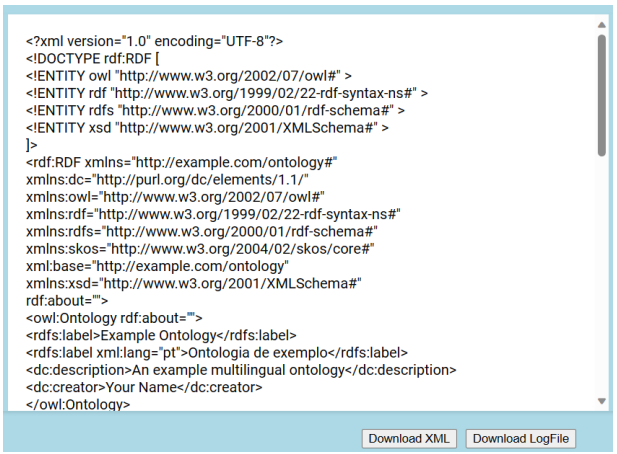


Figure 11: Example Output box with buttons to download

This is just for aesthetic purposes and so that the user can see the final result without having to download it. At the same time as the final ontology is displayed in the white box, two buttons are also created, like the example in Figure 11. One allows for downloading the output OWL file and the other allows for downloading the Log file with information about the decisions made by the program.

Finally, in the top right-hand corner, there is a button that gives access to the other ontology creation process, the triangulation process.

4.6.2 *Triangulation Process Page*

The Triangulation Process Page (Figure 12), where the process of creating ontologies by triangulation is located, is similar to the home page, with only slight differences due to the different requirements of this functionality.

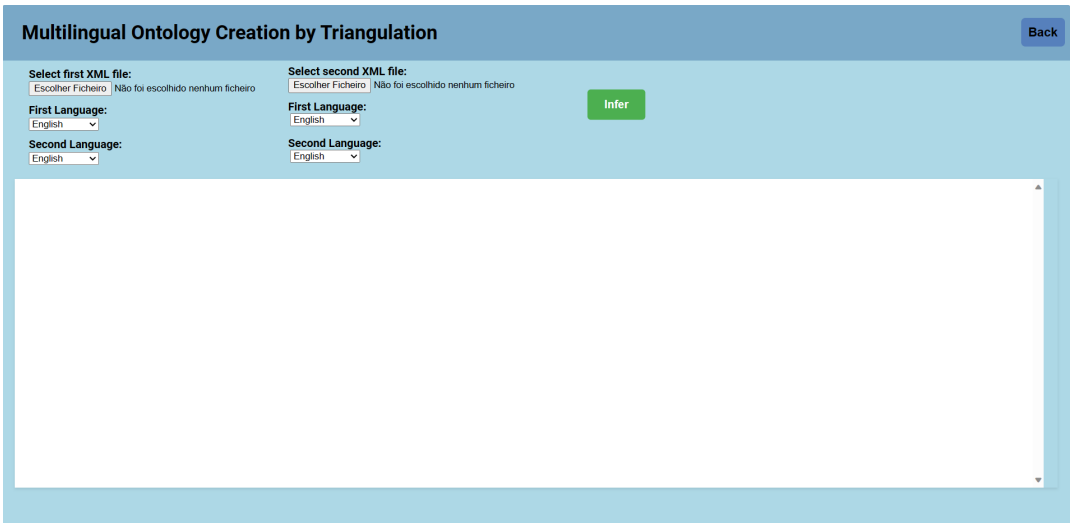


Figure 12: Triangulation process page

As on the home page, there is an element for the user to upload the ontology. In this case, as it is a triangulation, the user needs to upload two ontologies, both in OWL. For logistical reasons, as in this case it is necessary to upload two ontologies, this page does not preview the ontologies given as input.

Below each ontology input box, there are two other language selection boxes where the user must, for each ontology, choose the 2 languages the ontology is in. These values are then used to check whether or not triangulation between the two ontologies is possible.

Once the inputs have been entered, the user can press the green "Infer" button to get the programme to do the triangulation. If the user wants to go back to the home page, he just needs to press the button in the top right-hand corner labelled "Back".

Once the programme has run and the final ontology has been produced, it is displayed in the white box on the page, and a button appears where the user can click to download the final ontology.

4.7 CASE STUDY

The testing phase is one of the most important steps in the process of developing an application, because it is during this phase that it is possible to realise if everything is working properly and if it fulfils the previously defined requirements. Although this phase is often associated with a "final" phase of the project, it should be noted that tests were always carried out throughout the development of the program, which helped a lot to detect certain errors and adapt the code to the different obstacles that different ontologies offered. This has helped this tool to work with almost any ontology. Of course, the ontologies used

in the tests are just a small sample, given the amount of ontologies that exist all over the internet.

Bearing in mind that this is a tool that aims to work with any type of ontology, which makes it unique, it made it impossible to make comparisons with other types of tools, of which there aren't many either. That said, the tests focused more on verifying that the tool worked without any errors, for a variety of inputs.

The quality of the translations was also taken into account, but this is a somewhat less objective point because to be more thorough in this regard, it would be necessary to consult experts, both in the domain of each ontology and in the language itself. Therefore, having seen opinions on the internet about the quality of the DEEPL translator, and adding the decision algorithm developed for cases of ambiguity, it was considered that, for such a general tool, the quality of the translations has been achieved.

To show the results, a small, complete example is used, where it is possible to find all the types of elements to be translated (labels, comments, tags with the "lang" attribute) and cases of ambiguity (e.g. Book).

This example, shown in Listing 4.19 was given to the programme as input, where the original language chosen was English and the language to be added to the ontology is Portuguese.

```
<?xml version="1.0" encoding="UTF-8"?>
  <!-- Classes -->
  <owl:Class rdf:about="Person">
    <rdfs:label xml:lang="en">Person</rdfs:label>
    <rdfs:label xml:lang="fr">Personne</rdfs:label>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>
  <owl:Class rdf:about="Book">
    <obo:IAO_0010000 xml:lang="en">Book</rdfs:label>
    <obo:IAO_0010000 xml:lang="fr">Livre</rdfs:label>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>
  <!-- Properties -->
  <owl:ObjectProperty rdf:about="hasAuthor">
    <rdfs:label xml:lang="en">has author</rdfs:label>
    <rdfs:label xml:lang="fr">a pour auteur</rdfs:label>
    <rdfs:domain rdf:resource="#Book"/>
    <rdfs:range rdf:resource="#Person"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

Listing 4.19: Example ontology in English-French

Note that this example is already a multilingual ontology, although it could perfectly well be a monolingual ontology. Listing 4.20 and Listing 4.21 show both the final ontology and

the Logfile produced respectively.

```
<?xml version="1.0" encoding="UTF-8"?>
  <!-- Classes -->
  <owl:Class rdf:about="Person">
    <rdfs:label xml:lang="en">Person</rdfs:label>
    <rdfs:label xml:lang="pt">pessoa</rdfs:label>
    <rdfs:label xml:lang="fr">Personne</rdfs:label>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>
  <owl:Class rdf:about="Book">
    <obo:IAO_0010000 xml:lang="en">Book</rdfs:label>
    <obo:IAO_0010000 xml:lang="pt">livro</rdfs:label>
    <obo:IAO_0010000 xml:lang="fr">Livre</rdfs:label>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>
  <!-- Properties -->
  <owl:ObjectProperty rdf:about="hasAuthor">
    <rdfs:label xml:lang="en">has author</rdfs:label>
    <rdfs:label xml:lang="pt">tem autor</rdfs:label>
    <rdfs:label xml:lang="fr">a pour auteur</rdfs:label>
    <rdfs:domain rdf:resource="#Book"/>
    <rdfs:range rdf:resource="#Person"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

Listing 4.20: Output ontology for the previous input

```
Multiple translations found for line 4 for Person. Choosing 'pessoa' from
options ['pessoa'].
Multiple translations found for line 9 for Book. Choosing 'livro' from
options ['livro', 'reservar', 'agendar'].
```

Listing 4.21: Logfile for the previous input

As can be observed in the previous listings, everything is working as intended. The structure of the original ontology is maintained, and the new lines, with the Portuguese language, are added correctly. It's also interesting to see the impact that providing context has on the results. In this previous example, no context was provided, so the translation chosen was **livro** because it is the most common translation. However, if the user provides the context of "cinema tickets" as an example, the output produced will be different.

As can be seen in Listings 4.22 and 4.23, the word Book now has a new meaning given this context and so the decision as to which translation to use has changed.

```
<?xml version="1.0" encoding="UTF-8"?>
  <!-- Classes -->
  <owl:Class rdf:about="Person">
    <rdfs:label xml:lang="en">Person</rdfs:label>
    <rdfs:label xml:lang="pt">pessoa</rdfs:label>
    <rdfs:label xml:lang="fr">Personne</rdfs:label>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>
  <owl:Class rdf:about="Book">
    <obo:IAO_0010000 xml:lang="en">Book</rdfs:label>
    <obo:IAO_0010000 xml:lang="pt">reservar</rdfs:label>
    <obo:IAO_0010000 xml:lang="fr">Livre</rdfs:label>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>
  <!-- Properties -->
  <owl:ObjectProperty rdf:about="hasAuthor">
    <rdfs:label xml:lang="en">has author</rdfs:label>
    <rdfs:label xml:lang="pt">tem autor</rdfs:label>
    <rdfs:label xml:lang="fr">a pour auteur</rdfs:label>
    <rdfs:domain rdf:resource="#Book"/>
    <rdfs:range rdf:resource="#Person"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

Listing 4.22: Output ontology for the previous input with the "cinema tickets" context

```
Multiple translations found for line 4 for Person. Choosing 'pessoa' from
options ['pessoa'].
Multiple translations found for line 9 for Book. Choosing 'reservar' from
options ['livro', 'reservar', 'agendar'].
```

Listing 4.23: Logfile for the previous input with context "cinema tickets"

Speaking of the triangulation process, the results are also satisfactory because not only is the process quick, but the results are as expected. To show the results of the triangulation process, two small example ontologies are used, the first can be seen in Listing 4.19 and the second in Listing 4.24.

```
<?xml version="1.0" encoding="UTF-8"?>
  <!-- Classes -->
  <owl:Class rdf:about="Person">
    <rdfs:label xml:lang="en">Person</rdfs:label>
    <rdfs:label xml:lang="es">Persona</rdfs:label>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>
```

```
8       <owl:Class rdf:about="Book">
9         <obo:IAO_0010000 xml:lang="en">Book</rdfs:label>
10        <obo:IAO_0010000 xml:lang="es">Libro</rdfs:label>
11        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
12      </owl:Class>
13      <!-- Properties -->
14      <owl:ObjectProperty rdf:about="hasAuthor">
15        <rdfs:label xml:lang="en">has author</rdfs:label>
16        <rdfs:label xml:lang="es">tiene autor</rdfs:label>
17        <rdfs:domain rdf:resource="#Book"/>
18        <rdfs:range rdf:resource="#Person"/>
19      </owl:ObjectProperty>
20    </rdf:RDF>
```

Listing 4.24: Second input ontology

The ontology created can be seen in Listing 4.25.

```
1     <?xml version="1.0" encoding="UTF-8"?>
2       <!-- Classes -->
3       <owl:Class rdf:about="Person">
4         <rdfs:label xml:lang="fr">Personne</rdfs:label>
5         <rdfs:label xml:lang="es">Persona</rdfs:label>
6         <rdfs:subClassOf rdf:resource="&owl;Thing"/>
7       </owl:Class>
8       <owl:Class rdf:about="Book">
9         <obo:IAO_0010000 xml:lang="fr">Livre</rdfs:label>
10        <obo:IAO_0010000 xml:lang="es">Libro</rdfs:label>
11        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
12      </owl:Class>
13      <!-- Properties -->
14      <owl:ObjectProperty rdf:about="hasAuthor">
15        <rdfs:label xml:lang="fr">a pour auteur</rdfs:label>
16        <rdfs:label xml:lang="es">tiene autor</rdfs:label>
17        <rdfs:domain rdf:resource="#Book"/>
18        <rdfs:range rdf:resource="#Person"/>
19      </owl:ObjectProperty>
20    </rdf:RDF>
```

Listing 4.25: Final input ontology

In summary, the testing process was highly beneficial in helping to decide which techniques to apply.

The testing method for the algorithm provides insurance that this tool is producing positive and successful results.

4.8 TECHNOLOGIES USED

The development of this tool involved the use of different technologies and some already mentioned libraries. Here is a short summary of the technologies used.

When considering the development of the web application to host the tool, Flask[3] was selected because it is a lightweight web framework that simplifies the creation of small web applications.

The **DEEPL Api** and the **LingueeTranslator** library from the **deep_translator** python library was used to help with translations.

To help with the parsing and extraction of information from the ontology, the **re** python library was chosen.

3 https://flask.palletsprojects.com/en/3.0.x/

5

CONCLUSION

In this document the topic of multilingual ontologies was explored. Firstly an introduction is given which talks about why ontologies exist, what their usefulness is and what has led to people working on their development. It is also explained that, due to the problem of portability, the first languages that allow expressing ontologies appeared. Those first languages led to the development of current standards such as the OWL language.

Afterwards, multilingual ontologies were presented. An explanation of what they are and what their purpose is, is given. Then, a general discussion on how to create an ontology was included.

In section 2.4, focused on the creation of multilingual ontologies, Cross lingual enrichment and Ontology Localization, two techniques used to build these ontologies, were discussed. Moving on to the next section, some tools that already exist for creating multilingual ontologies were listed and presented in detail.

After the chapter devoted o the background and state-of-the-art, it was presented the proposed solution for this Master's project, including a brief overview of the system architecture and a description of the various components that make up the tool. After the proposed solution is presented, the development process is explained. In a first moment, it is given an understanding of the rationale and logical foundation of the algorithm within the tool that facilitates the creation of a multilingual ontology through translation of elements, along with an exploration of the techniques employed to achieve this. Initially it was planned to include an intermediate module between the ranking and the output generator that would serve to make a semi-automatic choice of the "ideal" translation, i.e. the user could choose manually his prefered translation or let the system decide. This idea was discarded because for ontologies of considerable size, it would be impractical for the user to select all the translations manually. This was solved with the creation of LogFile. The system decides on all the ideal translations, but writes the decision made and the alternatives to the LogFile, so that the user can manually change the translations if he is not happy with the result. This feature enables users to retrieve comprehensive data about the decisions that culminated in the final ontology. The process of creating ontologies through triangulation is then explained

in detail. Finally, an elucidation of the Web Application designed to host the tool is provided, offering insights into the various interfaces it offers and guidance on their utilization.

The two primary objectives of this Master's Project have been accomplished. The initial aim was to get a deeper knowledge about techniques for creating multilingual ontologies, as well as study existing tools, facilitating a comparative analysis of their strengths and weaknesses. This study aided in establishing precise goals for the application that should be built in this Master's work, ultimately leading to the development of an autonomous tool that operates without human interaction thus realising the second objective, that was the successful development of this tool. The application was named MOC (Multilingal Ontology Creation) and is now available at http://moc.epl.di.uminho.pt/.

Although the main objectives have been achieved, there is still some room for manoeuvre to improve the tool. For future work, the optimization of the execution time of the ontology creation algorithm through translations is, maybe, one of the most important improvements to be performed. This can be a difficult task as it depends a lot on the APIs that assist the program in translation, however it is a challenging direction for future research. It would also be interesting to find an automatic way of validating the final ontology, to see if all its structure has been maintained and if the file is as intended. For small ontologies, for example, this task can easily be done manually, but it would be inconceivable for large ontologies. Adding support for more languages available for translations is also something to be considered. And finally, add support for ontology languages other than just XML, such as Turtle or OntoDL+(Dias, 2021). To do this, adaptations would have to be made to the parser so that it could extract the necessary information from the files of the different languages. In principle, no other changes would be necessary as all the other components already work with the data in a predefined format. More specifically, for the OntoDL+ language, you would need to use the OntoDL+ parser to read the input and then use a term translator to produce the new output.

A

CASE STUDY: ONTOLOGY ABOUT BASKETBALL

This appendix shows an OWL document, developed to show different examples throughout this work, that represents an ontology about basketball.

```
<?xml version="1.0" encoding="UTF-8"?>

<Ontology xmlns="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"  ontologyIRI="http://www.
    semanticweb.org/simia/ontologies/2020/10/Basquetebol">
  <Prefix name="" IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol"/>
  <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
  <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
  <Prefix name="xml" IRI="http://www.w3.org/XML/1998/namespace"/>
  <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
  <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>

  <Declaration>
    <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/Schroder"/>
  </Declaration>

  <Declaration>
    <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/Brown"/>
  </Declaration>

  <Declaration>
    <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/James"/>
  </Declaration>

```

```
28  <Declaration>
29    <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Booker"/>
30  </Declaration>
31
32  <Declaration>
33    <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Walker"/>
34  </Declaration>
35
36  <Declaration>
37    <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Payne"/>
38  </Declaration>
39
40  <Declaration>
41    <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Carter"/>
42  </Declaration>
43
44  <Declaration>
45    <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/aauav"/>
46  </Declaration>
47
48  <Declaration>
49    <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Davis"/>
50  </Declaration>
51
52  <Declaration>
53    <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Bridges"/>
54  </Declaration>
55
56  <Declaration>
57    <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Pop"/>
58  </Declaration>
59
60  <Declaration>
61    <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Craig"/>
62  </Declaration>
63
64  <Declaration>
```

```
65      <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/Ayton"/>
66    </Declaration>
67
68    <Declaration>
69      <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/uminho"/>
70    </Declaration>
71
72    <Declaration>
73      <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        TwentyFourSecond"/>
74    </Declaration>
75    <Declaration>
76      <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Blocking"/>
77    </Declaration>
78    <Declaration>
79      <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Game"/>
80    </Declaration>
81    <Declaration>
82      <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Smallforward"/>
83    </Declaration>
84    <Declaration>
85      <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Slapping"/>
86    </Declaration>
87    <Declaration>
88      <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Charging"/>
89    </Declaration>
90    <Declaration>
91      <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Player"/>
92    </Declaration>
93    <Declaration>
94      <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Position"/>
95    </Declaration>
96    <Declaration>
97      <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Point"/>
98    </Declaration>
99    <Declaration>
```

```
100     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Goaltending"/>
101   </Declaration>
102   <Declaration>
103     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Referee"/>
104   </Declaration>
105   <Declaration>
106     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Travelling"/>
107   </Declaration>
108   <Declaration>
109     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Flagrant"/>
110   </Declaration>
111   <Declaration>
112     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Intentional"/>
113   </Declaration>
114   <Declaration>
115     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Technical"/>
116   </Declaration>
117   <Declaration>
118     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        HeldBall"/>
119   </Declaration>
120   <Declaration>
121     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Powerforward"/>
122   </Declaration>
123   <Declaration>
124     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Freethrow"/>
125   </Declaration>
126   <Declaration>
127     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        EightSecond"/>
128   </Declaration>
129   <Declaration>
130     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Hitting"/>
131   </Declaration>
132   <Declaration>
133     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        DoubleDribble"/>
134   </Declaration>
```

```
135  <Declaration>
136    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
       IllegalScreen"/>
137  </Declaration>
138  <Declaration>
139    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
       Team"/>
140  </Declaration>
141  <Declaration>
142    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
       ShootingGuard"/>
143  </Declaration>
144  <Declaration>
145    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
       Holding"/>
146  </Declaration>
147  <Declaration>
148    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
       Violation"/>
149  </Declaration>
150  <Declaration>
151    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
       Pushing"/>
152  </Declaration>
153  <Declaration>
154    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
       PointGuard"/>
155  </Declaration>
156  <Declaration>
157    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
       BackcourtViolation"/>
158  </Declaration>
159  <Declaration>
160    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
       Coach"/>
161  </Declaration>
162  <Declaration>
163    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
       TwoPointer"/>
164  </Declaration>
165  <Declaration>
166    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
       Carrying"/>
167  </Declaration>
168  <Declaration>
169    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
       Center"/>
```

```
170   </Declaration>
171   <Declaration>
172     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Foul"/>
173   </Declaration>
174   <Declaration>
175     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      FiveSecond"/>
176   </Declaration>
177   <Declaration>
178     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      ThreePointer"/>
179   </Declaration>
180   <Declaration>
181     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/playsAgainst"/>
182   </Declaration>
183
184   <Declaration>
185     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/playsFor"/>
186   </Declaration>
187
188   <Declaration>
189     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/playsAs"/>
190   </Declaration>
191
192   <Declaration>
193     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/coaches"/>
194   </Declaration>
195
196   <Declaration>
197     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/whistles"/>
198   </Declaration>
199
200   <SubClassOf>
201     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Charging"/>
202     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Foul"/>
203   </SubClassOf>
204
205   <SubClassOf>
```

```
206     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        PointGuard"/>
207     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Position"/>
208   </SubClassOf>
209
210   <ObjectPropertyDomain>
211     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/playsAs"/>
212     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Craig"/>
213     </ObjectPropertyDomain>
214   <ObjectPropertyRange>
215     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/playsAs"/>
216     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Smallforward"/>
217   </ObjectPropertyRange>
218
219   <SubClassOf>
220     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Blocking"/>
221     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Foul"/>
222   </SubClassOf>
223
224   <ObjectPropertyAssertion>
225     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/coaches"/>
226     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/Pop"/>
227     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/aauav"/>
228   </ObjectPropertyAssertion>
229
230   <SubClassOf>
231     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        ShootingGuard"/>
232     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Position"/>
233   </SubClassOf>
234
235   <ObjectPropertyDomain>
236     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/whistles"/>
237     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Referee"/>
```

```
238     </ObjectPropertyDomain>
239   <ObjectPropertyRange>
240     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/whistles"/>
241     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Game"/>
242   </ObjectPropertyRange>
243
244   <SubClassOf>
245     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        IllegalScreen"/>
246     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Foul"/>
247   </SubClassOf>
248
249   <ClassAssertion>
250     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Player"/>
251     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/Craig"/>
252   </ClassAssertion>
253
254   <ObjectPropertyDomain>
255     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/playsAs"/>
256     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Schroder"/>
257     </ObjectPropertyDomain>
258   <ObjectPropertyRange>
259     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/playsAs"/>
260     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        PointGuard"/>
261   </ObjectPropertyRange>
262
263   <SubClassOf>
264     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        EightSecond"/>
265     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Violation"/>
266   </SubClassOf>
267
268   <ClassAssertion>
269     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Team"/>
270     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/uminho"/>
```

```
271   </ClassAssertion>
272
273   <SubClassOf>
274     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Technical"/>
275     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Foul"/>
276   </SubClassOf>
277
278   <ObjectPropertyAssertion>
279     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/playsFor"/>
280     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Craig"/>
281     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/aauav"/>
282   </ObjectPropertyAssertion>
283
284   <SubClassOf>
285     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      TwoPointer"/>
286     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Point"/>
287   </SubClassOf>
288
289   <ClassAssertion>
290     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Player"/>
291     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Bridges"/>
292   </ClassAssertion>
293
294   <ObjectPropertyAssertion>
295     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/coaches"/>
296     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Carter"/>
297     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/uminho"/>
298   </ObjectPropertyAssertion>
299
300   <ClassAssertion>
301     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Coach"/>
302     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Carter"/>
303   </ClassAssertion>
```

```
304
305 <SubClassOf>
306   <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
    Flagrant"/>
307   <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
    Foul"/>
308 </SubClassOf>
309
310 <SubClassOf>
311   <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
    Powerforward"/>
312   <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
    Position"/>
313 </SubClassOf>
314
315 <SubClassOf>
316   <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
    Holding"/>
317   <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
    Foul"/>
318 </SubClassOf>
319
320 <ObjectPropertyDomain>
321   <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/playsAs"/>
322   <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
    James"/>
323   </ObjectPropertyDomain>
324 <ObjectPropertyRange>
325   <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/playsAs"/>
326   <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
    Powerforward"/>
327 </ObjectPropertyRange>
328
329 <ObjectPropertyDomain>
330   <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/playsAs"/>
331   <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
    Bridges"/>
332   </ObjectPropertyDomain>
333 <ObjectPropertyRange>
334   <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/playsAs"/>
335   <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
    Powerforward"/>
336 </ObjectPropertyRange>
```

```
337
338 <SubClassOf>
339   <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
    Center"/>
340   <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
    Position"/>
341 </SubClassOf>
342
343 <ObjectPropertyAssertion>
344   <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/playsFor"/>
345   <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/James"/>
346   <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/uminho"/>
347 </ObjectPropertyAssertion>
348
349 <ObjectPropertyAssertion>
350   <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/playsFor"/>
351   <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/Brown"/>
352   <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/uminho"/>
353 </ObjectPropertyAssertion>
354
355 <ObjectPropertyAssertion>
356   <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/playsFor"/>
357   <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/Schroder"/>
358   <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/uminho"/>
359 </ObjectPropertyAssertion>
360
361 <ClassAssertion>
362   <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
    Player"/>
363   <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
    Basquetebol/Brown"/>
364 </ClassAssertion>
365
366 <SubClassOf>
367   <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
    TwentyFourSecond"/>
368   <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
    Violation"/>
```

```
369   </SubClassOf>
370
371   <ClassAssertion>
372     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Player"/>
373     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Ayton"/>
374   </ClassAssertion>
375
376   <ClassAssertion>
377     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Player"/>
378     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Schroder"/>
379   </ClassAssertion>
380
381   <SubClassOf>
382     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      HeldBall"/>
383     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Violation"/>
384   </SubClassOf>
385
386   <SubClassOf>
387     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Travelling"/>
388     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Violation"/>
389   </SubClassOf>
390
391   <ClassAssertion>
392     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Coach"/>
393     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Pop"/>
394   </ClassAssertion>
395
396   <ObjectPropertyAssertion>
397     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/playsFor"/>
398     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Booker"/>
399     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/aauav"/>
400   </ObjectPropertyAssertion>
401
402   <ObjectPropertyAssertion>
```

```
403     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/playsFor"/>
404     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/Payne"/>
405     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/aauav"/>
406   </ObjectPropertyAssertion>
407
408   <ObjectPropertyAssertion>
409     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/playsAgainst"/>
410     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/uminho"/>
411     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/aauav"/>
412   </ObjectPropertyAssertion>
413
414   <SubClassOf>
415     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        ThreePointer"/>
416     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Point"/>
417   </SubClassOf>
418
419   <ClassAssertion>
420     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Player"/>
421     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/Payne"/>
422   </ClassAssertion>
423
424   <SubClassOf>
425     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Slapping"/>
426     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Foul"/>
427   </SubClassOf>
428
429   <SubClassOf>
430     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        BackcourtViolation"/>
431     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Violation"/>
432   </SubClassOf>
433
434   <ObjectPropertyDomain>
```

```
435     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/playsAs"/>
436     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Davis"/>
437     </ObjectPropertyDomain>
438   <ObjectPropertyRange>
439     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/playsAs"/>
440     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Center"/>
441   </ObjectPropertyRange>
442
443   <SubClassOf>
444     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        FiveSecond"/>
445     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Violation"/>
446   </SubClassOf>
447
448   <SubClassOf>
449     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Goaltending"/>
450     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Violation"/>
451   </SubClassOf>
452
453   <ObjectPropertyDomain>
454     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/playsAs"/>
455     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Booker"/>
456     </ObjectPropertyDomain>
457   <ObjectPropertyRange>
458     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/playsAs"/>
459     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        ShootingGuard"/>
460   </ObjectPropertyRange>
461
462   <ObjectPropertyAssertion>
463     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/playsFor"/>
464     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/Bridges"/>
465     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/aauav"/>
466   </ObjectPropertyAssertion>
```

```
467
468  <SubClassOf>
469    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
     Carrying"/>
470    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
     Violation"/>
471  </SubClassOf>
472
473  <ClassAssertion>
474    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
     Player"/>
475    <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
     Basquetebol/Davis"/>
476  </ClassAssertion>
477
478  <ObjectPropertyDomain>
479    <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
     Basquetebol/playsAs"/>
480    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
     Walker"/>
481    </ObjectPropertyDomain>
482  <ObjectPropertyRange>
483    <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
     Basquetebol/playsAs"/>
484    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
     ShootingGuard"/>
485  </ObjectPropertyRange>
486
487  <ClassAssertion>
488    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
     Player"/>
489    <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
     Basquetebol/Booker"/>
490  </ClassAssertion>
491
492  <ClassAssertion>
493    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
     Team"/>
494    <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
     Basquetebol/aauav"/>
495  </ClassAssertion>
496
497  <SubClassOf>
498    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
     Smallforward"/>
499    <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
     Position"/>
```

```
500   </SubClassOf>
501
502   <ObjectPropertyAssertion>
503     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/playsFor"/>
504     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Ayton"/>
505     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/aauav"/>
506   </ObjectPropertyAssertion>
507
508   <SubClassOf>
509     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Freethrow"/>
510     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Point"/>
511   </SubClassOf>
512
513   <ObjectPropertyDomain>
514     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/playsAs"/>
515     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Brown"/>
516     </ObjectPropertyDomain>
517   <ObjectPropertyRange>
518     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/playsAs"/>
519     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Smallforward"/>
520   </ObjectPropertyRange>
521
522   <ObjectPropertyAssertion>
523     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/playsFor"/>
524     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/Davis"/>
525     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/uminho"/>
526   </ObjectPropertyAssertion>
527
528   <ObjectPropertyDomain>
529     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
      Basquetebol/playsAs"/>
530     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
      Payne"/>
531     </ObjectPropertyDomain>
532   <ObjectPropertyRange>
```

```
533     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/playsAs"/>
534     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        PointGuard"/>
535   </ObjectPropertyRange>
536
537   <SubClassOf>
538     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        DoubleDribble"/>
539     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Violation"/>
540   </SubClassOf>
541
542   <SubClassOf>
543     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Intentional"/>
544     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Foul"/>
545   </SubClassOf>
546
547   <ObjectPropertyDomain>
548     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/playsAs"/>
549     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Ayton"/>
550     </ObjectPropertyDomain>
551   <ObjectPropertyRange>
552     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/playsAs"/>
553     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Center"/>
554   </ObjectPropertyRange>
555
556   <SubClassOf>
557     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Pushing"/>
558     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Foul"/>
559   </SubClassOf>
560
561   <ClassAssertion>
562     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Player"/>
563     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/Walker"/>
564   </ClassAssertion>
565
```

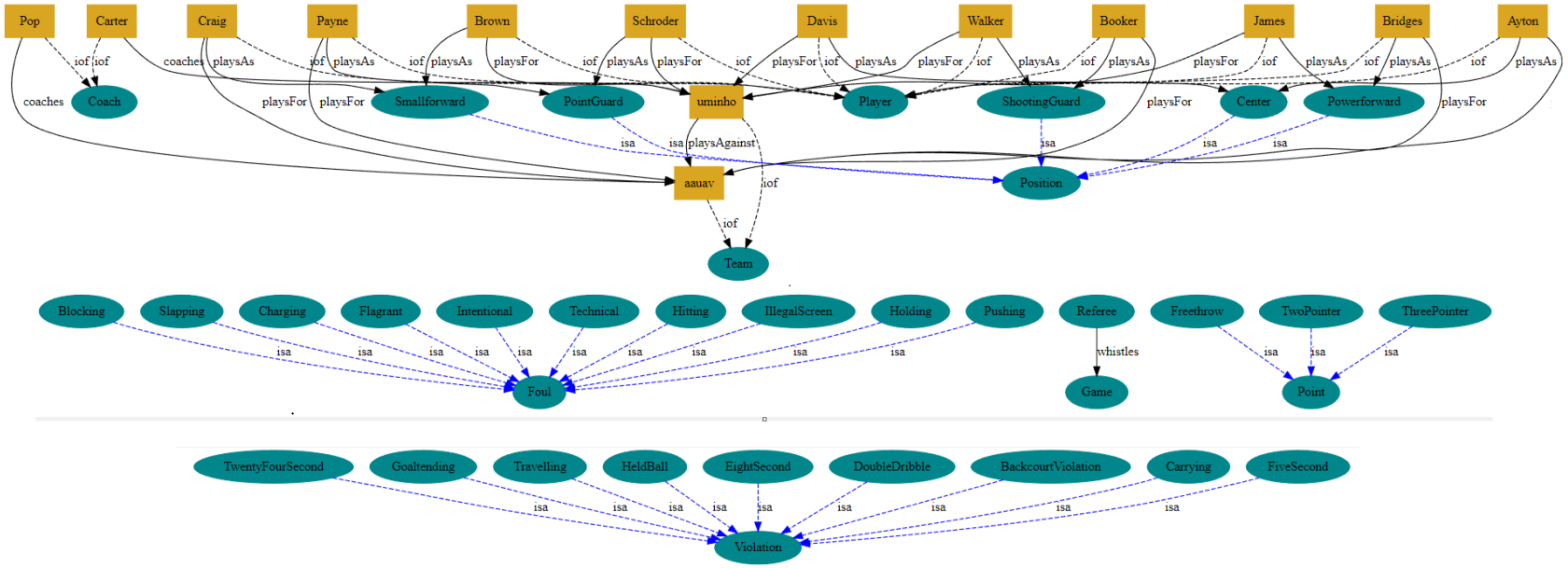
```
566   <ClassAssertion>
567     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Player"/>
568     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/James"/>
569   </ClassAssertion>
570
571   <SubClassOf>
572     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Hitting"/>
573     <Class IRI="http://www.semanticweb.org/simia/ontologies/2020/10/Basquetebol/
        Foul"/>
574   </SubClassOf>
575
576   <ObjectPropertyAssertion>
577     <ObjectProperty IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/playsFor"/>
578     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/Walker"/>
579     <NamedIndividual IRI="http://www.semanticweb.org/simia/ontologies/2020/10/
        Basquetebol/uminho"/>
580   </ObjectPropertyAssertion>
581 </Ontology>
```

B

VISUALISATION OF THE BASKETBALL ONTOLOGY

This Appendix exhibits a graphic visualization of the basketball ontology introduced and discussed along Appendix A.

Notice that one of the graphs is a conceptual view, and the other one is a populated view of the ontology including individuals that instantiate some concepts.


Pop
Carter
Craig
Payne
Brown
Schroder
Davis
Walker
Booker
James
Bridges
Ayton
Coach
Smallforward
PointGuard
uminho
Player
ShootingGuard
Center
Powerforward
aauav
Position
Team
Blocking
Slapping
Charging
Flagrant
Intentional
Technical
Hitting
IllegalScreen
Holding
Pushing
Referee
Freethrow
TwoPointer
ThreePointer
Foul
Game
Point
TwentyFourSecond
Goaltending
Travelling
HeldBall
EightSecond
DoubleDribble
BackcourtViolation
Carrying
FiveSecond
Violation
coaches
playsFor
playsAs
iof
isa
playsAgainst
whistles

BIBLIOGRAPHY

Michelle Cheatham and Pascal Hitzler. String similarity metrics for ontology alignment. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web – ISWC 2013*, pages 294–309, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-41338-4.

Philipp Cimiano. *Ontology learning and population from text: algorithms, evaluation and applications*, volume 27. Springer Science & Business Media, 2006.

Philipp Cimiano, Johanna Völker, and Rudi Studer. Ontologies on demand? a description of the state-of-the-art, applications, challenges and trends for ontology learning from text. 2006.

Philipp Cimiano, Elena Montiel-Ponsoda, Paul Buitelaar, Mauricio Espinoza Mejía, and Asuncion Gomez-Perez. A note on ontology localization. *Applied ontology*, 5, 01 2010. doi: 10.3233/AO-2010-0075.

Alexandre Miguel Costa Dias. Ontodl+: an ontology description language and its compiler, 2021.

Mauricio Espinoza, Asunción Gómez-Pérez, and Eduardo Mena. Enriching an ontology with multilingual information. In *European Semantic Web Conference*, pages 333–347. Springer, 2008.

Mauricio Espinoza, Elena Montiel-Ponsoda, and Asunción Gómez-Pérez. Ontology localization. In *Proceedings of the fifth international conference on Knowledge capture*, pages 33–40, 2009.

Michael R Genesereth, Richard E Fikes, et al. Knowledge interchange format-version 3.0: reference manual. 1992.

Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies*, 43(5):907–928, 1995. ISSN 1071-5819. doi: https://doi.org/10.1006/ijhc.1995.1081. URL https://www.sciencedirect.com/science/article/pii/S1071581985710816.

Tom Gruber. Where the social web meets the semantic web. In *International Semantic Web Conference*, pages 994–994. Springer, 2006.

Tom Gruber. *Ontology*, pages 1963–1965. Springer US, Boston, MA, 2009. ISBN 978-0-387-39940-9. doi: 10.1007/978-0-387-39940-9_1318. URL https://doi.org/10.1007/978-0-387-39940-9_1318.

Manzano-Macho Gómez-Pérez. A survey of ontology learning methods and techniques. onto-web ist project, deliverable 1.5. 2003.

Shimaa Ibrahim, Said Fathalla, Hamed Shariat Yazdi, Jens Lehmann, and Hajira Jabeen. From monolingual to multilingual ontologies: The role of cross-lingual ontology enrichment. In Maribel Acosta, Philippe Cudré-Mauroux, Maria Maleshkova, Tassilo Pellegrini, Harald Sack, and York Sure-Vetter, editors, *Semantic Systems. The Power of AI and Knowledge Graphs*, pages 215–230, Cham, 2019a. Springer International Publishing. ISBN 978-3-030-33220-4.

Shimaa Ibrahim, Said Fathalla, Hamed Shariat Yazdi, Jens Lehmann, and Hajira Jabeen. Oecm: A cross-lingual approach for ontology enrichment. In Pascal Hitzler, Sabrina Kirrane, Olaf Hartig, Victor de Boer, Maria-Esther Vidal, Maria Maleshkova, Stefan Schlobach, Karl Hammar, Nelia Lasierra, Steffen Stadtmüller, Katja Hose, and Ruben Verborgh, editors, *The Semantic Web: ESWC 2019 Satellite Events*, pages 100–104, Cham, 2019b. Springer International Publishing. ISBN 978-3-030-32327-1.

Maria Keet. *Toward Multilingual Ontologies*. 2020.

Thomas Krichel. The semantic web and an introduction to rdf. 2002.

Ted Pedersen, Satanjeev Banerjee, and Siddharth Patwardhan. Maximizing semantic relatedness to perform word sense disambiguation. *University of Minnesota supercomputing institute research report UMSI*, 25:2005, 2005.

Mehrnoush Shamsfard and Ahmad Abdollahzadeh Barforoush. The state of the art in ontology learning: a framework for comparison. *The Knowledge Engineering Review*, 18 (4):293–316, 2003.

The Apache Software Foundation. Apache Jena. http://jena.apache.org/, 2023.

Raquel Trillo, Jorge Gracia, Mauricio Espinoza, and Eduardo Mena. Discovering the semantics of user keywords. *J. Univers. Comput. Sci.*, 13(12):1908–1935, 2007.

PJTM Vossen. Eurowordnet: a multilingual database for information retrieval. In *Proceedings of the DELOS workshop on Cross-language Information Retrieval, March 5-7, 1997 Zurich*. Vrije Universiteit, 1997.

W3C. OWL 2 Web Ontology Language: Semantics and Abstract Syntax. https://www.w3.org/TR/owl-semantics/syntax.html.

W3C. Resource Description Framework (RDF): Concepts and Abstract Syntax. https://www.w3.org/TR/rdf-concepts/, 2014a.

W3C. Resource Description Framework (RDF) Schema. https://www.w3.org/TR/rdf-schema/, 2014b.

W3C. Turtle - Terse RDF Triple Language. https://www.w3.org/TR/turtle/, 2014c.