Semantic modelling of the Portuguese inquisition archives

João Cardoso Caldas Pinto

Mestrado em Engenharia Informática
Especialização em Interação e Conhecimento

Dissertação orientada por:
André Osório E Cruz De Azeredo Falcão

2022
“From childhood's hour I have not been
   As others were
   I have not seen
   As others saw
   I could not bring
My passions from a common spring
From the same source I have not taken
   My sorrow
   I could not awaken
My heart to joy at the same tone
   All I have loved
   I have loved alone.”

-Edgar Allan Poe

To my sweet Luna.
Aknowlegments

I would like to thank my parents that always nurtured and supported my curiosity and thirst for knowledge. All my success is by virtue of their actions. I would also like to thank my supervisor André Osório e Cruz de Azerêdo Falcão that suggested and supported me during this project. His ideas, suggestions and enthusiasm were a constant motivation in the labour of this work.
List of Figures

Figure 1: Web link to different plays ................................................................. 8
Figure 2: Indirect Graph Playwright Sophocles .................................................... 10
Figure 3: RDFS example in turtle format .......................................................... 12
Figure 4: Extract, Transform, Load .................................................................. 16
Figure 5: Original data available online .............................................................. 17
Figure 6: Original file vs curated file .................................................................. 18
Figure 7: Address information given by google maps API .................................... 21
Figure 8: Accusation in the processed JSON file .................................................. 22
Figure 9: Sentences in the processed JSON file ................................................... 22
Figure 10: Overall class view ontology ............................................................... 25
Figure 11: Social status scheme ......................................................................... 37
Figure 12: Triples of the profession “cabo” in the Ontology ................................. 41
Figure 13: Triples of the accusation “pacto com o diabo” in the Ontology ............. 41
Figure 14: Triples of the sentence “bens confiscados” in the Ontology .................. 42
Figure 15: Triples of the individual social status “étnia mestiça” in the Ontology .... 43
Figure 16: A typical address line to parse ............................................................ 43
Figure 17: Triples with the location Portugal in the Ontology .............................. 44
Figure 18: Triples of the “Castelo Branco” and “Covilhã” in the ontology .......... 44
Figure 19: Triples identifying “Sertã” as part of Province_BeiraBaixa .................. 45
Figure 20: Triples of the king reign “D.Pedro II” in the ontology .......................... 46
Figure 21: Triples of “Pombalismo” in the ontology ........................................... 46
Figure 22: An inquisition process in the ontology ............................................... 47
Figure 23: An Individual person in the ontology ................................................ 49
Figure 24: Graphic individual by class ............................................................... 51
Figure 25: Top ten most frequent accusations ..................................................... 51
Figure 26: Top ten most frequent convictions ..................................................... 52
Figure 27: Running the Pellet reasoner in the ontology ....................................... 53
Figure 28: SPARQL query in application ........................................................... 54
Figure 29: Searching by parameters ................................................................. 55
Figure 30: A individual person in the web app ..................................................... 56
Figure 31: Object Property in the web app ......................................................... 57
Figure 32: individual partner in the web app ....................................................... 58
Figure 33: Inquisition process in the web app ..................................................... 58
Figure 34: “Judaísmo” in the web app ............................................................... 59
Figure 35: Divorced class page in web app ......................................................... 59
List of Tables

Table 1: Triple information ................................................................. 9
Table 2: Person class object properties ................................................ 27
Table 3: Person class inferred object properties ...................................... 27
Table 4: Person class data properties ..................................................... 28
Table 5: Person class data restrictions .................................................... 28
Table 6: Inquisition class object properties ............................................ 29
Table 7: Inquisition class data properties ............................................... 29
Table 8: Inquisition class data restrictions .............................................. 30
Table 9: Inquisition Process class object restrictions ............................... 30
Table 10: Subclasses of accusation ......................................................... 30
Table 11: Accusation class data properties ............................................. 31
Table 12: Subclasses of conviction ......................................................... 31
Table 13: Conviction class data properties .............................................. 32
Table 14: Court class object properties .................................................. 33
Table 15: Court class data properties ..................................................... 33
Table 16: Historical period class data properties ..................................... 34
Table 17: Historical Period class object restrictions ................................. 34
Table 18: Subclasses of professions ....................................................... 35
Table 19: Profession class data properties ............................................. 36
Table 20: Subclasses of locations .......................................................... 36
Table 21: Location class object properties ............................................. 37
Table 22: Location class data properties ............................................... 37
Table 23: Social Status class data properties .......................................... 52
## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW</td>
<td>Semantic Web</td>
</tr>
<tr>
<td>ETL</td>
<td>Extract, transform, load</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>SPARQL</td>
<td>Simple Protocol and RDF Query Language</td>
</tr>
<tr>
<td>SWRL</td>
<td>Semantic Web Rule Language</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma Separated Values</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>QN</td>
<td>Qualified name</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
</tbody>
</table>
Abstract

In this work we present a semantic model of the Portuguese holy inquisition archives. This model is ontology based using semantic web techniques. It allows storing, querying and browsing the inquisition archives. From the ground up we extracted the Portuguese Holy Inquisition data from the archives of Torre do Tombo. This data included personal information of a given person like the name, address and profession as well as the process itself like the accusation and sentence. This data had errors, typos and erroneous information that was necessary to filter and transform using proper algorithms and techniques. Afterwards the structure of the ontology was build creating classes, subclasses, object properties, data properties and restrictions. The data was then inserted into the structure using algorithms developed in Python. Logic rules were constructed to allow a reasoner to infer family relationships between the persons present in the archives. The work was concluded by building a web application that allows to navigate all the information in the archives enabling to search among others by a person name, region, profession and accusation. In this work we exhibit the viability of the semantic web to store historical information and the fresh rich knowledge it can give to academics and historians.

Keywords: Semantic Web, Information Extraction, Ontology, History, Historic databases
Resumo Alargado

Desde a sua fundação, a ciência de computação tem evoluído a enorme velocidade alterando com frequência a metodologia de trabalho e investigação. Existe, no entanto, algumas áreas em que a evolução ficou parada ou os avanços são lentos. Um dessas áreas é o armazenamento e interação de dados de eventos históricos. O modelo tradicional que utiliza base de dados relacionais é por vezes inadequado pois tem dificuldades em preservar o contexto da informação. Nesta dissertação propomos uma forma alternativa de armazenar e interagir com dados e eventos históricos apresentando um modelo semântico para os arquivos da inquisição portuguesa.

O projeto foi iniciado com o estudo do trabalho já realizado na área do armazenamento digital de material histórico. Foram analisadas soluções tradicionais utilizando base de dados relacionais e soluções utilizando técnicas de web semântica. Escolhemos os dois trabalhos em cada solução que nos pareceram mais indicados, a base de dados histórica de Estocolmo onde é preservado os dados de todos os habitantes da cidade durante cinquenta anos entre os séculos dezenove e vinte. Escolhemos igualmente a ontologia “stole” que utilizando técnicas da web semântica mantem uma base histórica de documentos legislativos Italianos. Após este processo extraímos a informação dos processos da inquisição portuguesa contidos na página de internet dos arquivos da Torre do Tombo através de um scrapper para um ficheiro de texto. Depois deste processo estar concluído revelou-se necessário iniciar o tratamento e transformação dos dados obtidos, corrigindo erros de codificação e da inserção original dos dados, extraíndo dos dados informação extra como por exemplo as coordenadas de latitude e longitude da morada e local de nascimento e a categorização de diferentes informações como por exemplo a profissão, estado social ou acusações. A informação foi estruturada e inserido num ficheiro JSON.

Concluído o processo iniciou-se uma segunda fase do projeto. Foi feito um estudo dos dados contidos no arquivo da Inquisição Portuguesa de modo a construir um modelo semântico com capacidade de albergar a informação previamente tratada. Num processo progressivo e construíu-se o modelo utilizando a linguagem de modelação conhecida por “Web Ontology Language”. Criaram-se as classes, propriedades de dados, propriedades de objetos e restrições adequadas ao modelo e resultado pretendido. De seguida utilizando a linguagem de programação Python, a biblioteca owlready2 e técnicas de computação, iniciou-se o exigente e demorado processo de inserção dos dados tratados no modelo. A primeira preocupação foi inserir e fazer as respetivas ligações dos vários países, distritos, municípios e localidades presentes nos arquivos da Inquisição Portuguesa. Após esse processo inserimos as instâncias das profissões, acusações, sentenças e estatuto social tendo o cuidado de fazer a respetiva ligação entre as instâncias e a sua respetiva categorização. De seguida iniciamos o processo de inserir as instâncias das pessoas propriamente ditas. Primeiro inserimos os dados literais como o nome e idade, em seguida fez-se a ligação entre a instância da pessoa e os objetos já inserdos como a profissão, localidade, acusação e estatuto social. Por fim realizamos o processo de inserir o pai, mãe e cônjuge da pessoa, como queríamos evitar criar pessoas se estas já existissem no modelo utilizamos algoritmos de pesquisa para tentar procurar se estes já existiam no modelo, se existissem o algoritmo fazia a ligação caso contrário criava a pessoa e só depois a ligação era efetuada. Construímos de seguidas uma sèrie de regras na linguagem SWRL que permitem a inferência de relações familiares que não estão explicitas nos arquivos da Inquisição como por exemplo avô, avó, irmão, irmã, tio, tia, etc.
Utilizando a linguagem de query SPARQL foi verificado se o resultado obtido era o pretendido, foram necessários múltiplos ajustes aos algoritmos para obter uma representação correta dos dados no modelo. Este processo foi particularmente demorado devido ao elevado número de processos presente no arquivo da Inquisição Portuguesa superior a 25.000 processos fazendo com que a inserção dos dados com os algoritmos de pesquisa fosse superior a doze horas.

A última fase do projeto foi a construção de uma página de internet que permitisse a navegação e exploração do modelo. Através do framework Django e da biblioteca owlready2 procedeu-se ao desenvolvimento de um portal que permitisse a navegação do modelo. Neste portal é possível procurar processos de Inquisição de pessoas pelo seu nome, localidade, profissão, acusação, etc. O portal permite a pesquisa através de um query na linguagem SPARQL. Após a pesquisa, o portal permite a exploração do processo em si detalhando não só toda a informação do processo em si como permitindo através de links explorar toda a informação associada aos vários campos como, por exemplo a categoria da profissão ou das várias acusações. Outra funcionalidade é que permite igualmente ir diretamente para os dados dos familiares da pessoa em questão. O portal permite ainda uma listagem automática das classes e propriedades do modelo possibilitando a exploração do seu nome, restrições e meta informação.

**Palavras-chave:** Web Semântica, Extração de informação, Ontologias, História, Base dados históricas
## Contents

1 - Introduction
- 1.1 - Work Introduction .......................................................... 1
- 1.2 - Aims and Scope .............................................................. 2
- 1.3 - Significance of the study .................................................. 2
- 1.4 - Overview ..................................................................... 3

2 - Related Work
- 2.1.1 - Introduction ................................................................. 4
- 2.1.2 - Stockholm Historical Database ................................. 4
- 2.1.3 - Stole: An Ontology for Historical Research Documents .. 5

3 - Semantic web tools
- 3.1 - Introduction .................................................................. 7
- 3.2 - Semantic Modeling ......................................................... 7

4 - Modelling Historical records with semantic web
- 4.1 - Introduction ................................................................. 15
- 4.2 - Extracting and transformation of the data ......................... 16
- 4.3 - The Ontology ............................................................... 24
- 4.4 - Conclusion ................................................................. 38

5 - Inquisition data in the model
- 5.1 - Insertion of the data ....................................................... 39
- 5.2 - Exploration of the data .................................................. 49

6 - Web Application
- 6.1 - Introduction ................................................................. 51
- 6.2 - Application ................................................................. 53
- 6.3 - Conclusion ................................................................. 59

7 - Conclusions
- 7.1 - Introduction ................................................................. 60
- 7.2 - Struggles and challenges ............................................... 60
- 7.3 - Future Work ............................................................... 60

Bibliography ............................................................................. 62
Appendix A ............................................................................. 62
Appendix B ............................................................................. 62
Appendix C ............................................................................. 71
1- Introduction

1.1 - Work Introduction

While western civilization has been fascinated with history since in ancient Greece Herodotus has gifted us with his histories (Waterfield et al., 2008) and a large quantity of studies and books have been written using historical data and documentation we still lack in computer science a universal accepted way of storing and connecting those historical records. Along the years several studies using historical data from a given country, region or continent have been made. One example is the study of the gross domestic product (GDP) over a certain period of time to understand the evolution of a given economy and the consequences it produced. Another might be the study the evolution of temperature and air quality to understand the progress of global warming in a given area. However, if we wish to study the life of an Ottoman soldier that fought in the first siege of Vienna in the 1529, we have to be thoughtful about the data. It may not be enough to store the name of the place he was born or lived as this name may have changed, changed culture and language or even do not exist anymore. In fact, the Ottoman empire has ceased to exist, and its regions are now occupied by a vastly different people with different cultures and beliefs.

To use the information in all its potential we need to find a way to have contextual and semantic information of the location. So, for example if that soldier was from the ancient Ottoman city of “Buda” we must find a way to represent the information that “Buda” is now part of the city of Budapest that in its term is the capital of modern-day Hungary. If we accomplish this if we want to study the people that fought for the Ottoman empire that originated from the location that is now Budapest, we will find our soldier. Finding a way to connect different sources of information is also important. If one database has the information about the birth and family of the soldier and a second database has the information about its military career, then if we are capable of seamlessly connect both databases then we can have a richer overview of his life and history. History is brimming with events and institutions that are complex, interconnected and worth the effort to experiment different technique in order to extract and study all the knowledge and information that it can give us.

One of those institutions is the Holy Inquisition which was particularly prevalent in Portugal and Spain (Netanyahu, 1995). The holy inquisition had its foundation in the twelve century France. Its aim was to combat heresy by investigating and conduction trials of suspected heretics. During this early period, known as medieval inquisition, it also expanded to Italy targeting mainly the spiritual Franciscans, the Hussite's and the Beguines (Netanyahu, 1995). In later Middle Ages in a response to the protestant reformation it expanded and grew mainly in the Portuguese and Spanish empires whose main target where converted Jews and Muslims (Marcocci & Paiva, 2016).
The Portuguese holy inquisition was active from 1536 to 1821 (Marcocci & Paiva, 2016). During this period thousands of people were accused and put on trials. This defunct institution left behind a rich registry of data that can be used to study and understand not only the practices of the Holy Inquisition but also the history of the country, its families and their costumes. While this data was all naturally connected in its time storing this data while keeping these connections is a challenge.

In the rich and complex world of today computer science there are various techniques and technologies that can be used to approach this problem. Semantic Web techniques can be used to formally represent metadata and information. It can use ontologies to represent the definition of categories, properties and the relations between concept, data and entities. It is today mainly employed and researched in medicine applications but by no means it potential cannot be applied in other fields. In this work we will explore these techniques and methods with the intention to apply them to the data available of the Portuguese Inquisition.

1.2 - Aims and Scope

In this work the aim will be to build a working Ontology of the Portuguese Inquisition, curate the available data and load it into the ontology. It should be constructed in a way that allows proper historians to correct any mistakes made in the curation of the data (for example a defunct profession categorization, two people with slightly different names being the same, etc.) and to be expanded in the future.

The data used was digitalized by Torre do Tombo in Lisbon, Portugal available online at their website. The data was scraped to a text file and then treated and curated. This data result from a handwritten process over the span of three centuries and was in natural need of heavy curation, from simple writing errors, different words to represent the same facts, to bad filling of a given template (that might have changed in the centuries that the inquisition was active).

1.3 - Significance of the study

With this work we will present a valid and alternative method to store historical data. Is is our perspective that semantic web is underutilized and unfairly forgotten in the overgrowing world of computer science. With its ability to connect data and sources it can allows a bridge to a better understanding and identification with our collective past.

It is also the objective to contribute for the preservation and research of historical data that is unique to Portugal. Unfortunately to often with the troubles of our stormy present we tend to forget the rich and powerful history that our ancestors left us to learn and preserve.

1 https://digitarq.arquivos.pt/details?id=2299704
1.4 -Overview

This thesis consists of six further chapters. In chapter 2 we will make an overview of the state of art, studying some previous and relevant models of structured and semi-structured historical databases. Afterwards the main concepts of the semantic web will be briefly discussed so a better framing of the work can be understood. In chapter 3 we will discuss the challenges and decisions necessary to model historical records with the semantic web. In the following chapter (4) We will present the results that were achieved. In chapter 5 we will discuss these results and the web application developed to navigate the solution. Finally in the last chapter (6) the final conclusions will be presented and discussed.
2 - Related Work

2.1.1 - Introduction

In this chapter, we will begin by a small state of art review discussing some structured or semi-structured solutions to store, preserve and study historical information. Afterwards we will go through the main concepts of the semantic web so this work can be better followed and understood.

2.1.2 - Stockholm Historical Database

The Stockholm Historical Database (SHD) is a unit within the Stockholm City Archives that has been digitising historical source material since the middle of the 1970s (Agneta Geschwind, Stefan Fogelvik, The Stockholm Historical Database - IPUMS). Their main source is the Roteman Archives created by the Roteman institution. This institution was founded under a government decree in November 10 of the year 1877. In January of the next year Stockholm was divided into sixteen wards called rotar, with about 8.000 to 10.000 inhabitants each. Every rote was assigned to one roteman that administered the Roteman office. This roteman population registration system was in operation for almost fifty years (1878-1926). The database is constituted by all inhabitants that lived in Stockholm between those years.

The information was originally kept in a longitudinal population register—mantalsbok (ledger). On the front page of each ledger some general information was kept: the ledger name, the period covered, the number of the ward, the address and street number. Inside each double-spread page contained forty-one columns with printed headings that gave information about the population. This information included the name, occupation, place of birth, sex, civil status. Etc.

Due to the fragility of ledgers and the determination to make it available to a large range of people including historians, researchers and families a process to transform this data into machine readable form was begun in 1970 and has yet to be finished due to the size of the archives. The SAS system is used for data entry, data management, and as a tool to serve researchers with extraction and processing of information from the Roteman Files.

To store the information a relational database model is used. The data is stored as a flat file that are divided into libraries in accordance with the ward-divisions. For each one of these wards there exists several flat files with relational linking capabilities in the form of variables in the tables. This is done to make the digitized information system independent. By linking the primary entries (sex, data of birth, names, birthplace, etc.) a biography for each individual can be built. If we used the page and line for the head of household in combination with the family standing families and households can also be
reconstituted. For future work they intend to conclude the digitization of the remaining wards and to build a web-application that allows its navigation.

### 2.1.3 - Stole: An Ontology for Historical Research Documents

In the paper entitled “An Ontology for Historical Research Documents” (Adorni et al., 2015) is introduced the ontology STOLE and ontology-based digital archiving with the goal of helping researchers to organize data.

The authors state that the objective of this ontology is to model historical concepts while allowing historians to discover unexplored aspects about particular event or person. STOLE collects data about the legislative history of public administration in Italy using the most important journal papers published between 1848 and 1946. The creation process of the ontology had three main phases:

i. Identification of key concepts.
ii. Identification of the proper language and how to implement it.
iii. Ontology population.

In the first phase with the help of domain experts the definition of the key issues of the application domain were identified. The main focus was to identify the main categories of the historical documents. They computed a taxonomy composed of three elements:

- Data concerning the author of the article, e.g., name, surname and biography.
- Data concerning the journal and the article, e.g., article title, journal name, date and topics raised in the article.
- Data concerning some relevant facts and persons cited in the article, e.g., persons, historical events, institutions

The focal point is the interrelations between the data, they give the example that the link between an author and the people cited in an article can provide invaluable information to historians. e.g., if an author has often referred to King Vittorio Emanuele II probably it can easily be interpreted as favorable to the monarchy.

In the second phase the STOLE ontology was designed upon existing standards and meta-data vocabularies such as Dublin Core, FOAF, the Bio Vocabulary, the Bibliographic Ontology and the Ontology of the chamber of deputies (Ontology of the Chamber of Deputies) they used the modelling language OWL2 DL (Horridge & Bechhofer, 2011), classes and properties were created to model the data. Some examples of classes are Article that represents the library, event that designates relevant events and subject that represents the various topics contained in the historical journals.

In the third phase the ontology was loaded by means of a JAVA program built on top of the OWL APIs (Horridge & Bechhofer, 2011), semantic annotations were provided by a team of domain experts. While the ontology is built for a very specific domain the authors claim that the ontology can be used for other applications that relates to the history of the Italian administrations and institutions. For future
work the authors explain that a Graphical User Interface to support the ontology population state is being designed and they also intent to use DL reasoners for both classification and query answering tasks.
3 - Semantic web tools

3.1 - Introduction

Since the dawn of the world wide web, also known as the internet, that we have been interested in how to represent knowledge and how to define the meaning of that knowledge. This field is called semantics and is a branch of linguistics and logic that is concerned with meaning. We always have been interested in this because we strive to have an intelligent web and not a static one.

Imagine one finds itself in holidays in England and desire to visit the breathtaking botanic garden of Kew (Kew) one might visit their website to learn the best way to travel there. One might find they provide a convenient bus that get one person from where they are staying to the gardens, one might then visit the bus company website to buy a bus fare but is unable to find the route he wanted as it is no longer offered by the bus company. One might, understandingly, get irked and complain about computer scientists and web-page managers for not being synchronized with each other in their platforms and services. This kind of phenomenon is not driven by a lack of intelligent applications, in fact amazing applications are built every day that are capable to do things that would seem impossible just a few years ago. The issue is that most of the times, they are small islands that are incapable to share their resources (in this case information) and instead they only use their own. Even the most insightful and intelligent application is limited to the information it possesses at a given time. This is ideally, where semantic web comes to the rescue by attempting to bridge the gap of information between applications allowing them to be as intelligent as they can possibly be. It is not the job of semantic web to be intelligent, it leaves that job to the applications, its goal is to make an infrastructure that is most appropriate for the job of integrating information on the Web.

3.2 - Semantic Modeling

One of the issues (and maybe its malison) of the world wide web is that anyone can say anything about everything (known as AAA). Scientifically we know that tomato is a fruit (Is a tomato a fruit) but a nutritionist may consider it a vegetable as it have all the benefits of one in their eyes. Some people indeed say they can be both and who knows if in the future some smart person will come along and prove it is neither. We need a way to sort all this information in a coherent way. A rich variety of tools have been development over the year to attempt to solve this issue, but they all have something in common: They help people understand the world by forming an abstract description that hides selected details and exposes others. These abstractions are called models.
The semantic Web has been created not only to facilitate the sharing of information between different people and different applications but as a medium in which collaboration on models is possible. One way this is accomplished is with a simple idea any model can be built up from contributions and information from different sources (Allemang et al., 2012). One way to visualize this idea is to consider a Semantic Web model as a trunk of an old tree. Each ring comes from a different source and time frame while still being part of the same trunk.

We understand semantics in natural languages as the rules that allows to give meaning to the combination of symbols (letters). In programming languages, we understand it as the syntax of a given language so that its compiler can transform it in machine language that than is run and produces the results the programmer intended. In the Semantic Web we intent to give structured connections to information.

In the standard web that we have been using for several decades we expect a network of documents connected by links. For example, we can have as illustrated in figure 1 a webpage with a link to the play Antigone from the playwright Sophocles (Sophocles, 2013) and a link to the play Electra from the playwright Euripides (Euripides et al., 2013). We do not expect any information to be given between the links, any connection between both is normally made by the person reading/using the documents.

![Figure 1: Web link to different plays](image)

He might know that both playwrights are from Ancient Greece, or he may not. In the Semantic Web everything in the world is a resource. A resource is anything that anyone may want to talk about. May it be a famous play, a person or a simple individual member of the genus *rattus* (Rattus). These resources can be in different servers from different parts of the globe, in fact in the context of the semantic web we expect them to be. In our example one server might have the Electra play while other have who Euripides was while a third one might have the information that they were both citizens of Ancients.
Greece. For this distributed solution to be able to work the servers need to be able to “speak” in the same language. This basic block that semantic web relies on is called RDF which is short for Resource Description Framework.

**Resource Description Framework (RDF)**

The Resource Description Framework (RDF) (RDF - Semantic Web Standards) is a World Wide Web Consortium standard originally designed as a data model for metadata. It has come to be used as a general method for description and exchange of graph data. The basic block of RDF it’s how it represents a cell of information. If, for example, we want to express the knowledge expressed in the following table row:

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sophocles</td>
<td>Occupation</td>
<td>Playwright</td>
</tr>
</tbody>
</table>

Table 1: Triple information

We could write *Sophocles :: occupation :: playwright*. With these three values we can define the information we intended. In RDF this is called a triple. If we think this as a row in a table (as it can be seen in table 1) then the identifier for the row is called the subject of the triple, The identifier for the column is called the predicate of the triple, the value itself is called the object of the triple. If there are multiple triples referring to the same identify they can, in fact, be seen as indirect graphs.

For example, if we add extra information to Sophocles:

- sophocles :: occupation :: playwright
- sophocles :: wrote :: antigone
- sophocles :: wrote :: oedipus the king
- sophocles :: wrote :: ajax
- sophocles :: cityOfBirth :: colonus
- sophocles :: foughtAt :: marathon

We can construct with the indirect graph shown in figure 2:
We can continue to add information via triples to express for example that *Colonus* was a city which was part of Attica that was controlled by the city state of Athens. By the way of triples, we can add almost limitless information about the resources we are interested in. As mentioned previously it is expected that the information can be found in multiple servers across the web so if server A has the information of the plays that Sophocles wrote, and server B has the information about Sophocles military accomplishments we need to guarantee that the Sophocles of server A is the same person of the one of server B. In *RDF*, this problem is resolved through the use of Uniform Resource Identifiers (URI).

URI look and feel to the naked eye very similar to regulars Uniform Resource Locators (URLS) that we use every day in the regular web. For example, a *URI* referring to Sophocles could be http://www.semanticmodel.org/theater.owl#Sophocles (In fact a URL is technically a special case of a URI). They are nonetheless a powerful way to uniquely identify and object. Two different web applications located in distinct parts of the world can make reference to the same thing by referencing the same URI. One of the advantages of the syntax of the URI is it allows to use all the information it contains (name, protocol, server, etc.) to locate the file in the web (Allemang et al., 2012). In that way if two servers use these same unique identifiers, we know that they are refereeing the same object (Semantic Web also has the property *SameAs* to indicate two objects are the same (RDF - Semantic Web Standards), for example we can indicate that object A is the same as object B). Merging two graphs located in different servers is then easier as it is done by combining objects with the same URI.
While URIs work efficiently for computers not surprisingly they are not very pleasing to our human brain due to their length. To ease this issue, we use something called qualified names (qnames) to abbreviate them. Qnames consist of two different parts: a namespace and an identifier that are separated by a colon. So for an example we could reference Sophocles by SM:Sophocles. SM would be short for the name space http://www.semanticmodel.org/theater.owl and Sophocles the object identifier.

RDF is at its core a flexible way to model data. Its main goal is accommodated and easily merge data that can be store all around the world. It accomplishes this by having every single relationship between two elements explicitly represented. While it loses to other models in compactness and readability it has the advantage of eliminating the necessity of arranging the columns of tables to make sure they match up. We also do not need to be worried that there may be some missing data that compromises everything about an element. This is a crucial quality in a world where the information is increasingly disconnected across the globe.

Resource Description Framework query

It is unavailing to store information about something if we cannot retrieve it. This is true with RDF and during the early days of its development some querying engines were built. From the various solutions and with the invaluable contribution of W3C and its standardization practices a standard RDF query language has emerged: SPARQL. This query language is used thought this work, an introduction to the language can be found in appendix A.

Resource Description Framework Schema (RDFS)

RDF capabilities can be expanded using variants like RDF-Schema(RDF schema 1.1).While Semantic Web modelling with RDF is mostly concerned with graphs RDFS focus is on sets (Allemang et al., 2012). It provides the ability to talk about which and how individual relate to each other, how the properties used to define an individual relate to other individuals and to each other. It allows to group individuals in sets and answer questions about them (Allemang et al., 2012).

The basic method to construct a set in RDFS is called a rdfs:class. To do this rdfs uses a triple with the predicate rdf:type and the object rdfs:class. By specifying that something is a set we gain a description of the meaning of membership in a set (Allemang et al., 2012). If for example want to connect Alice to the class human we can say than the individual Alice is rdf:type writer this indicates that Alice is part of the subset writes. Sometimes this can be confusing as we can also use object properties for this affect and it’s not clear which is the more appropriate. For example, in the previous example, we may also create the object property HasProfession and say Alice HasProfession Writer. In this way we are also indicating that Alice is part of the subset writers. The difference is subtle and sometimes difficult to choose the better approach. If we indicate that Alice is rdf:type human and human is a Subclassof writer, then the inference engine will infer that all writers are humans which may not be true (as we know AI machines can now write stories and who knows what the future entails in that aspect) but if we are not preoccupied with that kind of concerns the only difference would be in the way we construct the SPARQL queries.

The description that sets can offer only has meaning in rdfs if more than one class exists and at least some relations between the classes are knows. If something is a member of a class, then this means it is also members of the superclass (Allemang et al., 2012). This is useful and allows the introduction of the rdfs:subClassOf which allows us to in indicate that if something is part of a class and that class is a subclass of another class than that something is also member of the other class. This can easily be
understood if we think about a football player. A football player is a professional athlete so if we have a class `footballPlayer` that is a subclass of the class `professionalAthlete` then if we say that a player is member of the set of the class of football players then he is also a member of the class professional athletes.

RDFS also allows to introduce this idea in properties, so if we have the property `has_location` and the property `has_city` and by using a triple we say that `has_city` is a `rdfs:subPropertyOf` of `has_location` then any object that has the property `has_city` also has the property `has_location`.

**Web Ontology Language (OWL)**

Other evolution of RDF is OWL that provides a systematic treatment of information description. OWL allows to describe classes in terms of other things that have already been modelled which opens a new world of modelling possibilities.

Let’s return to our previous example of ancient Greek drama. Suppose we have a class called `TheatrePlay` with a `rdfs:subClassOf` `AncientGreekPlay`, and another class called `TheatreActor` that is a subset of the class `profession`. Any actor of a given play would be connected to it by the property `ActorIn` that connects `TheatrePlay to TheatreActor`. In figure 3 we can see this declared in the Terse RDF Triple Language (Turtle) which is a syntax and file format for expressing data in the RDF data model (Turtle 2022). A few actors would also be connected to the `AncientGreekPlay`. A play also has a writer and director so classes and properties for them would need to be created and so on until we have the play completely defined.

```
### http://www.semanticweb.org/InquisitionThesis#ActorIn
:ActorIn rdf:type owl:ObjectProperty ;
   rdfs:domain :TheatreActor ;
   rdfs:range :AncientGreekPlay .

### http://www.semanticweb.org/InquisitionThesis#AncientGreekPlay
: AncientGreekPlay rdf:type owl:Class ;
   rdfs:subClassOf :TheatrePlay .

### http://www.semanticweb.org/InquisitionThesis#Profession
:Profession rdf:type owl:Class .

### http://www.semanticweb.org/InquisitionThesis#TheatreActor
:TheatreActor rdf:type owl:Class ;
   rdfs:subClassOf :Profession .

### http://www.semanticweb.org/InquisitionThesis#TheatrePlay
:TheatrePlay rdf:type owl:Class .
```

Figure 3: RDFS example in turtle format

One of the great powers of the Semantic Web is that information that has been specified by one person in one context can be reused either by that person or by others in different contexts (Allemang et al., 5 2012). It allows the creation of powerful descriptions of classes using restrictions.
A restriction in OWL is a class that is defined by the description of individuals it can contain (Allemang et al., 5 2012). To be able to use the information from different sources stored in different servers we need a method that is able express concepts from different contexts in terms of the concepts of different objects or classes. OWL does this by being able to describe a new class based in class that have already been created. For example, an actor is someone that is a person, and it acts in plays (being as a profession or a hobby), a person is someone that has a name, gender, age, etc. By adding restrictions, we can clearly describe almost anything we want to. In OWL the semantic web mantra that anyone can say anything (AAA) still stands. All classes are subclass of the “mother” class Things (owl:thing) that is completely unrestricted. A restriction is therefore defined by providing descriptions that will limit the things that can be attributed to one or more class.

OWL defines three main restrictions: `owl:allValuesFrom`, `owl:someValuesFrom` and `owl:hasValue`. Each constrains the possible asserted values of a class in a different way:

**AllValuesFrom:** This restriction is used to indicate that the individual for which all values of the property A come from class X. This means that if we have a new class AntigoneActors that has a allvaluesfrom restriction to the class AncientGreekActors then all members of the antigoneActors class is a member of the AncientGreekActors.

**SomeValuesFrom:** This restriction is used to indicate that all individuals for which at least one value from the property A comes from class X. This means that we can define a ancient Greece theatre actor to all individuals that acted in at least one play from ancient Greece.

**HasValue:** This restriction acts only in a particular property. It indicates that all individuals have the value A for the property X. So for example all females would be all individuals that are persons and have in the property HasGender with the value female.

**Inference**

One of the strongest abilities of OWL and RDF and its variants is the capability to make inferences. This capability allows a system to infer new facts from existing data based on inference rules or ontologies. This simply means that given some stated information, we can determine other, related information that we can also consider as it has been explicitly stated (Allemang et al., 5 2012). Its mechanism can be a simple or complex process that takes powerful processing. For the purpose of integrating data is often preferable to rely on simple inference than complex ones that risk not being what we intended or taking so long to calculate that they cease to be useful (Allemang et al., 5 2012). We can help the inference engine by stating clear constrains to the relations instead of allowing the engine to do guess work.

**Ontologies**

An ontology is an old concept in human though, it is a philosophy branch that studies concepts such as existence and reality (Ontology 2022) it includes the question of how entities are grouped into categories and how they exist and communicate with each other. In computer science an Ontology is a model for a given knowledge domain and a way to navigate and reason on that domain.
OWL is a language to build and manage ontologies and it designed for use by applications that need to process the content of information instead of just presenting information to humans (Owl Web Ontology Language Overview). It has three increasing-expressive languages:

- **OWL Lite:** It’s the simpler and more efficient of the implementations, it’s made for uses that primarily need a classification hierarchy and simple constraints.
- **OWL DL:** It is built for the users who want the most possible expressiveness while guaranteeing computational completeness. It includes all OWL constructs, but they can be used only under certain restrictions.
- **OWL Full:** Is built for the users that want all the expressiveness with no computational guarantee, it gives virtual total freedom of RDF, but it is unlikely that any reasoning engine is built in the near future that is able to process it. In OWL full a class can, be a collection of individuals or an individual on its own (Owl Web Ontology Language Overview).

In this work we will be using OWL lite to construct the ontology of the Portuguese Holy Inquisition.

**Expressions, Entities and Axioms:**

OWL has inbuilt three different kinds of elements to represent knowledge: Expressions, Entities and Axioms. Entities are used to describe and relate objects in the domain of the ontology, they can be classes, named individuals, object properties and data properties. Axioms are all the knowledge (or statements) present in the ontology, they are asserted true by default but can also be asserted false if certain conditions are met. Expressions are more complex entities defined by logically combining different entities.

**Different kind of entities:**

Named individuals refers to concrete entities while classes represent abstract ones. Let’s imagine that Alice is a forty-seven-year-old mathematics teacher she represents an actual person which is an entity, however we can have the class teachers that Alice is an element of but does not really exist. However, both can have properties. There are two main properties in OWL, object properties that connect different entities and data properties that connect an entity to a data value (called literal in semantic web). So, for example if we want to represent that she has certain parents we can do it via object properties (e.g Alice HasMother Vivianne, Alice HasFather Mathias). In the other hand if we want to represent her date of birth, we use data properties (e.g Alice DateBirth 21-03-1974).

**Properties:**

Properties in OWL can have important characteristics worth mentioning:

- **Functional:** A functional property is a property than can only have one unique value y for each instance x. For example, if we create the object property “HasFather” and make it functional then a person can only have one father in the ontology.
- **Transitive:** A transitive property is a property that relates pairs connected to each other. So if the pair (x,y) is an instance of P then (x,z) is also a instance of P. This property is mostly used when we want to represent regions of the ontology connected to each other. For example, if we say Boston is part of the state of Massachusetts and we have stated that Massachusetts is part of the United States then in a transitive property we know that Boston is also part of the United States.
Symmetric: A symmetric property is a property than if (x,y) is a an instance of P then (y,x) is also an instance of P. An example would be if Matthias is friend of Jay, then Jay is also friend with Matthias. Then FriendOf could be a symmetric property.

Inverse: An axiom of the form P1 “owl:inverseOf” P2 asserts that for every pair (x,y) in the property extension of P1, there is a pair (y,x) in the property extension of P2, and vice versa. Thus, “owl:inverseOf” is a symmetric property (Owl Web Ontology Language Overview) The idea is if Vivianne owns a radio x then it is also true that radio x is owned by Vivianne.

Cardinality: In OWL we assume that any instance of a class may have a arbitrary number of values for a given property. If we want to guarantee that a given property have a specific number of values, then we can use a cardinality restriction. If we want to restrict the maximum number of values we use owl:maxCardinality and if we want to restrict the minimum number of values we use owl:minCardinality (that exists also the owl:cardinality but it is redundant). (Owl Web Ontology Language Overview).

Equivalence and disjointedness: In OWL we can also define a class as Equivalent to other or to be disjointed from another. If we say that class A is equivalent to class B then we are explicit saying that all elements of class A are also members of class B, for example we can say that the class Automobiles is equivalent to the class Cars. In other hand if we want to explicit say that all elements of class A are not members of class B then we can indicate that class A and B are disjoint.

We can also indicate this with individuals, by using “OWL:SameAs ” we can indicate that the individual A is the same than B (imagine a book in different languages). We can also explicitly say than individual A is NOT individual B by using “OWL:DifferentFrom ”.

3.3 - Protégé

Protégé (Stanford Center for Biomedical Informatics Research) is a free, open-source ontology editor and framework for building intelligent systems. It was developed by the Stanford centre for biomedical informatics research at the Stanford University School of Medicine. It fully supports the latest OWL Web Ontology Language and RDF specifications from the World Wide Web Consortium. It provides a graphical interface to build and edit ontologies, it supports instances and inference engines. It was built in Java. Protégé was used in this work in to validate the data once it was loaded into the ontology, in appendix C this validation is detailed.
4 - Modelling Historical records with semantic web

4.1 - Introduction

In this third chapter we will go through the necessary steps to construct an ontology of the Holy Portuguese Inquisition. We will start by discussing the data extraction and necessary transformations to be able to have a workable and coherent data. Afterwards we will explore the structure of the ontology itself explaining the processes and decisions made.

4.2 - Extracting and transformation of the data

Introduction:

To begin the process of creating an ontology to fit the Portuguese inquisition first we need to look at the data that is available and make it go through an Extract, Transform, Load process. This is a process that typically a software engineer uses to extract information from a source, transform the data into a usable and trustworthy form and then load the data into the warehouse of data. In figure 4 we have an illustration of this process.

The original data from the Portuguese inquisition was digitalized from the archives by Torre do Tombo and made available online\(^2\). The original data has around 38,000 different inquisition processes. Of those around sixty five percent refer to men, thirty percent women and five percent of gender unknown.

\(^2\) [digitarq.arquivos.pt/details?id=2299704]
Their age ranges from as young as twelve to as old as seventy-five. While most of the cases naturally refer to Portuguese residents there are cases from around the world: Brazil, United States, Spain, Netherlands, Angola, among others. The archives where originally handwritten and are, as expected, full of typos, errors, and different forms of writing the information (it spanned several centuries). These aspects had not been corrected in the data available in the digital format available at *Torre Do Tombo*. This online information was scrapped to a text file and this work begun afterwards. Using python we processed the text-file and cleaned and transformed it so it could be used in the ontology. The data was loaded into a JSON structure.

![Original data available online](image)

**Figure 5: Original data available online**

In figure 5 you can see the original format of a process available in *Torre do Tombo* website.
In figure 6 we see the original file and the file resultant of the transformation of the data. In the following sub-chapters I will go through the process to get from one file to the other.

**Character Encoding**

The first issue that was tackled in this process was the encoding problem found in the original scrapped data. For example, let’s look to this row in the text file:

```
2299978 Processo de Inês Mendes, 1558-12-06/1560-12-11 (PT/TT/TSO-IL/028/00108)
```

As it can be seen instead of properly displaying *Inês* we have the unicode #234. This is called an encoding problem. This occurs because computers only deal with numbers and then text (like everything else) is stored as a sequence of numbers where each character has a unique number that has been previously agreed upon (this is called character encoding). The problem arises because there exists a considerable quantity of different standards and each one stores a different number to each character. In our example the ê is stored as 234 in the ISO-8859-1 standard but stored as two bytes number 50090 in the UTF-8 standard. Passing from one standard to the other then causes the problem we were facing.

*Python* has some built-in methods that are supposed to solve this problem but unfortunately, they did not find success in this particular case. We ended up building a function that receives a string, checks if it contains a Unicode and if so, substitute it for the proper character. So, in our example the function would find the &#234; and replace it with ê. While it is not a perfect solution timewise the data processing is only supposed to be run once which can be argued makes the time issue not very relevant in our work.

---

3 English Translation (by the author): 2299978 Process of Inês Mendes, 1558-12-06/1560-12-11 (PT/TT/TSO-IL/028/00108)
**Process ID**

Using the unique inquisition process identification, we can extract a multitude of information. A typical process id is written like this: 2299869 *Processo de Manuel Gonçalves Laranjo, 1759-10-10/1761-08-13 (PT/TT/TSO-IL/028/00003)*

By processing this string, we can extract the following information:

- Name of the person
- Process Start Date
- Process End Date
- Process number
- Location of the court

Firstly, we used the first six digits created by the scraper as a general identifier for each process. Then we extracted the person’s name by catching the characters between *processo de* (and its variants “do” and “da”). Afterwards we extracted the dates between the slash and save them (as a date format using the python method *strftime*) as process start date and end date respectively. We then extracted from the parenthesis the original inquisition process id. In this id is indicated in which court location the trial occurred. *Tso-il* for Lisbon, *ts-o-ie* for Coimbra, *ts-o-ie* for Évora and *ts-o-ip* for Porto. This information is extracted and explicitly saved.

**Name:**

In the original data a person name is given in a full string, and it was necessary to split it so we could extract the family name (and be able to search for it). To accomplish the division, we separated the given name string by space and then verified its length. If the length is equal to one the person only has a given name in the archives. If it is equal to two, then it has a given name and a family name (they are saved separately). If it is equal to three or more than a person has a given name, a middle name and a family name. They are all properly saved separately.

**Age:**

The age while simple at first glance is more complex because sometimes it has a range number of ages (presumably because the person writing did not know the exact age) so a person could have his age set as a simple numeric number like “23” or a mixture of number like “23 or 24” or even a range of number like “23-26”. In these particular cases I decided to keep only the largest number and disregard the rest. So, in the example 23-26 the number stored as the person age would be 26.

**Father, Mother, Partner:**

Father, mother and partner are all presented as a string and saved as such. Later they will need to be processed and separated into given, middle and last name (some of the persons identified as father, mother and partner may also have their own separated processes who need to be identified).

---

4 English Translation (by the author): 2299869  Process of Manuel Gonçalves Laranjo, 1759-10-10/1761-08-13 (PT/TT/TSO-IL/028/00003)

5 English Translation (by the author): 2299869  Process Of

6 In Portuguese the preposition changes based on the gender
Social Status:

Social status in the context of the Portuguese holy inquisition refers in most of the cases to the level of faith of the person and their ancestor. It makes a partition of Christians in old Christians and new Christians (and everything in between like half new christian, etc.), it also partitions people in other religions like Jews and Muslims. It has however a variety of exceptions from slaves to different ethnicity (sometimes using unacceptable slang words for today standards that I opted to change). It has also typos in writing like $1/2$ instead of $1/2$ or cristoo instead of cristão and difference expressions depending on the gender of the person. In other words, the data needed to be properly cleaned and processed.

We approached this problem by firstly programmatic printing to a file all the unique values found and then sorted them out. Then we programmed a function that would receive a string and compare with a long list of conditions to go through possible options.

For example, in the case of a person whose social status indicated “mameluco” (a typical warrior slave in Muslim countries (Encyclopædia Britannica)) there are three different forms of writing this in the archives: “mameluco”, “mamaluco” and “mameluca”. If the function finds any of these forms in a given process it will save the social status to slave. Being aware that the decisions being made may not being completely correct historically wise and not intending to lose any information in the process we opted to save the original social status name in an extra field called social_status_literal.

We also made the decision to simplify fractions. For example, if a social status is $1/2$ new Christian or $2/4$ new Christian, we assume that in practice it all means the same: a half new Christian.

Address and Birthplace:

In both fields birthplace and address we are given, if fortunate, a full address or a named locality as a village or city name. While this is useful and worth preserving it can also be remarkably deficient. Locations change name, place and some are even abandoned. Google maps however is most of the time able to understand where the address is even if it is ancient and no longer is in use. We took advantage of this and using google maps API(Google maps platform) we queried each address and Birthplace extracting the GPS coordinates. I was also able to extract extra information like the country and administrative area.

For example, in the process pt/pt/tso-il/028/00011 the person whose name is “Felicitas Uchoa” has as her birthplace “freguesia de S. Gonçalo de Tapecite “. Using google maps API I was able to extract that this is located in São Gonçalo that is part of Amarante, located in the Porto area and part of the country Portugal. Its GPS coordinates where also successfully extracted.
Google maps API is a professional product without, at the writing of this report, academic licensing. With the amount of addresses needed to be queried (around 80,000) we had to be careful and selective as one run would spend all their allowed bandwidth before a high price had to be paid. It is also fallible and some address it will get wrong (or in fact return no result), while the exact number due to the excessive number of data is hard to quantify, we predict it is low percentage wise.

**Civil State:**

While in theory civil state should range between married, single, divorced and widow in reality like before the archives are full of noise in the data. Sometimes the name of the person was written as the civil state, others had two or more options as for example “casado ou solteiro”\(^7\). Others even had a whole range of information that is hard to process as for example: “casado pela segunda vez. A primeira mulher foi Maria Antunes”\(^8\). It was necessary to filter this and correct the information (which meant losing the extra information).

**Accusations:**

Accusation is in the archive written as a loose phrase, sometimes with only one accusation, sometimes with several. For example: “2305406 Crime/Acusação: feitiçaria, sodomia, impedir o recto ministério do Santo Ofício.”\(^9\) To save this as a string in a database is not appealing for searching and categorization. It became apparent that it was necessary to split the accusation and identify the maximum of different accusations possible. It the example above we would want to identify witchery that can be characterized as a *religious crime*, sodomy a *costumes crime*, and “impedir o recto” also a *religious crime*. To accomplish this we begun by programmatically printing to a text file in an alphabetically order a list of all unique accusations, then in a painstaking process we manually selected the ones that made sense. This catalogue was then loaded to a list in python (*Python list*) , then utilizing a function the original string is divided and for each element checks if it is present in the list. When successfully it saves the accusation. While some information can be lost (as before the original

---

\(^7\) English Translation (by the author): Single or Married

\(^8\) English Translation (by the author): Married for a second time, first wife was Maria Antunes

\(^9\) English Translation (by the author): “2305406: Crime/Accusation witchery, sodomy, obstacles to the normal operation of the holy church”
string is saved in a field called `acusation_litera` (it works satisfactorily in most cases. In figure 8 is shown how to example was processed and saved by the algorithm.

```
"2305406": {
    "process id": "pt/tt-tso-il/028/05385",
    "loc_court": "Lisboa",
    "process start date": "21/05/1707",
    "process end date": "22/11/1707",
    "social_status_litera": "crisânto-velho",
    "social_status": "crixânto-velho",
    "idade": ["75"],
    "acusation_litera": "feitiçaria, sodomia, impedir o recto ministério do santo ofício",
    "acusation": [
        "feitiçaria",
        "sodomia",
        "impedir o recto"
    ]
},
```

Figure 8: Accusation in the processed JSON file

**Sentences:**

Sentences in the original archives are very similar to accusation by having descriptions of the sentence. Contrary to Accusations though they can be very long and convoluted making it harder to divide in blocks and characteristics that make sense. A typical sentence is for example: “2305405 Sentença: auto-da-fé de 06/11/1707. Ir ao auto-da-fé com vela acesa na mão, abjuração de veemente, cárcere a arbitrio dos inquisidores, instrução na fé católica, penitências espirituais, pagamento de custas”

As it can be seen it has a rich variety of text that due to its quantity is hard to sort out to smaller meaningful chunks. As consequence of the amount of data when broken down we firstly, attempted to use data mining techniques to statistically determine the most common used words in sentences (abjuration, instruction, penitence in that order) and then use them in the list. Sadly, the results were rather weak and to much information was being lost for our goals. We had no choice but to edit the file manually, a process that took a long time and was not very gratifying. The algorithm itself is very similar to the one described in accusations, the data in the edited file is loaded to a python list and then a function splits the string from the archive and searches the list for a match. If found it saves it.

In the end the results were adequate and as it can be seen in figure 9 the loss of information is not very drastic. Like before the original data is also maintained in a field called `sentence_litera` so improvements can be made later.

```
"Sentence_litera": "auto-da-fé de 06/11/1707. Ir ao auto-da-fé com vela acesa na mão, abjuração de veemente, cárcere a arbitrio dos inquisidores, instrução na fé católica, penitências espirituais, pagamento de custas",
```

Figure 9: Sentences in the processed JSON file

One of the problems of this approach is that sentence has a large quantity of dates. Some are related to specific events (in this case is the “acto da fé”) some are just the date as is complicated to understand

---

10 English Translation (by the author): “2305405 Conviction: Act of faith in 06/11/1707. Go to the act of faith with a lit candle in his hand, Abjuration, imprisonment at the free will of the inquisitors, instruction in the catholic faith, spiritual penitence, payment of cost.

11 English Translation (by the author): Act of faith
what they actually represent (most likely the date the sentence was given). This context is lost and is something that can be worked on in the future.

**Prison and sentence date:**

Sentence and prison dates are in most part obviously dates than only needed to be properly formatted. Sometimes they have extra information or unknown information (for example [?]/[?]/[1542?]). When there is extra information, we disregard it. Where there is unknown information, we replace it by one. So in the example [?][?][1542?] the final result would be 1/1/1542. Regular expressions are used to catch the dates.

**Profession:**

The field “professions” is as the name indicate the profession activity that the person in the process was practicing or doing at the time of the trial. It ranges from familiar professions like soldier or doctor to defunct professions that no longer exist. It includes questionable activities like slavery and even some professions that is not trivial to determine their meaning and likely needs an expert in that time period to decipher them. The challenge in extracting and transforming the data was for one the amount of data, the genderization of the profession name (a female measurer was called a “medideira” while a male one a “medidor”) different spelling of the same activity by different people in different times (for example a pilgrim could be called a “peregrino” or a “pergaminheiro”) and frequent typos in the writing.

We approached these problems in a similar way of the social status, firstly we programmatically printed to a file an ordered list of all unique professions. Then we built a function to process them. We solved the genderization problem by using simple regular expressions, for example for the profession pescador(fisherman) and its variations any word that started with “pescador” would be a fisherman, this would “caught” pescador and pescadora. To solve the other problems mentioned after some trial and errors we again had little choice but to do it by hand problematically coding a series of conditions than spanned for more than 800 lines of code. It works in the most basic way; it receives a string and checks by code if its matched. For example, it would mark a person profession as “entulhador” if the word received is “entalhador” or “entrelhador” or “entulhador”.

While a crude and time expensive method it did work satisfactorily, and the professions were successfully treated and processed. Later we will explain how we categorized the professions allowing us to search the processes where for example a person was in a military profession regardless of its given specificity.

**Conclusion:**

In this chapter we discussed the state of the original data and the steps necessary to extract and transform it in a format that is adequate for the semantic web. While a burdensome, time-consuming task and not particularly gratifying or intellectually challenging it was vitally necessary to be able to have success in reaching the goals of this work. As previously discussed, an application is only as intelligent as the quality of its data, likewise the semantic web is only able to support that intelligence if is data is well formatted and organized.
4.3 - The Ontology

Introduction:

In this chapter we go through the construction of the ontology itself. We will go through each class, property and restriction of the ontology and explain to the best of our abilities the choices made. In figure 10 we can observe an overall view of the classes in the ontology.

Figure 10: Overall class view ontology
Person:

This class is the abstract entity that represent all the people that are referenced in the inquisition archives may they have a process or are referenced as a father, mother or partner of someone that has a process. We make the connection from the object of a person to the class by stating that the object is part of the set named person (using subClassOf). This is the most natural mode of association in OWL and the one that makes more sense in this situation. To model this class, it was necessary to create object properties to store the connections between the instances and other objects, they are described in table 2. It was necessary to create data objects to store the literal information of the instances belonging to the class, they are described in table 4. We also created some restrictions to better frame the instances of the class, these restrictions are detailed in table 5.

Object properties:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>has Social Status</td>
<td>This property represents the social status of a person. It connects an individual of the class person to an individual of the class SocialStatus. It has as the domain the class Person and the range is the class SocialStatus</td>
</tr>
<tr>
<td>has Address</td>
<td>This property represents a person current address. It connects an individual of the class person to an object of the class locality. It has as the domain the class Person and the range is the class locality</td>
</tr>
<tr>
<td>has Birthplace</td>
<td>This property represents a person birthplace. It connects an individual of the class person to an individual of the class locality. It has as the domain the class Person and the range is the class locality</td>
</tr>
<tr>
<td>lived InHistoricalPeriod</td>
<td>This property represents a historical period that can be a king reign, a century or other historical period. It connects an individual of the class person to an individual of the class HistoricalPeriod. It has the domain in an object of the class Person and the range is an object of the class HistoricalPeriod. It has as the domain the class Person and the range it is the class HistoricalPeriod</td>
</tr>
<tr>
<td>Property Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>has_Partner</td>
<td>This property represents the wife or husband of an individual person. It connects an individual of the class person to an individual of the class person. It has as the domain the class Person and the range it is the class person</td>
</tr>
<tr>
<td>has_Mother</td>
<td>This property represents the mother of an individual person. It connects an individual of the class person to an individual of the class person. It has as the domain the class Person and the range is the class person</td>
</tr>
<tr>
<td>has_Father</td>
<td>This property represents the father of an individual person. It connects an individual of the class person to an individual of the class person. It has as the domain the class Person and the range is the class person</td>
</tr>
<tr>
<td>has_Profession</td>
<td>This property represents the profession or activity of a person. It connects an individual of the class person to an individual of the class profession. It has as the domain the class Person and the range is the class profession.</td>
</tr>
<tr>
<td>has_Inquisition_Process</td>
<td>This property represents the inquisition process of a given person (if it has one). It connects an individual of the class person to an individual of the class InquisitionProcess. It has as the domain the class Person and the range is the class InquisitionProcess</td>
</tr>
</tbody>
</table>

Table 2: Person class object properties

In table 3 we describe the object properties that are inferred by the inference engine:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>has_Brother</td>
<td>This property represents the brother of a person. It connects an individual of the class person to an individual of the class person. It has as the domain the class Person and the range is the class person</td>
</tr>
<tr>
<td>has_Sister</td>
<td>This property represents the sister of a person. It connects an individual of the class person to an individual of the class person. It has as the domain the class Person and the range is the class person</td>
</tr>
<tr>
<td>has_GrandFather</td>
<td>This property represents the grandfather of a person. It connects an individual of the class person to an individual of the class person. It has as the domain the class Person and the range is the class person</td>
</tr>
</tbody>
</table>
individual of the class person to an individual of the class person. It has as the domain the class Person and the range is the class person

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>has_GrandMother</td>
<td>This property represents the Grand Mother of a person. It connects an individual of the class person to an individual of the class person. It has as the domain the class Person and the range is the class person</td>
</tr>
<tr>
<td>has_Uncle</td>
<td>This property represents the uncle of a person. It connects an individual of the class person to an individual of the class person. It has as the domain the class Person and the range is the class person</td>
</tr>
<tr>
<td>Has_aunt</td>
<td>This property represents the Aunt of a person. It connects an individual of the class person to an individual of the class person. It has as the domain the class Person and the range is the class person</td>
</tr>
</tbody>
</table>

Table 3: Person class inferred object properties

Data properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>first_Name</td>
<td>This property represents the first name of an individual of the class person. It has as the domain the class person and the range must be a string</td>
</tr>
<tr>
<td>middle_Name</td>
<td>This property represents the middle name of a individual of the class person. It has as the domain the class person and the range must be a string</td>
</tr>
<tr>
<td>last_Name</td>
<td>This property represents the last name of an individual of the class person. It has as the domain the class person and the range must be a string</td>
</tr>
<tr>
<td>civilState_Literal</td>
<td>This property represents the original data of the Civil state of a person so information is not lost. It has as the domain the class person and the range must be a string</td>
</tr>
<tr>
<td>person_ID</td>
<td>This property represents the identification of a person. As it its used for other classes it has no domain, the range must be an integer.</td>
</tr>
<tr>
<td>birthplace_Literal</td>
<td>This property represents the original data of the Civil state of a person. It has as the domain the class person and the range must be a string</td>
</tr>
<tr>
<td>address_Literal</td>
<td>This property represents the original data of the address of a person, so information is not lost. It has as the domain the class person and the range must be a string</td>
</tr>
<tr>
<td>accusation_Literal</td>
<td>This property represents the original data of the accusation of a process, so information is not lost. It has as the domain the class person and the range must be a string</td>
</tr>
<tr>
<td>sentence_Literal</td>
<td>This property represents the original data of the sentence of a process, so information is not lost. It has as the domain the class person and the range must be a string</td>
</tr>
<tr>
<td>address_GPS</td>
<td>This property represents the GPS of the address. It has as the domain the class person and the range must be a string</td>
</tr>
<tr>
<td>social_Status_Literal</td>
<td>This property represents the original data of the accusation of a process, so information is not lost. It has as the domain the class person and the range must be a string</td>
</tr>
</tbody>
</table>
age | This property represents the age of a person. It has as the domain the class person and the range must be an integer

Profession_Literal: | This property represents the original data of the profession of a person, so information is not lost. It has as the domain the class person and the range must be a string

<table>
<thead>
<tr>
<th>Name</th>
<th>Cardinality</th>
<th>Range</th>
<th>Domain</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person_ID</td>
<td>1</td>
<td>Integer</td>
<td>Person</td>
<td>Must have an ID given programmatically</td>
</tr>
<tr>
<td>age</td>
<td>0</td>
<td>Integer</td>
<td>Person</td>
<td>Not all processes in archives have an age</td>
</tr>
<tr>
<td>literal Social Status</td>
<td>0</td>
<td>String</td>
<td>Person</td>
<td>Original social status data</td>
</tr>
<tr>
<td>middle Name</td>
<td>0</td>
<td>String</td>
<td>Person</td>
<td>Not all persons in process have middle name</td>
</tr>
<tr>
<td>last Name</td>
<td>0</td>
<td>String</td>
<td>Person</td>
<td>Not all persons in process have last name</td>
</tr>
<tr>
<td>first Name</td>
<td>1</td>
<td>String</td>
<td>Person</td>
<td>A person needs to have at least a first name</td>
</tr>
<tr>
<td>literal Address</td>
<td>0</td>
<td>String</td>
<td>Person</td>
<td>Original Address data</td>
</tr>
<tr>
<td>literal Accusation</td>
<td>0</td>
<td>String</td>
<td>Person</td>
<td>Original Accusation data</td>
</tr>
</tbody>
</table>

Data Restrictions:

Table 4: Person class data properties

Inquisition Process:

The class InquisitionProcess is an abstraction of the inquisition process itself, it should contain all relevant information of the process that do not involve personal information about the person. We choose to split this from the person data because there will be people in the ontology that do not have a process (most fathers, mothers and partners). It is connected to the person by object property Has_Inquisition_Process. It was chosen to characterize it this way and not by using subClassof because a process makes no sense without a connection to an individual person. To model this class, it was necessary to create object properties to store the connections between the instances and other objects, they are described in table 6. It was necessary to create data objects to store the literal information of the instances belonging to the class, they are described in table 7. We also created some restrictions to better frame the instances of the class, these restrictions are detailed in table 8. It table 9 the object restrictions are detailed.
Object properties:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>has_Accusation</code></td>
<td>This property represents one of the accusations of a process (1-N). It connects an individual of the class <code>InquisitionProcess</code> to an individual of the class <code>Accusation</code> (or one of its sub-classes). It has as the domain the class <code>InquisitionProcess</code> and its range its the class <code>Accusation</code>.</td>
</tr>
<tr>
<td><code>has_Conviction</code></td>
<td>This property represents one of the Convictions of a process (1-N). It connects an individual of the class <code>InquisitionProcess</code> to an individual of the class <code>Conviction</code> (or one of its sub-classes). It has as the domain the class <code>InquisitionProcess</code> and its range its the class <code>Conviction</code>.</td>
</tr>
</tbody>
</table>

Table 6: Inquisition class object properties

Data properties:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inquisitionP_ID</code></td>
<td>This property represents the ID of an individual of the class <code>InquisitionProcess</code>. It has as the domain the class <code>InquisitionProcess</code> and the range must be a integer.</td>
</tr>
<tr>
<td><code>process_Number</code></td>
<td>This property represents the process number of the original process in the archives and its attributed to a specific individual of the class <code>InquisitionProcess</code>. It has as the domain the class <code>InquisitionProcess</code> and the range must be of the type string.</td>
</tr>
<tr>
<td><code>start_Date</code></td>
<td>This property represents the start date of the original process in the archives and its attributed to a specific individual of the class <code>InquisitionProcess</code>. It has as the domain the class <code>InquisitionProcess</code> and the range must be of the type <code>DateTime</code>.</td>
</tr>
<tr>
<td><code>end_Date</code></td>
<td>This property represents the end date of the original process in the archives and its attributed to a specific individual of the class <code>InquisitionProcess</code>. It has as the domain the class <code>InquisitionProcess</code> and the range must be of the type <code>DateTime</code>.</td>
</tr>
<tr>
<td><code>prison_Date</code></td>
<td>This property represents the prison date of the original process in the archives and its attributed to a specific individual of the class <code>InquisitionProcess</code>. It has as the domain the class <code>InquisitionProcess</code> and the range must be of the type <code>DateTime</code>.</td>
</tr>
</tbody>
</table>

Table 7: Inquisition class data properties

Data Restrictions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Cardinality</th>
<th>Range</th>
<th>Domain</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>End Date</td>
<td>0</td>
<td>DateTime</td>
<td>-</td>
<td>Process End date.</td>
</tr>
<tr>
<td>Prison Date</td>
<td>0</td>
<td>DateTime</td>
<td>Inquisition Process</td>
<td>Process Prison Date.</td>
</tr>
<tr>
<td>Process Number</td>
<td>1</td>
<td>String</td>
<td>Inquisition Process</td>
<td>Original process ID, obligatory</td>
</tr>
<tr>
<td>Start Date</td>
<td>0</td>
<td>DateTime</td>
<td>-</td>
<td>Process Start date.</td>
</tr>
<tr>
<td>End Date</td>
<td>0</td>
<td>DateTime</td>
<td>-</td>
<td>Process End date.</td>
</tr>
</tbody>
</table>

Table 8: Inquisition class data restrictions
Object Restrictions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Cardinality</th>
<th>Range</th>
<th>Domain</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inquisition Process Of</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>An inquisition process is a process of some person. Inverse of Has Inquisition Process.</td>
</tr>
</tbody>
</table>

Table 9: Inquisition Process class object restrictions

Accusation:

Accusation class is an abstraction of all individual accusations contained in the Inquisition ontology. Like in most cases in OWL we could connect an individual to the class by the subClassOf OWL property or by a created object property like for example accusation X - OfType - Accusation. Both options are interchangeable and, in this context, only affect future SPARQL queries. In the ontology we have chosen to use subClassOf as it made more sense semantically.

In this class we have created four sub-classes to represent different kinds of accusations, they can be seen in the table 10.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blood_crime_accusation</td>
<td>This sub-class is an abstraction for individual accusations of crimes that today we would also criminalize in the courts as serious crime like murder or rape.</td>
</tr>
<tr>
<td>Costume_crime_accusation</td>
<td>This sub-class is an abstraction for individual accusations of crimes that would offend the social costumes of its time like for example homosexuality or polygamy.</td>
</tr>
<tr>
<td>Religious_crime_accusation</td>
<td>This sub-class is an abstraction for individual accusations of crimes that for the inquisition violated the religiosity of the time. The most common would-be Judaism.</td>
</tr>
<tr>
<td>Other_crime_accusation</td>
<td>This sub-class is an abstraction for individual accusations of crimes that do not fit any of the other criteria as for example someone that escaped and did not serve the original sentence</td>
</tr>
</tbody>
</table>

Table 9: Subclasses of accusation

The model this class it was necessary to create data properties to store the literals of the instances that are part of it, they are detailed in table 11.

Data properties:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accusation_ID</td>
<td>This property represents a unique identification of an accusation. As it its used for other classes it has no domain, the range must be a integer.</td>
</tr>
<tr>
<td>name</td>
<td>This property represents the name of the accusation (for example “Judaism”). As it is used for other classes it has no domain, the range must be a String.</td>
</tr>
</tbody>
</table>

Table 11: Accusation class data properties
This class is **disjoint** with the class person which means that the inference engine will not infer that a accusation is a person and vice-versa.

**Conviction:**

Conviction class is an abstraction of all individual convictions contained in the inquisition ontology. Like in most cases in OWL we could connect an individual to the class by the `subClassOf` OWL property or by a created object property like for example `sentence X - OfType - Conviction` Both options are interchangeable and, in this context, only affect future SPARQL queries. In the ontology we have chooses to use `subClassOf` as it made more sense semantically.

In this class we have created nine sub-classes to represent different kinds of accusations. They are detailed in the table 12.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acquit Sentence</td>
<td>When the person accused was found non-guilty.</td>
</tr>
<tr>
<td>arrest Sentence</td>
<td>When the person accused was found guilty and arrested.</td>
</tr>
<tr>
<td>banishment Sentence</td>
<td>When the person accused was found guilty and banned from the kingdom of Portugal.</td>
</tr>
<tr>
<td>corporal Punishment Sentence</td>
<td>When the person accused was found guilty and the sentence includes some form of corporal aggression like for example whipping.</td>
</tr>
<tr>
<td>date Sentence</td>
<td>When the conviction includes a date (most of the times only a date appears with no other context).</td>
</tr>
<tr>
<td>excommunication Sentence</td>
<td>When the person accused was found guilty and the sentence includes some form of ecclesiastical censure.</td>
</tr>
<tr>
<td>Financial Sentence</td>
<td>When the person accused was found guilty and the sentence includes financial consequences like a fine or seizure of assets.</td>
</tr>
<tr>
<td>light Sentence</td>
<td>When the person accused was found guilty but the sentence is not very serious this could include for example a mandatory attendance of church services or serve in a convent.</td>
</tr>
<tr>
<td>retraction Sentence</td>
<td>When the person accused was found guilty and the sentence includes some kind of abjuration.</td>
</tr>
</tbody>
</table>

Table 12: Subclasses of conviction

The model this class it was necessary to create data properties to store the literals of the instances that are part of it, they are detailed in table 13.

**Data properties:**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conviction_ID</td>
<td>This property represents a unique identification of a sentence. As it is used for other classes it has no domain, the range must be an integer.</td>
</tr>
<tr>
<td>name</td>
<td>This property represents the name of the sentence (for example “banished from the kingdom”). As it is used for other classes it has no domain, the range must be a string.</td>
</tr>
</tbody>
</table>

Table 13: Conviction class data properties
This class is **disjoint** with the class person which means that the inference engine will not infer that an accusation is a person and vice-versa.

**Civil State:**

Civil state class is an abstraction of all possible civil states that a person could have at the time.

- Divorced
- Married
- Single
- Widow

There are several possible approaches to represent this in an ontology. We could create the class *CivilState* that has as individuals the four possibilities or they can be sub-classes of the *civilState* class. It makes little difference in practical terms as we do not need the inference engine to work this out. I have chosen the sub-classes route as it made more sense semantically to use the *subClassOf* OWL property which would not be possible if they were individuals. As before we could connect a individual person to a specific civil state by a created object property (e.g *Has_CivilState*) or by the OWL *subClassOf* property. They are interchangeable and the later was chosen as it made more sense.

**Court:**

Court class is an abstraction of the different individual courts that judged inquisition processes. There were five different courts in the Portuguese Empire (Marcocci & Paiva, 2016): one located in Lisbon, other in Porto, other in Coimbra, other in Évora and one in Goa. As explained in the ETL sub-chapter this can be identified in the process number of each different inquisition file. One of the strengths of the Semantic Web is to add a web of information to the data, adding the information of where the process was trialed enriches the information.

As previously discussed, we could approach this class in two different ways. We could make the different courts as individuals of the class *Court*, or we can make them sub-classes of the class *court*. We wanted to give this court extra information like its formal name and its location (as also give room to any future extra information) so it made more sense to make them individuals of the class *Court*.

To model this class, it was necessary to create an object property to connect an instance of the class to another object, it is detailed in table 14. It was also required to create data property to store the literals of the instance, they are detailed in table 15.

**Object properties:**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>has_location</td>
<td>This property represents the location of something or someone. It connects an individual of the different classes to an individual of the class <em>location</em>. It has no specific domain as it can be used by multiple classes and its range it is the class <em>location</em>.</td>
</tr>
</tbody>
</table>

Table 14: Court class object properties
Data properties:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>This property represents the formal name of the court (e.g. “Santo Officio de Lisboa”). As it is used for other classes, it has no domain, the range must be a String.</td>
</tr>
</tbody>
</table>

Table 15: Court class data properties

This class is disjoint with the class Inquisitionprocess which means that the inference engine will not infer that a court is an inquisition process and vice-versa.

Gender:

Gender class is an abstraction of the two different genders a person could have at the time (this is not indicated in the archives and in the results an explanation will be given in how it was determined). We have been discussing the different ways to represent this in the ontology, by having different individuals of a class or sub-class of a class. While we have chosen the sub-class approach and the subclassOf property we also have explained that they are interchangeable and as long as we are aware of how its constructed it makes little difference. To explore and show this interchangeability the gender construction takes the individual route, so male and female are individuals of the class gender connected to it by the subclassOf property. A person is then connected to each individual by the object property has_gender detailed in the person class.

This class is disjoint with the class person which means that the inference engine will not infer that a gender is a person and vice-versa.

Historical Period:

One of the major strengths of the semantic web is the ability to add information on top of information. One additional information we wanted to enrich the ontology with was historical periods. Since the Portuguese Inquisition activity spanned for several centuries, it was part of different kings and time periods. By adding historical context we could for example search and understand what kind of processes were being made during enlightenment or during the reign of king D. João V.

To accomplished this, we created the class HistoricalPeriod. The first version of the ontology had the approach of any given individual was connected to the class by the SubClassOf property and had an object property Is_Type that could be a Reign, Century or Other. While this method works seamlessly it made the SPARQL queries more complex without any noticeable advantage. This motivated a backtrack and in the next revisions of the ontology we made the three categories sub-classes of the class HistoricalPeriod. To successfully model this class, it was necessary to create data properties to store the literal information of the instances belonging to the class, they are described in table 16. We also created some restrictions to better frame the instances of the class, these restrictions are detailed in table 17.

Therefore, the HistoricalPeriod class has three sub-classes:

- **Reign**: The various kingdom reigns
- **Century**: The various century's
**OtherPeriod:** Other periods as for example the first French invasions

**Data properties:**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>historical_ID</td>
<td>This property represents a unique identification of an Historical Period. As it its used for other classes it has no domain, the range must be an integer.</td>
</tr>
<tr>
<td>name</td>
<td>This property represents the name of the sentence (for example “banished from the kingdom”). As it its used for other classes it has no domain, the range must be a string.</td>
</tr>
<tr>
<td>start_Date</td>
<td>This property represents the start date of an individual Historical Period. As it is used in various classes has no domain, the range must be of the type Datetime</td>
</tr>
<tr>
<td>end_Date</td>
<td>This property represents the end date of an individual Historical Period. As it is used in various classes has no domain, the range must be of the type Datetime</td>
</tr>
</tbody>
</table>

Table 16: Historical period class data properties

**Data Restrictions:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Cardinality</th>
<th>Range</th>
<th>Domain</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>start Date</td>
<td>1</td>
<td>DateTime</td>
<td>-</td>
<td>Historical Period start date.</td>
</tr>
<tr>
<td>end Date</td>
<td>1</td>
<td>DateTime</td>
<td>-</td>
<td>Historical Period End date.</td>
</tr>
<tr>
<td>name</td>
<td>1</td>
<td>String</td>
<td>-</td>
<td>Name of the historical period</td>
</tr>
</tbody>
</table>

Table 17: Historical Period class object restrictions

**Human Maturity:**

As we have the age of most persons in the process it may be interesting to classify if the person is a child, adult or elderly. To do this the class HumanMaturity was created. As in some previous cases it was opted to create the notion of child, adult our elderly as sub-class of the class HumanMaturity.

- **Child:** Anyone with age under 18.
- **Adult:** Anyone with age between 18 and 64.
- **Elderly:** Anyone whose age is above 64.

An individual person is connected to the HumanMaturity class by the OWL property subClassOf. As before this choice was made because is what we though made more semantically sense but the other options we previously discussed could again be used with no major consequences (and can be easily changed via a relatively simple SPARQL query).

This class is disjoint with the class person which means that the inference engine will not infer that a HumanMaturity is a person and vice-versa.

**Profession:**

The profession class is an abstraction of all individual different professions contained in the Inquisition ontology. Again, it was chosen to connect and individual profession to is class by the
We connect an individual person to an individual profession by the object property Has_Profession whose properties we discussed while presenting the person class.

To be able to categorize the several types of profession several sub-classes within the profession class were created as it can be seen in table 18.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>administrator</td>
<td>Any given profession that has administrative functions as for example a governor or a commissar.</td>
</tr>
<tr>
<td>Agriculture</td>
<td>Any given profession that can be categorised as agriculture like for example a farmer or a shepherd.</td>
</tr>
<tr>
<td>Artist:</td>
<td>Any given profession dealing with artistic pleasure or entertainment like a singer or comedian.</td>
</tr>
<tr>
<td>Arts</td>
<td>Any given profession that involves arts or letters as for example a writer or a professor.</td>
</tr>
<tr>
<td>Commerce</td>
<td>Any given profession that involves commerce of goods like for example a fish salesman or a shop owner.</td>
</tr>
<tr>
<td>Craftsman:</td>
<td>Any given profession that involves the crafting of goods or services as for example a shoemaker or a locksmith.</td>
</tr>
<tr>
<td>Military</td>
<td>Any given profession that involves any possible military fighting as for example a soldier or a guard.</td>
</tr>
<tr>
<td>Religious</td>
<td>Any given profession involving religious practices as for example a priest or a friar.</td>
</tr>
<tr>
<td>Services</td>
<td>Any given profession that has some kind of services like a hairdresser or a seer.</td>
</tr>
<tr>
<td>Other_profession:</td>
<td>Professions that was difficult to put into specific categories as for example a slave or a beggar.</td>
</tr>
</tbody>
</table>

Table 18: Subclasses of professions

It was necessary to create and ID data property to store the unique identification of each category, this is detailed in table 19.

Data Properties:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profession_ID</td>
<td>This property represents a unique identification of a profession. As it is used for other classes it has no domain, the range must be an integer.</td>
</tr>
<tr>
<td>name</td>
<td>This property represents the name of the profession (for example “Procurador”). As it is used for other classes it has no domain, the range must be a String.</td>
</tr>
</tbody>
</table>

Table 19: Profession class data properties

This class is disjoint with the class person which means that the inference engine will not infer that a Profession is a person and vice-versa.
Location:

The Location class is an abstraction of all individual object locations present in the ontology. As it was discussing in the ETL part of this chapter the google maps API was used to extract information about the addresses referenced in the Inquisition Archive. The class location was created to encircle all these locations and their information. We have sub-classes for the parishes, municipality, district and countries and the connections between them. We also have sub-classes for several regions. As in some previous classes we connect a individual to its class by the OWL property subClassOf as it makes semantically sense. We connect each locality by object properties, for example a municipally Has_district certain district which by its turn has_country certain country. We connect a person to a location by the object property has_address and has_birthplace.

The class location has several sub-classes to encapsulate different kinds of locations, they are detailed in table 20.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Border_raia</strong></td>
<td>Represents the locations that were part of the defunct raia region near Spain.</td>
</tr>
<tr>
<td><strong>Country</strong></td>
<td>An individual location that represents a country e.g: Portugal</td>
</tr>
<tr>
<td><strong>District</strong></td>
<td>An individual location that represents a district e.g: Gujarat</td>
</tr>
<tr>
<td><strong>Municipality</strong></td>
<td>An individual location that represents a municipality e.g: Lisboa</td>
</tr>
<tr>
<td><strong>Parish</strong></td>
<td>An individual location that represents a parish</td>
</tr>
<tr>
<td><strong>Province_Algarve</strong></td>
<td>A location that is part of the old province of Algarve</td>
</tr>
<tr>
<td><strong>Province_altoalentejo</strong></td>
<td>A location that is part of the old province of Alto Alentejo</td>
</tr>
<tr>
<td><strong>Province_BaixoAlentejo</strong></td>
<td>A location that is part of the old province of Baixo Alentejo</td>
</tr>
<tr>
<td><strong>Province_BeiraAlta</strong></td>
<td>A location that is part of the old province of Beira alta</td>
</tr>
<tr>
<td><strong>Province_BeiraBaixa</strong></td>
<td>A location that is part of the old province of Beira Baixa</td>
</tr>
<tr>
<td><strong>Province_Estremadura</strong></td>
<td>A location that is part of the old province of Estremadura</td>
</tr>
<tr>
<td><strong>Province_Litoral</strong></td>
<td>A location that is part of the old province of Litoral</td>
</tr>
<tr>
<td><strong>Province_minho</strong></td>
<td>A location that is part of the old province of Minho</td>
</tr>
<tr>
<td><strong>Province_Ribatejo</strong></td>
<td>A location that is part of the old province of Ribatejo</td>
</tr>
<tr>
<td><strong>Province_Trasdosmontes</strong></td>
<td>A location that is part of the old province of Trás dos montes</td>
</tr>
</tbody>
</table>

Table 20: Subclasses of locations

To model this class, it was necessary to create object properties to store the connections between the instances and other objects, they are described in table 21. It was necessary to create data objects to store the literal information of the instances belonging to the class, they are described in table 22.

Object properties:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>has_Country</td>
<td>This property indicates the country of certain location (that is not itself a country). It connects a Parish, municipality and district to a certain country. It is a transitive property.</td>
</tr>
<tr>
<td>is_Country</td>
<td>Inverse of has_Country property.</td>
</tr>
<tr>
<td>has_District</td>
<td>This property indicates the district of certain location (that is not itself district or a country). It connects a Parish and/or municipality to a certain district. It is a transitive property.</td>
</tr>
<tr>
<td>is_District</td>
<td>Inverse of has_District property.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>has_Municipality</td>
<td>This property indicates the municipality of a certain location. It connects a Parish to a certain municipality. It is a transitive property.</td>
</tr>
<tr>
<td>is_Municipality</td>
<td>Inverse of has_Municipality property.</td>
</tr>
<tr>
<td>has_Parish</td>
<td>This property indicates the parish of a certain location.</td>
</tr>
<tr>
<td>is_Parish</td>
<td>Inverse of has_Parish property.</td>
</tr>
</tbody>
</table>

Table 21: Location class object properties

Data properties:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location_ID</td>
<td>This property represents a unique identification of a location. As it is used for other classes it has no domain, the range must be an integer.</td>
</tr>
<tr>
<td>name</td>
<td>This property represents the name of the location (for example “Covilhã”). As it is used for other classes it has no domain, the range must be a String.</td>
</tr>
</tbody>
</table>

Table 22: Location class data properties

Social Status:

The SocialStatus class is an abstraction of all individual different social status present in the inquisition archives. One of the challenges of this class is the sheer amount of different social status that are related to each other, for example a new Christian can be fully new Christian or 1/2, 1/4, 1/8, 1/16 and so on. If someone is half new Christian, then is reasonable to assume that he is probably also half one Christian which further complicates matter. One approach could be as in other classes to create a subclass for each category in figure 11 we can see an example of this approach applied to New Christians.

Figure 11: Social status scheme

However, by opting for this approach we risked being caught in a known trap of modelling ontologies called Rampant Classism (Allemang et al., 7 2012) which in short is simply to define everything as a class which makes querying and retrieving knowledge from the data more difficult and unpredictable. To avoid this issue and solve the problem we opted for a middle ground solution with some subclasses and some individuals. So for the example of the new Christians, we create the class New_Christian that has as subclasses full_new_christian and partial_new_christian. Then for each partial new christian we create a separate individual that is connected to the subclass partial_new_christian by the SubClassOf property. We do this approach to all other possible social status (like for example different ethnicities) and this way we are able to successfully represent the knowledge. We then connect a person to a specific social status by the object property has_social_status to the individual social status and by the SubClassOf to the class. So for example if Vivianne is a 1/16 New Christian we say Vivianne is a SubClassOf of partial_new_christian (and the same for the old Christian part).
The class SocialStatus has the following subclasses:

- **New_Christian**: All new Christians
  - **Full_new_christian**: Those who are specified as fully new Christians
  - **Partial_New_christian**: Those that are specified about by at least partial new Christians
- **Old_Christian**: All old Christians
  - **Full_old_christian**: Those who are specified as fully old Christians
  - **Partial_old_christian**: Those that are specified about by at least partial old Christians
- **Different_ethnicity**: Those who social status is being a different ethnicity e.g Gypsy’s
- **Jewish**: Those that are specified as being Jewish.
- **Other**: Individual Social status that do not fit in a category e.g “prisoner”
- **Other_Religion**: less common religion e.g Roman Catholic
- **Slave**: Social Status that are some kinds of slavery.

To model this class, it was necessary the creation of some data properties to store the literal information of the instances, they are detailed below in table 23.

**Data properties:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ss_ID</td>
<td>This property represents a unique identification of an individual social status. As it is used for other classes it has no domain, the range must be an integer.</td>
</tr>
<tr>
<td>name</td>
<td>This property represents the name of the individual social status (for example “1/2 new Christian”). As it is used for other classes it has no domain, the range must be a String.</td>
</tr>
</tbody>
</table>

Table 23: Social Status class data properties

### 4.4 -Conclusion

In this chapter we went through the necessary steps to transform the inquisition digitalized archives in data we could use for the semantic web. We then introduced in more detail what an ontology is and their properties. We went step by step the construction of the structure of a suggested ontology for the Portuguese inquisition. In the next chapter we will go through the loading of the data into the Ontology and the construction of SWRL inference rules that could give us extra information. We will then check the results to verify if we were successful in our efforts.
5 - Inquisition data in the model

5.1 - Insertion of the data

Succeeding the construction of the ontology it was now necessary to load the curated data into it in an organized and coherent manner. It was imperative that all triples and connections where properly made so the final ontology represents what was originally intended. We also wanted to find if any of the family of the people that had processes and connect them which required some form of automated searching and educated guessing.

This was done through programming in python using the package owlready2 (Lamy, 2017). This library package can load OWL ontologies as Python objects, modify them, save them and perform reasoning via the reasoner Hermit or Pellet (Pellet). It is also able to search an ontology using various methods and list the properties of a given object which was invaluable in several steps of this work. In this chapter we will go through all the steps taken to load the curated data in the ontology. We will then show the state of the Ontology after that process.

One of the first issues that we come across in this task what was to load first into the ontology. Starting by the person would be pointless because there would be no data in the accusations, social status, sentences. We had to start by the classes that had less connection to the others. Social status, professions, accusations and sentences where the most obvious choices.

Professions:

In the fourth chapter when describing the profession class, we discussed the interest in classifying the profession in various categories. Before we begin loading the data into the ontology it was necessary to classify each profession, this was done by creating an extra table with the professions in one side and the respective category in the other. Afterwards it was just a matter of reading each row and based on the category create the profession with the is_a OWL property to the selected category (that itself is an already created subclass). For example, if we wanted to load the profession “cabo” into the ontology the code would check what was its category and then insert it in the ontology.

1. Creates a sequentially generated id for the profession (e.g profession_X where X is a number)
2. Insert that the profession is_a military
3. Insert that the profession has the data property ID with X as value
4. Insert that the profession has the data property Name with “cabo” as value
Accusations:

In order to load the accusations into the ontology we follow a similar strategy to professions (as we will use in sentences). We should note that exists several methods to accomplish this goal, the most obvious one would be to build into the python code itself using a dictionary or a list. This however would make less friendly to change the various categories chosen that although done to the best of our knowledge may be wrong or inaccurate. By adopting the CSV file method only changing the file is necessary to fix any mistake.

Following the previous strategy, we proceed in copying all the unique accusations to an extra table and in a second row we manually attributed to each one a category. Now, as before, we read each row and based on the category we create the corresponding accusation using the is_a OWL property to the selected category (that itself is an already created subclass). So, for example if we wanted to load the accusation “pacto com o diabo” into the ontology the code would check what was the corresponding category and then insert it in the ontology.

1. Create a sequentially generated id for the accusation (e.g accusation_X where X is an integer)
2. Insert that the accusation_X is_a religious_crime_accusation
3. Insert that the accusation_X has the data property ID with X as value
4. Insert that the accusation_X has the data property Name with “pacto com o diabo” as value

Sentences:

The same technique used in professions and accusations was used to load the Convictions into the ontology. A third table was created with all the unique sentences, in a separated column we categorized each one and then the file was read by a python function.

Again, for each unique sentence in the table we insert it in the corresponding category previously created as a sub-class in the ontology. To insert the “bens confiscados” sentence the algorithm would to the following:

1. Create a sequentially generated id for the sentence (e.g sentence_X where X is an integer)
2. Insert that the sentence_X is_a financial_sentence
3. Insert that the sentence_X has the data property ID with X as value
4. Insert that the sentence_X has the data property Name with “bens confiscados” as value

English Translation (by the author): Pact with the devil

English Translation (by the author): Confiscation of goods
After the work of curation and transformation of the data we ended up with twenty-eight different possible social status variations:

- “1/16 cristão-novo” (translation: 1/16 New-Christian)
- “1/2 cristão-novo” (translation: 1/2 New-Christian)
- “1/2 cristão-velho” (translation: 1/2 Old-Christian)
- “1/4 cristão-novo” (translation: 1/4 New-Christian)
- “1/8 cristão-novo” (translation: 1/8 New-Christian)
- “3/4 cristão-novo” (translation: 1/16 New-Christian)
- “5/8 cristão-novo” (translation: 5/8 New-Christian)
- “católico romano” (translation: Roman Catholic)
- “Cristão” (translation: Christian)
- “Cristão-novo” (translation: New-Christian)
- “Cristão-velho” (translation: Old-Christian)
- “Elche” (no translation)
- “Escravo” (translation: Slave)
- “Fidalgo” (translation: nobleman)
- “Judeu” (translation: Jew)
- “Mouro” (translation: Moorish)
- “Outro” (translation: Other)
- “parte cristão-novo” (translation: part New-Christian)
- “parte cristão-velho” (translation: part Old-Christian)
- “Preso” (translation: prisoner)
- “étnia cigana” (translation: Gypsy ethnicity)
- “étnia criola” (translation: Creola ethnicity)
- “étnia indiana” (translation: Indian ethnicity)
- “étnia mestiça” (translation: mestizo ethnicity)
- “étnia mulata” (translation: mullato ethnicity)
- “étnia negra” (translation: Colored ethnicity)
- “étnia parda” (translation: brown ethnicity)
- “étnia índia” (translation: Indian ethnicity)

As mentioned in the last chapter we have opted for a middle-ground in social status with some being classes and some being individuals so in reality we do not need to insert all sixteen different social status in the ontology only those that are individuals (this is probably the only data that was feasible to insert by hand using Protégé). As we had already the possible values in a simple text file, we simple programmatically open it and if the line is one that we established as an individual we create it. So, for
example if the line contains “étnia” we know it’s one that we want to create as an individual of the subclass \textit{Different\_ethnicity}.

For example, if the line is “étnia mestiça”\footnote{English Translation (by the author): mestizo ethnicity} the algorithm inserts the individual in the same way as before:

1. Create a sequentially generated id for the social status (e.g SocialStatus\_X where X is an integer)
2. Insert that the SocialStatus\_X is a Different\_ethnicity
3. Insert that the SocialStatus\_X has the data property ID with X as value
4. Insert that the SocialStatus\_X has the data property Name with “étnia mestiça” as value

\begin{verbatim}
### https://eu.pythonanywhere.com/user/Sparrowhank/PInquisition#SocialStatus_16
PInquisition:SocialStatus_16 rdfs:type owl:NamedIndividual ,
PInquisition:Different\_ethnicity ;
PInquisition:ID 16 ;
PInquisition:Name "étnia mestiça"^^xsd:string .
\end{verbatim}

Figure 15: Triples of the individual social status “étnia mestiça” in the Ontology

We do this methodology for all different social status that are individuals (like for example the non-full new or old Christians). The remaining are sub-classes that need to be dealt with when we insert the individual persons in the ontology.

\textbf{Locations:}

We envisioned to populate the ontology not only with all the possible locations but also the connections between them. So, if someone lived in the municipality A we wanted to add the information that municipality A was part of district B that was located in country C. We had two feasible options to accomplish this, option one was to from an external resource to extract all possible parishes, municipalities, districts and countries and load them into the ontology. The second option was to use only the ones that we extracted from the google maps API. If the data in the ontology was mutable and additional people were expected to be added to it, then option one would be necessary (or some other similar alternative) as thankfully no new people are being subjugated to inquisition trials or any other archives are expected to be found the second alternative is more appealing as it allows the ontology to be a little lighter data-wise (which will be important later on).

The first thing the algorithm does is to go through the JSON file created during the ELT process and go through each line. When it finds lines that refers to location data it parses it.

\begin{verbatim}
  "morada\_geo\_details": {
    "localidade": "Covilhã",
    "administrative\_area\_level\_1": "Castelo Branco District",
    "pais": "Portugal"
  },
\end{verbatim}

Figure 16: A typical address line to parse

The opening action of the algorithm is to search the ontology for each location element to understand if it already exists or not, it does this using the owlready2 method \textit{search\_one} that allows to find elements in the ontology based on several characteristics including data properties (in this case the
name). If it is not found, then it is loaded into the ontology in the same logic as before. So, for example if Portugal did not yet exist in the Ontology it would behave like this:

1. Create a sequentially generated id for the location (e.g location_X where X is an integer)
2. Insert that the location_X is a Country
3. Insert that the location_X has the data property ID with X as value
4. Insert that the location_X has the data property Name with “Portugal” as value

If the location does not yet exist and it is not a country then the algorithm connects it to the proper locality (given it have that particular information) and a fifth step is made adding the has_municipality, has_district or has_country object property. For example, in the example of “covilha” the algorithm would create the individual as a location that has as district “Castelo Branco” which by its turn would have as country “Portugal”.

One of the problems we noticed while testing and checking the results of the algorithm is that for some unknown reason google maps API sometimes gives location name in English and other times in the local language for example sometimes in would say that a location was in “açores” others “azores” which are the same location in different languages. To correct this a python correction function had to be built to normalize all the more common mistakes causes by this abnormality (some may persist as there are several thousand locations in the ontology).

It is worthy to note that in their final form these algorithms are only supposed to be run once to load the ontology so their performance time wise was not an important consideration (except in searching for people like we will explain). There are certainly more efficient algorithms to accomplish what was intended, the only concern was to have all locations properly connected in the ontology.

Provinces:

Provinces are an old administrative division of the territory of Portugal. While they were effectively extinct in 1976 in the democratization process following “estado novo” dictatorship they still represent an effective historical division of the territory, and a significant amount of people still identify with them. For this reason, we thought to be worthy to include them in the ontology as they would enable searching for people and processes in given province and even track possible prevalence of costumes in different areas of the territory.
In the chapter dedicated to constructing the ontology we already discussed that a province is structured as a subclass of the class locality, this is also true for a district, municipality and parish. So, what we wanted to do is that if a locality was part of a province, we would connect to it via the is_a OWL property. So, for example “Covilhã” is a municipality is the district of “Castelo Branco” and is also part of the old province of “Beira Baixa”. We wanted to represent it by insert in the ontology that “Covilhã” is both part of the municipality class and the Province_BeiraBaixa class.

To accomplish this, we create separate lists for each province that contains each of the locations that constitute the province, for example in the “Beira Baixa” province list we have the following locations:

- "Belmonte"
- "Castelo Branco"
- "Covilhã"
- "Fundão"
- "Idanha-a-Nova"
- "Oleiros"
- "Penamacor"
- "Proença-a-Nova"
- "Sertã"
- "Vila de Rei"
- "Ródão"
- "Pampilhosa da Serra"
- "Mação"

Now when inserting a new location, we can check all province lists one by one and if it is part of a province, we add the is_a Province X to its properties.

```python
# https://eu.pythonanywhere.com/user/Sparrowhawk/PTinquisition#locality_1005
PTinquisition:locality_1005 rdf:type owl:NamedIndividual ,
PTinquisition:Municipality ,
PTinquisition:Province_BeiraBaixa ;
PTinquisition:Has_district PTinquisition:locality_21 ;
PTinquisition:ID 1006 ;
PTinquisition:Name "Sertã"^^xsd:string .
```

Figure 19: Triples identifying “Sertã” as part of Province_BeiraBaixa

**Historical Periods:**

Unlike the classes we have been discussing so far historical periods are not derived from the curated data so all that was necessary was to decide what periods made sense to be in the Ontology and to insert them by code using owlready2. As previously discussed, we decided for three different kinds of historical periods: reigns, centuries and other important periods. Reigns correspond on which king was in power at a given time, centuries are as the name indicates the century of a given year and other are important events that occurred and effected its time like for example the enlightenment.

For example, if we wanted to insert into the ontology the reign of the king “D. Pedro II” we would do the following:
1. Create a sequentially generated id for the reign (e.g. rei_X where X is an integer)
2. Insert that the rei_X is_a Reign
3. Insert that the rei_X has the data property ID with X as value
4. Insert that the rei_X has the data property Name with “D. Pedro II” as value
5. Insert that the rei_X has the datetime property startDate with 01/01/1683 as value
6. Insert that the rei_X has the datetime property endDate with 31/01/1706 as value

As you may notice in the figure 31 the actual code inserts extra information (is_a HistoricalPeriod and Type). This is because we were not sure we would be able to run an inference engine in the full ontology due to its size. Although later this was fortunately wrong there was no need to change the code as in semantic web extra information makes no harm. An inference engine is able to deduct that if X is a reign and a reign is a HistoricalPeriod then X is also a HistoricalPeriod so we are just helping in out. Of course, if we stated a wrong fact like for example X is a reign and a InquisitionProcess then the inference engine would complain that the ontology was inconsistent.

We do this process for centuries and other interesting HistoricalEvents, in the figure 24 we have the triples that insert the important Portuguese period “Pombalismo”

At this stage we have created all the individuals that support the actual inquisition processes, we have curated and carefully connect them to each individual or class that we created. This was a lengthy but fundamental process to undertake and any error in this department would hinder if not invalidate our goals. We need to create a person and its process in sequential steps as we cannot connect a person to a process that does not exists and vice versa. We also need to carefully look to the father, mother and partners fields, we want to make them individual objects of the type of person and not just data fields of type string. In addition, we also want to only create them if they do not already exist, this denotes the necessity of constructing some-kind of search to decide if they are the same person and if so to connect them. To do this we have to create all persons that directly exist in the inquisitions processes because we cannot search something that do not exists. In this section we will go through each step made to create a person and his/her respective process.

---

15 English Translation (by the author): Pombalism
Inquisition Process:

The first step is to open the JSON file resultant of the ETL process, we then in a loop function sequentially add the data of a given process to a python dictionary and process it. Next, we start by creating and inserting in the ontology the inquisition process. The ID, start_date, end_date and prison_date are simple data properties, and we can add them as they are. loc_Court (Court location), Sentences and Accusations are, as discussed previously, objects so we have to search the ontology in order to be able to connect the inquisition process to the specific individual. We use the owlready2 method Search_one to do this. A process can have several sentences and accusations while there is only one court location. So, we have to go through individual accusation or sentence, search for its object and connect it.

Figure 22: An inquisition process in the ontology

Person:

After the inquisition process is created, we proceed to create the individual person itself. Again, the data properties are trivial to load in the database as they can be added as they are without any further actions. These field are the following:

- FirstName
- LastName
- MiddleName:
- Social_Status_Literal
- Address GPS
- ID
- CivilState_Literal
- Accusation_Literal
- Address_Literal
- Birthplace_literal

Afterwards we focused on the object properties. This required connecting the individual person that we are making to the individuals of the classes that we already did create. For the profession, address, Civil state and birthplace we again use the OWLREADY2 method search_one by the data property name and connect the person to the individual that is found by the appropriate object property. The social status we need to execute some extra steps, as we saw in chapter 3 its architecture is a mixture of classes and individuals. So, a function was created that receives the name of social status and parses it determining if it should be connected to a class via the is_a OWL property or to a given individual by-means-of the appropriate object property.
The gender is not given in the digitalized archives of the inquisition but as we have the name we can make and educated guess. This is where the strength of python and its libraries comes through and by using the library package gender guesser (Gender-guesser) we can successfully determine the gender of most persons successfully. There is the problem that archives are from another time and contain a multitude of names no longer in use that the package is unable to determine its gender. Example of this names are “eufrási” or “frazina “, luckily that are not so numerous that with a little patience we can create lists of names that help a function succeed even when the gender guesser fails. Using a mixture of the gender_guesser package and our own lists we then have a function that receives a name and is able to determine if it’s a male or a female name and all we have to is make the appropriate connection between the individual person and its gender.

The next object property we need to insert in this phase is the Lived_InHistoricalPeriod which connects a person to several possible historical periods. We cannot use the previous method of owlready2 search_one method as we needed to search all events between two dates and the library does not support this (at the time of this work it is an active library with frequent updates) so we had to resort to the SPARQL query language that was introduced in chapter two. The query needed is relatively simple all we need is to list all individual of the class HistoricalPeriod whose dates are within the start date and end date of a process (which is the only date information we have in a process). All that remains to conclude the process is to connect the person to the historicalProcess class by the Lived_InHistoricalPeriod object property. All that is left now is to connect the person and the inquisition process we created by the has_inquisition_process object property.

**Mother and father:**

By this stage we had successfully loaded all the processes of the inquisition into the ontology, but we still had the family in the processes to deal with. Like previously discussed we wanted to insert the family members into the ontology as individual persons if they did not exist already in the ontology. The standing block was how we determine if two persons are the same when we have little more to use than the name of the person? We could not in any way assume that just because they share the same name two persons were the same individual, nor was in the range of this project to do field work in church registries (not to mention the hard-lock downs of covid19). The option remaining was to use all the family fields available to us and try to find a connection. If a person X had as father person A and mother person B then a person A and Z would be the same if the person Z had the same name as person A and had as partner the person B. This approach has some problems we may not have the partner information (or different marriages), the names can be written in different ways with different completions, etc. Lacking more information this is what we could do, and it is effective in a considerable amount of instances.

To be able to perform this search we need to sequentially search everyone and their partner in a minimally efficiency manner. If we use a traditional method, we would pick the person we wanted to verify open the JSON file and go through it looking for the information we wanted. We would need to do this for every single person in the archives which could take weeks given that even with a small sample date it took hours (python is also famously inefficient in cycles like this). To avoid this situation a technique called hashing was used. This technique consists in converting a range of key values into a range of indexes in an array and with it we can build a hash table (a structure that stores data in an associative manner) that contains the information of a person and its family that can be relatively quickly searched.
For example, in the case of a person called “José Nunes” that has as wife “Filipa Nunes”, as mother “Isabel Góis” and as father “Gaspar Nunes”. When we first insert him in the ontology, we also create an entry in a hash table in the format Person Name > Mother Name > Father Name > Partner name with the key to the entry equal to the unique ID of the person. In the case of “José” his entry in the hash table would be: ‘2299881’: ‘José Nunes > Isabel de Góis > Gaspar Nunes> Filipa Nunes ‘. We do this for every person as we insert them in the ontology and when we are done, we have a hash table with every person that has an inquisition process in the archives.

To insert the mothers and fathers into the ontology we now read again the JSON file entry by entry and if a process has a mother and a father, we first go through the hash table and verify if any position in a table has a person whose wife is the same as the mother (and vice-versa for the mothers) in the specific entry of the JSON file. If the algorithm finds one, it simply connects both persons with the has_father object property. If the search returns no results, then we need to create a new individual person. We generate a custom ID and divide the father or mother number in first name, middle name and last name using the function described in chapter 3. We call these properties as data properties and connect both persons with the has_father or has_mother object properties. As we know that are father or mother, we know the gender and can also connect them to the appropriate individual, they have no further properties. If a person has no father or mother information in the archives, then we cannot verify if they already exist and we need to create them using the method just described.

**Partner:**

In case of partners, we use the same technique that we just describe but instead of looking for a mother and father name in the hash table we look for a person that has the partner’s name whose partners has the same name as the person in the JSON file entry we are analyzing. If a person is found we connect both individual persons by-use-of the object property, if returns zero results we create them. While the hash table technique improves the algorithm efficiency dramatically it still takes over twelve hours in a modern computer to go from an empty ontology to a full one. This process is supposed to be run only once assuming no mistakes are made so it’s not problematic. To improve the time, we could use difference techniques like divide and conquer or dynamic programming or even change the programming language as python is notorious slow running this kind of loop cycles. In figure 26 we can see a person, in triple format, inserted in the ontology with all the available information.

Figure 23: An Individual person in the ontology
Semantic web rule language rules:

Semantic web rule language known as SWRL is an expressive OWL-based rule language for the semantic web. It allows users to write rules that can be expressed in terms of OWL concepts to provide more interesting and powerful reasoning capabilities than OWL alone. These rules are of the form on an implication between an antecedent (body) and a consequent (head) (SWRL 2022). This can be understood as the following: If the conditions specified in the antecedent are true then the conditions specified in the consequent also must be true. For example, our society has established that if two people share the same mother and father they are siblings. An inference engine on its own cannot deduce this but if we create a rule stating that if individual A and B are not the same and share the same individual mother and father then it can deduct all siblings in the ontology. Some SWRL rules have been inserted into our ontology that we think improves the information it provides. This is only an example of rules that can be implemented in an ontology, we are only limited to the structure of the ontology and what we want to accomplish. Naturally we need to be careful about the rules we implement, if we are to demanding the inference engine will run out of memory and crash. Rules based on values are especially demanding and can easily overflow the engine.

In this work SWRL rules were used to determine family relations like grandfather, grandmother, aunt, uncle, brother, sister, etc. In appendix C it can be found a full detailed explanation of the rules used.

5.2 - Exploration of the data

Now that we have loaded all the curated data into the ontology, we can verify the results of our work. The first we have to acknowledge is that the resulting ontology is heavy in size. Without the data in occupied barely 60 kilobyte and with the data in goes well over the 200 megabytes. This makes browsing the data problematic and protege crashes frequently when browsing the ontology (and always when trying to initialize a reasoning engine). To circumvent these limitations, we built a web tool to navigate the ontology which will be presented in the next chapter.

A loaded ontology has now 83843 different individuals of the class person in which 33131 have inquisition processes. 44140 are the gender male, 36946 females (with the rest unknown). We have 95 different accusations, 5623 different convictions, 33500 different locations and 362 different professions.

Statistics:

With the help of SPARQL and owlready2 we can also retrieve some numerical information about the individuals in the ontology. In figure 27 we have the graph of all individual present in ontology by class, as expected the class person contains the vast majority of individuals.
We can also do this to individual classes, in figure 28 we have the ten most common accusations. With no surprise we see that by far the most accusation is “Judaismo” which is the “religious crime” of Judaism that the inquisition was built to combat (Netanyahu, 1995).

![Figure 24: Graphic individual by class](image)

![Figure 25: Top ten most frequent accusations](image)
Finally in figure 29 we have the most common convictions, the most common convictions “confiscation of goods”, “payment of cost” and “destitution” which are in the category of financial or light convictions.

![Most Common Convictions](image-url)

**Figure 26: Top ten most frequent convictions**

**Conclusion:**

In this chapter we went through step by step the methods used to load the curated date into the Ontology. We then checked if the data was coherent with our goals for this work. In the next chapter we will present the web application developed to navigate the ontology in a more pleasant and user-friendly way while discussing the possibilities of semantic web in this kind of historical data.
6 - Web Application

6.1 - Introduction

Now that we have a fully loaded Ontology, we desired to develop a way to navigate it. One solution was to use a SPARQL endpoint server like fuseki (Apache Jena Fuseki 2022) to query the ontology but that would require some knowledge of the language and we wanted to find a way to a non-technical user to navigate the ontology. To accomplish we built our own web application using Django and owlready2.

There are several languages that can be used to build a web application but since we have been using python and had already a large quantity of useful function it was just logical to continue to use the same language. There are two major frameworks to build web application within python Django (Django 2022) and Flask (Flask 2022). While Flask would most likely be better to this project as Django is very relational database centric, we were not familiar with it and as time is always a factor in this kind of projects, we opted for a framework we already knew how to operate. There are some libraries for django to work with SPARQL like for example djubby (Wikier) but their development, unlike owlready2, has been long discontinued.

The main challenge of building the web application is that the ontology needs to be loaded into memory only once or everything will be painfully slow and django likes to reload all methods when a page is requested. We also need to be weary on the amount of queries we make to get information as this can make the application to slow to be used. Another problem is that while obviously we want to run an inference engine over our ontology they need huge amount of memory to run that still cannot be parallelized (research is ongoing in this area). To successfully run a reasoner against our ontology we required over 10 gigabytes of ram.

Since the data in the ontology is not expected to be updated nor new information added there is no practical reason to not run the reasoner just once and save the inferences it made as simple triples of the ontology. This way we can avoid having the reasoner active with the application consuming memory. To solve the ontology loading problem was not as easy and evolving a little hacking of the settings of the Django app, the ontology is loaded at the same time the web server first starts and is accessed by means of a special call to the settings class. While Django was not constructed with this kind of calls in mind, they seem to work without too many problems. The drawback is that we need to keep the
ontology in memory at all times which occupies around 300 megabytes which is quite heavy for a web application.

We retrieve the information with a mixture of SPARQL and owready2. We use SPARQL for the queries (e.g., all persons with a certain last name) and owready2 to retrieve the properties of a given individual. We need to convert the information into a python dictionary format which is what django expects to be able process the information. Caution has to be made to not extract too much information in any given time as this makes the process too slow for web browsing.

This web application is temporarily available at https://sparrowhank.eu.pythonanywhere.com. It is important to note that it is made as an academic showcase only, it does not have the security or performance concerns necessary for the use of the general public. SPARQL queries are open to use, they are extremely powerful allowing the complete change or deletion of any triple in the ontology. It also needs a better server than pythonanywhere standard web hosting gives and can sometimes be slow in its loading or querying the ontology. These problems can easily be solved migrating into a better server in the future.

6.2 - Application

The first feature we wanted in this application was the ability to run any SPARQL code we wanted to be able to query anything and everything in the ontology. To do this we programmed a function that receives a valid SPARQL query, runs it and returns the result. If the query contains errors, it transmits it to the user as it does if the query result is nothing found.

![Figure 28: SPARQL query in application](image)

We also wanted to show we can abstract from SPARQL and allow the users to search for information as we would in a normal website using a normal database. We allow the user to search by username, ID or other parameters like historical region, type of profession or type of accusation. What the application does behind the curtain is a simple SPARQL query nothing more, nothing less. In figure 32
we have a search for any person in the ontology whose Birthplace is located in the historical region of “Beira Alta” and the profession was military one.

This is important to note because we are not in any way limited the parameters in the application in what we may search. Using SPARQL we can search for anything in the ontology with the only limitation being the data itself and our imagination. We could for example search for people with a kind of sentence or even for people with an inquisition process whose father also have an inquisition process.

Person:

One of the goals of the web application was allowing to navigate a person property in a user-friendly way. Going back to the figure 32 if we click the button “view” of the first person found we are presented with the following page (Figure 33):
Figure 30: A individual person in the web app
As it can be seen in this page, we have the complete information of a given person. In the top corners we have his ID and Process Number (that itself is a link to his process). Afterwards we have the data properties followed by the object properties (that are all links to other pages with more information). In a more thorough look, we notice that the effort to search and connect individuals in conjunction with the SWRL rules implemented allowed the reasoner to infer the grandparents of this individual person. He also was able to determine that he was an adult and the four persons he is descendant of. Notice that in the is_descendant_of object property we do not do extra queries to get the FirstName and LastName data properties in a effort to reduce the loading time.

Object properties:

All object properties are interactive and link to a page that list all its properties. Continuing our example if we click in the object property social status (not the data property in the upper part of the person page) of the individual we are presented with figure 34. Here we can see the object class (which again is interactive to see its properties) and the respective data properties.

![Figure 31: Object Property in the web app](image)

Family:

All family members are interactive and lead to their own page. In figure 35 we have the page of the individual partner. In a carefully look we notice that she also has an inquisition process herself which indicates that while certainly flawed the algorithm that searched and connected different people does have some success.
Inquisition Process:

The inquisition process of each person is equally intractable and transport the user to a page with all its details. As can be seen in figure 36 we again have the ID and who the process refers to (interactable) in the top page, followed by the data properties and ending with the object properties.

As is the case with the person page the object properties are interactable, if we click in the “Judaísmo” accusation we are transported to a page that details its class and data properties (fig 54).
Properties:

As an extra we implemented the ability to navigate through the classes, objects and properties of the ontology and explore their restrictions, domains, ranges and metadata. In the ontology section we list the classes, object properties and data properties. This is automatically generated and dynamically reflects any alteration made to the ontology.

In the individual class pages, we have the superclass of each class, its subclasses and respective restrictions. Everything is interactable except its own class and the superclass Thing which is a special class of OWL where all other classes descent from and has no properties or restrictions.
6.3 - Conclusion

In this chapter a web application to navigate the ontology was presented. With it we hoped to have demonstrated that ontologies are not limited to rich query able data but that interactable and fun application can use it to display the information. In chapter 7 we will finish this work by presenting our conclusions.
7 - Conclusions

7.1 - Introduction

It was the purpose of this work to collect the historical rich data that the Portuguese Holy Inquisition have left us and using semantic web techniques build a data structure that could harbor and enrich it. Its chapter three we have seen how the data could be extracted and transformed. We then proposed a possible ontology to harbour this data. In chapter four we have seen a possible methodology to import the data into the ontology. Finally in chapter five we demonstrated that a full ontology could be navigated and queried. We show that by using simple rules and an inference engine unstated information could be extracted from the data. We conclude this work by having met all initial goals and with the hope of having shown the potential of semantic web in storing and working with historical data.

7.2 - Struggles and challenges

The sheer amount of data available in the archives was the main challenge throughout this work. From the data extraction process, to loading the data into the ontology and finally to be able to navigate it the data size has always been a factor and forced the search of different solutions. The fact that there are remarkably few works done in using semantic web for historical data was also a challenge and motivation. We had to find the solutions for the problems and challenges we faced in our own without the comfort of previously though and works, this is always challenge as the best solutions are built in the foundation built by others.

7.3 - Future Work

Academic and scientific work never stops. As future work this project would benefit greatly with a historian look at the decisions we made in the transformation of the data, verifying that the areas attributed to professions, accusations and sentences make sense and are correct in the rich field of study of history. Another area that should be looked at and improved is the family connections, while some were found and politely guessed it is certain that a significant amount was missed. If a way is found to find them (by a smarter algorithm and/or by consulting other archives) it would be possible to build impressive gemmological trees of families along the centuries the inquisition lasted. One of the beauties of the semantic web is that if we found that actually person A and B are the same, we would only need to use the OWL property sameas and the reasoner would take care of the rest.
A last suggestion for future work is the possible connection to other sources of information. One of the strengths of the semantic web is that it is able to connect data in different locations and enrich a given knowledge. Spain also had its own holy inquisition with an equally rich history, if a way is found to connect both sets of data it is very likely we will find individuals or family of individuals that are the same. If it is possible to connect them, we could open a whole new door of knowledge of their lives and even how different institutions and sensibilities would behave in the face of the same person and habit's.
Bibliography


Is a tomato a fruit or a vegetable and why? Eufic. (n.d.). Retrieved September 14, 2022, from https://www.eufic.org/en/healthy-living/article/is-a-tomato-a-fruit-or-a-vegetable-and-why#:~:text=The%20botanical%20classification%3A%20Tomatoes%20are%20fruits.,&text=A%20botanical%20fruit%20would%20have%20the%20flower%20o

t%20the%20plant%20&amp;text=With%20this%20definition%20in%20mind%2C%20flowers%20or%20the%20t

omato%20plant

62


Appendix A: An introduction to SPARQL

SPARQL has a core building block named triple pattern. This triple pattern is very similar to triples, but it allows to have variables instead of resources in the subject, predicate and object. Variables are designated as symbols preceded by the special character “?” (Allemang et al., 2012).

In order to understand SPARQL lets expand our previous graph to include Euripides:

![Diagram of Euripides and Sophocles]

Suppose we want to query the plays that were written by someone that was born in the city of Colonus. One SPARQL query could be:

```
SELECT ?play, ?birthplace
```
WHERE {
  ?person wrote ?play.
  ?person WasBorn ?birthplace.
  FILTER (?birthplace = "Colonus")
}

SPARQL uses a notation format and tries to mimic natural language, here we are asking a play that was written by a person that was born in the city of Colonus. Needless to say our graph is very simplified and in reality a play would have a variety of properties including a name. The same for Colonus and Euripides. So in a “real world scenario” our question would have to ask for the name of the plays written by someone that is a playwright whose name of the place of birth was Colonus.

Since a set of RDF triples is viewed as a graph, a more interesting query is one in which the query specifies a graph pattern. A graph pattern is specified as a set of triple patterns, with the stipulation that any variable that appears in two or more triple patterns must match the same resource in the graph. In SPARQL syntax, graph patterns are given as a list of triple patterns enclosed within braces (\{} and \}\)(Allemang et al., 2012). SPARQL has a mode of operation called select form that consists in certain variables being selected from the graph pattern which results in all related binding related to them being returned in a more standard machine-readable form. The SELECT form in SPARQL can be thought of as converting a graph to a table; the graph pattern matches parts of the graph, and the resulting bindings are returned as a table of values for the corresponding variables.

SPARQL like any RDF query engine is deeply tied to the RDF store, it relies on the internal representations of the data to perform its queries which results in only being as efficient as the RDF store is fine tuned.
Appendix B: SWRL RULES

Maturity: In this rule we want that the ontology is able to infer if a person is a child (under 18), adult (18 to 65) or elderly (over 65). To do this we use the following SWRL rules:

\[
\begin{align*}
\text{Person}(\text{?X1}) \land \text{Age}(\text{?X1},\text{?Z}) \land \text{lessThanOrEqual}(\text{?Z}, 17) & \rightarrow \text{Has_Maturity}(\text{?X1}, \text{Minor})' \\
\text{Person}(\text{?X1}) \land \text{Age}(\text{?X1},\text{?Z}) \land \text{greaterThanOrEqual}(\text{?Z}, 18) \land \text{lessThanOrEqual}(\text{?Z}, 64) & \rightarrow \text{Has_Maturity}(\text{?X1}, \text{Adult})' \\
\text{Person}(\text{?X1}) \land \text{Age}(\text{?X1},\text{?Z}) \land \text{greaterThanOrEqual}(\text{?Z}, 65) & \rightarrow \text{Has_Maturity}(\text{?X1}, \text{Elderly})'
\end{align*}
\]

Sibling: In this rule we want to establish two individuals of the class person are siblings if they are not the same and have the same mother and father. To do this we use the following SWRL rules:

\[
\begin{align*}
\text{Person}(\text{?X1}) \land \text{Is_ChildOf}(\text{?X1},\text{?X2}) \land \text{Person}(\text{?Y}) \land \text{Is_ChildOf}(\text{?Y},\text{?X2}) \land \text{differentFrom}(\text{?X1},\text{?Y}) & \rightarrow \text{Is_SiblingOf}(\text{?X1},\text{?Y})
\end{align*}
\]

Brother: A individual person is the brother of other if they are siblings and he is of the gender male:

\[
\begin{align*}
\text{Person}(\text{?X1}) \land \text{Person}(\text{?Y}) \land \text{Is_SiblingOf}(\text{?X1},\text{?Y}) \land \text{Has_Gender}(\text{?Y},\text{?XI}) \land \text{resolveURI}(\text{?XI}, \text{https://eu.pythonanywhere.com/user/Sparrowhank/PTinquisition#Male}) & \rightarrow \text{Has_Brother}(\text{?X1},\text{?Y})
\end{align*}
\]

Sister: A individual person is the sister of other if they are siblings and he is of the gender female:

\[
\begin{align*}
\text{Person}(\text{?X1}) \land \text{Person}(\text{?Y}) \land \text{Is_SiblingOf}(\text{?X1},\text{?Y}) \land \text{Has_Gender}(\text{?Y},\text{?XI}) \land \text{resolveURI}(\text{?XI}, \text{https://eu.pythonanywhere.com/user/Sparrowhank/PTinquisition#Female}) & \rightarrow \text{Has_Sister}(\text{?X1},\text{?Y})
\end{align*}
\]

Uncle: If person A and B are siblings then if A is male and B has children then person A is the uncle of person B children

\[
\text{Has_Father}(\text{?X1},\text{?X2}) \land \text{Has_Brother}(\text{?X2},\text{?Y}) \rightarrow \text{Has_Uncle}(\text{?X1},\text{?Y})
\]

Aunt: If person A and B are siblings then if A is female and B has children then person A is the uncle of person B children

\[
\text{Has_Father}(\text{?X1},\text{?X2}) \land \text{Has_Sister}(\text{?X2},\text{?Y}) \rightarrow \text{Has_Aunt}(\text{?X1},\text{?Y})
\]

Grandfather: If person A is father of person B and person B is father or mother of Person C then person A is grandfather of person C.

\[
\text{Has_Father}(\text{?X1},\text{?X2}), \text{Has_Father}(\text{?X2},\text{?Y}) \rightarrow \text{Has_Grandfather}(\text{?X1},\text{?Y})
\]
Has_Mother(?X1, ?X2), Has_Father(?X2, ?Y) \rightarrow \text{Has_Grandfather(?X1, ?Y)}

**Grandmother:** If person A is mother of person B and person B is father or mother of Person C then person A is grandfather of person C.

Has_Father(?X1, ?X2), Has_Mother(?X2, ?Y) \rightarrow \text{Has_Grandmother(?X1, ?Y)}
Has_Mother(?X1, ?X2), Has_Mother(?X2, ?Y) \rightarrow \text{Has_Grandmother(?X1, ?Y)}

**Descendant:** If person A is mother or father of person B then person B is descendant of person A. If person C is descendent of person B and person B is descendant of person A then person C is descendant of person A (and so on).

\[
\text{Person(?X1) \land Person(?X2) \land Is_ChildOf(?X1, ?X2)} \rightarrow \\
\quad \text{Is_Descendant(?X1, ?X2)}
\]

\[
\text{Person(?X1) \land Person(?X2) \land Is_Descendant(?X1, ?X2) \land Person(?X3) \land} \\
\quad \text{Is_Descendant(?X2, ?X3)} \rightarrow \text{Is_Descendant(?X1, ?X3)}
\]
Appendix C: Ontology in Protégé

Using the Stanford developed application we can also browse the loaded ontology we just created, we just need to edit the default memory than Protege uses to over one gigabyte or the program will crash.

When we open the ontology, we are given some information like the description (figure 40) and some metrics of the ontology (figure 41)

Figure 37: Ontology Comments in Protégé

Figure 38: Ontology Metrics in Protégé
We can now navigate through the classes, properties and individuals. In fig 42 we have “Isabel Loba” that we searched previously using SPARQL.

Figure 39: "Isabel Loba" individual in Protégé

We can also navigate her process by clicking on it:
We can see the properties of “Accusation_1”

We can also navigate the properties or her father (figure 44) and mother (figure 45) which are in their own created individuals that did not exits in the archives.
Figure 42: “Isabel Loba” father in Protégé

Figure 43: “Isabel Loba” mother in Protégé

Sadly, Protégé becomes very unstable and more often that do not crash when we try to use its reasoners in the Ontology in its full capacity probably caused by some limitation of java that needs to be adjusted.