INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

MESTRADO EM ENGENHARIA INFORMÁTICA

Integração automatizada de informação de horários de transportes

JOÃO BAPTISTA MONTEIRO WESTERBERG

Outubro de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Automated integration of transport timetable information

João Westerberg

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Computer Engineering, Specialisation Area of Software Engineering

Supervisor: Jorge Santos

Porto, October 15, 2020

Abstract

The ever-growing Web contains a large amount of data. This large amount of data is useful when combined with applications that can refine it and use it to improve its users' lives. However, using the data available is not an easy task since most of the information is not represented in machine-friendly formats.

Instead, this information is represented in formats ideal for human users, resulting in an additional effort for having machines interpreting, extracting, and integrating it, while at the same time ensuring the consistency of information from different sources.

In this project, a solution using an ontology-based integration combined with web robots' extraction automates the process required for updating information regarding schedules of public transports. An already existing application receives that information and uses it to calculate efficient routes for commuters.

The proposed solution can extract information from multiple online sources and transform it into different formats. It can extract and transform the information from PDFs and HTML. The system provides a web service for the exportation of these formats by a route optimization system.

This document contains the detailed process of the design and construction of the integration system. It describes the alternatives and selections that lead to the application created. Lastly, it evaluates the solution by performing extraction from several sources relevant to the project's domain.

Keywords: Information Retrieval, Web crawling, Information Integration, Ontology, PDF extraction

Contents

List of Figures

List of Tables

List of Rules

List of Acronyms

DIKW	Data, Information, Knowledge, Wisdom.
DTO	Data Transfer Object.
EDOAL	Expressive and Declarative Ontology Alignment Language.
ETL	Extract Transform Load.
FAST	Function Analysis System Technique.
GTFS	General Transit Feed Specification.
IE	Information Extraction.
IR	Information retrieval.
NLP	Natural Language Processing.
OS	Operating system.
QFD	Quality Function Deployment.
RIF	Rule Interchange Format.
SMTP	Simple Mail Transfer Protocol.
SWRL	Semantic Web Rule Language.
URI	Uniform Resource Identificator.
URL	Uniform Resource Locator.
W3C	World Wide Web Consortium.

Chapter 1

Introduction

1.1 Introduction

In a time when society views public transportation as a cost-effective and eco-friendly medium of dislocation (Hong et al. 2016), there are opportunities for solutions that focus on the publication of relevant information to the topic.

Different categories of solutions focus on the publication of relevant information, such as real-time transport tracking, price optimization, topological map generation, or route optimization. Most of these solutions have in common the prerequisite of information about the schedules available in timetables.

For developing a solution to publicize information about timetable information or route optimization, it is essential to understand the business of public transportation and the decisions that improve the efficiency of the services offered. The transports timetable's information is in constant change, and, as a response to that environment, the system needs a mechanism to handle the changes, maintaining access to precise and up-to-date information. However, it is critical to address and resolve some concerns for the development of such a system.

From a user perspective, the benefits of using a unique information service are not only to have access to accurate information but also to search a mixed group of transport providers resulting in the freedom of not being restricted to a specific provider or a single medium of transportation.

When developing an information distribution system, considering the user necessities is essential to integrate the data from the various providers into a standard format.

1.2 Context

The project described in the present document is the result of a collaborative effort with OPT [1]. This organization develops projects in the field of transports management and optimization. One of those projects is a tool for performing an algorithmic generation of optimized schedules to providers. This particular tool contains an export functionality that allows the integration of the generated timetables with the OPT internal system, allowing for other company tools to use that information.

[1]OPT website available at `http://www.opt.pt/`

However, some public transportation providers prefer to manually design the timetables instead of using a tool-assisted system or use a different tool. The information about the schedules of those providers' vehicles is essential for other projects developed by OPT, for example, a mobile application designated "Move.Me" that performs route optimization[2].

The company manually verifies if the information available on the web was updated to address this necessity of information and provide better services for its users. This information is usually available on the websites of the providers that use an alternative to their software. To remove the effort of this periodic task of verification, OPT wants to develop a system to automatically detect changes and integrate the information available online on the providers' website.

1.3 Problem

To maximize its vehicles usage and drivers' efficiency, each public transportation company has adopted different approaches when defining the vehicles' route and the respective timetable. The result is a complex set of rules and domain concepts that can vary depending on operation conditions, time of the year, and transportation medium. This ambiguity leads to an increase in the difficulty of the data integration phase.

Furthermore, most of the timetable information provided by those transport companies is only available on their respective websites, where it is portrayed in a non-machine friendly manner since the target audience is human users. Such information is then typically displayed using an HTML table or a PDF file.

The nonexistence of APIs or machine convenient formats such as RDF, CSV, JSON, or XML, results in an inefficient process for updating the information distribution system. In extreme cases, the team in charge of the system must update the information after detecting the changes manually. This process is also prone to human error, and without a service that can detect the changes to the website information, the updating speed might not be up to the user's standards. If a user of a transports information application misses a train due to outdated information, he could change to a competitor application in the time that takes the next vehicle to arrive.

Since one of the predicaments when developing this type of solution is that the available data is typically not machine friendly, there is also an issue when trying to infer metadata from the available information formats.

It is impossible to infer the metadata of a given data set and guarantee its accuracy in some cases. A possible solution is the development of a tool to assist the process of data extraction. This approach can still be an improvement when comparing it to manually updating since it reduces the resources needed to integrate the information from the various sources into a centralized system.

The main benefit of this improvement is that once the mapping rules are defined, as long as there are no future changes in the data structure, updates to the data will automatically integrate with the information distribution system. However, before the phase of integration, there must be a task of information extraction.

[2]Route Optimization App available at `https://play.google.com/store/apps/details?id=com.moveme`

There is a wide variety of tools and techniques that specialize in extracting information from semi-structured web-pages. To take advantage of the useful information existent on the wide web is common the development of procedures designated by wrappers or web scrapers that have the functionality of extracting a particular web resource's content (Kushmerick, Weld, and Doorenbos 1997).

Besides the problems and processes already presented, the solution changes depending on the geographical and social-economical environment for which it will operate. In the European panorama, there are already well-defined standardized data formats to represent public transport information. As an example, the DatexII surges as a response for exchanging traffic information and traffic data in a standardized way, and promises to be a foundation for communication protocols in self-driving cars (ICEACSA-TEKIA 2019).

This project targets the Portuguese public transportation's environment, more specifically the operators in Portugal's capital (Lisbon) and the northern region of Portugal. By selecting a broad set of websites in this case study, it was possible to document and understand the different presentation methods used by operators.

It is essential to mention that the patterns identified in this group of operators can differ depending on several factors, such as the operators' team's technical programming knowledge or the budget for the website development.

Using the analysis of the 25 operators and websites presented in appendix A, the graph shown in figure 1.1 represents the distribution per presentation method.

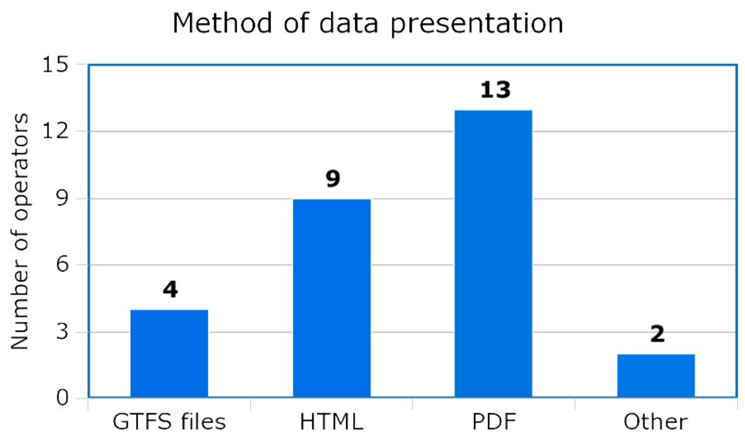


Figure 1.1: Distribution of operators by method of timetable presentation

The total number of results is superior to the number of operators because some operators have more than one method for presenting their schedules.

Its easily notable that the most popular method of presentation is PDFs, followed by HTML and General Transit Feed Specification (GTFS) files. From the selected operators, only 1 had a working API and another a Flash application, both represented in the "other" group.

A few of the operators use GTFS, a google maintained data format to represent public transportation timetable and location coordinates (Google 2019). The operators presenting the data using this format are all located in Lisbon, as a result of a project name "Open Data Lisbon," which provides well-formatted and open data in several fields of management (for instance, education, environment, mobility) (OpenLisbon 2018).

1.4 Objectives

Considering the problem presented, the main objectives of the project described in this document can be divided in :

- Detect updates in the information sources.
- Perform information retrieval from the websites/files when changes are detected.
- Define input/output formats and respective mapping rules.
- Guarantee the consistency of information retrieved from multiple sources, allowing inference and querying of information from transports with different transport providers
- Provide an interface to access and export the converted information.

The solution proposed to solve this problem is a set of software artifacts capable of automating the process of information retrieval and information integration. This solution handles information retrieval by applying web crawling and web parsing techniques to the providers' website combined with ontology-based information integration.

The following research questions can be formulated:

- What is an automated way to retrieve and integrate the schedules displayed in the timetables?
- How is the quality of the results generated by this automated approach?
- How does the proposed solution compare with the manual approach?

With these questions in mind, the following optimizing hypothesis or enhancement hypothesis is proposed:

An ontology-based system using web crawling and scraping can automate information retrieval and integration, reducing the resources necessary while retaining the information's quality if compared to a manual extraction and integration process.

1.5 Document Structure

This document is divided into the following chapters: Introduction, Project Value Analysis, State of the art, Selection of technologies and solutions, Design, Evaluation of the solution.

In the first and current chapter, Introduction, the contextual framework of the project is present. It is in this chapter that the problem and objectives of the project are specified.

The following chapter, the project value analysis, presents the project as a product, analyzing the value of its features and functionalities for the costumers and stakeholders.

In the third chapter, state of the art, different approaches are studied and summarized from books, articles, surveys, and technical papers describing solutions to similar problems. This chapter also compiles the information on technologies to apply in the solution.

The fourth chapter, the Selection of technologies and solutions, compares the specificities and characteristics of the technologies identified in the previous chapter, and selects the ones suitable for the solution development.

The Design chapter contains the technical details for the system's architecture. This chapter represents several high-level architectural alternatives, displaying the design pattern resultant from a software engineering methodology. Furthermore, it provides the detailed design of the proposed solution that was selected.

The sixth chapter, titled Implementation, describes the process for the solution's codification alongside implementation details, algorithms, and code snippets.

In the "Evaluation of the solution" chapter, the methodologies, criteria, metrics, and measurements to evaluate the solution are described.

The final chapter presents the document's conclusion, mentioning the development's constraints and future work suggestions.

Chapter 2

Project Value Analysis

This chapter presents the value analysis of the project. It defines the process for creating, and explores the advantages and disadvantages of the solution proposed from the point of view of the clients.

2.1 The innovation process

When developing a new product, whether it is a physical product, an alternative process, or a straightforward service, the organization or individual must use a systematic process to assess the benefits of the product's features.

By applying a systematic process, it becomes possible to ensure that the features' cost is not higher than the cost necessary to carry out its functions. In other words, by using a process to analyze the product's value, it is possible to guarantee that the value is positive.

Considering that the project is being developed in cooperation with a company (OPT), it is expected that the initial steps for an idea generation were addressed. More specifically, the opportunity identification and opportunity analysis.

At the time, the company has over 25 years of experience in the market of public transportation, and alongside the techniques of market research and technology trend analysis.

By understanding the external factors and opportunities, it is possible to initiate the genesis of ideas. Most of these ideas and their consequential selection result from processes such as brainstorming sessions.

It is then possible to use a systematic approach to value analysis. There exist several models, protocols, and techniques, but an example is the Function Analysis System Technique (FAST), used to identify functions and accurately define the problem. According to Borza (2011), FAST Diagrams provide a graphical representation of how functions are linked and work together in a system to deliver the intended results.

In the figure 2.1, a possible FAST diagram for the element of concept definition, proposes the higher function of improving the quality of the services offered by OPT by increasing the offer of accurate information available in their system. The low order function is to detect the changes in the data sources.

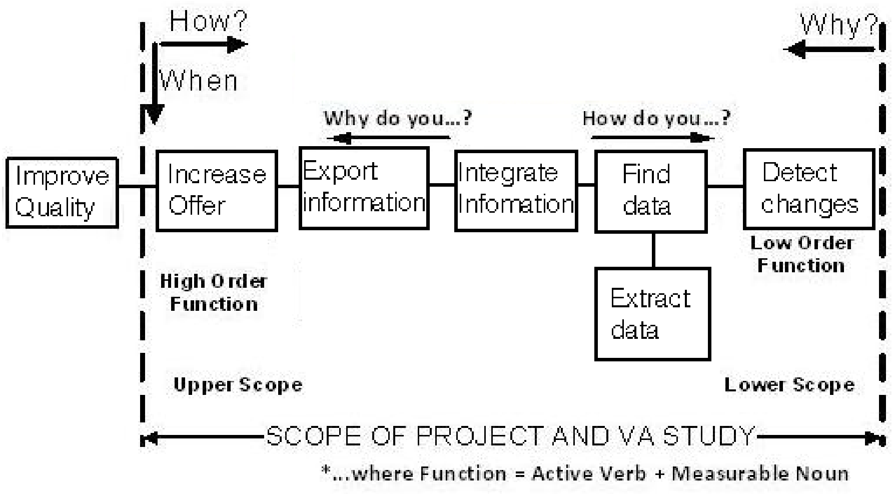
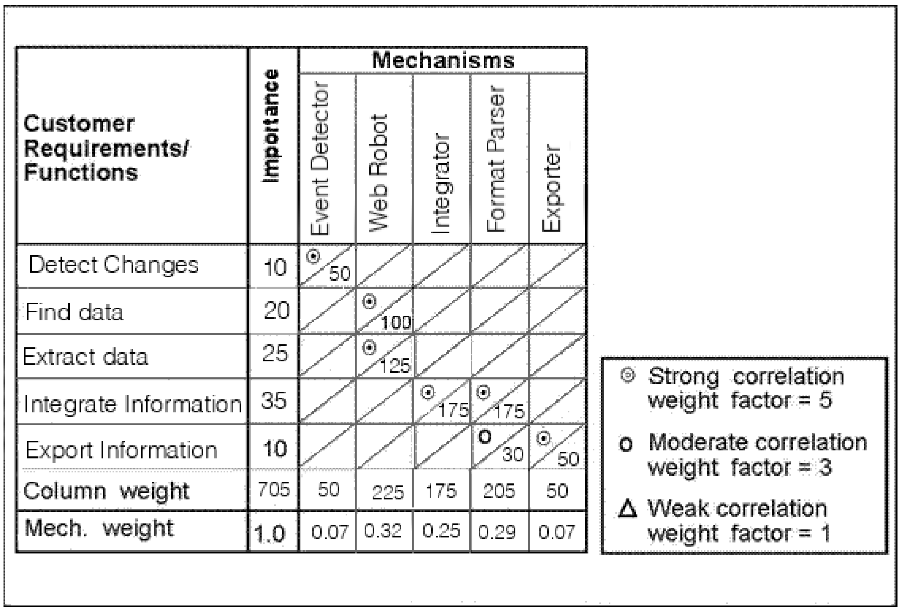


Figure 2.1: Proposed FAST diagram

FAST can also be used in conjunction with a different technique Quality Function Deployment (QFD), which allows determining the importance of each mechanism, or in this case, each software component, by using a Value Analysis Matrix. As such, the matrix in figure 2.2 represents the importance of each mechanism considering the relevant functions defined in the FAST diagram.



Customer Requirements/ Functions	**Importance**	**Mechanisms**				
		Event Detector	Web Robot	Integrator	Format Parser	Exporter
Detect Changes	10	⊙ 50				
Find data	20		⊙ 100			
Extract data	25		⊙ 125			
Integrate Information	35			⊙ 175	⊙ 175	
Export Information	10				o 30	⊙ 50
Column weight	705	50	225	175	205	50
Mech. weight	1.0	0.07	0.32	0.25	0.29	0.07

⊙ Strong correlation weight factor = 5

o Moderate correlation weight factor = 3

Δ Weak correlation weight factor = 1

Figure 2.2: Proposed FAST diagram

The analysis suggests that web robots, integrators, and format parsers are the system's most relevant components.

2.2 Value definition

Value creation can be defined as a trade-off between benefits and sacrifices perceived by customers (Walters and Lancaster 2000). Since the primary purpose of this project is to increase the offer and quality of products/services already present in the organization and reduce the effort of processes used by the organization, the customer segments of this solution can be defined as the final users of the organization's tools and the own organization.

For the end-users of the organization's products/tools, this project does not provide any sacrifices. However, it provides value by having the benefits of a higher quality of routes, resultant in an increase in the information in the system, and a faster process for updating information.

However, for the organization and the organization's affiliates, the solution presents the sacrifices of increasing the number of projects that require maintenance in their platform environment, increasing the maintenance costs and effort. Contrasting these sacrifices, the product has the benefit of reducing the time required for the integration process, increasing the quality and value of their products, and consequently improving the company's image and their products/services.

It is then possible to define the perceived value of the product for each of its customer segments. For the end-users of the OPT products, the solution created has a null perceived value. For these users, the solution's existence is not known, and they can only notice a quality increase in the services they already use. For the organization, the solution provides an increase of information available in their system and an improvement for their credibility and trust.

2.3 Value proposition

When defining a solution or product, it is essential to look at what the consumers need and what can be additionally offered. A value proposition is ideal for defining these two concepts.

The value proposition is constituted by costumer profile (jobs, pains, and gains), alongside the value map (products and services and their pain relievers and the gain creators) (Osterwalder et al. 2014).

The customer profile section of the value proposition focuses on what the costumers have to do, their jobs, the difficulties they find when executing their tasks, and what can be done to improve the lives of these costumers.

The value map puts the business side of the equation in the spotlight. Its analyses are on what are the products and services offered to the costumers, how these products ease the customer pains, and how the gains the consumer would appreciate to have or be surprised by the existence of can be created.

2.3.1 Jobs

The two very distinct types of customers identified have different tasks and necessities, so for each user type is necessary to specify to which job they are assigned.

The organization and its elements have the jobs of verifying if the timetables' information was changed, update the information on their services, and support the information of new providers.

Since the end-user of the organization systems is not aware of the existence of this internal solution, having a null perceived value, it is possible to delegate their necessity of quality information to the organization. Thus, the organization also has the job of providing its clients with accurate and diverse information regarding public transportation services' timetables.

2.3.2 Pains

The pains of the organization are that the tasks necessary for integrating this information are time-consuming and costly. Also, a dissatisfied client results in additional effort handling complaints and a reduction in the organization's image.

2.3.3 Gains

Considering the organization's participation in the development of the solution, it becomes difficult to identify its gains. The solution's functionalities are the result of a collaborated effort, and as such, the majority of those are designed to attenuate the pains. However, a possible gain could be having a single view of the providers' information not associated with them. By versioning that data/information, it would now be possible to create a history or timeline, where before that information would be lost once replaced by the providers.

2.3.4 Products and Services

As previously mentioned, the product is a software component that enables automatic extraction of information about timetables of public transportation schedules from websites and integrates it into standard formats.

2.3.5 Pain relievers

By automating the job of detecting and transforming the changes in timetables, part of the effort and cost is reduced. Since this new information also increases the quality of the company products, the number of complaints should be lower, and the organization's image should improve in the view of their clients.

2.3.6 Gain creators

If the system provides a tool for visualizing the information before exportation, the information that previously was scattered on the web will be available in a single place under the same view. Since the information is now integrated into the organization's system, that information can be versioned and accessed, even when it becomes unavailable on the original website.

2.4 Canvas Model

The canvas model is a simple but essential management tool to visually represent a business model. It accomplishes to answer the questions about the business customers, partners, resources, and activities such as "Who are our clients?", "What do our clients want?" or "How will the clients be reached?"

This is achieved by summarizing the key aspects of that business model and representing it by using an infographic.

In this project, the canvas model can be applied. However, it is important to mention that although the project creates value for the company using it, it will only create value indirectly for the end-users by improving the quality of the systems that they already use.

The canvas model represented in figure 2.3 was created considering the customer segment as the company to which the product was created. As such, the customer relationships are techniques used for a co-creation environment.

Since the benefits that this project offers for the company are reducing the time required for the integration process, increasing the quality and value of their products, and consequently improving their products' image, there is no direct revenue stream for this product. Although there is no direct monetary retribution, the product is capable of reducing the human effort required by the task, and therefore should reduce the monetary amount spend on human labor.

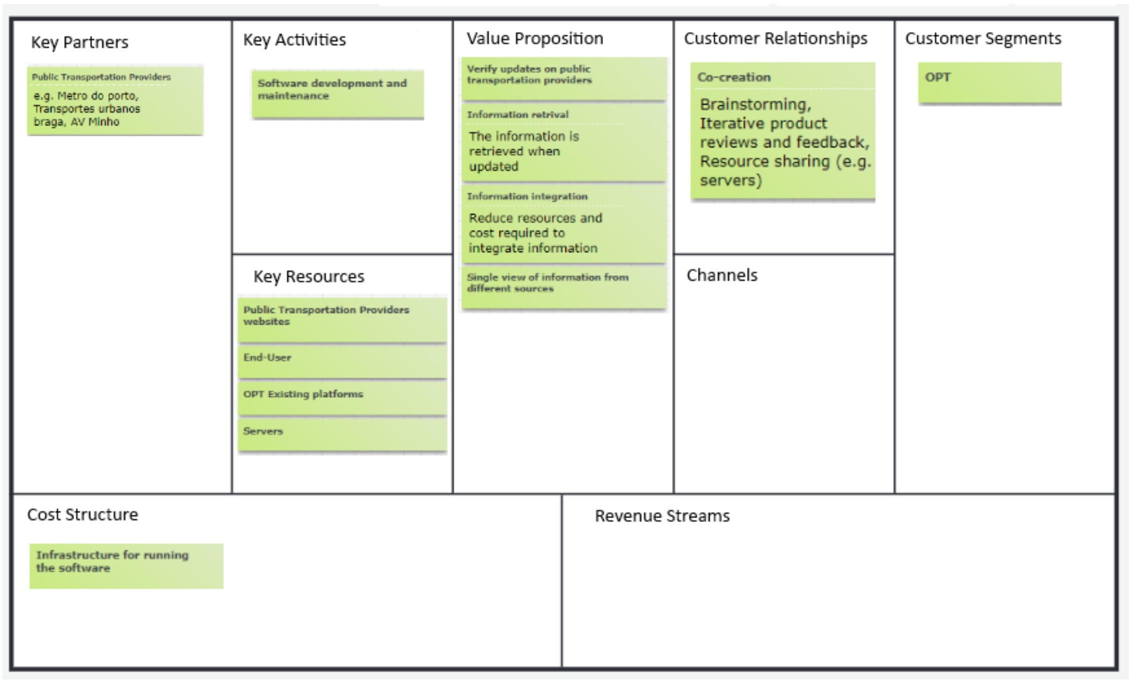

Key Partners
Public Transportation Providers
e.g. Metro do porto, Transportes urbanos braga, AV Minho
Key Activities
Software development and maintenance
Key Resources
Public Transportation Providers websites
End-User
OPT Existing platforms
Servers
Value Proposition
Verify updates on public transportation providers
Information retrival
The information is retrieved when updated
Information integration
Reduce resources and cost required to integrate information
Single view of information from different sources
Customer Relationships
Co-creation
Brainstorming, Iterative product reviews and feedback, Resource sharing (e.g. servers)
Channels
Customer Segments
OPT
Cost Structure
Infrastructure for running the software
Revenue Streams

Figure 2.3: Proposed canvas model

It would be possible to create a canvas for the value propositions of the end-users customer segments. This canvas would remain the same in the backstage (Key Partners, Key Activities, Key Resources, and Cost Structure), while in the front stage, the customer relationships and channels would reflect the OPT's already existing channels.

Chapter 3

State of art

In this chapter, it is documented the current approaches to resolve the concerns addressed in the problem section of the document and existing technologies that can be applied in the development of the solution. It presents an overview of the state of the art for information extraction and information integration, with special focus on ontology-based approaches.

3.1 Information extraction

When dealing with information managing systems it is important to have a clear definition and distinction between data, information, and knowledge. These three concepts are studied in the philosophical field of epistemology, however, as a consequence of their use in computer science, there are several interpretations and definitions of data, information, and knowledge which results in inconsistencies and often conflicting results (Chen et al. 2008).

Donald Kraft provides a succinct, easy to understand and relevant definition. He defines that Data are atomic facts, basic elements of "truth," without interpretation or greater context. Information is a set of facts with processing capability added, such as context, relationships to other facts about the same or related objects, implying increased usefulness. The information provides meaning to data. Knowledge is information with more context and understanding, perhaps with the addition of rules to extend definitions and allow inference (Zins 2007).

A complementary definition of knowledge is given by Donald Hawkins where he states that knowledge "emerges from analysis, reflection upon, and synthesis of information" (Zins 2007)

Ackoff (1989) proposed this concept of continuous refinement and processing of the previous result, this idea can be visualized using the pyramid represented in the figure 3.1 and is usually referred as knowledge pyramid/hierarchy or Data, Information, Knowledge, Wisdom (DIKW) pyramid/hierarchy. The concept of wisdom goes beyond the concept of knowledge, and it allows not only to infer facts but also to make predictions and decisions to improve the effectiveness of a process (Ackoff 1989).

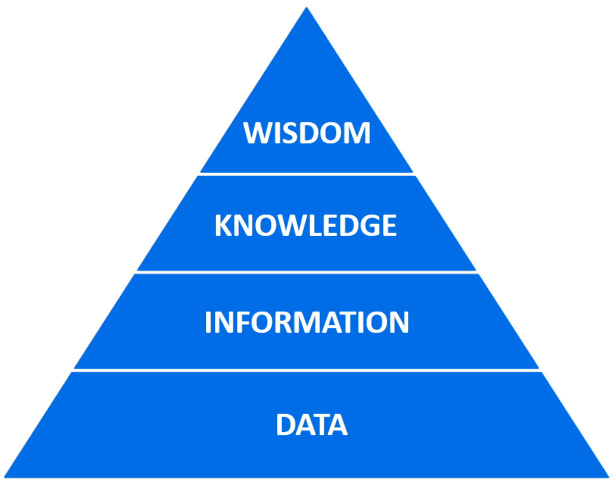


Figure 3.1: Knowledge pyramid

The desired system has to perform information extraction from different data sources. As such, one of the main components in this solution is a set of tools to extract the desired data.

Information Extraction (IE) is a term that has come to be applied to the activity of automatically extracting configurable sorts of information from an input text (Gaizauskas and Wilks 1998).

The techniques used for information extraction are dependent on the type of data source. As previously identified, the majority of timetable information is available at the providers' website as an HTML document or as downloadable PDF files. This creates the need to access those web pages and extract the available documents.

The process of extracting data from the web is typically referred to as web scraping and it is used in combination with the techniques of web crawling. In web crawling a set of web robots (also known spiders) traverse the web indexing new web pages and downloading meaningful documents (Mitchell 2018).

According to Gudivada et al. (1997) the web crawling process can be divided into three different approaches for web traversal:

- Providing a seed Uniform Resource Locator (URL) (also known as base URL), and find new URLs recursively in a breadth-first or depth-first fashion.
- Start with a set of URLs based on a Web site's popularity and searching recursively.
- Partitioning the Web space based on Internet names or country codes and assigning one or more robots to explore the space exhaustively.

The process of web crawling is better defined as a technique of Information retrieval (IR) (Gudivada et al. 1997; Inkpen 2007), rather than IE. Since the main objective of the spiders is only to collect documents, it is necessary to parse the collected documents to obtain the desired information.

Information Retrieval can be co-related to the knowledge pyramid 3.1 where the facts in the unstructured data and documents accessed by the spiders will be converted to information after the parsing process. This process will add context to the data and, most importantly, add usefulness to the facts.

Considering the possibilities that arise from making use of the web and its vastness, it should be no surprise that the topic of how to extract user-interested information automatically or semi-automatically has become a research topic from researchers worldwide (Wei-Guo et al. 2010).

The current information extraction approaches can be divided into the categories of wrapper based information extraction or concept model-based information extraction methods (Wei-Guo et al. 2010).

3.1.1 Wrapper based information extraction

A wrapper is a software artifact, designed to extract content from a particular information source. In a Web context a wrapper is a set of extraction rules suitable to extract information from a website (Flesca et al. 2004).

Wrapper generation can be accomplished manually or automatically (Nekvasil 2007). There is also the possibility of using a semi-automatic approach (Flesca et al. 2004).

When generating a wrapper manually, it is necessary to have a good knowledge in the domain of the information to extract and the structure of the document with that information. However there are some reasons why this is undesirable. Besides the high level of expertise required, the process of writing rules is tedious and time-consuming (Muslea, Minton, and Knoblock 1999). Another disadvantage of manually generated wrappers is the lack of ability to adapt to document format changes.

As an alternative, it is possible to generate wrappers in a semi-automatic or automatic fashion.

The automatic generation of wrappers is usually referred to as wrapper induction (Kushmerick, Weld, and Doorenbos 1997). This generation can be done by training machine learning models using sample web pages containing data similar to the desired. Another option less documented is to interpret a user query, transforming it to the required set of rules using them for extracting the information (Wei-Guo et al. 2010). The wrappers generated by this approach have the main disadvantage of performing worst when compared to hand-written wrappers (Flesca et al. 2004).

3.2 Ontology

The concept of Ontology stems from the philosophical field of metaphysics where it focuses on the nature of being as the topic of study. A good definition comes from Heidegger (1967) presented in his book Being and Time: "The task of ontology is to explain Being itself and to make the Being of entities stand out in full relief".

In the computer science field, ontologies take a broader view, being applied to a domain, and not only focusing on the Being of the entities but also the relationships between them. The concept Ontology from Philosophy is usually distinguish in writing, from the one used in the field of computer science/knowledge engineering, by being capitalized.

Although there is no formal consensus on the meaning of ontology, the most accepted definition in computer science communities is that "An ontology is a specification of a conceptualization" as described by Gruber (1993). As Gruber states, an ontology is a description of the concepts and relationships that can exist for an agent or a community of agents.

As stated by Chandrasekaran, Josephson, and Benjamins (1999), the importance of analyzing the ontologies of a given system's domain is that an ontology allows the clarification of the structure represented by the knowledge existent in that system. "Given a domain, its ontology forms the heart of any system of knowledge representation for that domain. Without ontologies, or the conceptualizations that underlie knowledge, there cannot be a vocabulary for representing knowledge" (Chandrasekaran, Josephson, and Benjamins 1999).

In a practical sense, a solution's metadata is a part of that system's ontology since it represents its data structure, therefore specifying a narrow view of the concepts and relationships present in that system's domain.

Another difference from the characteristics of an ontology in computer science when compared to Philosophy is that the ontology used to define a given entity, changes depending on the aspects of reality selected to be represented in the encoding process (Chandrasekaran, Josephson, and Benjamins 1999).

The objective of an ontology in computer science, is not to define the concepts as they are in reality, but to define those concepts under the lens of a specific context. For example, in the domain of vehicles, we would focus on particular aspects of reality if we were developing the ontology for choosing an appropriate route, and focus on different aspects when developing an ontology for an insurance company.

Therefore when creating an ontology it is important to not only understand the concepts to be depicted but also take into account the desired context and the important characteristics of the domain's concepts under in that scope.

Authors have classified ontology's on their levels of generality and the purpose of that ontology. A categorization of the different types of ontologies with a succinct definition is given by Staab and Studer (2010), where the following categories are identified (cf. Guarino 1997; Stephan, Pascal, and Andreas 2007; Van Heijst, Schreiber, and Wielinga 1997):

- Top-level ontologies or foundational ontologies, capture general concepts that are domain-independent. This type of ontology specifies the conceptualization of commonsense knowledge such as space or time.
- Domain ontologies specify concepts and relations that are relevant for a specific domain.
- Task ontologies describe concepts that are specific for a task or activity.
- Application ontologies have the lowest level of abstraction and combine domain and task ontologies, extending them with more refined domain and task specific concepts and relations. The concepts specified must have the necessary detail to achieve the requirements of the application that will use the ontology.

As mentioned some concepts specified in lower level ontologies can be a specialization of a concept specified in an ontology of a higher level. As an example the concept. This hierarchical behavior is presented in figure 3.2 created by Guarino (1998) where the arrows represent specialization relationships .

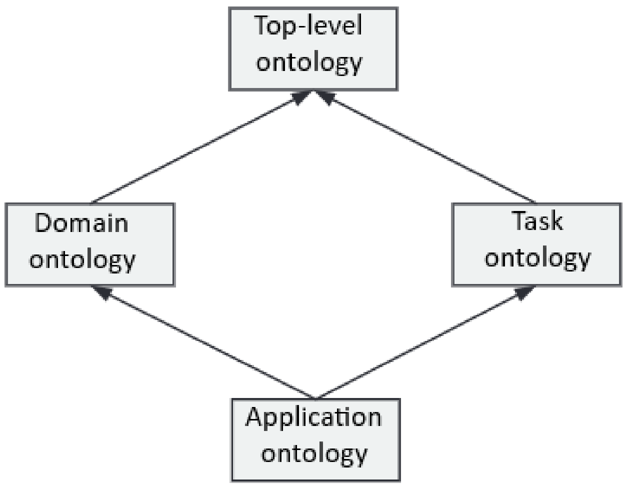


Figure 3.2: Ontology types hierarchy

By applying ontologies in the solution of the timetable problems it is possible to specify some of the complex business rules. These complex rules are presented on the website, alongside the timetable information, as an observation text in Natural Language. This rules can include restrictions on the schedules such as "The schedules are not in service on holidays except if it is a Sunday. If the holiday is on Mondays, they do not take place on the previous Sunday, thus moving to that day." [1]

3.2.1 Concept model-based information extraction

An alternative to using wrappers for information retrieval is to extract the information using concept models.

This type of information extraction is preferred to use for unstructured text and to apply in Natural Language Processing (NLP). The model is typically defined by the domain lexical knowledge, extraction rules, and an ontology (Maedche, Neumann, and Staab 2003).

The result is usually information of concepts defined on the ontology, and optionally an updated version ontology with some new concepts mined in the text analysis process (Wei-Guo et al. 2010).

3.3 Data integration and information integration

Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of these data (Lenzerini 2002).

[1]retrieved and translated from `http://www.avminho.pt/horarios`

According to White (2005) the techniques existent for data integration in the enterprise environment can primarily be divided in:

- Data consolidation, a technique where several data sources are converted or consolidated into a single database. This process is usually executed using an Extract Transform Load (ETL) tool.
- Data federation, a technique where the data sources are displayed using a virtual view. The data is not locally saved being fetched from the data source when the visualization is required. It is adequate to use in highly volatile data, or when the data is difficult to consolidate.
- Data propagation, appropriate for when the enterprise has control of the data sources, using event-driven propagation allowing to integrate distributed database using asynchronous communication or by using a shared database / shared files.

Considering that information is just processed data with added usefulness, the same techniques can be used for information integration as long as such usefulness is not lost in the process.

For an ontology-driven system, since the information/knowledge is structured by a specified format represented as an artifact and a set of rules, it is good practice to use ontology integration and transformation to handle metadata changes and necessary conversions.

3.4 Integration and transformation of ontologies

After Berners-Lee, Hendler, and Lassila (2001) presented at the beginning of the twenty-first century their idea for the semantic web or web 3.0, a World Wide Web where the information would be categorized and easily understandable by software as a result of web pages enriched with embedded semantics in machine-friendly formats, a large number of ontologies surged in a wide variety of field, generating new challenges and opportunities including the integration of different ontologies and knowledge bases.

The topic of ontology integration is vastly studied, giving origin to several definitions with different meanings in the ontology engineering field. Some authors have differentiated the several types of transformation of ontologies (cf. Choi, Song, and Han 2006; N. F. Noy 2004; Pinto, Gomez-Perez, and Martins 1999).

3.4.1 Ontology integration

The ontology integration process consists of the construction of an ontology by reusing a part, or the integrity, of other ontologies. The source ontologies can have different domains that are reused by composing them into a new ontology.

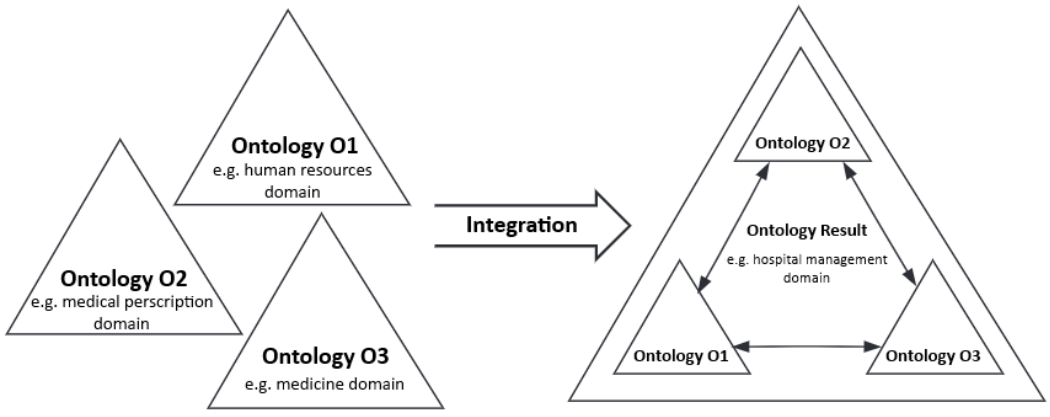


Figure 3.3: Ontology integration

In exceptional cases a whole ontology can be built just from assembling other ontologies, however, in the majority of cases of ontology integration, the new ontology has to specialize the concepts provided by the source ontologies. As an example, this process can be used to create application ontologies from domain and task ontologies.

3.4.2 Ontology merging

In ontology merging, two or more ontologies of the same knowledge domains are integrated, creating a new ontology that represents the unification of these ontologies. The ontology merging process differentiates from the process of ontology integration by unifying the knowledge of several ontologies of the same domain in to a single one.

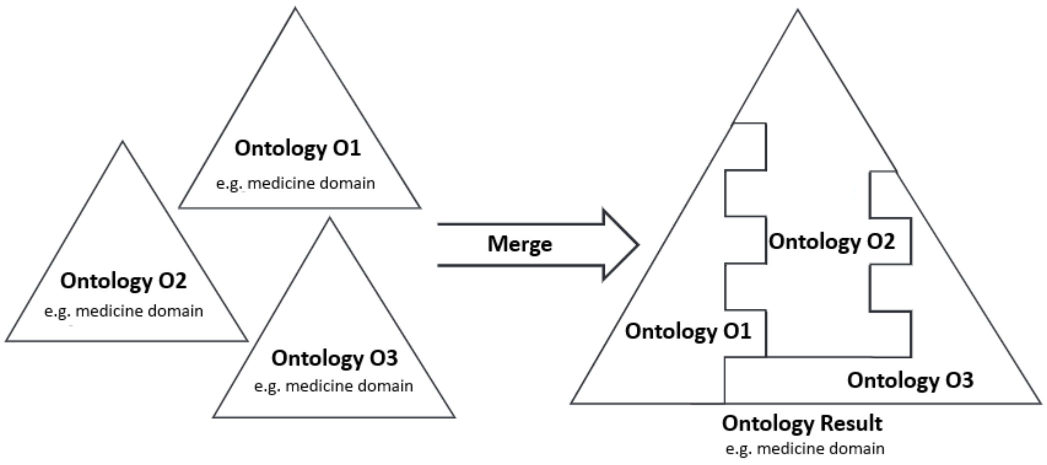


Figure 3.4: Ontology merging

3.4.3 Ontology mapping

Since ontologies are only a perception of reality, being a context-dependent projection, different ontologies can be used to represent the same knowledge.

Ontology mapping is the process whereby two ontologies are semantically related at the conceptual level and the source ontology instances are transformed into target ontology entities according to those semantic relations (Silva and Rocha 2003).

This is achieved by defining, either manually or automatically, the rules that allow relating two or more ontologies. This task is usually referred to as ontology alignment, a process that achieves consistency between the ontologies while keeping them separate. This is achieved by creating links between the original ontologies.

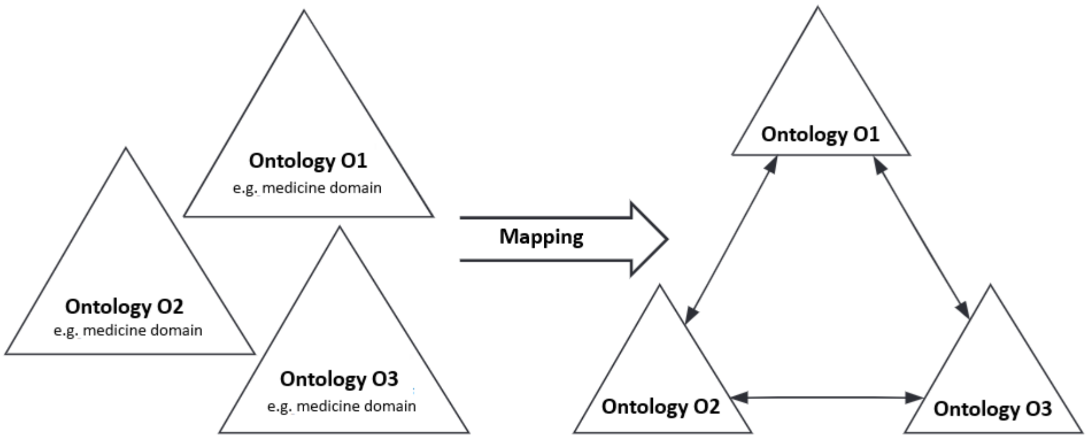


Figure 3.5: Ontology mapping

An advantage of using this type of integration, when compared with the ones previously mentioned, is that by allowing the preservation of the heterogeneity of the different specifications of the same domain (Santos 2008).

Some case studies in ontology mapping propose the creation of semantic bridges between ontologies allowing the conversion of data to different data structures without damaging the original information (Silva 2004).

3.5 Technologies

Knowing the approaches to solve the problem is not enough. To apply them, it is important to find, study and select relevant technologies and tools to use in the solution's development.

3.5.1 Web crawling and information retrieval

Starting with the technologies found for web crawling and information retrieval, the majority of tools found were built for development in the python programming language. This preference for the python programming language could be attributed to it being prepackaged with a handful of general-purpose libraries and a well-established library environment (e.g. NumPy library) for data science processes (Oliphant 2007), which usually require data sets that can be obtained using information retrieval.

For the task of web crawling and information retrieval, the developer has some options when it comes to the selection of the technologies to use within the python library environment.

3.5.2 Beautiful Soup

One approach for web crawling and information retrieval is to use two separate libraries, one responsible for the HTTP requests handling and a different library for the parsing of the collected documents.

When using this approach, the most common technological selection is to use a simple request handling library (e.g. python-requests [2]) and Beautiful Soup, a python library designed to extract data from HTML and XML files. Its approach to parsing is to format the HTML by fixing messy sections and simplifying it by presenting it as a traversable python object which represents the original XML (Mitchell 2018).

The spiders created using this approach work as python scripts. This allows them to be run in a stand-alone manner using a standard command line.

3.5.3 Scrapy

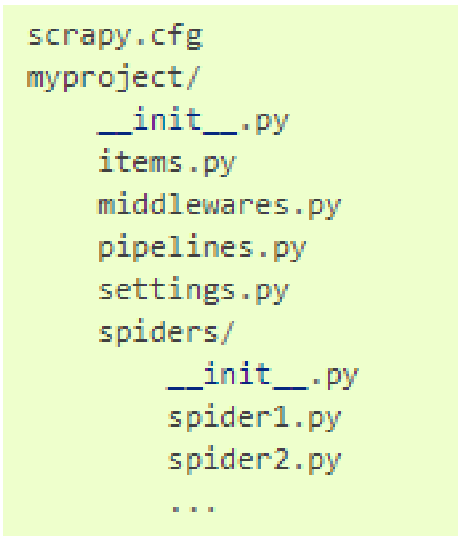
A different approach is to use a full-fledged framework designed for the task of web crawling/web scraping. Scrapy is an open source and collaborative framework for extracting the data you need from websites. It encourages the usage of good programming practices, such as code re-usage by using a specified project structure (3.6) that supports the addition of middleware code.

In the "items.py" the developer adds the Items, objects representing the structure of the scraping output, to be returned after the extraction of information by the spiders. It is possible to define different pipelines, that will be used depending on the type of the Item returned.

As previously mentioned, it is possible to add middleware code, that is executed when using the downloading pipeline and the spider processing pipeline. This allows modifying a request and response, for example, to add/remove headers to every request before sending.[3]

[2]Detailed information and documentation can be found at `https://github.com/psf/requests`

[3]More information on the Scrapy architecture can be found at `https://docs.scrapy.org/en/latest/topics/architecture.html`



```
scrapy.cfg
myproject/
    __init__.py
    items.py
    middlewares.py
    pipelines.py
    settings.py
    spiders/
        __init__.py
        spider1.py
        spider2.py
        ...
```

Figure 3.6: Standard Scrapy project directory structure

The main advantage of using Scrapy is that it eases the process of web crawling by handling URL collecting/extraction, evaluating the identified URLs as external or internal and selecting the next web page to scrape (Mitchell 2018).

When writing extraction rules, the Scrapy frameworks uses a python library named parsel[4] that allows to define Selectors specified either by XPath or CSS expressions.

Regarding deployment, the spiders created using Scrapy can be run in a single or group fashion. There are platforms specialized for deploying Scrapy projects in the cloud[5], allowing the automation/scheduling of crawling actions and providing options to visualize and save the extracted data and links.

3.5.4 Regex

Regex or regular expressions can be used alongside both approaches and it is key to a powerful, flexible and efficient text processing. Regular expressions present a set of general pattern notation[6], allowing to describe and parse text using a pattern matching technique (Friedl 2006).

Regular expressions are supported in different programming languages, including the python standard library re [7] , making its usage a good alternative for defining extraction rules.

3.5.5 XPath

XPath is one of the possible ways to access specific parts of a DOM tree when using Scrapy. In contrast with using CSS expressions, XPath allows the manipulations of Node Axes,

[4]Parsel documentation available at: `https://parsel.readthedocs.io/en/latest/`

[5]Scrapy cloud service available at `https://scrapinghub.com/scrapy-cloud`

[6]A regular expressions' cheat sheet can be found at `https://regexr.com` or in Friedl 2006

[7]Detailed information and documentation can be found at `https://docs.python.org/3/library/re.html`

making it easy to navigate the Tree and easily accessing the relationships of the nodes (e.g. children, parent, siblings).

More specifically, XPath is query language specialized for XML documents, allowing it to address parts of those documents, while also providing basic facilities for manipulation of strings, numbers, and booleans. It works by using a compact, non-XML syntax to facilitate allowing to navigating through the hierarchical structure, which is similar to the path notation used in URLs (Clark, DeRose, et al. 1999).

3.5.6 Portia

Portia is a tool that allows to visually scrape websites. By using Portia is possible to annotate a web page to identify the data to extract, and with the knowledge of those annotations, Portia will attempt to scrape data from similar pages.

It provides a user interface with an embedded browser, and it allows the manual selection of the type of HTML fields to be scraped by click. By default, Portia crawls by following all in-domain URLs, but it works on a basis of configuration, where constraints can be specified.

Portia can be define as a wrapper induction tool, where the output of a crawling and a scraping configuration is an executable spider which uses the Scrapy framework.

3.5.7 Semantic web stack

The Semantic Web was created to extend the potency of the web from merely sharing documents via HTML to sharing data by enabling links as Uniform Resource Identificator (URI). This allows linked data to be shared effectively by wider communities and provides an opportunity for that data to be processed automatically by robotic tools (Hoyland et al. 2014). Presented in the figure 3.7 is the architectural and technological stack for the semantic web, provided and maintained by the World Wide Web Consortium (W3C), based in the original definition described by Berners-Lee (2000) and retrieved from Hoyland et al. (2014).

Understanding the group of technologies and concepts presented in the Semantic Web architecture is key to develop an ontology-based solution to a web-driven problem. The technologies and concepts presented are considered the standard for solving such problems and promise to work as the foundation of the idea of the semantic web.

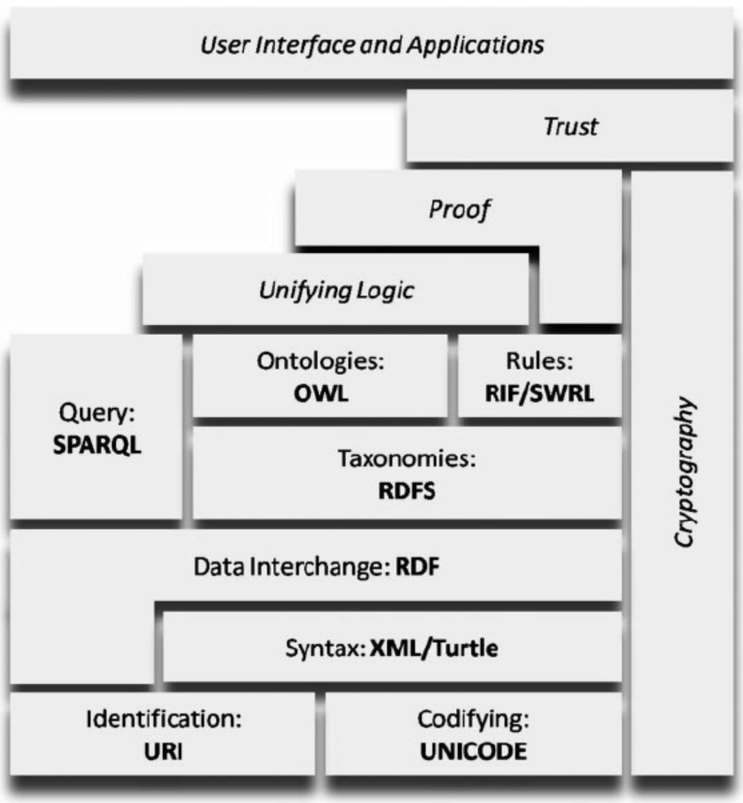


Figure 3.7: Semantic Web architecture stack

The first layer consists of URIs and the Unicode character set. These tools are present on the web most users currently use, where it is used to represent resources and text. Unicode is the standard for encoding character sets worldwide. It allows that all human languages and most symbols (from emoticons to Egyptian hieroglyphs) can be used on the web using one standardized form. A URI is simply a string of characters used to identify a name or resources on the internet.

The syntax layer consists of technologies to encode and distribute documents over the web, where XML or Turtle are the viable options. XML is the standard for encoding documents and for defining a syntax for knowledge encoding within the Semantic Web. In the context of the Semantic Web, Turtle has the same purpose as XML allowing it to textually represent an RDF Graph.

RDF is the standard model for data interchange on the Semantic Web. RDF is designed to represent information in a minimally constraining and flexible way. It was originally designed for describing Web resources such as Web pages. However, RDF resources may be physical objects, abstract concepts, in fact anything that has identity. Thus, RDF defines a language for describing just about anything (Brian McBride 2004).

It can be used in isolated applications, usually for simplifying the definition of the format of resources, but RDF's generality offers greater value for sharing data (Klyne, Carroll, and B McBride 2004).

As an evolution of RDF, RDFS or RDF Schema [8] can be used in the Taxonomy layer. RDF

[8]RDFS properties can be found at `https://www.w3.org/TR/rdf-schema`

Schema adds a mechanism to describe taxonomies of classes and properties. It distinguishes between a class and that class' instances (Brickley, Guha, and Brian McBride 2014).

The W3C OWL is a Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things. As mentioned by McGuinness, Van Harmelen, et al. (2004) the OWL is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema by providing additional vocabulary along with a formal semantics.

The ontologies created with OWL are used to define knowledge bases which contain the following parts :

- Abox - Assertion axioms (e.g., instances)
- Tbox - Terminology axioms (e.g., classes)
- Rbox - Role hierarchy axioms (e.g., property chains)

In the query layer, SPARQL allows querying RDF. A SPARQL query can also be executed on any database that can be viewed as RDF via middleware. For example, a relational database can be queried with SPARQL by using a Relational Database to RDF mapping software. However, SPARQL is more than just a simple query language. It is also an HTTP-based transport protocol, where any SPARQL endpoint can be accessed via a standardized transport layer. This allows the making of queries that can access multiple data stores[9].

On the rules layer of the architecture, the technologies defined to support the semantic web are RIF (Rule Interchange Format) and Semantic Web Rule Language (SWRL) . SWRL is a rule language which was designed as an extension to OWL and Rule Interchange Format (RIF) was designed as an interchange format for exchanging rules between rule systems, such as those that implement SWRL. An OWL ontology, in the abstract syntax, contains a sequence of axioms and facts where axioms may be of various kinds, e.g., "subClass" axioms or "equivalentClass" axioms. The main purpose of SWRL is to extend this with rule axioms using the syntax defined in the generic formula presented in the SWRL Rule 3.1, where p is a predicate symbol and var1, . . . varn are the arguments of the expression (Braga 2013).

$$p(?var1, ...varn) \wedge ...p(?var1, ...varn) \rightarrow p(?var1, ...varn) \wedge p(?var1, ...varn)$$

Rule 3.1: SWRL generic rule

It provides simple function to manipulate data properties such as "swrlb: multiply" or "swrlb: stringConcat" and allows to handle the knowledge using first order logic with the objective of asserting new individuals or their properties. To improve the compression of this document and add separation between the logic rules created and the technologies used, first order logic notation is used for representing SWRL rules. A concrete simple example of a SWRL rule would be to assert that the combination of the hasParent and hasBrother properties implies the hasUncle property (Horrocks et al. 2004) as showned in swrl rule 3.2, now presented with first order logic notation.

[9]The fundamentals of SPARQL queries can be found at `https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/`

$$\forall x, y, z : \; hasParent(x, y) \land hasBrother(y, z) \rightarrow hasUncle(x, z) \land isUncle(z, True)$$

Rule 3.2: Rule for hasUncle

Since OWL operates on an open world assumption, the definition of rules and axioms are important to assert the ontologies consistency, since inconsistencies can be detected by using a reasoner.

Cryptography, Unifying Logic, Trust and Proof layers are just the concepts to work as foundation for the semantic web reliability, accuracy and trustworthiness (Braga 2013).

3.6 PDF Extraction

As previously demonstrated in chapter 1.3, the majority of operators use PDF files to represent their timetables.

A solution to address the extraction of information from timetable would not be complete without proving a solution to that representational format.

Developing a solution or technique to handle PDF extraction would be a time consuming task. An easier alternative is to use a open source software program, and tailor it to the domain of public transportation and transport timetables.

Various approaches have been put forward to solve this issue. If the focus is shift at the extraction of tabulated data, the most relevant solutions are Tabula (Aristarán et al. 2018) and Camelot (Mehta 2020). Both programs are executable via a terminal, but also provide a web application that allows support to the PDF extraction.

A comparison between both application is presented in table 3.1

Table 3.1: Comparison between Camelot and Tabula

	Tabula	Camelot / Excalibur
Has a Graphic User Interface	Yes	Yes (named Excalibur)
Programming Language	Java, JRuby	Python, Javascript
License	MIT	MIT
Has an API to receive external PDFs	No	No
Requires additional software	No	Yes (Ghostscript)
Extraction algorithm	Lattice and stream	Lattice and stream
Last version release date	4 June 2018 (v1.2.1 release)	17 July 2020 (v0.4.3 for excalibur)

Source code url	`https://github.com/tabulapdf/tabula`	`https://github.com/camelot-dev/excalibur`

Both programs use two separate extraction methods, lattice and stream. In lattice the algorithm identifies the lines that form the table structure. In stream the spacing between the text is analysed to form the column and rows of the table (Aristarán et al. 2018) (Mehta 2020). Stream is more versatile than lattice since it can be executed in tables that contain lines and table which do not[10].

As a side note, there is also a simple Python wrapper implementation of Tabula, however it does not work as a Graphic User Interface, but as a python library only offering support to the tabula core.

[10]A comparison between the algorithms of both solutions can be found at `https://github.com/camelot-dev/camelot/wiki/Comparison-with-other-PDF-Table-Extraction-libraries-and-tools`

Chapter 4

Selection of technologies and solutions

In this chapter, the information retrieval and information integration technologies and approaches identified in the State of the art are compared and selected.

4.1 Selection for information retrieval

Starting with information retrieval, the approaches identified were concept model-based extraction and wrapper based extraction. The first one is mostly used for Natural Language Processing and, as such, is not a good alternative for retrieval where the text is scattered in several web pages. The alternative is using wrappers, which can be hand-written or created by using wrapper induction. The technologies identified for creating hand-written wrappers were using Beautiful Soup or Scrapy, and Portia for wrapper induction. To ensure the value of this solution, it is necessary to guarantee the output information's quality and credibility. Since wrappers generated by induction perform worst compared to hand-written wrappers, Portia is less viable than the other options for this specific problem.

The choice between Scrappy and Beautiful Soup depends on the complexity of the crawling and scraping problem. Considering the number of websites to extract and the diversity of the documents and information, a Scrapy project will help the code maintainability, extensibility, and reusability. Since the wrappers created using either alternative use the same rule and query technologies (XPath, CSS selectors, Regex), a wrapper created using Beautiful Soup can be implemented in Scrapy and have an extraction with the same accuracy. Differences can occur in metrics like execution time, but considering the requirements of the solution, those have a low to none impact in the selection and are highly dependent on request and response times and, therefore, it is dependent on the network connection's velocity and stability at the time of execution.

Nevertheless, considering that both approaches are viable, two different prototypes can be created and evaluated using metrics relevant to the execution environment, such as memory usage, while also considering the difficulty of creating wrappers in each technology with effort estimation.

Although concept model-based extraction cannot be applied in this solution, ontologies can and should be used. The proposed approach is to extract the information using either Beautiful Soup or Scrapy, downloading any PDF files found and storing the information in a temporary format that can be processed to dynamically generate the ontology's individual-s/instances.

Regarding the extraction of PDF data/information, since the objective is to create a prototype tool capable of extracting the tabulated data while at the same time adding information related to the transportation business, the program selected as a codebase must be simple and fast to modify.

The tailored version will only be usable on a subset of the operators. As such, the PDF extractor should work as a proof of concept for this type of representation formats.

With this in mind, the Camelot / Excalibur was selected as it offers the advantages of having the technological stack as the other components and having a simpler architecture, making it simpler and faster to modify. In contrast, Tabula lacks documentation, and changes to the code base can be time-consuming, as mentioned in recent criticisms on Tabula, which are summarized in Rosén (2019).

4.2 Selection for data and information integration

From the types of data integration identified (data consolidation, data federation, and data propagation), the proposed approach to solve the problem at hand uses data consolidation combined with techniques used in data federation. Considering there is interest to preserve versions of the information, data federation can be removed from the viable approaches, since it only creates a virtual view without persisting the data.

The proposed solution would be using the event-driven part of data propagation to initiate a data consolidation process. However, the event-driven technique used in data propagation is designed to be used when the developer has control over the data source and can fire the events on updates. In this process, the push approach of firing events would have to be replaced by a periodic pull operation, where the system verifies if updates have occurred on the external data sources (providers websites).

The tasks of data consolidation presented in the ETL (extract, transform, load) tools would be equivalent to extracting using wrappers, transforming using ontologies defined in OWL / RDFS and ontology mapping, and the loading is equivalent to storing the python object-s/ontologies in a database, RDF store or data warehouse.

Chapter 5

Analysis and Design

This chapter of the document contains the analysis of the solution's requirements and the descriptions of the solution's design alternatives.

5.1 System requirements analysis

With the public transportation operators selected, the websites to search for the information are already defined. It is then necessary to design a system that can extract that data and transform it into information understood by the system. The use case diagram presented in figure 5.1 represents two actors, where each one executes a different set of use cases. The first actor is the system itself, responsible for the scraping, crawling, parsing, and applying data mappings. The other is the end-user, which requires information for its end system and wants to consult the information extracted from the websites and export it using standard formats.

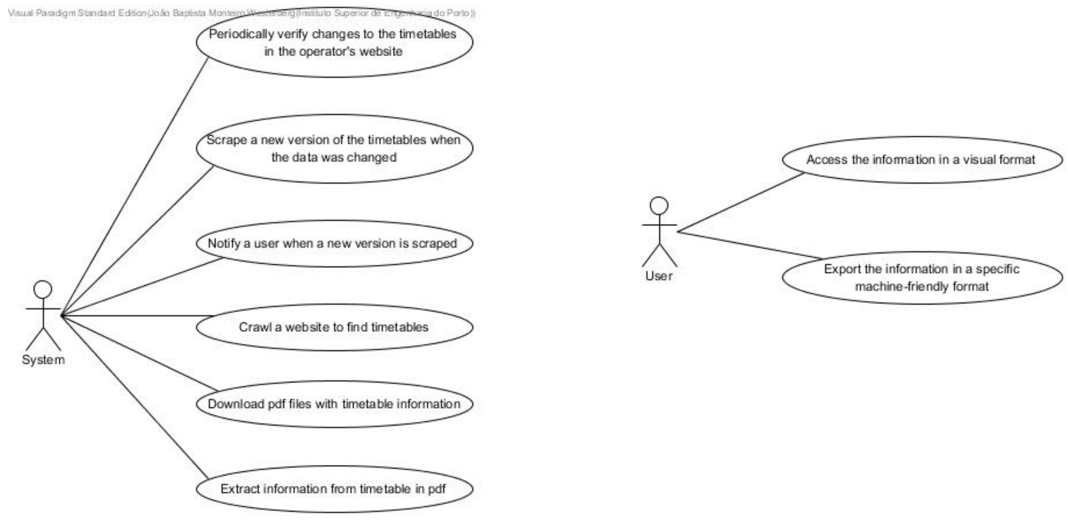

Periodically verify changes to the timetables in the operator's website
Scrape a new version of the timetables when the data was changed
Notify a user when a new version is scraped
Crawl a website to find timetables
Download pdf files with timetable information
Extract information from timetable in pdf
System
Access the information in a visual format
Export the information in a specific machine-friendly format
User

Figure 5.1: System's Use Cases

The system's operation can be described as a single pipeline, where each execution will integrate the information contained in the websites. The operations in this pipeline follow the logic represented in the activity diagram represented in the figure 5.2, and are initiated by a scheduled task that verifies if the timetables were updated.

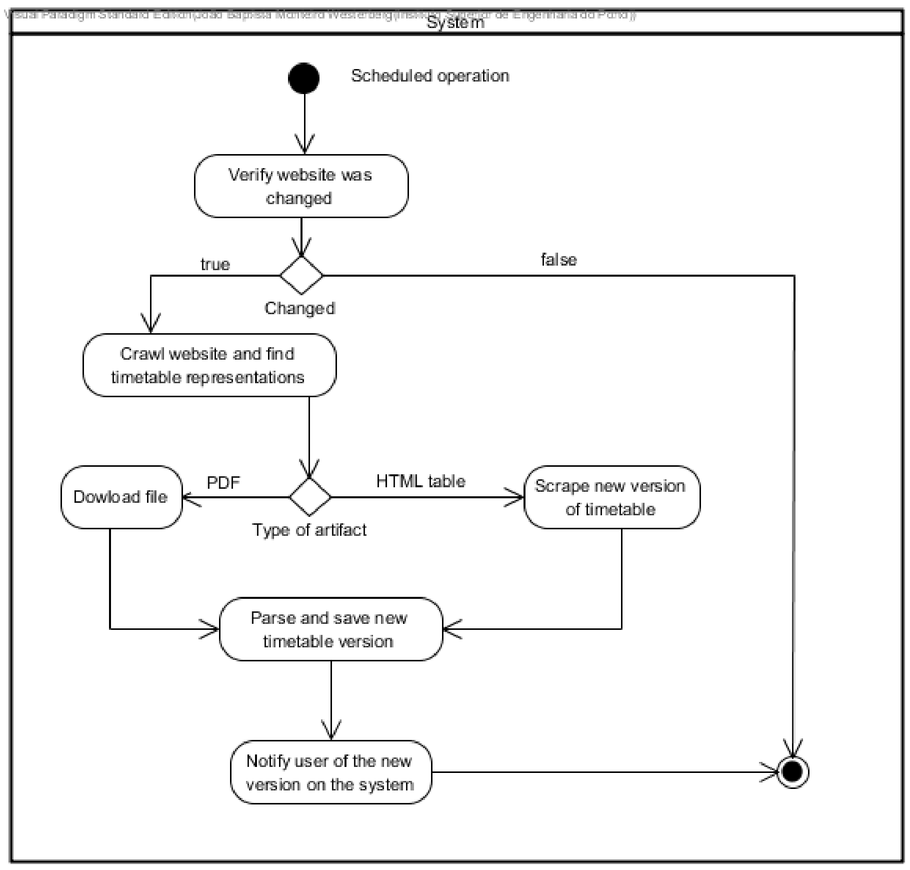

Visual Paradigm Standard Edition(João Baptista Monteiro Westerberg(Instituto Superior de Engenharia do Porto))
System
Scheduled operation
Verify website was changed
true
false
Changed
Crawl website and find timetable representations
Dowload file
PDF
HTML table
Scrape new version of timetable
Type of artifact
Parse and save new timetable version
Notify user of the new version on the system

Figure 5.2: Activity diagram for high-level view of action in the integration process

5.2 High-Level design alternatives

Before starting to declare and organize the system's components, it is essential to understand and define the term spider. Spider is just a different name for web crawlers, small software artifacts to search and crawl the web. As previously mentioned, there are different approaches when using developing a project with manually created spiders to crawl web pages. The first one is to create individual spiders; each is self-contain or standalone and works in a script-like fashion. Their purpose is to access a website and crawl it while recovering any crucial data. For this first approach, an example of a component diagram with a coarse-grained can be found in figure 5.3.

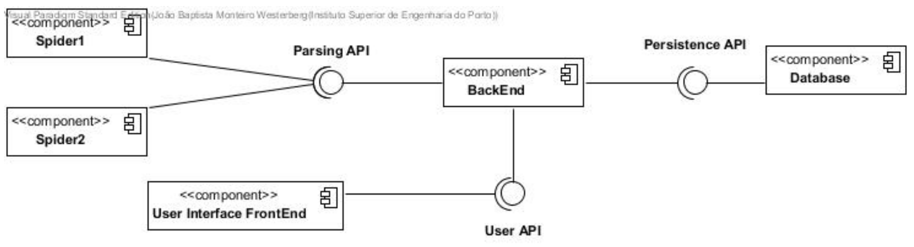

Visual Paradigm Standard Edition(João Baptista Monteiro Westerberg(Instituto Superior de Engenharia do Porto))
<<component>> Spider1
<<component>> Spider2
Parsing API
<<component>> BackEnd
Persistence API
<<component>> Database
<<component>> User Interface FrontEnd
User API

Figure 5.3: High-level, coarse-grained component diagram for multiple spiders

Besides the spiders, the diagram also suggests a possibility to the components that give a response to the user-related requirements using a client-server architecture. This approach structures the components using patterns to enforce that the database is only accessible by the "BackEnd" component. This approach results in a higher cohesion and the reduction of the coupling between the database and the other components.

The previous diagram only shows two spiders; however, the number of spiders required is a lot higher in the problem's solution. The solution requires different types of spider, such as :

- Spiders to verify if the website was changed, by verifying values like the last updated.
- Spiders to locate tables and scrape the information.
- Spiders to download the relevant PDFs.

.

It is vital to consider that some providers only change the timetable information seasonally, for instance, semesterly or per trimester and that the scheduling of the spiders can be variable. Having a set of spiders to verify if the website was changed allows reducing the number of times where the scraping is required, reducing bandwidth usage, improving the speed of the operation, and the spiders' politeness. An example of this process's sequence of actions can be found in the diagram in figure 5.4.

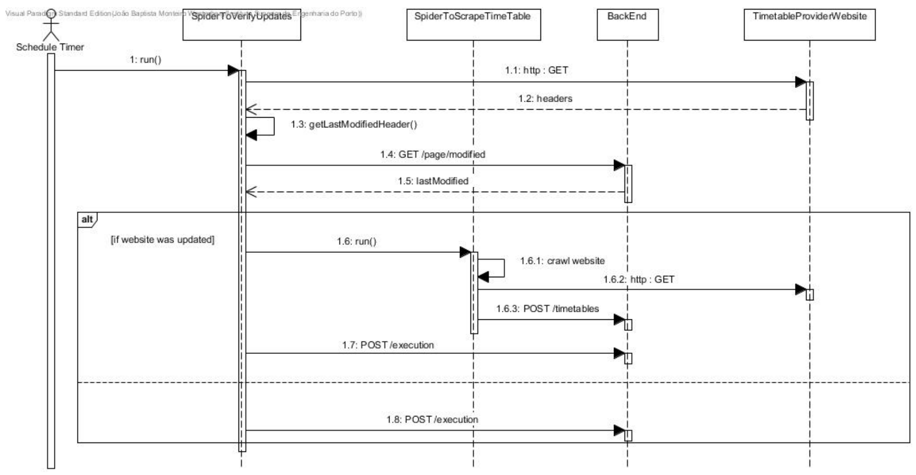


Figure 5.4: Example of a sequence diagram using different types of spiders

This high number of spiders reinforces the necessity to use a client-server architecture and exposes some issues with the first alternative. The most noticeable is that there is no code reusage from spider to spider. The lack of code reusage introduces complications in the maintenance process, where for example, a change to the Parsing API location would require changes to every spider due to code duplication. As beautifully put by Hunt and Thomas (1999), "every piece of knowledge must have a single, unambiguous, authoritative representation within a system".

A possible solution to resolve this is to create a single project containing every spider, removing the self-contained property by introducing dependency with shared code. A scraping framework can be used to achieve this, further enforcing the usage of good programming practices and design patterns. Having a single project to work allows us to accelerate the process of creating a new spider and allows the simplification of communication with other components of the solution.

Another possible optimization is to remove the need for the "SpidersToVerifyUpdates" (figure 5.4) to communicate with the "BackEnd" component to inform the results of execution and instead replace it for a caching database (e.g., Redis[1]). An alternative that combines the commons component for the spiders and a Caching Database is represented in the diagram in the figure 5.5.

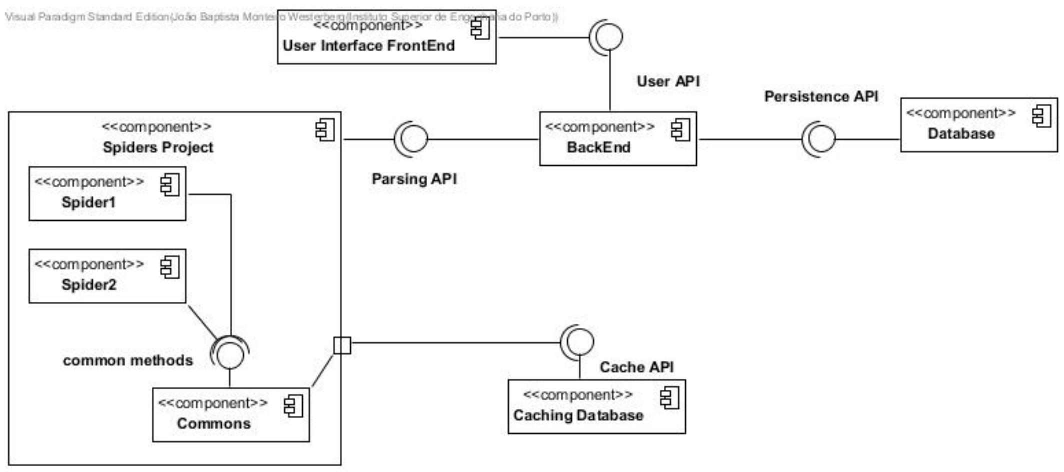

Visual Paradigm Standard Edition(João Baptista Monteiro Westerberg(Instituto Superior de Engenharia do Porto))
<<component>> User Interface FrontEnd
User API
Persistence API
<<component>> Spiders Project
<<component>> BackEnd
<<component>> Database
<<component>> Spider1
Parsing API
<<component>> Spider2
common methods
<<component>> Commons
Cache API
<<component>> Caching Database

Figure 5.5: High-level component diagram for alternative using a spider project and a caching database

5.3 Event driven communication

So far, the design alternatives present an issue when it comes to defining the order of execution between spiders. As an example, the process where the execution of a "SpiderToScrapeTimetable" is explicitly triggered when "SpiderToVerifyUpdates" identifies an update, introduces dependencies that can be difficult manage with the growth of the number of spiders.

As a response to this issue, an event-driven approach implementing a publisher/subscriber pattern using a message bus was designed in the diagram represented in the figure 5.6. When using a publisher/subscriber pattern, the responsibility of continuing the flow of execution becomes independent from the publisher becoming decoupled in both time and space from the subscribers (Eugster et al. 2003). As a result, the open/closed principle is applied, and an addition of a new spider to the system does not require to change previously created spiders to add logic for the sequence of executions.

[1]More information about Redis and documentation can be found at `https://redis.io/documentation`

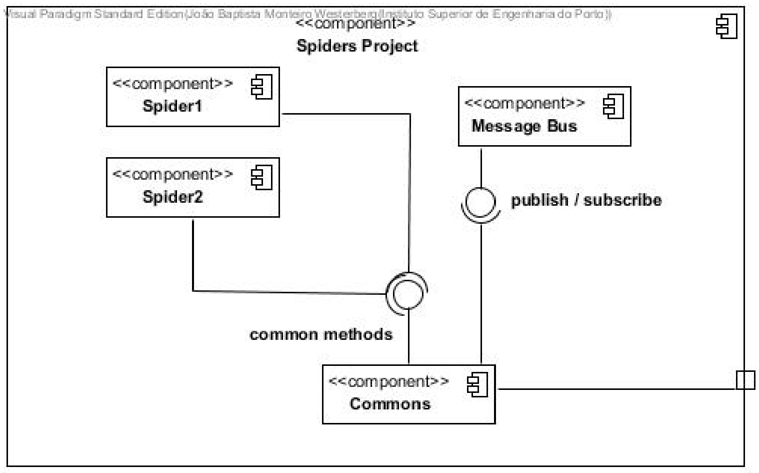


Figure 5.6: Component diagram for alternative using a message bus in the spider project

This event-driven approach can be taken to an extreme by designing an alternative with a messaging approach for asynchronous communication.

The alternatives presented so far use a synchronous technique for communicating between the system components. Some operations, such as the spiders' requests to the webserver containing the pages, need to be executed synchronously. However, the communication between the spiders and the Backend can be done asynchronously.

A way to achieve this is by replacing the message bus internal to the spiders' project for an external message broker. This can not only reduce the dependencies that result from the spiders' scheduling but also remove the need to design an API for parsing. By introducing a message broker, it is possible to divide the "BackEnd" component in different services, allowing for flexibility in the technology selection, increasing the components' cohesion, and reducing their coupling.

For example, a service responsible for handling the execution of ontology mapping can be developed in a language with better support for the Web Ontology Language (OWL) and a different language for creating the user's service to export timetables. By dividing the server-side into several components, it is also introduced some modularity to the system. The dependency between these two functionalities becomes weakened, and the service to export timetables can unknowingly receive data from different publishers. This applies from a high-level perspective, the open/closed principle, and the single responsibility principle.

Considering the nature of the domain and the application's functionalities, a microservice approach using decomposition by sub-domain would not present benefits, since the user operations require the information to be accessed as a whole. However, a decomposition by business capabilities is a viable approach, resulting in something similar to the architecture represented in the figure 5.7 and in the table 5.1

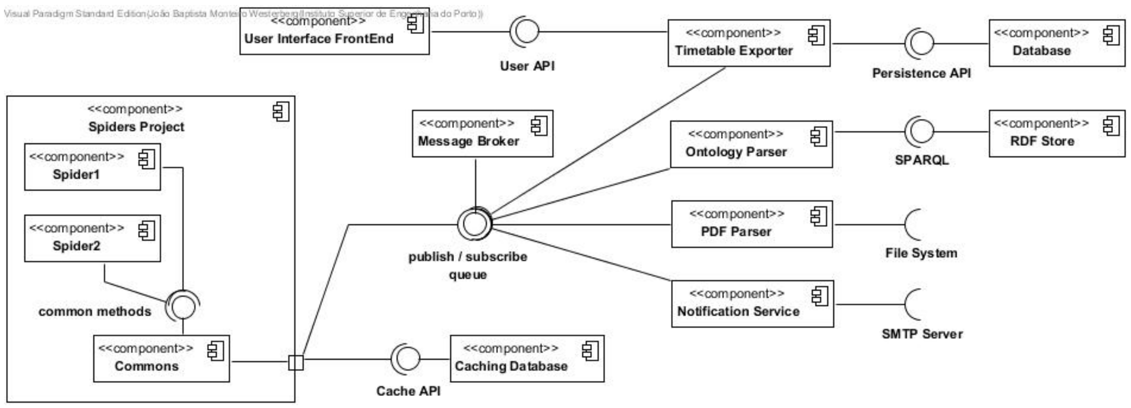

Visual Paradigm Standard Edition(João Baptista Monteiro Westerberg(Instituto Superior de Engenharia do Porto))
<<component>> User Interface FrontEnd
User API
<<component>> Timetable Exporter
Persistence API
<<component>> Database
<<component>> Spiders Project
<<component>> Spider1
<<component>> Spider2
common methods
<<component>> Commons
<<component>> Message Broker
publish / subscribe queue
<<component>> Ontology Parser
SPARQL
<<component>> RDF Store
<<component>> PDF Parser
File System
<<component>> Notification Service
SMTP Server
<<component>> Caching Database
Cache API

Figure 5.7: High-level component diagram for alternative using a message broker and multiple Backend services

Table 5.1: Components for an alternative using a message broker and multiple Backend services and description

Component Name	Brief Component Description
Ontology Parser	Transforms the events received from the spiders about the scrapings into ontologies representing the information in each export format
RDF Store	Option database if ontology persistence is beneficial. Can be used to keep a history of the several versions of the timetables
PDF parser	Performs PDF mining from tables with embedded text. Uses the Operating system (OS) File System.
Notification Service	A service to notify the users. Depending on the preferred channel of notification can use different external APIs. In the representation, it uses an Simple Mail Transfer Protocol (SMTP) Server for email notification as an example.
Timetable Exporter	Allows the user to view the timetables extracted and export the information according to several formats

The main advantage of using a microservice approach is the improved scalability/elasticity of the system. These characteristics are essential to high demand systems where its reliability is essential and may be beneficial to achieve, even at the price of losing performance per instance due to an increase in network bandwidth overhead. The main disadvantage of this alternative results from the fact that using a message broker introduces a single point of failure to the application and the possibility of message loss. This issue can be resolved by scaling the application by deploying several instances of the message broker or applying reliability patterns (e.g., sending an acknowledgment of message received).

5.4 Ontology design

To achieve the system's primary objective, the information extracted could be represented as an ontology referent to each data structure, allowing the extraction of information from several sources and conversion into different exportable data structures. Although the use of an ontology is more complicated than a typical ETL approach, this could be useful considering the complexity of the data. Besides the timetable information, it is also important to acquire the observations text that typically represents an exception to the regular schedule presented in the timetable.

Not only the input information is complex, but the different output formats also contain different vocabulary and concepts. One of the output format desired is the one already supported for importation in the OPT system environment, a set of CSV files with the structure represented in the figure 5.8.

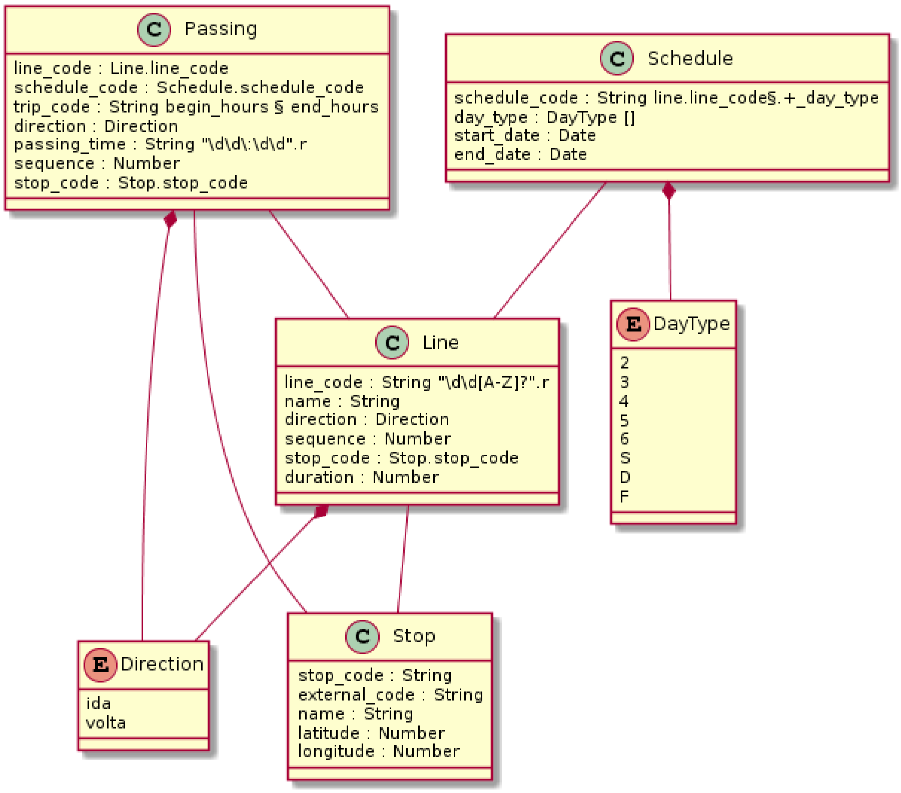


Figure 5.8: Domain model for the OPT import files

A different output format interesting for the field of timetable declaration is the GTFS format, which is specified in the figure 5.9 (Mouncif and Boulmakoul 2014).

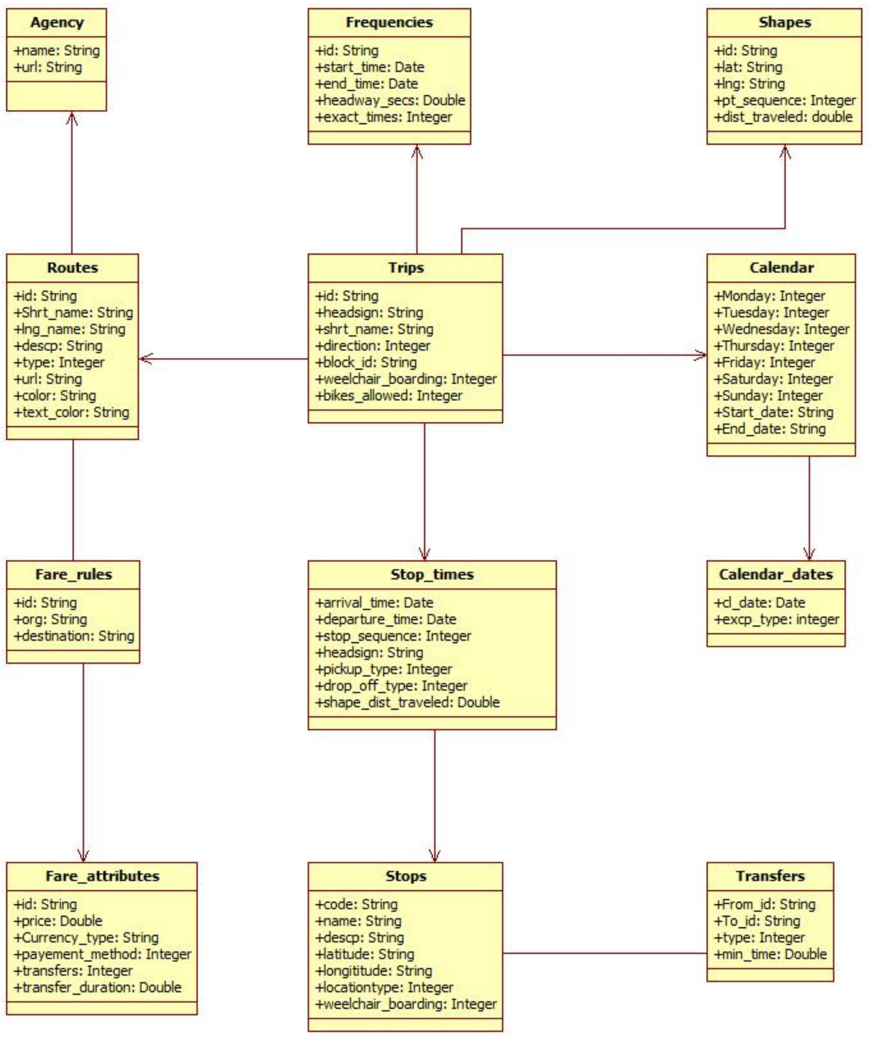


Figure 5.9: Domain model of the GTFS format

Considering the necessity to map data from multiple sources formats to different ontology formats, it is advantageous to create an intermediate ontology that can be used as a canonical format and function as a bridge to facilitate the mapping process and the data to knowledge conversion.

This process's components are illustrated in figure 5.10, in which the Web resources' data is extracted and mapped onto the intermediate/bridge ontology's ABox, and further mapped into the file representation ontologies.

This approach reduces the number of mappings required for adding a new web resource. Adding a new Web resource only requires a mapping to the intermediate ontology. The same happens when adding a new representation.

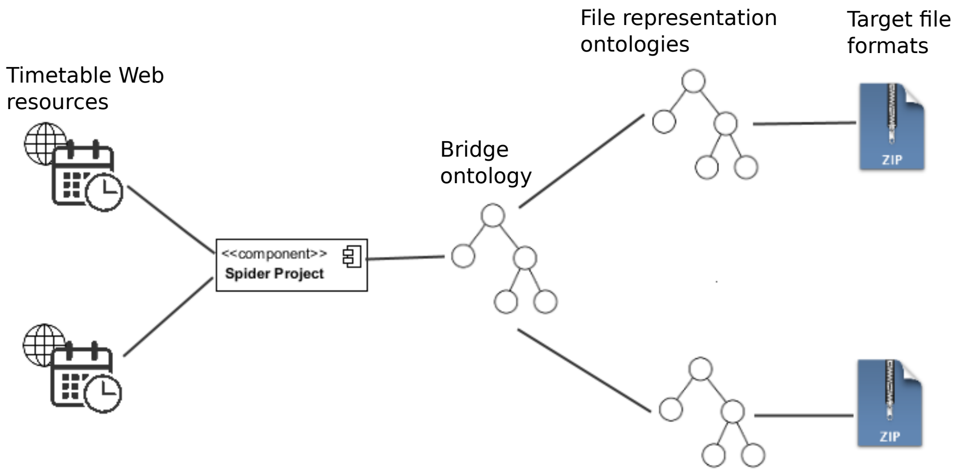


Figure 5.10: Mapping pipeline showing the ontologies involved in the format conversion process

5.5 Proposed design selection

The proposed architecture embodies the following key design principles or design guidelines:

Loose coupling - By creating well-defined interfaces between the different components, it becomes possible to protect each component from external changes. Any changes on one side do not affect the other as long as the interface remained stable.

Language independence - The system uses web services and messaging channels to establish communication between the different components. Both techniques are standard practice in the industry, and a wide array of languages provides implementations. Using these techniques allows for the addition of new components or tailored projects without having to be restricted to the other components language (Mak 2007).

Modularity - An advantage of a loose coupling system is that the components can be replaced with an alternative implementation. A well designed modular system not only allows the interchangeability of its parts but also allows for its components to be used in other systems, improving the overall system's reusability.

Reusability - The system ensures reusability through different ways for each component. For instance, the web scraping service provides a set of configurable web scrapers/ spiders. Configuration allows reusing the same spider for different websites of different providers by simply changing a few initial variables. Not only is there an abstraction of the information's source, but there is also an abstraction of the information's domain. This means that, hypothetically, the scraping component can extract information outside of the public transportation domain, as long as it is configured and executed with the correct initial parameters.

Extensibility - An extensible system allows the addition of new functionalities and assures the reduction of the required to implement the new functionality. One of the main design choices that improve this system extensibility is polymorphism and dependence injection. For example, a new extraction algorithm could be implemented without changing the code responsible for the algorithm execution. Developing a system for extensibility "leads to fewer, cleaner, dependencies, well defined interfaces and abstractions with corresponding reduction in coupling and higher cohesion" (Kelly 2002).

Due to the iterative nature of the development process, it was expected that the proposed solution design would progressively diverge from the initial alternatives.

From a high-level perspective, the proposed solution contains most of the components defined in the diagram presented in figure 5.5 while taking into account the problems which lead to the changes in the diagram in the figure 5.7.

The communication between the spiders/robots and the Backend happens asynchronously, using the previously proposed publish/subscribe pattern. In addition, the spider project provides an HTTP API designed for the execution of the spiders. The result is the system represented in figure 5.11.

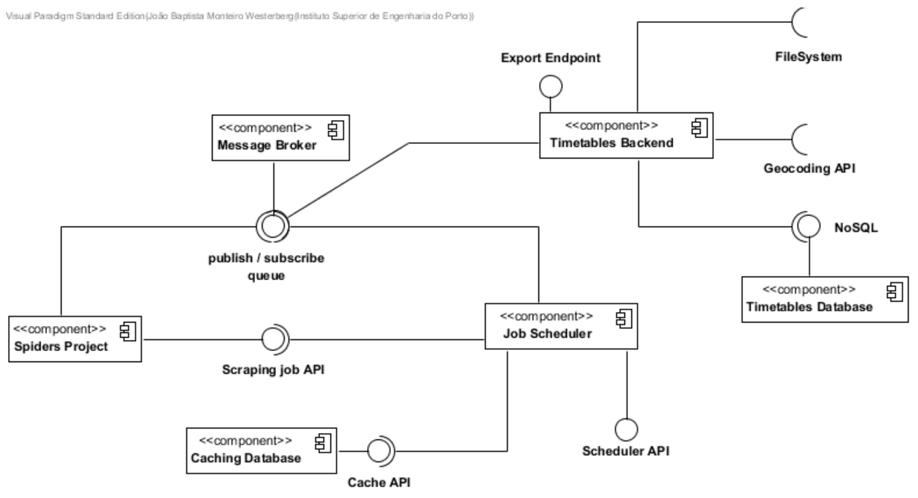

Visual Paradigm Standard Edition(João Baptista Monteiro Westerberg(Instituto Superior de Engenharia do Porto))
Export Endpoint
FileSystem
<<component>> Message Broker
<<component>> Timetables Backend
Geocoding API
NoSQL
publish / subscribe queue
<<component>> Timetables Database
<<component>> Spiders Project
<<component>> Job Scheduler
Scraping job API
<<component>> Caching Database
Scheduler API
Cache API

Figure 5.11: High level implementation / development view of proposed solution's system

As previously mentioned, although hand-written wrapper(spiders) perform better when compared to those generated by wrapper induction, they have the disadvantage of being tedious and time-consuming to create (Flesca et al. 2004).

A way to minimize the cost of developing these spiders is to create generic spiders that are parameterized to become configurable.

A scraping project that contains generic spider with executions configurable by their parameters can also contain custom made spiders for specific websites. As such, if a specific website requires a different approach from the ones available in the generic spiders, its scraping can be accommodated. As such, for a data retrieval solution, there are few disadvantages of using this approach, although it requires additional care to the security of the application when defining the parameters to avoid malicious code injections.

The Job Scheduler is responsible for scheduling scraping requests periodically and for configuring the execution of generic spider. Consequently, the job scheduler provides a remote facade (Fowler 2002) to the spiders' project API (represented as Scraping job API).

The Timetables Backend is responsible for handling the scraping responses, mapping the data onto the ontologies, and handling the persistence of that information. The Timetables Backend also offers the export endpoint from which users can request the generated files.

5.5.1 Spiders Project low-level design

This component contains a set of spiders/robots that instantiate items subjected to a sequence of pipelines, similar to the pipes-and-filters architectural pattern (Monroe et al. 1997). From a fine-grained perspective, the spiders' project has the design simplified in figure 5.12.

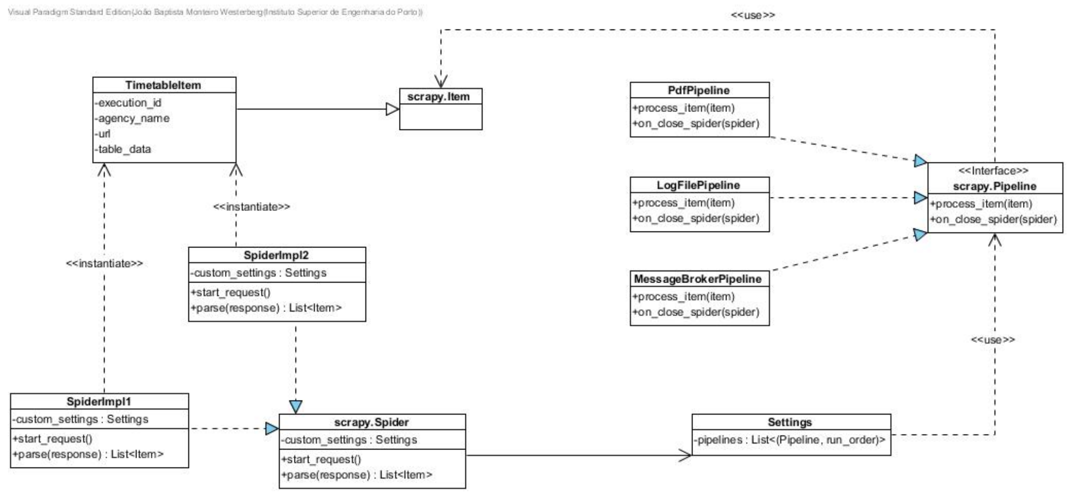


Figure 5.12: Low level logical view of Spider Project

The pipelines' order of execution is configured by an integer defined in the settings object, starting with the lower value to higher valued classes. An example of this process is simplified in the sequence diagram 5.13, where the Engine / Mediator is the Scrapy engine.

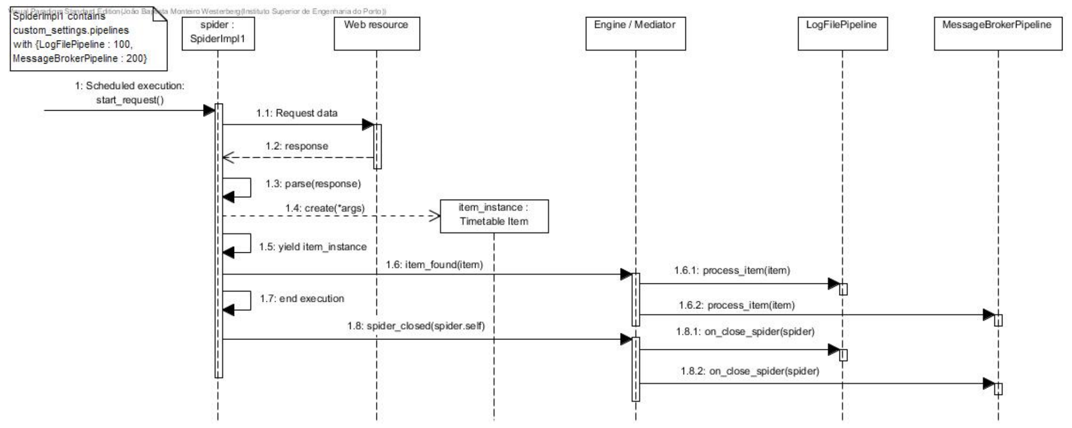

Visual Paradigm Standard Edition(João Baptista Monteiro Westerberg(Instituto Superior de Engenharia do Porto))
SpiderImpl1 contains custom_settings.pipelines with {LogFilePipeline : 100, MessageBrokerPipeline : 200}
spider : SpiderImpl1
Web resource
Engine / Mediator
LogFilePipeline
MessageBrokerPipeline
1: Scheduled execution: start_request()
1.1: Request data
1.2: response
1.3: parse(response)
1.4: create(*args)
item_instance : Timetable Item
1.5: yield item_instance
1.6: item_found(item)
1.6.1: process_item(item)
1.6.2: process_item(item)
1.7: end execution
1.8: spider_closed(spider.self)
1.8.1: on_close_spider(spider)
1.8.2: on_close_spider(spider)

Figure 5.13: Spider and pipeline execution sequence in Spider Project

The settings object can also include additional properties for improved configurability. Since the settings object is not private to the spider, it can be accessed by the pipelines. For example, adding a property for the message broker queue name allows different spiders to have different destination queues for their data.

5.5.2 Job Scheduler low level design

This component schedules spider executions from the spiders' project. The Job Scheduler communicates with the spider's project using the Scraping job API and provides a Scheduler API that works as a facade to the Scraping job API. A fine-grained design diagram is represented in figure 5.14

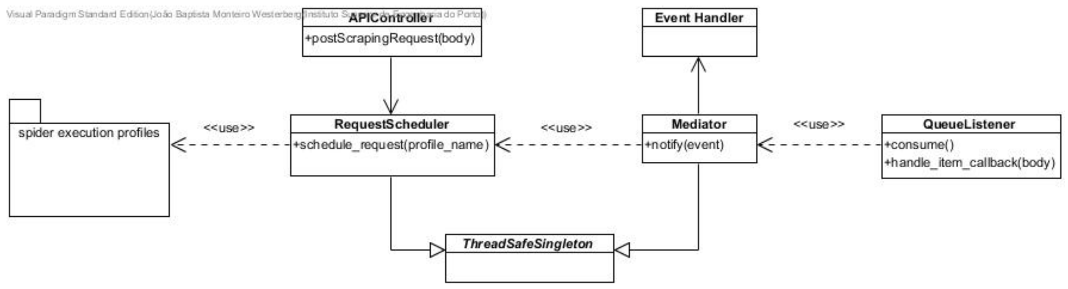

Visual Paradigm Standard Edition(João Baptista Monteiro Westerberg(Instituto Superior de Engenharia do Porto))
APIController
+postScrapingRequest(body)
Event Handler
spider execution profiles
<<use>>
RequestScheduler
+schedule_request(profile_name)
<<use>>
Mediator
+notify(event)
<<use>>
QueueListener
+consume()
+handle_item_callback(body)
ThreadSafeSingleton

Figure 5.14: Low level logical view of Job Scheduler

The generic spiders in the spiders' projects are executed via the Scraping job API, using the set of profiles available to the Request Scheduler. A profile is the group of parameters that configure the execution of a generic spider.

This component is also responsible for resolving the previously identified issue of handling dependencies and triggers in spiders' executions (refer to 5.3). This problem is resolved by consuming an execution's response from the message broker and handling the events by scheduling new spider executions if necessary.

The definition of which executions are triggered from which events is done via configuration and the Mediator object has the responsibility of managing it (Gamma et al. 1995). This allows to reduce the coupling between the system's components, and by applying the single responsibility principle, it improves the program's comprehension and maintenance (Gamma et al. 1995).

5.5.3 TimeTables Backend low level design

The timetables Backend is responsible for consuming the extracted data and exporting the data mapped into the desired formats. This component interacts with the ontologies, creating individuals of the Timetable ontology and mapping them into the file representation ontologies. A fine-grained overview of this component is represented in figure 5.15

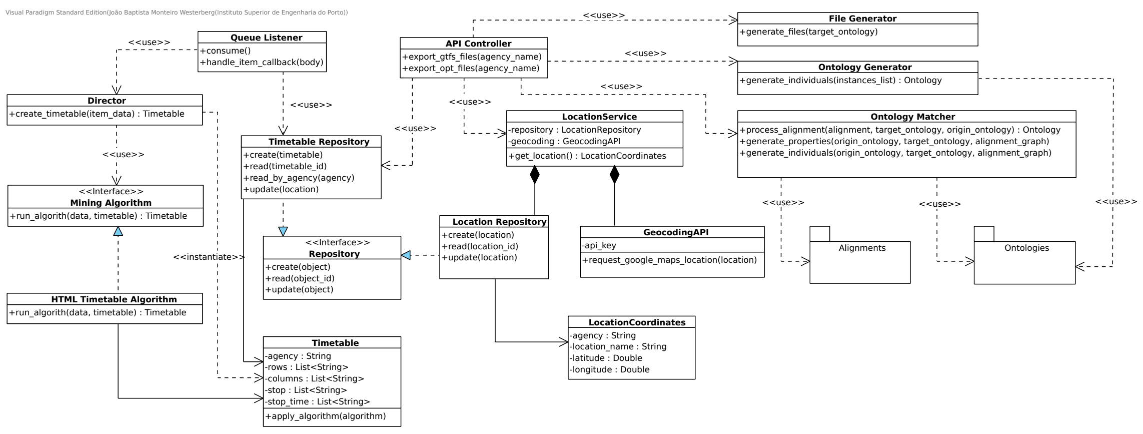

Visual Paradigm Standard Edition(João Baptista Monteiro Westerberg(Instituto Superior de Engenharia do Porto))
Queue Listener
+consume()
+handle_item_callback(body)
Director
+create_timetable(item_data) : Timetable
<<Interface>>
Mining Algorithm
+run_algorith(data, timetable) : Timetable
HTML Timetable Algorithm
+run_algorith(data, timetable) : Timetable
Timetable Repository
+create(timetable)
+read(timetable_id)
+read_by_agency(agency)
+update(location)
<<Interface>>
Repository
+create(object)
+read(object_id)
+update(object)
Timetable
-agency : String
-rows : List<String>
-columns : List<String>
-stop : List<String>
-stop_time : List<String>
+apply_algorithm(algorithm)
API Controller
+export_gtfs_files(agency_name)
+export_opt_files(agency_name)
LocationService
-repository : LocationRepository
-geocoding : GeocodingAPI
+get_location() : LocationCoordinates
Location Repository
+create(location)
+read(location_id)
+update(location)
GeocodingAPI
-api_key
+request_google_maps_location(location)
LocationCoordinates
-agency : String
-location_name : String
-latitude : Double
-longitude : Double
File Generator
+generate_files(target_ontology)
Ontology Generator
+generate_individuals(instances_list) : Ontology
Ontology Matcher
+process_alignment(alignment, target_ontology, origin_ontology) : Ontology
+generate_properties(origin_ontology, target_ontology, alignment_graph)
+generate_individuals(origin_ontology, target_ontology, alignment_graph)
Alignments
Ontologies
<<use>>
<<instantiate>>

Figure 5.15: Low level logical view of Timetables Backend

This component consumes the responses from spiders' execution, which create and persist timetable object. When a message is received in the queue, a mining algorithm is selected, creating and persisting a Timetable instance. This process is represented in figure 5.16.

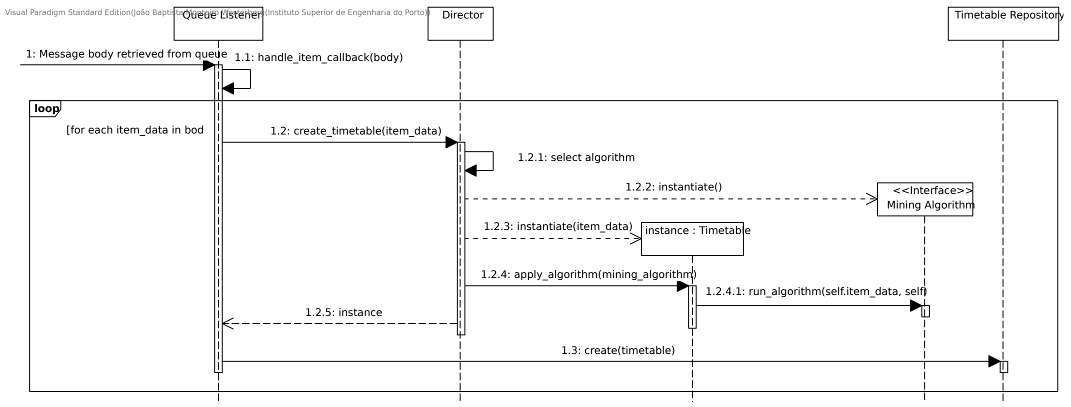

Visual Paradigm Standard Edition(João Baptista Monteiro Westerberg(Instituto Superior de Engenharia do Porto))
Queue Listener
Director
Timetable Repository
1: Message body retrieved from queue
1.1: handle_item_callback(body)
loop
[for each item_data in bod
1.2: create_timetable(item_data)
1.2.1: select algorithm
1.2.2: instantiate()
<<Interface>>
Mining Algorithm
1.2.3: instantiate(item_data)
instance : Timetable
1.2.4: apply_algorithm(mining_algorithm)
1.2.4.1: run_algorithm(self.item_data, self)
1.2.5: instance
1.3: create(timetable)

Figure 5.16: Process of creating and persisting timetables

The Mining Algorithm's concrete implementations encapsulate the logic for extracting information from the raw data. Simultaneously, the timetable contains the logic for creating its internal components (for example, instance.create_stop(stop_data)).

When creating the concrete implementations of the Mining Algorithm, the composite design pattern can be applied alongside the chain of responsibility pattern, improving the separation of concerns (Gamma et al. 1995) and following the Open/Closed Principle (figure 5.17 (Meyer 1997)).

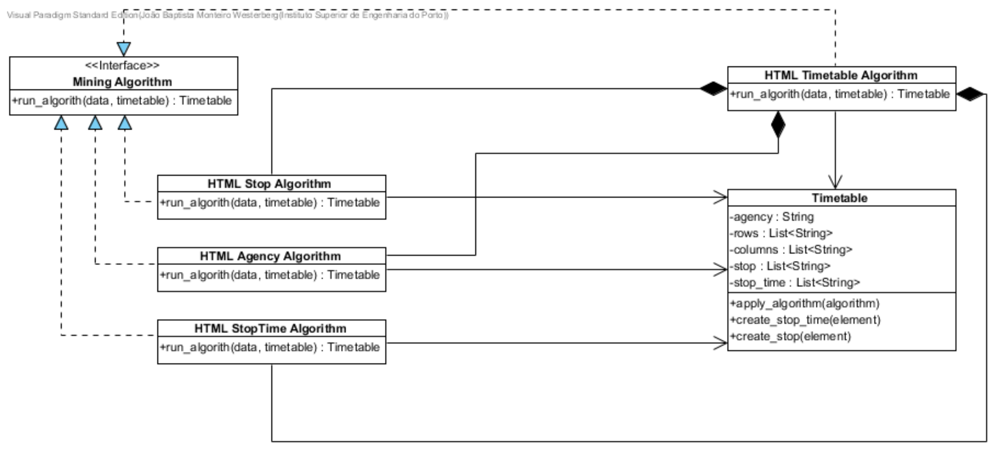

Visual Paradigm Standard Edition(João Baptista Monteiro Westerberg(Instituto Superior de Engenharia do Porto))
<<Interface>>
Mining Algorithm
+run_algorith(data, timetable) : Timetable
HTML Timetable Algorithm
+run_algorith(data, timetable) : Timetable
HTML Stop Algorithm
+run_algorith(data, timetable) : Timetable
HTML Agency Algorithm
+run_algorith(data, timetable) : Timetable
HTML StopTime Algorithm
+run_algorith(data, timetable) : Timetable
Timetable
-agency : String
-rows : List<String>
-columns : List<String>
-stop : List<String>
-stop_time : List<String>
+apply_algorithm(algorithm)
+create_stop_time(element)
+create_stop(element)

Figure 5.17: Mining Algorithm concrete implementations with separation of concerns

The timetable object represents a set of instances in the Timetable ontology. The Timetable Backend component also provides the API to export the files. When it receives an export request, the timetable's instances are retrieved from the database using the repository, and the process of generating individuals is initiated, as represented in 5.18.

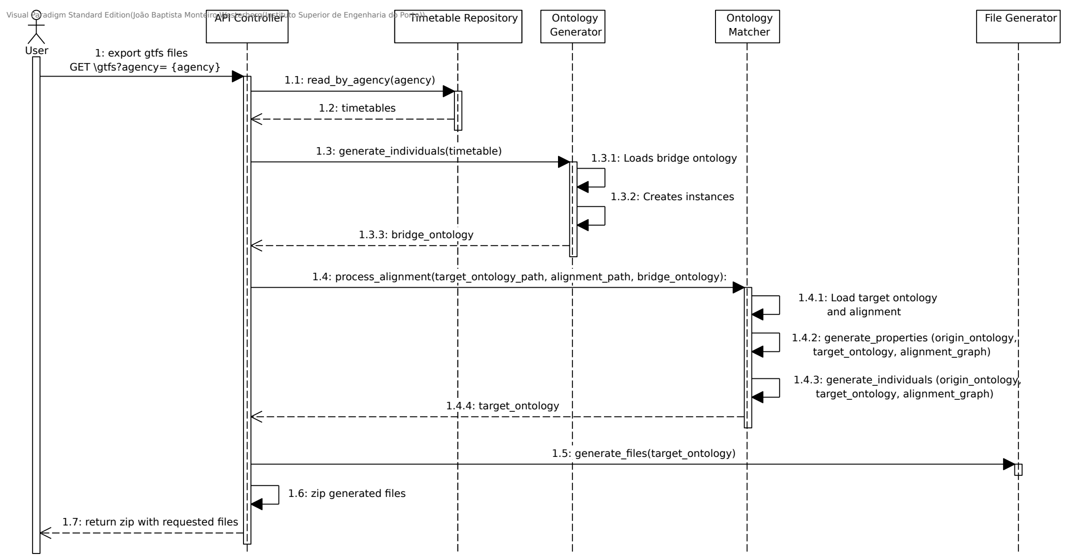

Visual Paradigm Standard Edition(João Baptista Monteiro Westerberg(Instituto Superior de Engenharia do Porto))
User
API Controller
Timetable Repository
Ontology Generator
Ontology Matcher
File Generator
1: export gtfs files GET \gtfs?agency= {agency}
1.1: read_by_agency(agency)
1.2: timetables
1.3: generate_individuals(timetable)
1.3.1: Loads bridge ontology
1.3.2: Creates instances
1.3.3: bridge_ontology
1.4: process_alignment(target_ontology_path, alignment_path, bridge_ontology):
1.4.1: Load target ontology and alignment
1.4.2: generate_properties (origin_ontology, target_ontology, alignment_graph)
1.4.3: generate_individuals (origin_ontology, target_ontology, alignment_graph)
1.4.4: target_ontology
1.5: generate_files(target_ontology)
1.6: zip generated files
1.7: return zip with requested files

Figure 5.18: Low level logical view of Timetables Backend

5.5.4 Solution deployment

Regarding the solutions' deployment, the designed proposed provides flexibility on where and how the system executes. By using virtual environments, it is possible to manage the packages and modules that do not come as part of the standard library. These virtual environments are a self-contained directory tree that contains a Python installation and additional packages. As expected, the virtual environments facilitate the deployment in a new physical device or a cloud environment.

Applying other good practices, such as setting/configuration files to configure the location of components and the location of their external APIs, makes it possible to scale the application using horizontal duplication. With horizontal duplication, the same application runs multiple copies behind a load balancer [2].

The diagrams in the figure 5.19 and figure 5.20 represent the two deployment approaches for this solution. By providing two contrasting approaches, it is possible to understand the limits of the solution regarding its distribution and the flexibility of its deployment.

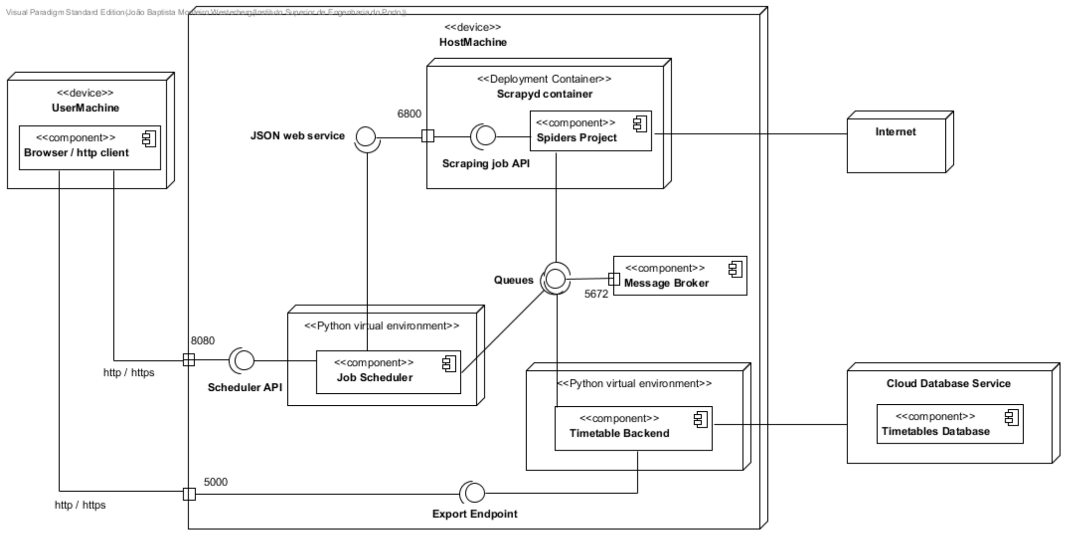


Figure 5.19: Deployment view of the solution in a single device

The first approach provides a design view, where a single device is used to contain multiple components. Each component is executed inside its virtual environment, allowing for different python versions and library versions to co-exist within the same device.

A scrapyd container encapsulates the Spiders Project component. Scrapyd is a service daemon for running Scrapy projects and spiders that provide a JSON web service and transforms the web service request into the Scrapy console commands required to run a spider.

[2]More information on horizontal duplication and other scalability dimensions can be found at `https://microservices.io/articles/scalecube.html`

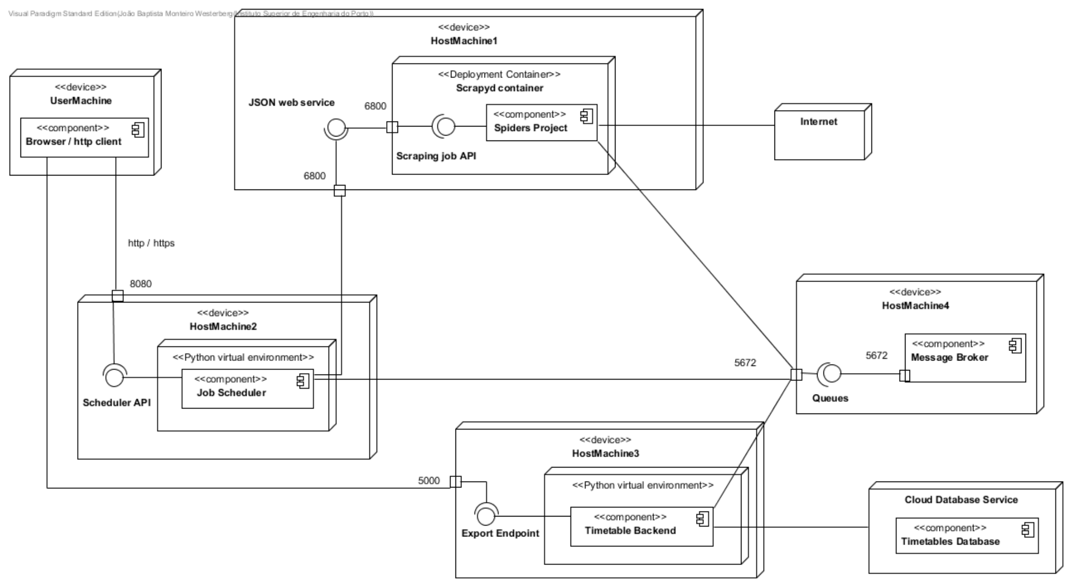


Figure 5.20: Deployment view of the solution distributed in several devices

The second deployment view provides a scenario on the other side of the spectrum. In this approach, every service is deployed on a different physical device. As previously mentioned, it is possible to scale the application service by running a new copy of a service and a load balancer. As an example, it would be possible to add a new copy of the Timetables Backend in a different machine and use the message broker with multiple queues for an exchange to achieve round-robin consumption.

5.6 Excalibur design for PDFs' information extraction

The open-source Excalibur project was selected and tailored to fit the solution's requirements for extracting the data and information from the agencies' PDFs. From a coarse-grained perspective, the Excalibur project contains the components illustrated in figure 5.21.

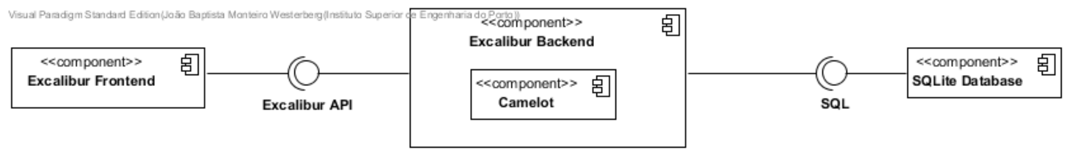


Figure 5.21: Deployment view of the solution distributed in several devices

From a more detailed perspective, the project can be represented as illustrated in figure 5.22. The Excalibur project contains a Frontend developed using Jinja[3], a Model-View-Template, that fills HTML templates with the responses received by the Backend API client.

The Backend contains an HTTP API to access the Camelot library's functionalities while also containing the classes required for the data persistence and the file importation.

[3]Documentation regardings jinja can be found at `https://jinja.palletsprojects.com/en/2.11.x/`

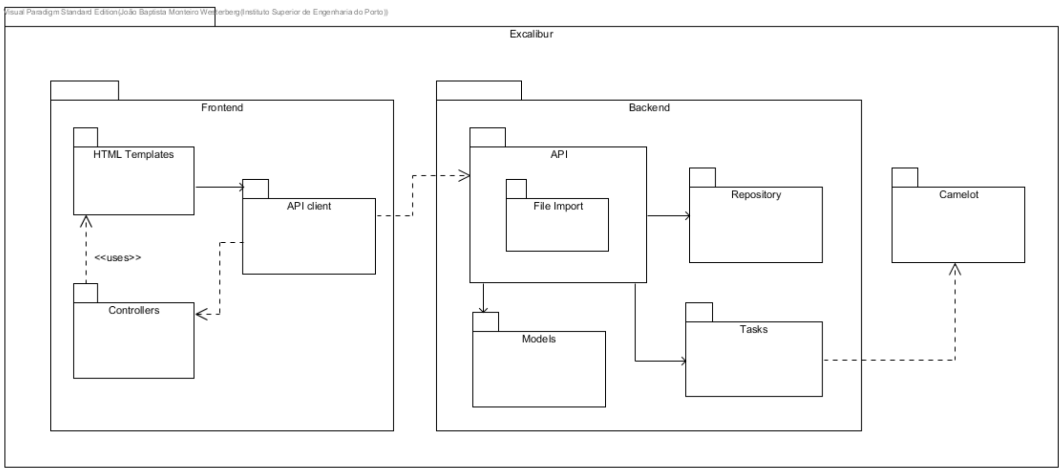

Visual Paradigm Standard Edition(João Baptista Monteiro Westerberg(Instituto Superior de Engenharia do Porto))
Excalibur
Frontend
HTML Templates
API client
<<uses>>
Controllers
Backend
API
File Import
Repository
Models
Tasks
Camelot

Figure 5.22: Deployment view of the solution distributed in several devices

A message broker client was added to the Excalibur code base to integrate the Excalibur project into the solution's system. The Excalibur Backend can then communicate with the other solution's components using the channels already defined in the message broker.

When receiving a list of URLs identified by the spider as PDF, the files are downloaded and connected with Excalibur's file import features.

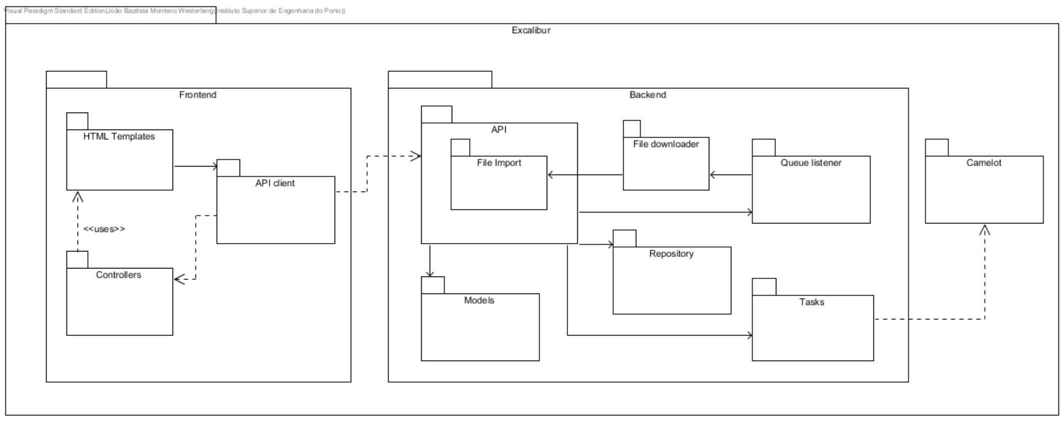


Figure 5.23: Deployment view of the solution distributed in several devices

Finally, after extracting the files' information, a message is sent to the appropriate message broker exchange to be consumed by its subscribers.

Chapter 6

Implementation

This chapter of the document describes the implementation process of the solution from the design described in 5.5. It begins by describing the generic spiders' implementation and job scheduler. It continues with the ontology engineering process and, finally, with the Backend services' implementation details.

6.1 Spider project implementation

The spider project is the component responsible for data/information retrieval. This component consists of spiders and common/shared classes that promote code reusage from spider to spider. The shared classes are either items or pipelines.

"Items" is the Scrapy terminology for the simple python objects returned from a spider's parse method. These objects are implementing the pattern usually referred to in software engineering as Data Transfer Object (DTO) (Fowler 2002), and, as its name would suggest, are objects whose purpose is to specify the structure of data transferred in the communication between system's components.

Scrapy pipelines are middleware components that can be used for:

- cleaning HTML data
- validating scraped data (checking that the items contain certain fields)
- checking for duplicates (and dropping them)
- storing the scraped item in a database

Pipelines are classes implementing the process_item and close_spider methods as demonstrated in code 6.1.

```
class RabbitMQPipeline(object):
    ... # Setting up channel and connection

    def close_spider(self, spider):
        data = self.encoder.encode(self.items) #encoder =
    ScrapyJSONEncoder()
        if len(data) > 0:
            self.items = []
            self.channel.basic_publish( exchange=self.
    exchange, routing_key=self.routing_key, body=data)
        self.channel.close()
        self.connection.close()

    def process_item(self, item, spider):
        self.items.append(item.copy())
        return item
```

Listing 6.1: RabbitMQ pipeline (Python).

Finally, as previously explained, generic spiders were constructed to reduce the development time required to scrape a website. These generic spiders share the same pipelines and items. The generic spiders implemented are "URL Extractor," "Single timetable per page," spider, and "Multiple timetables per page" spider.

6.1.1 URL Extractor

This generic spider is responsible for crawling a web site and finding its relevant URLs. It receives a root URL and a set of parameters that restrict the scope of the crawling. Those parameters are presented in table 6.1 alongside a brief description.

Table 6.1: URL Extractor Attributes

Attribute Name	Brief Description
event_name	Event to be published in the response
agency_name	Name of the agency
root	Starting URL of execution
allow_domains	Domains allowed
allow	Restrictions to subdirectories using regex
allow_pdf	Boolean value where if true logs the URL of the PDFs found

Using the initial URL, the spider finds URLs embedded in the HTML and iterates them recursively with a depth-first search algorithm. The extraction of embedded URLs is done using the LinkExtractor object from Scrapy [1].

The process of URL crawling is presented in a simplified format in the algorithm 6.1

[1]Documentation about LinkExtractor at `https://docs.scrapy.org/en/latest/topics/link-extractors.html`

Algorithm 6.1 Algorithm for extracting a URL item

```
 1: Input: Response object $response$, List of items $list$
 2: Output: Items created
 3:
 4: procedure CreateURLItems($response$)
 5:     $urls \leftarrow extract\_urls(response)$          ▷ Current URL is replaced by new URLs
 6:     if $urls$ $not$ $empty$ then
 7:         $list.add(create\_item(response))$
 8:         for each $url \in urls$ do
 9:             if $url$ $not$ $pdf$ then
10:                 return $Request(url, callback = CreateURLItems)$
11:             end if
12:         end for
13:     else
14:         return $list$
15:     end if
16: end procedure
```

The Scrapy framework takes advantage of a trait present in the python programming language. By requiring the spiders to use the yield operator, instead of returning the obtained values, the spider parse function becomes a generator.

In simple terms, a generator is a function that behaves as an iterable and can be thought of as an iterator that contains a frozen stack frame (Van Rossum et al. 2007). When a function asks for the next element in the generator, the operation resumes until the next yield value.

Considering this, although the algorithm might suggest that it implements a depth-first search, the responsibility of managing the URLs navigating and selecting the crawling order is delegated to the Scrapy core. This implementation allows a developer to swap the searching algorithm by altering the project configuration file's respective property.

Besides finding the hyperlinks of an HTML page, this spider can also be configured to extract the URL of PDF files using the allow_pdf property. This boolean property has the default value of false, but if changed, the ".pdf" extension is removed from the deny_extensions used by the LinkExtractor.

If the execution finds any PDFs, these responses are passed to the PDFPipeline and are downloaded, as demonstrated in code 6.2.

```python
class PDFFilePipeline(FilesPipeline):
    def file_path(self, request, response=None, info=None):
        original_path = super(PDFFilePipeline, self).file_path(
    request, response=None, info=None)
        sha1_and_extension = original_path.split('/')[1]  #
    delete default 'full/' from the path
        name = request.url.split("/")[-1].split(".")[0]  # get
    original name of document
        folder = request.meta['agency_name']  # set the folder
    name as the agency_name
        return f'{folder}/{name}_{sha1_and_extension}'

    def get_media_requests(self, item, info):
        return [scrapy.Request(x, meta={'agency_name': item['
    agency_name']}) for x in item.get('file_urls', [])]
```

Listing 6.2: PDF pipeline (Python).

6.1.2 Single Timetable per page

While the previous spider is responsible for extracting the relevant URLs for the scraping, this generic spider receives a list of those URLs and returns items according to the parsing specifications. The configuration attributes are represented in table 6.2 alongside a brief description.

Table 6.2: Spider attributes

Attribute Name	Brief Description
agency_name	Name of agency
urls	List of urls
config	Dictionary of XPath expression and respective name

The config property specifies the target HTML element and how those XPath expressions can parse/access the elements. For each entry in the config, spider parses the HTML as demonstrated in algorithm 6.2.

Algorithm 6.2 Algorithm for extracting a timetable item

```
1: **Input**: HTML data named $html$ an a dictionary, $config$
2: **Output**: Item created
3:
4: **procedure** CreateItems($html, config$)
5:     $item \leftarrow new\ timetable_item$
6:     **for each** $(Element, XPathExpression) \in config$ **do**
7:         $item[Element] \leftarrow html.extract(XPathExpression)$
8:     **end for**
9:     **return** $item$
10: **end procedure**
```

The process of extraction presented in the algorithm is slightly simplified to improve interpretation. The algorithm can is complemented with the code in 6.3, showcasing the response handling. Showing the code is the easiest way to communicate how the Scrapy framework handles the extraction of HTML elements from raw data using XPath expression.

```python
def start_requests(self):
    for url in self.urls:
        yield scrapy.Request(url, callback=self.parse)

def parse(self, response):
    item = TableItem()
    for key in self.config:
        element = response.xpath(self.config[key]).extract()
trans_table = {ord(c): None for c in u'\r\t\n\xa0'}  # remove
   undesirable unicode chars from extracted data
        item[key] = [elem.translate(trans_table) for elem in
   element]
    yield item
```

Listing 6.3: Simple generic spider for HTML response handling (Python).

In this operation, the data from the web pages begin to be processed into information. The dictionary described in the config property will be the catalyst in the data to information conversion. During the scraping process, the spider identifies and names specific HTML elements, adding context to the retrieved data. Thus, it is possible to use a finite set of names/flags that can be used by the Backend components for identifying the correct method of data/information mining. The names/flags used are:

- direction - element with the timetable name and direction, usually a heading tag(e.g.,h1,h2)
- service - element with the timetable schedule(e.g., Monday to Friday), usually a heading tag or an HTML label
- table - element with the table information, usually an HTML table or div
- observations - element with additional information, usually a heading tag or HTML paragraph

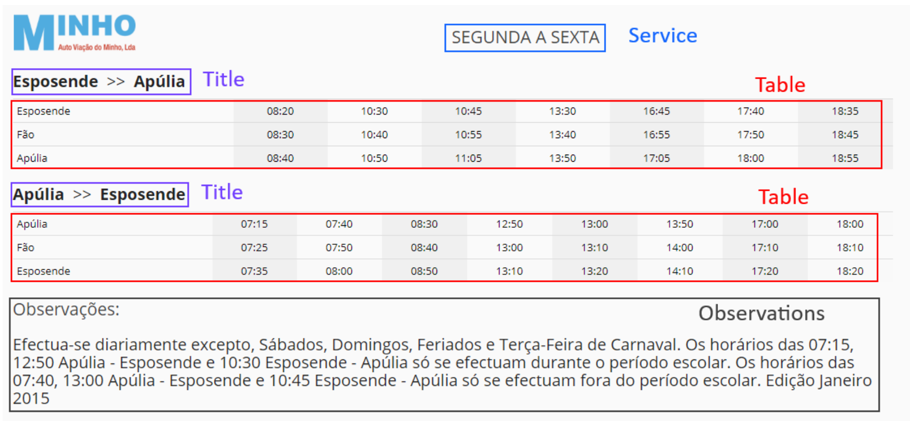
6.1.3 Multiple Timetables per page

The previous spider handles websites where each URL has a specific timetable. However, some public transportation agencies use websites in which a single page displays multiple timetables.

These webpages should not be confused with scraping single-page applications. In this scenario, the objective is still to scrape a static web page, but instead of retrieving a single item for URL, it retrieves multiple items (timetables).

Although XPath extractors still perform the extraction of data, the result of those extractors will be a list of HTML elements instead of a single element.

Using the Web page in figure 6.1 and the elements identified, the spider can generate two items containing a service, a title, a table, and observations. Some elements need to be shared by multiple items (in this example, the service, and observations).


SEGUNDA A SEXTA

Esposende >> Apúlia

Esposende	08:20	10:30	10:45	13:30	16:45	17:40	18:35
Fão	08:30	10:40	10:55	13:40	16:55	17:50	18:45
Apúlia	08:40	10:50	11:05	13:50	17:05	18:00	18:55

Apúlia >> Esposende

Apúlia	07:15	07:40	08:30	12:50	13:00	13:50	17:00	18:00
Fão	07:25	07:50	08:40	13:00	13:10	14:00	17:10	18:10
Esposende	07:35	08:00	08:50	13:10	13:20	14:10	17:20	18:20

Observações:

Efectua-se diariamente excepto, Sábados, Domingos, Feriados e Terça-Feira de Carnaval. Os horários das 07:15, 12:50 Apúlia - Esposende e 10:30 Esposende - Apúlia só se efectuam durante o período escolar. Os horários das 07:40, 13:00 Apúlia - Esposende e 10:45 Esposende - Apúlia só se efectuam fora do período escolar. Edição Janeiro 2015

Figure 6.1: Example of multiple timetables per page

This spider requires some additional parameters that are not present in the "single timetable per page" spider to achieve that behavior. The parameters and their brief description are presented in table 6.3.

Table 6.3: Spider attributes

Attribute Name	Brief Description
agency_name	Name of agency
urls	List of urls
config	Dictionary of XPath expression and respective object name in page_structure
page_structure	Dictionary with position order of elements in the web page's HTML and respective XPath
item_structure	List of items to be created using the elements found

Figure 6.2 presents an example for the property values required to extract multiple items from 6.1.

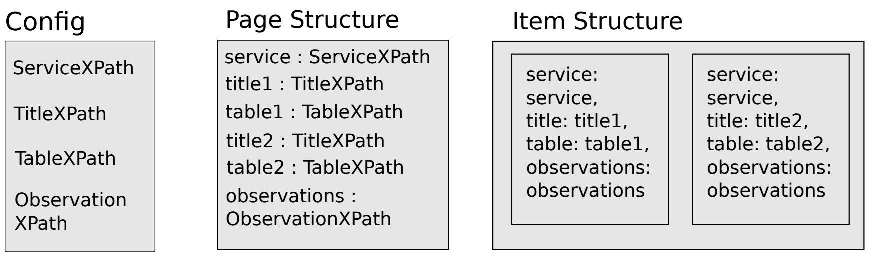


Figure 6.2: Properties for multiple timetables spider example

The page structure-property does not require the definition of the entire web page's structure. Instead, it identifies groups of HTML elements. If the web page contains four HTML div elements, each with child elements following the structure presented in 6.2, the result will be eight items. The algorithm 6.3 demonstrates how this can be achieved.

Algorithm 6.3 Algorithm for extracting Multiple Timetables per page

```
1: Input: HTML data named $html$, two dictionaries, $config$, $page\_structure$ and a list of dictionaries $item\_structure$
2: Output: List of items created
3:
4: procedure CreateItems($html, config, page\_structure, item\_structure$)
5:     $item\_list \leftarrow new\ list$
6:     $html\_elements \leftarrow new\ dict$
7:     for each ($XPathExpressionId, XPathExpression$) $\in config$ do
8:         $html\_elements[XPathExpressionId] \leftarrow html.extract(XPathExpression)$
9:     end for
10:    while $html\_elements.has\_elements()$ do
11:        $elements\_value \leftarrow new\ dict$
12:        for each ($ElementName, XPathExpressionId$) $\in item\_structure$ do
13:            $elements\_value[ElementName] \leftarrow html\_elements[XPathExpressionId].remove\_first$
14:        end for
15:        for each $structure \in item\_structure$ do
16:            $item \leftarrow new\ item$
17:            for each ($ItemAttribute, ElementName$) $\in structure$ do
18:                $item[ItemAttribute] \leftarrow elements\_value[ElementName]$
19:            end for
20:            $add\ item\ to\ item\_list$
21:        end for
22:    end while
23:    return $item\_list$
24: end procedure
```

6.2 Event mediation with job scheduler

The Job Scheduler is the software component responsible for making execution requests to the spiders and handles the events published with the spiders' response.

This component requires for the spiders to be deployed using a scrapyd container since it depends on the API provided by the scrapyd container for its full operation [2].

The scrapyd API should be accessed exclusively by the job scheduler. As previously noted, the spiders require a set of parameters for their execution. This way, the job scheduler only allows the scheduling of spider executions by using pre-specified configuration files called execution profiles.

These execution profiles are JSON files, located in the "jsonResources" folder of the project root, that define a set of parameters to configure a generic spider execution.

The Job Scheduler runs three different threads, each representing a different type of action that leads to an execution's scheduling. These three different actions are:

- Scheduled execution, for example, every day at 01:00, use a group of execution profiles to run specific executions
- User request via the Scheduler API (refer to subchapter 5.5.2)
- Consumed events that triggers new requests

For the first type of action, using the schedule[3] and time packages, the following code 6.4 represents scheduling, where "scrape/avminho" is the name of the profile to be executed every day at 1:00 am.

```python
def daily_schedule():
    schedule.every().day.at("01:00").do(
   daily_schedule_callback)
    while True:
        schedule.run_pending()
        time.sleep(60)

def daily_schedule_callback():
    RequestScheduler().schedule_request_with_urls("scrape/
   avminho")
```

Listing 6.4: Daily Scheduling (Python).

For the second action type, the API implemented receives a query parameter from the API consumer sends a query parameter, representing the profile's name. The endpoints were implemented using the Flask[4] library / micro-framework as demonstrated in the example code 6.5.

[2]Detailed information about the scrapyd API is available at https://scrapyd.readthedocs.io/en/stable/api.html

[3]Documentation for the schedule package at https://schedule.readthedocs.io/en/stable/

[4]More information about Flask and documentation is found at https://flask.palletsprojects.com/en/1.1.x/

```python
app = Flask(__name__)

def run_app():
    with open('./jsonResources/config.json') as
   config_file:
        config = json.load(config_file)
    app.run(port=config['API_Port'])

@app.route('/extract')
def extract():
    scheduler = RequestScheduler()
    profile = request.args.get('profile')
    scheduler.schedule_request(f"extract/{profile}")
```

Listing 6.5: API Scheduling (Python).

The last action that leads to an execution's scheduling comes from event handling. After executing a URL Extraction spider, an event is created and published to the RabbitMQ exchange/queue. The job scheduler will then consume this event, as demonstrated in code 6.6.

```python
def consume():
    channel.basic_consume(queue='item',
   on_message_callback=items_queue_callback, auto_ack=
   True)
    channel.start_consuming()

def items_queue_callback(ch, method, properties, body):
    mediator = Mediator()
    json_body = json.loads(body)
    mediator.notify(json_body["event_name"].pop(), list(
   json_body["urls"]))
```

Listing 6.6: Event Scheduling (Python).

After consuming the message with that event, the job scheduler's Mediator object accesses a configuration file containing a dictionary where the keys are the event_names, and the values are the list of execution profiles to be scheduled. The Mediator verifies which profiles should execute after the event is triggered and schedule those executions, as presented in code 6.7

```python
class Mediator():
    def __init__(self, events_config="events", scheduler=
   RequestScheduler()):
        with open(f"./jsonResources/{events_config}.json")
    as json_file:
            self.events = json.load(json_file)
        self.scheduler = scheduler

    def notify(self, event: str, urls: list):
        if event in self.events:
            for filename in self.events[event]:
                self.scheduler.schedule_request_with_urls(
   filename, urls)
```

Listing 6.7: Event mediator from job scheduler (Python).

6.3 Implemented ontologies

As already established, for the development of this solution, ontologies were developed to describe the data's structure and mediate the mapping between output formats.

The proposed solution includes three ontologies. Two of these ontologies represent the structure of the desired output files. In contrast, the other ontology works as a bridge connecting the different data formats of the same domain with the file representation ontologies. These ontologies were developed using Stanford's Protege tool (N. Noy et al. 2001) and are expressed in RDF/XML (Beckett and Brian McBride 2004).

The development of the GTFS and OPT ontologies was simple. The objective of these ontologies is to represent the classes and taxonomy represented in the desired output. For the OPT format, the classes and properties created in OWL/RDF-S result from the domain model previously described in the diagram in figure 5.8. For GTFS, an outdated GTFS ontology [5] was repurposed and customized to represent the current file formats.

The two file representation ontologies are domain ontologies about public transportation, each under a slightly different scope of view. Using the words of Gruber (1993), "An ontology is a specification of a conceptualization", so in this concrete example, there are two different specifications of the same domain concepts.

Each of these ontology uses different terminology to describe similar concepts. An approach that would allow the representation format's interchangeability is to design the timetable ontology as a domain ontology that contains the terms of both representations. As previously identified in 3.4.2, this type of integration is categorized as ontology merging, where a new ontology that represents the unification of the ontologies to integrate is created to work as the common ground.

However, it is essential not to lose focus on the issue at hand. The support of multiple output formats is secondary to the main responsibility of the bridge ontology. This ontology connects the unorganized and unformatted data retrieved by the spiders with the organized and formalized output formats.

Although using a domain ontology as the bridge ontology can enable the support of multiple file representation, its usage increases the difficulty of the initial data mapping, mapping the unorganized data to instances of the ontology.

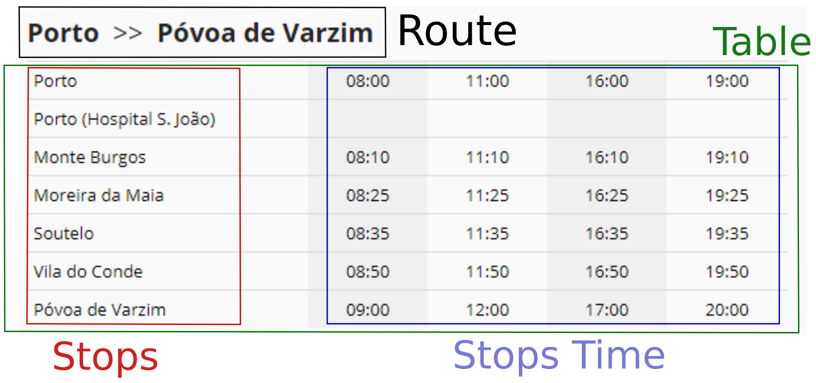
A different approach is to design the bridge ontology as a task ontology that specifies the format of how this type of data is generally found online. This approach allows the spiders to map the data to the bridge ontology's Abox easily.

When it comes to representing public transportation data, most representation includes the name of a route and a table with the stops and passage times. As such, the bridge ontology needs to contain the terms necessary to describe the data in that original format, alongside axioms that provide context and understanding, enabling the extraction of information and knowledge.

The bridge ontology can contain terminology from the target ontologies as necessary, and the instances of those terms can be inferred using axioms.

[5] GTFS ontology can be found at `http://vocab.gtfs.org/gtfs.ttl#`, credited to Pieter Colpaert and Andrew Byrd

Using the timetable in the figure 6.3 as an example [6], some concepts typical of these representations can be identified and specified.



Porto	08:00	11:00	16:00	19:00
Porto (Hospital S. João)				
Monte Burgos	08:10	11:10	16:10	19:10
Moreira da Maia	08:25	11:25	16:25	19:25
Soutelo	08:35	11:35	16:35	19:35
Vila do Conde	08:50	11:50	16:50	19:50
Póvoa de Varzim	09:00	12:00	17:00	20:00

Figure 6.3: Example of online timetable with concepts identified

The process of ontology engineering involves not only understanding and codifying the concepts represented but also codify the concepts, axioms, and rules for how tabulated data is presented and how a human reader perceives this data under the scope of transport schedules.

For instance, this includes describing the parts of a table, the difference between a row and column, and other logical assumptions subconsciously understood by a human reader due to their exposure to these types of formats.

By analyzing the concepts of a scheduling table (such as the one in figure 6.3), it is possible to obtain the ontology's classes, properties, and taxonomy represented in figure 6.4.

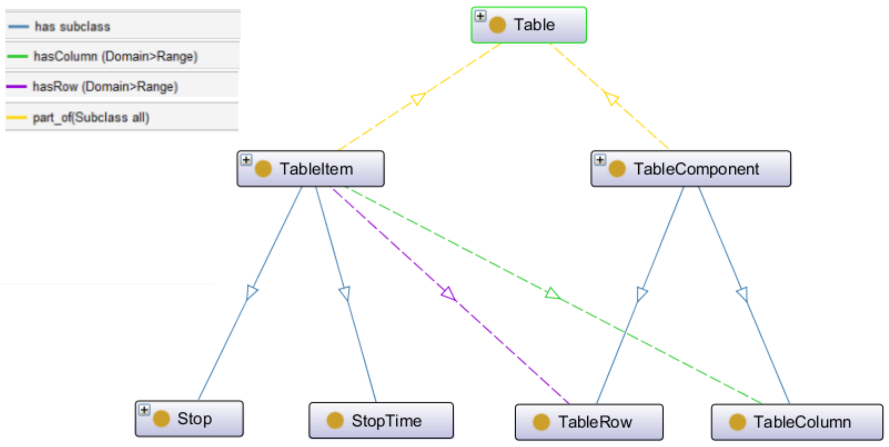


Figure 6.4: Table related OWL classes and properties

This representation contains the property partOf used in the following axioms 6.1 and 6.2[7].

[6] Timetable retrieved from `http://www.avminho.pt/horarios`

[7] An overview of the description logics notation is explained in Baader (2003)

$$\text{TableItem} \sqsubseteq \exists \text{partOf.Table} \tag{6.1}$$

$$\text{TableComponent} \sqsubseteq \exists \text{partOf.Table} \tag{6.2}$$

Besides the properties represented, the ontology contains the inverse properties presented in table 6.4.

Table 6.4: Inverse properties of timetable ontology

Property	Inverse Property	Inverse Property Domains	Inverse Property Range
partOf	hasPart	Table	TableComponent ⊔ TableItem
hasColumn	hasItemInColumn	TableColumn	TableItem
hasRow	hasItemInRow	TableRow	TableItem

Considering the topic and purpose of the ontology, the ontology concepts may not be related to the domain of public transportation but instead to the representation of a table. For example, to differentiate the subClasses of TableComponent, which are TableRow and TableColumn, the class Orientation was added, resulting in the following axioms.

$$\text{TableRow} \sqsubseteq \text{hasOrientation(Horizontal)} \tag{6.3}$$

$$\text{TableColumn} \sqsubseteq \text{hasOrientation(Vertical)} \tag{6.4}$$

These axioms also required the addition of the instances "Vertical and Horizontal" to the Abox of the ontology. When a reasoner executes, as a consequence of these axioms, every instance of TableRow will have the property hasOrientation with Horizontal, and the same applies to the TableColumn and Vertical.

At first sight, these axioms might seem simplistic and unrelated to the topic of public transportation or even integration. However, these embedded semantics can be used to define complex classes or to query the knowledge graph.

The next step after defining a timetable visualization's basic concepts was to add domain concepts to the ontology. Preferably the concepts share the same name as the ones present in the target outputs.

Using the domain model of the GTFS format presented in figure 5.9 the table 6.5 was created representing the added domain concepts. Since most of the concepts present in the diagram were optional, the table only contains the essential and relevant concepts with a brief description. These concepts were then defined as OWL classes.

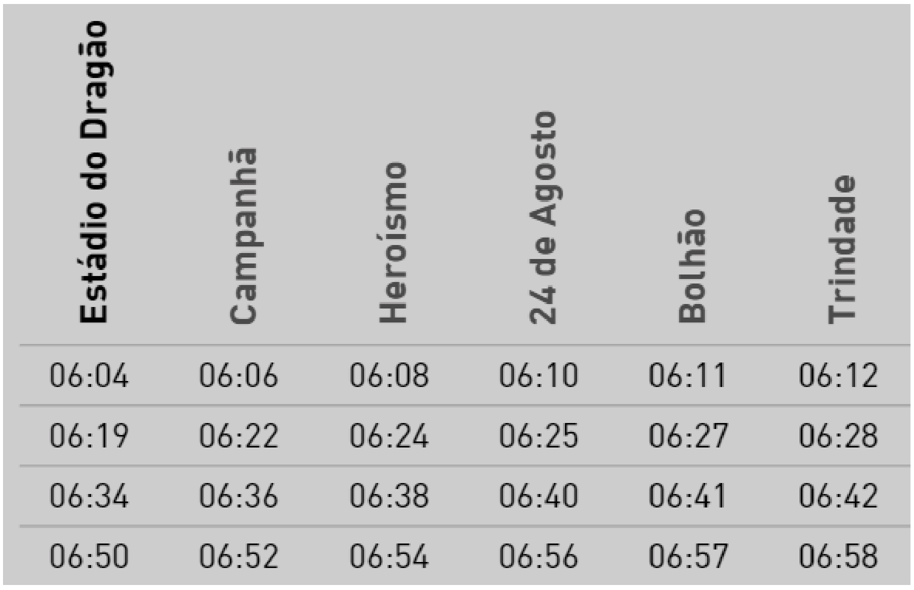
Table 6.5: Domain concepts from GTFS format

Concept	Brief Description
Agency	Public transportation company or public transport provider
Stop	Geographic location where vehicles pick up or drop off riders

Stop Time	Time that a vehicles departs from a specific stop
Trip	A trip is a sequence of two or more stops that occur during a specific time period
Route	A route is a group of trips under the same name, usually the origin stop and the destination stop
Service	A service is the weekly schedule of a given route. A route can have multiple weekly schedule each with different trips. For example a Sunday service that has a trip starting at 9:30, and a Monday service with different trips

Except for "Trip", all other concepts can be extracted from the spiders' execution. However, it is still required for the ontology to maintain or create the relationships between the extracted data.

Using a timetable different from the one represented in the figure 6.3, the timetable represented in the figure 6.5 possesses the same visual components [8] but the grouping of elements is different.



Estádio do Dragão	Campanhã	Heroísmo	24 de Agosto	Bolhão	Trindade
06:04	06:06	06:08	06:10	06:11	06:12
06:19	06:22	06:24	06:25	06:27	06:28
06:34	06:36	06:38	06:40	06:41	06:42
06:50	06:52	06:54	06:56	06:57	06:58

Figure 6.5: Example of online timetable with stops grouped in a row

In the previous example (figure 6.3), the stops were displayed in a column, where in this example (figure 6.5) are displayed in a row.

This ambiguity causes issues when trying to identify the individuals that are Trips. In the first table, a trip was a column with the exception of the column with stops. The second table presents the reverse situation, and a trip becomes a row. The following axioms and rules solve this ambiguity.

[8]The figure only represents a portion of the timetable retrieved from `https://www.metrodoporto.pt/frontoffice/pages/337`

$$\begin{aligned}\mathsf{TableComponent} \sqcap (\geq 2\ \mathsf{hasItemInColumn.Stop} \sqcup \geq 2\ \mathsf{hasItemInRow.Stop})\\ \sqsubseteq \mathsf{StopsTableComponent}\end{aligned} \tag{6.5}$$

$$\begin{aligned}\forall x, y, z:\ & \mathsf{Table}(x) \land \mathsf{StopsTableComponent}(y) \land \mathsf{hasPart}(x, y)\\ & \land \mathsf{hasOrientation}(y, z) \rightarrow \mathsf{routeOrientation}(x, z)\end{aligned}$$

Rule 6.1: Rule for defining the orientation of a table's trips

At first, the axiom 6.5 defines that if a TableComponent contains more than one stop, it has the superClass StopsTableComponent (which is a subclass of TableComponent).

With this information, it is then possible to identify the orientation in which the Trips are represented for that table because the Trips will have the same orientation as the TableComponent with the routes stops.

Using this information is then possible for the reasoner to infer which instances of TableComponent are Trips. Using a closed world assumption, an axiom such as 6.2 would work. However, due to OWL's open-world assumption, the reasoner cannot assume that every TableComponent that is not asserted as a StopsTableComponent is an instance of $\neg StopsTableComponent$.

$$\begin{aligned}\forall x, y, z:\ & \mathsf{Table}(x) \land \mathsf{TableComponent}(y) \land \mathsf{hasPart}(x, y) \land \mathsf{routeOrientation}(x, z)\\ & \land \mathsf{hasOrientation}(y, z) \land (\neg\mathsf{StopsTableComponent})(y)\\ & \rightarrow \mathsf{TripTableComponent}(y)\end{aligned}$$

Rule 6.2: Rule for defining trips individuals with negation

The issue with the open-world assumption could be easily solved if the Abox of the solution was static. If that were the case, that property could be asserted about the individuals. However, since the Abox of the ontology needs to be dynamic and filled in by an external source, it is necessary to find an alternative.

Therefore the SWRL rule in 6.2 was changed to 6.3. Instead of verifying that if a TableComponent is not a StopsTableComponent, this was changed for premise that verifies if the TableComponent has a StopTime item.

$$\begin{aligned}\forall x, y, z:\ & \mathsf{Table}(x) \land \mathsf{TableComponent}(y) \land \mathsf{hasPart}(x, y) \land \mathsf{routeOrientation}(x, z)\\ & \land \mathsf{hasOrientation}(y, z)\\ & \land (\geq 1\ \mathsf{hasItemInColumn.StopTime} \sqcup \geq 1\ \mathsf{hasItemInRow.StopTime})(y)\\ & \rightarrow \mathsf{TripTableComponent}(y)\end{aligned}$$

Rule 6.3: Rule for defining trips individuals

Additionally, it is necessary to create a property to describe the relationship between the class StopTime and the newly created TripTableComponent. This can be done by defining the rule 6.4

$$\forall x, y : \ \text{TripTableComponent}(x) \wedge (\text{hasRow} \sqcup \text{hasColumn})(y, x) \rightarrow \text{hasTrip}(y, x)$$

Rule 6.4: Rule for inferring hasTrip property

When defining axioms/rules for inferring properties in OWL, there are two different approaches. The first is to use SWRL rules as represented, for example, in rule 6.1 for creating defining new properties/roles using first order logic. A different approach that does not require SWRL is to apply complex role inclusion axioms to construct complex roles from more straightforward / atomic properties. This can be achieved by using property chains.

However, since some premises of the rule can be class assertions, to express the rule in OWL, its necessary to use a transformation called rolification. The rolification of a concept A results in a new role R_A defined by the axiom $\text{A} \equiv \exists \text{R}_A.\text{Self}$ (Krisnadhi, Maier, and Hitzler 2011). As such by defining the axiom 6.6 it is possible to transform the SWRL rule 6.4 into the OWL axiom 6.7 and axiom 6.8 where hasTrip has the domain StopTime and range TripTableComponent.

$$\text{TripTableComponent} \equiv \exists \text{R}_{Trip}.\text{Self} \tag{6.6}$$

$$\text{hasRow} \circ \text{R}_{Trip} \sqsubseteq \text{hasTrip} \tag{6.7}$$

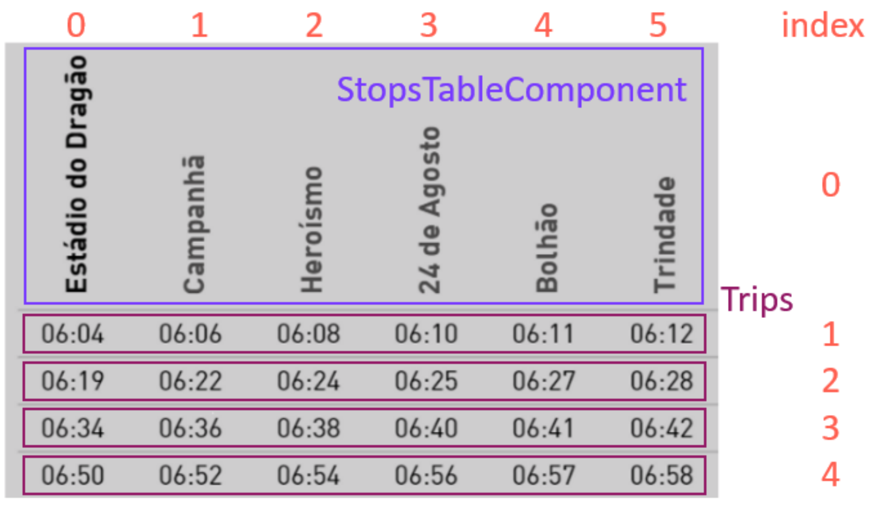
$$\text{hasColumn} \circ \text{R}_{Trip} \sqsubseteq \text{hasTrip} \tag{6.8}$$

The technique of rolification for defining complex roles can also be applied to other concepts of this ontology. The example expressed in axiom 6.9 uses rolification for defining the class of both individuals. The axiom 6.9 establishes the property hasStop between individuals with the classes StopTime and Stop that shares the same column.

$$\text{R}_{StopTime} \circ \text{hasColumn} \circ \text{hasItemInColumn} \circ \text{R}_{Stop} \sqsubseteq \text{hasStop} \tag{6.9}$$

Unfortunately, property chains cannot be used for data properties. For this reason, inferring data properties still require the creation of SWRL rules. As an example of an important data property in the integration process is the property of stopSequence. This property represents in GTFS the order of stops for a particular trip, and it is represented as a property of a stop time instance.

The stopSequence is co-related to the index of the column or row in which that stopTime is located, and that is perpendicular to the TripTableComponent. This property is easily visualized using the timetable in figure 6.6. This figure represents the timetable in figure 6.5 with the domain concepts of StopsTableComponent, Trip, and index identified. In this example, every component, its simple to verify that the value of stopSequence is equal to the index of the TableColumn.



Estádio do Dragão	Campanhã	Heroísmo	24 de Agosto	Bolhão	Trindade
06:04	06:06	06:08	06:10	06:11	06:12
06:19	06:22	06:24	06:25	06:27	06:28
06:34	06:36	06:38	06:40	06:41	06:42
06:50	06:52	06:54	06:56	06:57	06:58

Figure 6.6: Example of online timetable with trips and index representation

Since property changes cannot be used for data properties, the following SWRL rules (6.5 and 6.6) were defined to handle this property's inference.

$$\forall x, y, z, i : \ \text{StopTime}(x) \land \text{hasTrip}(x, y) \land \text{TableRow}(y) \land \text{hasColumn}(x, z) \\ \land \ \text{index}(z, i) \rightarrow \text{stopSequence}(x, i)$$

Rule 6.5: Rule for defining stopSequence if trip is a row

$$\forall x, y, z, i : \ \text{StopTime}(x) \land \text{hasTrip}(x, y) \land \text{TableColumn}(y) \land \text{hasRow}(x, z) \\ \land \ \text{index}(z, i) \rightarrow \text{stopSequence}(x, i)$$

Rule 6.6: Rule for defining stopSequence if trip is a column

6.4 Backend service

The components described thus far are able to scrape the data from websites, add context to that data by naming the HTML elements extracted, and describe the knowledge and structure of the information presented in those visual formats.

However, in order to transform the data extracted to knowledge, it is first necessary to transform the data into information. In other words, it is necessary to create the ontology Abox from the extracted data.

This process of creating information and mapping it to the Timetable ontology Abox is one of the Backend service responsibilities.

6.4.1 Backend information extraction

After receiving the response of a spider's execution, the data will be handled by a group of algorithms that implement the "MiningAlgorithm" interface (refer to 5.17). The tags previously defined in the execution profiles will be used to identify which part of the data will be used by each algorithm. As defined in the design, the algorithms are organized in a tree structure, and the root algorithm is demonstrated in code 6.8.

```python
class Timetable:
    def apply_algorithm(self, algorithm : MiningAlgorithm):
        algorithm.do_algorithm(self.data, self)
        return self  # for optionally chaining method callings


class TimetableAlgorithm(MiningAlgorithm):
    def do_algorithm(self, data, instances=None):
        if instances is None:
            instances = timetable.Timetable(data=data)

        return instances.apply_algorithm(AgencyAlgorithm()).\
            apply_algorithm(ServiceAlgorithm()).\
            apply_algorithm(DirectionAlgorithm()).\
            apply_algorithm(TableAlgorithm())
```

Listing 6.8: Timetable Algorithm root (Python).

The leaf algorithms aim to define the properties of a timetable object using the retrieved data. These objects will then be persisted, and mapped into the Timetable ontology, when a request for that agency is received.

These leaf mining algorithms vary in complexity, depending on the portion of the data that it targets. The simpler algorithms target sections of the data that translate into a single element on the ontology. For example, the algorithm in the code snippet in 6.9 retrieves the portion of the HTML data identified in the execution profile as a "direction" and extracts the text. The parsing library lxml[9], which is the same library used by Scrapy selectors, is used by the Backend service to handle the HTML elements received from the spiders.

```python
from lxml import html

class DirectionAlgorithm(AlgorithmStrategy):
    def do_algorithm(self, data, instances):
        direction_data = data["timetable"]["direction"]
        tree = html.fromstring(direction_data)
        text = tree.xpath('//text()')
        instances.Direction = format_string_list(text, "_"
    )
        return instances
```

Listing 6.9: Direction Algorithm (Python).

In contrast, TableAlgorithm and ServiceAlgorithm are more complex. The complexity from the TableAlgorithm comes from having to create multiple instances from an HTML tree. An HTML table elements instantiate a list of columns, a list of rows, a list of stops, and a list of stop_times. This algorithm is presented in the code 6.10 and 6.11

[9]Documentation for the lxml parsing library can be found at `https://lxml.de/`

```python
class TableAlgorithm(AlgorithmStrategy):
    def do_algorithm(self, data, instances):
        table_data = data["timetable"]["table"]
        tree = html.fromstring(table_data)
        tr_tags = tree.xpath('//tr')
        for row_index, tr in enumerate(tr_tags, start=0):  #
    iterates the rows
            td_tags = tr.xpath('./td/text()')
            row_name = instances.create_row(row_index)
            if td_tags:
                for column_index, td_text in enumerate(td_tags,
    start=0):   # iterates the columns
                    column_name = instances.create_column(
    column_index)
                    self.instantiate_element(td_text, instances,
    row_name, column_name)
        return instances
```

Listing 6.10: Table Algorithm (Python).

For each row and column combination, the algorithm calls the instantiate_element. This method, presented in 6.11, uses the regular expressions also defined in that code snippet to select which of the TableItem sub-classes the element is. These regular expressions are presented in the table 6.6 with a brief description.

```python
stop_time_regex = re.compile(r'\d{1,2}(:|,)\d\d')
stop_regex = re.compile(r'^[a-zA-Z]+[^0-9]+$')
empty_regex = re.compile(r'^\.*$')

@staticmethod
def instantiate_element(element, instances, row, column):
    element = element.strip()
    regex_case = [(stop_time_regex, instances.create_stop_time),
                  (stop_regex, instances.create_stop),
                  (empty_regex, instances.create_empty)]

    for regex, case in regex_case:
        if re.search(regex, element):
            case(element, row, column)
```

Listing 6.11: TableItem subclass selection (Python).

Table 6.6: Regular expressions for identifying table items

Item Type	Regular expression	Brief Description
Stop	`^[a-zA-Z]+[^0-9]+$`	Items that start with a letter without containing numbers
Stop Time	`\d{1,2}(:|,)\d\d`	Items with one or two numbers followed by a separator and two numbers (e.g. "1:30" and "11,30"). For military time/24-hour clock
Empty item	`^\.*$`	Filler items usually expressed in tables as a string of dots

The ServiceAlgorithm also takes advantage of regular expressions to identify the days of the week where a service is running. However, due to several informal representations of services, there exist ambiguity in the extracted data. Some agencies might present the service as "Saturday and Sunday" while other agencies use "Weekend". The responsibility of the ServiceAlgorithm is to define a formal format easily mappable to the Timetable ontology.

An approach using lexical analysis was applied to tackle this problem. For this process, a dictionary (defined as a thesaurus) contains the alphabet of tokens as the keys with the values representing a list of regular expressions. The thesaurus used is available in appendix B.

The algorithm applies these regular expressions to the text strings extracted from the service data extracting the known tokens. Since the tokens' order is important, a string without the sub-strings not identified by the regular expressions is created. The result is a string that follows the formal grammar 6.1.

$$\begin{aligned}
\langle\text{expression}\rangle &::= \langle\text{day}\rangle\langle\text{term}\rangle \mid \langle\text{weekday}\rangle\ \textit{to}\ \langle\text{weekday}\rangle\langle\text{term}\rangle \\
\langle\text{term}\rangle &::= ,\ \langle\text{day}\rangle\langle\text{term}\rangle \mid ,\ \langle\text{weekday}\rangle\ \textit{to}\ \langle\text{weekday}\rangle\langle\text{term}\rangle \mid \epsilon \\
\langle\text{day}\rangle &::= \langle\text{weekday}\rangle \mid \textit{Holiday} \\
\langle\text{weekday}\rangle &::= \textit{Monday} \mid \textit{Tuesday} \mid \textit{Wednesday} \mid \textit{Thursday} \mid \textit{Friday} \\
&\quad\ \mid \textit{Saturday} \mid \textit{Sunday}
\end{aligned}$$

Grammar 6.1: Formal grammar of service string

The string which follows the grammar is then transformed into a list of weekdays (or "holiday") according to the code in snippet 6.12.

```python
def string_to_week_days(s: str):
    # Converts string to a list of week days
    s_list = s.split(",")
    week_days_result = set()
    for string in s_list:
        if "to" in string:
            week_days_result |= (set(range_str_to_week_days(string)))
        else:
            string = string.replace(" ", "").lower()
            if string in week_days.keys() or string == "holiday":
                week_days_result.add(string.capitalize())
    return list(week_days_result)

def range_str_to_week_days(str_list):
    # Converts a range string to a list of week days
    temp = list(filter(lambda string: string.lower() in week_days.
   keys(), str_list.split(" ")))
    if len(temp) >= 2:
        start, *_, end = temp # list unpacking
        for i in range(week_days[start], week_days[end]+1 if
   week_days[end] > week_days[start] else week_days[end]+8):
            yield day_int_to_str(i)
```

Listing 6.12: Tranform string in list of days (Python).

6.4.2 Information persistence and external information requests

After extracting the information from the data and parsing it into the Timetable python object, the information needs to be stored in order to outlive the process that created it. Since each item received by the spiders' execution generates a Timetable object in a one to one conversion, and the desired output files contain information of multiple timetables of the same agency, by storing this information in a database, it is possible to query and group it when filling the ontology's Abox.

This solution uses a MongoDB database for saving the Timetables as JSON documents. Transactions with the database are done using the MongoDB python driver encapsulated by Repository objects.

Unfortunately, in most situations, the information extracted from the Timetable formats is not enough to fulfill the required outputs.

The solution to this problem is to extract this information from an additional source. The most relevant example is in identifying the location of stops. Most agencies represent their stops in a timetable without machine-readable information of the stops' locations.

For this concrete example, the google places API was selected as the external API selected to handle the stops' geocoding. This API allows for the retrieval of a place's location from textual input. Additionally, each place is categorized by place type, and the API can restrict the search's scope using the type as a query parameter. The place types that identify stops are the following:

- train_station
- bus_station
- subway_station
- light_rail_station
- transit_station

Each of these place types identifies a different type of vehicle. The exception is the transit_station type, which is the root type of this sub-domain.

The requests to the API uses the stop name as the query input and the transit_station type for restricting the results. These results are then persisted using a Location object and a Location Repository.

Unfortunately, most of the context known about the stop is not passed to the external API. This results in occasionally receiving incorrect location values from the API response. The figure illustrates a concrete example of this situation. It shows two stops with the same name and just a distance of 33,6 Km. A query for the stop "Trindade" near "Paços de Ferreira" will instead return the "Trindade" near the shore.

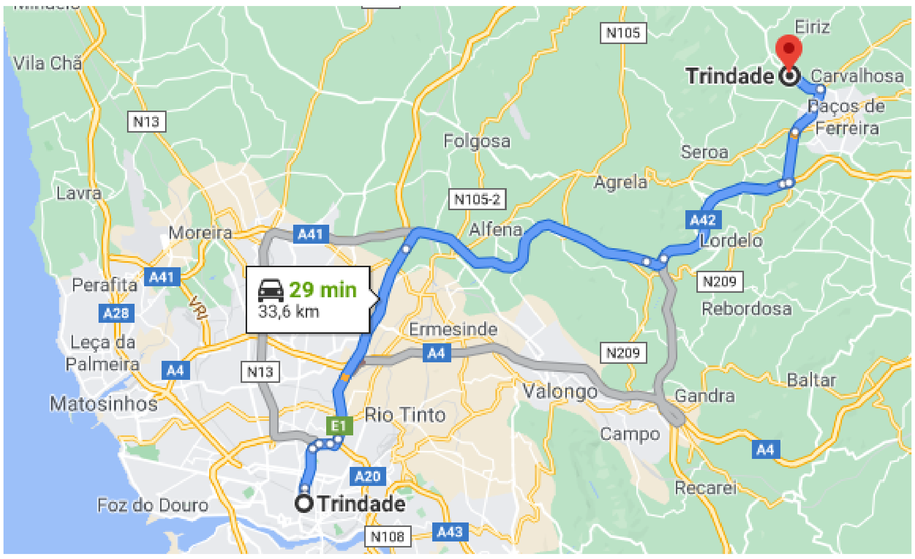


Figure 6.7: Example of ambiguity in stops' names

This problem with the ambiguity was identified, no solution for resolving the ambiguity was studied since the problem did not belong to the thesis scope.

A workaround was implemented, by having the locations persisted in a separate document, and using the agency's name and the stop's name as combined keys, it facilitates the managements of the location content and allows fixes to any incorrect values.

Thus, when searching for a stop location, the system first checks its database, and if it finds no results, it requests the external API for that information. After receiving that response, the system saves it to its database for further location searches.

```python
class LocationService:
    def get_location(self, agency, location_name):
        query_result = self.repository.
    read_by_agency_and_location_name(agency, location_name)
        if query_result:
            return LocationCoordinates.build_from_json(query_result)

        location = LocationCoordinates(agency=agency, location_name=
    location_name)
        result = self.geocoding.request_google_places_location(
    location_name)
        location.handle_geocoding_response(result)
        self.repository.create(location)
        return location
```

Listing 6.13: Location service (Python).

Although not ideal, this approach also helps with reducing the dependency that the Backend service has with the external API. Reducing the number of requests to the external API also reduces the monetary cost, since the Places API has a charge by request policy.

6.4.3 Ontology alignment and Backend API

The Backend service provides an API that allows a user to retrieve a specific agency's information in a specific format.

As previously mentioned, the selected formats were the OPT format and the GTFS format. Consequently, the API consists of two HTTP endpoints, each representing one of those formats. Each endpoint also requires the agency name as a query parameter.

When a request is received, the system retrieves the agency's information specified in the parameter from its database. The information is mapped into the ontology, and the reasoner is executed.

To handle the ontologies, the Backend uses the owlready2 package that integrates the OWL ontology model with the Python object model (Lamy 2017). The Backend is then able to create the ontologies Abox using an object-oriented approach.

The package provides the functionality of executing the Hermit reasoner or the Pellet reasoner. Unfortunately, in the current version, the general class axioms are not correctly imported by the owlready2. The owlready2 package uses close word assumption for defining knowledge, taking advantage of the python programming language. As a solution for the failing imports, the general class axioms were also implemented under a close world assumption, introducing redundancy to the operation.

It is important to note that this is not the ideal approach. Although redundancy can be beneficial in some engineering processes, such as in improving fault tolerance (Carzaniga, Gorla, and Pezzè 2009), when it comes to a software solution, it can impact the maintainability of a project throughout its lifetime.

With the Timetable Ontology Abox able to import the agency instances, next it is necessary to define the mappings between the Timetable Ontology and the OPT ontology and the GTFS ontology. The set of correspondence/matches between entities of two ontologies are called alignments (David et al. 2011). A simplified version of the Expressive and Declarative Ontology Alignment Language (EDOAL) (Euzenat 2015) was used to create the alignments. Each alignment is represented by an RDF/XML file containing mappings between the entities of two different ontologies and their relationships. To handle RDF parsing, the Backend uses the rdflib[10] library.

The AgreementMakerLight ontology matching system (Faria et al. 2013) was used to automatically create the base alignment file between the ontologies. Some matches were added manually, since the system could not detect the less straightforward matches, for example, TripsTableComponent to Line.

For creating the files, the systems follow the process previously illustrated in the design chapter using figure 5.18. The method for creating the individuals in the target ontology is displayed in algorithm 6.4, where the input variables represent the ontologies and the alignment files, located in memory and properly formatted.

[10]Documentation for rdflib can be found at `https://rdflib.readthedocs.io/en/stable/`

Algorithm 6.4 Filling target ontology's Abox

```
1: Input: The timetable ontology $origin\_ontology$, the target representation ontology, $target\_ontology$ and
   the matches named $alignment$
2: Output: Filled target ontology
3:
4: for each $match \in alignment$ do
5:     $origin\_entity \leftarrow match.origin$
6:     $target\_entity \leftarrow match.target$
7:     if Entities not null and exist in ontology then
8:         $target\_entity.equivalent\_to.origin\_entity$
9:         for each $origin\_individual \in origin\_ontology.get\_instances\_of(origin\_entity)$ do
10:            $target\_individual \leftarrow$ new $target\_entity(origin\_individual)$
11:            $target\_individual.equivalent\_to.origin\_individual$
12:        end for
13:    end if
14: end for
```

Lastly, the target ontology is used by a serializer to create the desired files locally. These files are then zipped using the zipfile[11] python module and returned as the response.

6.5 PDF extraction

Considering the percentage of agencies that use PDF files to represent their timetables, the system could not be complete without an approach for handling that type of data.

As previously mentioned, the camelot/Excalibur project was selected as the base for the PDF extraction prototype.

The changes made to the forked version of the Excalibur project were the following:

- Add asynchronous communication support
- Add subroutine to locally download the PDF files
- Change user interface to allow user input related to the timetables service and name
- Add duplication identification and handling

The first step for adding the support for asynchronous communication was to define a new thread for the queue listener.

Besides communicating as a consumer, the PDF extraction module acts as a publisher, sending the extracted information to the Backend service after a successful extraction. The code responsible for asynchronous communication is presented in 6.14

[11] Documentation about zip file can be found at `https://docs.python.org/3/library/zipfile.html`

```python
def consume():
    # ... create channel
    channel.basic_consume(queue='item_pdf', on_message_callback=
    items_queue_callback, auto_ack=True)
    channel.start_consuming()

def publish(message):
    # ... create connection, channel, exchange and queue
    channel.basic_publish(
        exchange="excalibur",
        routing_key="item",
        body=message,
    )
    connection.close()
```

Listing 6.14: Excalibur asynchronous communication (Python).

For downloading the PDF files locally, the coroutine in the code snippet 6.15 was created. This coroutine has to run in a non-blocking fashion since otherwise, it would block the queue listener thread and receive a timeout from the message broker.

```python
def download_file(file_url):
    name = file_url.split("/")[-1]
    request = requests.get(file_url)
    print(f"here - {request.content}")
    file = BytesIO(request.content)
    return file, name


async def download_files(file_urls, agency_name):
    for url in file_urls:
        file, name = download_file(url)
        print(f"content = {file}")
        content = FileStorage(stream=file, name=name, filename=name,
    content_type='application/pdf')
        create_files(content, agency_name=agency_name, url=url)
```

Listing 6.15: PDF downloader (Python).

The Excalibur project only had support for importing PDF using the WebApp graphical interface. The functionality of importing the PDFs had to be separated from the API endpoint for manual importation.

After receiving a message from the spiders via the queue listener, if the message contains any PDFs' URLs, the callback function will download the files and use the function for importing PDF, allowing to automate the importation of files.

During an importing the PDFs are transformed into image files. These image files are normally used by Excalibur to present the file in the user interface. By comparing the generated image files with the Pillow package, a new functionality of duplication identification was added. If a image from a newly imported file is detected as a duplicate, the system flags the file and warns the user.

Other small changes had to be done both in the Frontend and the Backend API, to accommodate the user input related to the timetables service and name, but those mostly include changes to the HTML templates and the download endpoint.

6.6 Putting everything together with asynchronous communication

As previously defined, the system applies an asynchronous communication approach using messaging with an intermediary component responsible for handling message distribution, also known as a message broker pattern. This improves the system's reliability because considering that the time of some operations is variable, for example, a spider execution, it would be unreasonable for a component to actively wait for the recipients' response.

The system uses a RabbitMQ instance to implement the message broker pattern. Figure 6.8 represents the exchanges and queues used by the system's components. This is an implementation of the publisher/subscribe pattern, where the publishers are represented in the figure as the producers and the subscribers as the consumers.

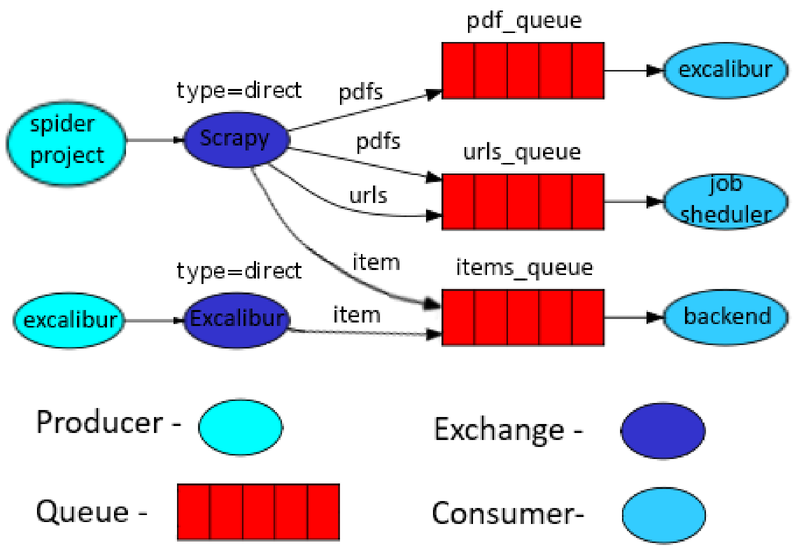


Figure 6.8: RabbitMQ exchanges and queues routing

Moreover, in this system, the exchanges are direct exchanges, meaning the communication occurs using multicast via a binding routing key. This approach differs from a simpler publisher/subscribe that can also be implemented in RabbitMQ using fanout exchanges, allowing for mindless message broadcast.

Chapter 7

Evaluation of the solution

This chapter contains a description of methodologies, criteria, metrics, and measurements to evaluate the solution.

7.1 Information retrieval criteria

For evaluating an information retrieval solution, some standard criteria and measurements allow verifying the system's effectiveness.

When executing an information retrieval system, two parameters are responsible for the system's effectiveness. The first is Indexing exhaustivity; this reflects the degree to which an IR system recognizes and indexes the terms present in the documents. A higher indexing exhaustivity results in a bigger, but more disperse collection of terms (Gudivada et al. 1997).

On the other hand, the second parameter is term specificity, which refers to the breadth of the terms used for indexing. The less specific the terms are, the higher the number of documents is retrieved. However, this also results in the retrieval of documents irrelevant to the search. By increasing the term specificity, the relevancy of the documents increases, but there is also the possibility of losing important documents (Gudivada et al. 1997).

To measure these two criteria the following metrics are defined:

- Recall, which is the ratio of the number of relevant documents retrieved to the total number of relevant documents in the collection.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{7.1}$$

- Precision, which is the ratio of the number of relevant documents retrieved to the total number of documents retrieved.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{7.2}$$

7.2 Ontology criteria

Brank, Grobelnik, and Mladenic (2005) defines ontology evaluation as "the problem of assessing a given ontology from a particular criterion of application". This evaluation is done from a verification perspective and a validation perspective.

Ontology verification allows the detection of errors in the encoding, allowing to verify the quality of the ontology specification (Gómez-Perez 2004).

On the other hand, ontology validation aims to confirm if the ontology fulfills its purpose. The objective is to validate if the formal definition of the ontology is compatible with the real world representation under the context where the ontology is applied (Gómez-Perez 2004).

The evaluation can then be performed using a diverse set of criteria, that either focus on the ontology correctness or the ontology quality (Hlomani and Stacey 2014). Denny (2009) identifies several criteria that can be used for verification and validation of ontologies. The criteria are divided as follows:

- Accuracy focus on if the ontology correctly captures and represents the aspects of the real world. It is one of the criteria used for validating the ontology correctness and the same metrics used on information retrieval (precision and recall) can be applied to an ontology and its knowledge base.
- Adaptability refers to the capacity of an ontology to be used in different scenarios. An ontology that is easily extensible and can easily support the addition of new axioms is more adaptable than an ontology less open to change. This criterion is used to evaluate the ontology from a quality perspective, analyzing the choices selected in the design phase, and as such, the study of this criteria can be done by verifying metrics showcasing the coupling (e.g., number of external classes referenced) and the cohesion (e.g., Average Depth of Inheritance Tree of Leaf Node) of the ontologies classes (Hlomani and Stacey 2014).
- Clarity refers to how the ontology communicates the intended meaning of the defined terms effectively. Hlomani and Stacey (2014) suggests the usage of word sense as a quantifiable measurement to these criteria.
- Completeness or competency, as the name suggests, is a criterion focused on how complete the ontology is when compared to the domain. This criterion can be quantified by measuring the coverage of the domain.
- Computational efficiency is a quality criterion that is correlated with the size and complexity of the ontology. Thus, those measurements can be used to quantify the computation efficiency of an ontology.
- Conciseness measures "how weak are the assumptions regarding the ontology's underlying philosophical theory about reality" (Denny 2009). It focuses on the axioms' relevancy and how significant those axioms are to the subject being specified in the ontology. Using an extreme example, if a domain ontology for medicine contains axioms about software engineering, it is not concise.
- Consistency or coherence is an evaluation criterion for verifying the correctness of an ontology. When examining this criterion, the target is to identify inconsistencies and errors in the ontologies axioms or terms. The number of inconsistent results can be used as a measurement for this criteria.
- Organizational fitness or commercial accessibility focuses on the commercialization of the ontology and attempts to measure the suitability of the ontology among the potential stakeholders.

7.3 Evaluation methodology

Considering the criteria presented is then possible to define a concrete methodology for evaluation.

The first characteristic to evaluate is the quality of the results of the solution. In an ideal scenario, the information extracted should be as precise as possible. Although a low value of recall is never desirable, for the context where the solution will be applied, a low value of precision is worst. In this context, it is better to have missing information than incorrect information.

To measure these criteria of information retrieval, tests should be designed where the spiders run in previously selected web sites, and the results are compared to actual values. Since the crawling process is deterministic, to evaluate each spider, only a single execution is required. An acceptable threshold should be specified to each criterion and validated after an execution.

Besides using the static criteria of ontology evaluation, a group of software tests should be designed to ensure the mapping is performed correctly to evaluate the information integration. Besides unit testing, a set of integration tests can be used to guarantee that the information retrieved by the spiders is correctly mapped to the canonical ontology or the domain ontology.

7.3.1 Evaluating quality and resource cost

This project aims to ensure that the quality of this solution is on par with the current manual process and that the resources necessary for this task are reduced.

As such, a methodology/approach should be defined to address these concerns.

A possible approach would be to use a survey to validate that the solution developed by questioning the company associates. However, since in this context, the company has a low number of employees, the results can be skewed due to a low number of responses.

A different approach is to compare the quality and resource cost to values before the installation of the solution, allowing the verification of improvements or decline in those values. The problem with this approach is that the results were obtained under different conditions irrelevant to the solution, which might have a correlation with the values and skew the results. This approach also requires the existence of data about the process before the installation of the solution.

The ideal approach would be to perform A/B testing, a test where two groups are created, the first uses the solution, and the second acts as control by not using the solution. Both groups' quality and resource costs are measured and compared to prove or disprove the optimizing/enhancement hypothesis.

Unfortunately, it was not possible to apply A/B testing in the current context of the OPT business. The alternative proposed is to create two different experimentation scenarios using the same data sample. In the first scenario, the data is to be converted into files using a manual approach, while in the second, the files are created using the developed solution. This differs from A/B testing, by previously selecting the data sample and by having the same person perform both tasks.

7.4 Experimentation scenarios

It is then necessary to consider the method of evaluation a scenario resource usage. Considering that time is the most relevant resource used when creating the files (more specifically, the time spent by human workers), the experiment scenarios designed aim to measure the time required to perform the same transformation using both approaches.

As previously defined, the data can be transformed manually or automatically, and each of these methods ensures a specific scenario. However, the process for obtaining information from a PDF is different from the process for obtaining it from an HTML. Therefore, four different experimentation scenarios were created.

The experimentation scenarios are group into two pairs. Each pair represents a different input format and contains the two different methods of transformation, either manual or automated. The scenarios defined are the following:

- Create OPT files manually from HTML files
- Create OPT files from HTML files automatically by using the created solution
- Create OPT files manually from PDF files
- Create OPT files from PDF files automatically by using the created solution

The OPT output file was selected since it is the internal file format created by the company, and it is the output format used to integrate external information into the company's systems. Consequently, without an automated process, this format would be manually created.

Each scenario contains the set of steps required for the extraction. In the experiments, the time required for each step is measured. The time required for manual steps is rounded up to the minute, while the steps performed by the software solution are rounded up to the second.

7.4.1 Create OPT files from HTML files automatically

For creating the files from HTML files using the developed solution, it is necessary first to define the spiders' execution. The steps after the configuration phase are automated by the spiders and Backend.

Since the information referring to a stop location has the issues detailed in chapter 6.4.2, the last step involves "validating and fixing location coordinates". This task combines a manual and an automated approach resulting in the measurements to be rounded up to the minute.

Luckily, there exists a vast amount of tools using the GTFS format for different purposes[1]. Some of these tools provide validation or visualization of the information serialized in the GTFS files. Since tools that provide the validation of OPT files do not exist, and since the developed solution can generate GTFS files portraying the same information as the OPT files, the following GTFS tools were selected to detect incorrect information.

[1]List of GTFS tools can be found at `https://github.com/CUTR-at-USF/awesome-transit`

- transitfeed-feedvalidator[2] - Validator tool maintained by google, which validates GTFS structure and content. It is able to detect incorrect stop locations by calculating the transport travel speed and verifying if it is physically achievable.
- gtfs-to-html[3] - Visualizer tool to create a HTML pages from GTFS files. The pages display a map which allows the detection of incorrect locations.

The steps required to extract the timetables from HTML files are presented in table 7.1.

Table 7.1: Steps for automatic generation of OPT files from HTML

Step	Description	Step type
Configure url extraction spider	Create the json configuration file for the url extraction spider and add the event to the events' dictionary	Manual
Configure scraping spider	Create the json configuration file for one of the spiders, creating the XPath queries	Manual
Run Url extractor	Execute the url extractor with the new spider profile by using the job scheduler API	Automated
Run Scraping spider	Automatic execution of the url extractor with the new spider profile by using the job scheduler API	Automated
Backend's ontology mapping and file generation	Request files from Backend API	Automated
Validating and fixing location coordinates	Validate the information in the generated files and fix erroneous information regarding the location	Manual and Automated

It could be argued that the step of "validating and fixing location coordinates" can be separated into the two different steps of "validating" and "fixing". However, since in some situations, these two tasks follow an iterative process, this task is presented as a single step.

7.4.2 Create OPT files from PDF files automatically

The process of creating the files from PDF files using the developed solution is similar to the previous scenario. As with the previous scenario, it is necessary first to define the spider's execution. For PDF extraction, the user only creates the URL extractor profile.

By specifying in the profile that PDFs are allowed, the URL extractor collects the URLs referencing PDFs and publishes it. The Excalibur component downloads the PDF and waits for the user instructions.

At this point, the user has to select the tables to send for the Backend. The following steps are the same as the ones specified in the HTML extraction scenario.

The steps of process for extracting timetables from PDF files are presented in table 7.1.

[2]Transitfeed-feedvalidator available at `https://github.com/google/transitfeed`

[3]gtfs-to-html available at `https://github.com/BlinkTagInc/gtfs-to-html`

Table 7.2: Steps for automatic generation of OPT files from PDFs

Step	Description	Step type
Configure url extraction spider	Create the json configuration file for the url extraction spider allowing pdf formats	Manual
Run Url extractor	Execute the url extractor with the new spider profile by using the job scheduler API	Automated
Download PDFs	Download the PDFs found in the previous step	Automated
Select tables to extract	Use the Excalibur user interface to select the tables to extract	Manual and Automated
Backend's ontology mapping and file generation	Request files from Backend API	Automated
Validating and fix location coordinates	Validate the information in the generated files and fix erroneous information regarding the location	Manual and Automated

The task of selecting the table is categorized as "Manual and Automated" since Excalibur provides an auto table detector functionality. However, the detected tables usually require small fixes, and the user needs to select the extraction algorithm and define its parameters.

7.4.3 Create OPT files manually

As should be expected, the steps required to create the OPT files manually are independent of the original format. As such, the two manual experimentation scenarios previously identified can be defined as a single scenario independent from the method of representation used by the source, as displayed in the table 7.3. Each step represents the creation of a different file. Every step in this scenario is manual, and as such, the measured times were rounded up to the minute.

Table 7.3: Steps for manual generation of OPT files

Step	Description
Find the information	Find the target timetable
Search the stops' location	Search the geographic location for each stop
Create the Stops file	Create the Stops file with the information on the stops name and location.
Create the Lines file	Create the Lines file with the name, and stop sequence
Create the Schedules file	Create the Schedules file with the Day types and interval dates
Create the Passings file	Create the Passings files including the stop time, direction and stop code

7.4.4 Agencies and sources selection

Considering that the process of manually creating a timetable is time-consuming, only two timetables were selected for each source type (HTML or PDF). Each of the selected timetables is provided by a different agency. This ensures that the steps of creating the spiders' profiles are done in every experiment.

Although the manual scenarios only focus on one timetable, several steps in the automated scenarios cover multiple timetables. For example, the spiders' execution profile for an HTML extraction must be able to collect the data of every timetable on the agency's website, even if the following manual steps only target a specific timetable.

Regarding the selection, the chosen agencies and timetables are represented in the table 7.4

Table 7.4: Timetable selected for evaluation

Agency	Route name	Timetable url	Source Type
Gondomarense	Porto - Valchão	`http://www.gondomarense.pt/horarios-2006/index.htm`	PDF
AVPacense	Paços de Ferreira – Valongo – Porto	`http://www.avpacense.pt/PACOS%20DE%20FERREIRA%20-%20VALONGO%20-%20PORTO.pdf`	PDF
AVMinho	Póvoa de Varzim - Porto	`http://www.avminho.pt/horarios`	HTML
Rodonorte	Felgueiras - Porto	`http://beware.pt/IP/MotorBusca/rodonorte/Horario.aspx?id=26553`	HTML

7.5 Evaluation results and discussion

This section documents the results of the timetables' extractions. For the timetables selected, the time required for each step was measured. Since the timetables and the respective URLs were already selected, the default value of one minute was attributed to the step of "finding the information source" for the manual scenario.

Starting with the extraction from the Gondomarense agency, the time measured for each step is represented in table 7.5.

Table 7.5: Steps' time for automated extraction and manual extraction of Gondomarense timetable.

Automated steps	Time (mm:ss)
Configure URL extraction	6:00
URL extractor execution	0:13
PDF Download	1:52
Table selection	5:00
File generation	0:40
Validation and fix	3:00
Total	16:45

Manual steps	Time (mm:ss)
Find the information	1:00
Search the stops' location	7:00
Create the Stops file	4:00
Create the Lines file	5:00
Create the Schedules file	2:00
Create the Passings file	16:00
Total	35:00

A different view of the times measured from the automated extraction is presented in the graph in figure 7.1. With this visualization, it is possible to verify that 83,6% of the time required for this extraction was spent in the solutions manual steps, in contrast with the remaining 16,4% used for the automated tasks. It is important to note that most of these measurements can differ from several conditions and depending on the user familiarity with the system. Another variable is the download speed of the internet connection. From this extraction, a total of 95 PDF files were retrieved totaling 34,9 Megabytes.

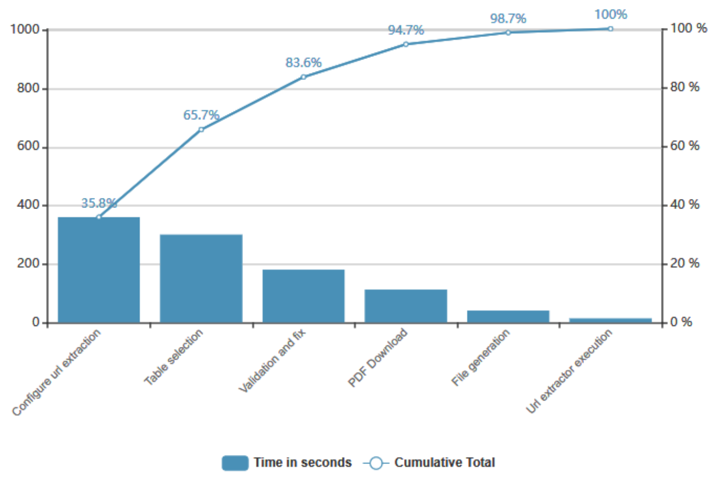


Figure 7.1: Time in seconds for step in automated extraction of ETG

Considering that the automated extraction retrieved a total of 95 PDF documents from the source website and that the number of PDF documents in the website that contained timetable information was 48, the Recall and Precision for this agency were calculated and displayed in the table 7.6.

Table 7.6: Timetable selected for evaluation

True Positives	40
False Negatives	8
False Positives	55
Recall	$\frac{40}{48} \approx 0,833$
Precision	$\frac{40}{95} \approx 0,421$

The timetable for the route "Porto - Valchão", which was used in the manual scenario and also selected from the PDFs generated for the manual steps, contained two tables with different services that were transformed into 4 stops, 14 trips, and 51 stop times.

The next agency also displayed information using PDF. In these scenarios, the target timetable for the manual steps was the timetable for the route "Paços de Ferreira – Valongo – Porto" from the AVPacense agency. The result time measurements are displayed in table 7.7.

Table 7.7: Steps' time for automated extraction and manual extraction of Avpacense timetable.

Automated steps	Time (mm:ss)
Configure URL extraction	3:00
URL extractor execution	0:10
PDF Download	0:25
Table selection	9:00
File generation	0:11
Validation and fix	3:00
Total	15:46

Manual steps	Time (mm:ss)
Find the information	1:00
Search the stops' location	9:00
Create the Stops file	6:00
Create the Lines file	3:00
Create the Schedules file	1:00
Create the Passings file	24:00
Total	

Similar to the extraction from Gondomarense, the manual step in the automated process required the majority. Since the spider's result found 24 PDFs, totaling to 2,08 Megabytes, the download time vastly decreased compared to the previous agency. For this execution, the manual steps' time represents 95,1% of the entire scenario, as presented in figure 7.2.

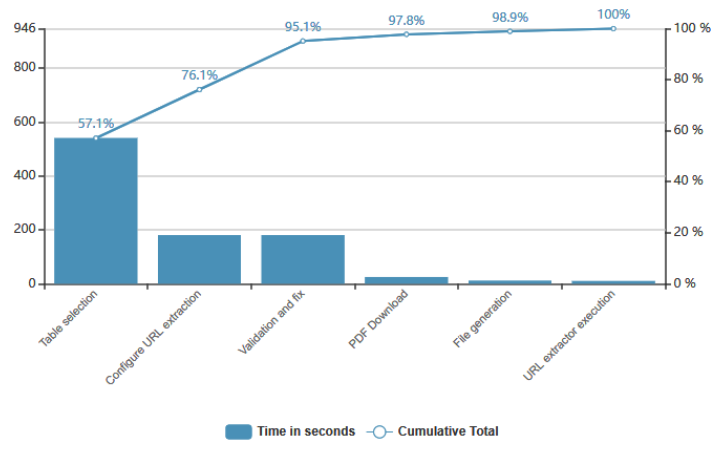


Figure 7.2: Time in seconds for step in automated extraction of AVPacense

For this agency, a total of 24 PDF documents were found, while the relevant documents available on the website totaled 16. The Recall and Precision for this scenario were calculated and are displayed in table 7.8.

Table 7.8: Timetable selected for evaluation

True Positives	16
False Negatives	0
False Positives	8
Recall	$\frac{16}{16} = 1$
Precision	$\frac{16}{24} \approx 0,667$

Proceeding to the scenarios using HTML as the source of information. Starting with the route "Póvoa de Varzim - Porto" by the AVMinho agency, the spider extraction required one simple profile for the URLExtractor spider and two distinct profiles for the scraping. The time measurements for the automated extraction and the manual extraction are represented in table 7.9

Table 7.9: Steps' time for automated extraction and manual extraction of AVMinho timetable.

Automated steps	Time (mm:ss)
Configure URL extraction	1:00
Configure scraping	32:00
URL extraction execution	0:25
Scraping execution	0:09
File generation	0:12
Validation and fix	1:00
Total	34:46

Manual steps	Time (mm:ss)
Find the information	1:00
Search the stops' location	7:00
Create the Stops file	5:00
Create the Lines file	2:00
Create the Schedules file	1:00
Create the Passings file	12:00
Total	28:00

These measurements are the first example where the manual extraction performed better from a time-consuming perspective. By analyzing the graph in figure 7.3, it is visible that the majority of the automated extraction was spent in the manual step of defining the scraping spider profiles, more specifically the XPath expressions required to extract the information.

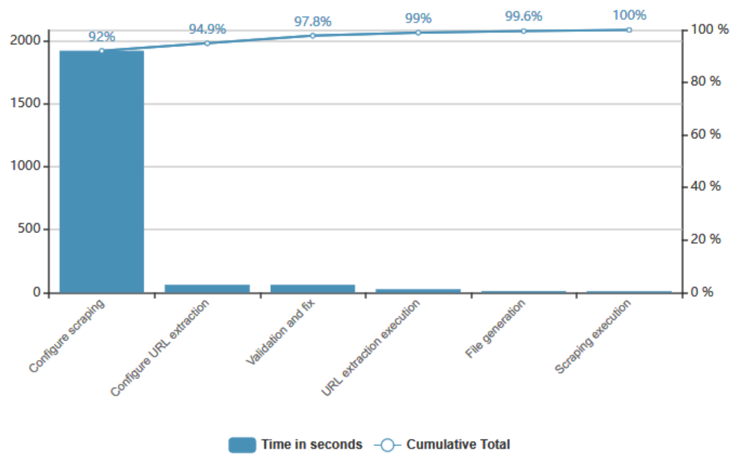


Figure 7.3: Time in seconds for step in automated extraction of AVMinho

However, while the manual scenario only serialized one of the timetable's information, the automated scenario managed to extract every timetable on the website.

This is a consequence of the simplicity in the AVMinho website. The website contains every timetable on the same web page (under the same URL) and represents every timetable using the same HTML structure. This makes it extremely easy for a web spider to extract structured data without missing a single timetable. With this in mind, the recall and precision have a perfect score, as represented in table 7.10.

Table 7.10: Timetable selected for evaluation

True Positives	58
False Negatives	0
False Positives	0
Recall	$\frac{58}{58} = 1$
Precision	$\frac{58}{58} \approx 1$

The final timetable extracted for these experiments was the "Felgueiras - Porto" and the rodonorte agency. Similarly to AVMinho, this agency uses HTML to represents their timetable. However, instead of displaying every timetable on the same page, it instead displays a list of hyperlinks, each containing a different timetable. The measurements for this timetable extractions are represented in the table 7.11.

Table 7.11: Steps' time for automated extraction and manual extraction of Rodonorte timetable.

Automated steps	Time (mm:ss)
Configure URL extraction	3:00
Configure scraping	23:00
URL extraction execution	2:17
Scraping execution	1:15
File generation	0:21
Validation and fix	7:00
Total	36:53

Manual steps	Time (mm:ss)
Find the information	1:00
Search the stops' location	12:00
Create the Stops file	6:00
Create the Lines file	2:00
Create the Schedules file	1:00
Create the Passings file	7:00
Total	29:00

Just as with the previous HTML source, the manual extraction performed better from a time-consuming perspective. Even though the configure scrape was faster than the previous example, the naming convention used by rodonorte for their stops negatively influences the geocoding API accuracy.

As expected, the configure scraping step requires the majority of the time at 62,4%, followed by the validation and fixing of the location of the stop taking 7 minutes or 18,9%, as presented in figure 7.4

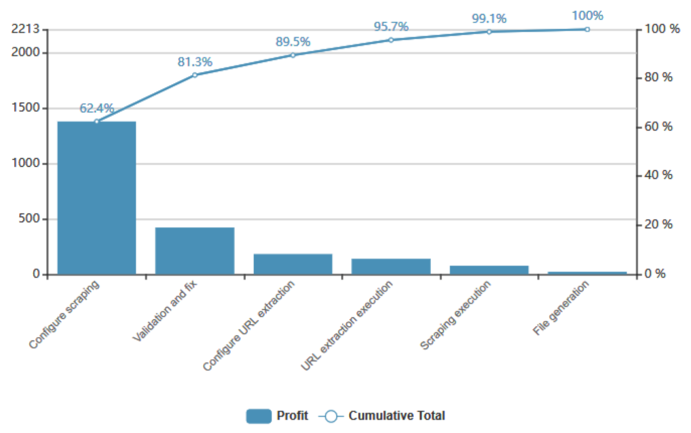


Figure 7.4: Time in seconds for step in automated extraction of rodonorte

With the analysis of retrieved timetables, it is possible to verify that number of items extracted is higher than the number of timetables, as demonstrated in table 7.12.

Table 7.12: Timetable selected for evaluation

True Positives	264
False Negatives	0
False Positives	94(scrape), 0(persisted)
Recall	$\frac{264}{264} = 1$
Precision	$\frac{264}{358} \approx 0.737$

The retrieval process detected a total of 94 incorrect timetables. Luckily, these timetables were identified by the Backend service during the mining phase.

This issue can be attributed to the inability of the "spider for multiple timetables per page" to adapt the number of items extracted if there is a varying number of services. Consequently, the spider creates and returns items with empty fields beside the actual timetable items for some pages. For this specific example, the precision of the scraping can be improved by creating a custom spider.

The results obtained allow us to understand the system's limitations and shortcomings better. The high percentage of false positives for PDF extraction indicates a necessity for an improved filtering mechanism. The current solution addresses these false positives by requiring confirmation from a user. Therefore an improved filtering mechanism can increase the system's automation/independence.

With the results from the HTML extraction, it becomes clear that the precision of this type of extraction is higher compared to the PDF. However, this type of extraction also required a lot more time for configuration, most of which was spent creating the XPath queries required for the information retrieval.

For both agencies/providers of this type, the automated approach was more time consuming that the manual extraction. Of course, the number of timetables retrieved makes the extra effort/time used worth it.

Lastly, a common issue with both types of extraction lies in the inaccuracy of the stops' geocoding operation. This specific issue led to a demand of the user time and attention to fix incorrect or incomplete values.

Chapter 8

Conclusions and future work

This work presented a practical case study where the application of existing information retrieval and integration techniques converges into the engineering, design, and development of a software system capable of converting data from the web into information, and capable of integrating the information into a pre-existing system.

While the current work is a promising start in integrating information using an automated approach, it has the potential to improve with the correct changes.

First, the current solution has a severe difference in extraction quality when comparing the source types, PDF, or HTML. The solution only takes a superficial study of what can be done when extracting PDF. The work developed on that topic comes off as an unpolished product, not ready for production in an enterprise environment. This component's simplicity was expected, considering the decision to create a basic prototype service to deal with PDF extraction. Throughout the development of the prototype, the areas that require improvements became visible.

The most crucial improvement is to provide a process for identifying and categorizing a PDF considering its contents. With this improvement, it is expected a reduction of the False Positives detected in the experimentation process, improving the overall system's precision. This improvement can also be the first step in increasing the automation present in information extraction for the PDF sources.

When it comes to the components for extracting information from HTML, a good starting point for deciding the next steps for the future work is to understand which parts require the most attention when maintaining the correctness of the system information, and as a result, are the most time-consuming.

A good start would be to improve the accuracy of the stop's geolocation process. For example, to use the context of a route when determining the location of a stop, and create complex queries to the geolocation service. This improvement should result in a reduction for the time required to the step of "validating and fixing" the stops.

8.1 Revisiting the objectives of the project

Back in chapter 1.4, we defined the objectives of this project. At the end of the document, it is essential to revisit the objectives and review if those were achieved or not. As such, an overview of the objectives and their completion level are demonstrated in table 8.1.

Table 8.1: Objectives and completion level

Objectives	Completion Level
Detect updates in the information sources	Completed
Perform information retrieval from the websites/files when changes are detected	Completed
Define input/output formats and respective mapping rules	Completed
Guarantee the consistency of information retrieved from multiple sources, allowing inference and querying of information from transports with different transport providers	Uncompleted
Provide an interface to access and export the converted information	Partially completed.

With this, its possible to conclude that the most crucial objectives of the project were successfully complete.

The exceptions are the objectives of guarantee the consistency across multiple agencies/providers and developing a graphical user interface to visualize the information before exporting.

Unfortunately, the current geocoding functionality implementation is not accurate to allow guaranteeing the consistency across multiple agencies, since the inference of these connection requires accurate stops' longitude and latitude.

Although a graphical user interface would improve the system's feedback and usability, the development of such a service was outside of the thesis' scope. As such, the system simply contains a web service for the exportation of the serialized files, justifying the partially completed score for the objective.

Bibliography

Ackoff, Russell L (1989). "From data to wisdom". In: *Journal of applied systems analysis* 16.1, pp. 3–9.

Aristarán, Manuel et al. (2018). *Tabula*. url: `https://tabula.technology`.

Baader, Franz (2003). "Appendix: description logic terminology". In: *The Description logic handbook: Theory, implementation, and applications*, pp. 485–495.

Beckett, Dave and Brian McBride (2004). "RDF/XML syntax specification (revised)". In: *W3C recommendation* 10.2.3.

Berners-Lee, Tim (2000). *Semantic Web on XML*. url: `https://www.w3.org/2000/Talks/1206-xml2k-tbl/all.htm`.

Berners-Lee, Tim, James Hendler, and Ora Lassila (2001). "The semantic web". In: *Scientific american* 284.5, pp. 34–43.

Borza, John (2011). "FAST diagrams: The foundation for creating effective function models". In: *General Dynamics Land Systems*.

Braga, Luís Miguel Barbosa (2013). "Engineering Temporal and Spatial Aspects in OWL using Patterns". PhD thesis.

Brank, Janez, Marko Grobelnik, and Dunja Mladenic (2005). "A survey of ontology evaluation techniques". In: *Proceedings of the conference on data mining and data warehouses (SiKDD 2005)*. Citeseer Ljubljana, Slovenia, pp. 166–170.

Brickley, Dan, Ramanathan V Guha, and Brian McBride (2014). "RDF Schema 1.1". In: *W3C recommendation* 25.

Carzaniga, Antonio, Alessandra Gorla, and Mauro Pezzè (2009). "Handling software faults with redundancy". In: *Architecting Dependable Systems VI*. Springer, pp. 148–171.

Chandrasekaran, Balakrishnan, John R Josephson, and V Richard Benjamins (1999). "What are ontologies, and why do we need them?" In: *IEEE Intelligent systems* 1, pp. 20–26.

Chen, Min et al. (2008). "Data, information, and knowledge in visualization". In: *IEEE Computer Graphics and Applications* 29.1, pp. 12–19.

Choi, Namyoun, Il-Yeol Song, and Hyoil Han (2006). "A survey on ontology mapping". In: *ACM Sigmod Record* 35.3, pp. 34–41.

Clark, James, Steve DeRose, et al. (1999). *XML path language (XPath) version 1.0*.

David, Jérôme et al. (2011). "The alignment API 4.0". In: *Semantic web* 2.1, pp. 3–10.

Denny, Vrandevcic (2009). "Ontology evaluation". In: *Handbook on ontologies*. Springer, pp. 293–313.

Eugster, Patrick Th et al. (2003). "The many faces of publish/subscribe". In: *ACM computing surveys (CSUR)* 35.2, pp. 114–131.

Euzenat, J (2015). "EDOAL: Expressive and Declarative Ontology Alignment Language". In: *URL: http://alignapi.gforge.inria.fr/edoal.html (visited on 22/08/2020)*.

Faria, Daniel et al. (2013). "The agreementmakerlight ontology matching system". In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, pp. 527–541.

Flesca, Sergio et al. (2004). "Web wrapper induction: a brief survey". In: *AI communications* 17.2, pp. 57–61.

Fowler, Martin (2002). *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc.

Friedl, Jeffrey EF (2006). *Mastering regular expressions.* " O'Reilly Media, Inc."

Gaizauskas, Robert and Yorick Wilks (1998). "Information extraction: Beyond document retrieval". In: *Journal of documentation* 54.1, pp. 70–105.

Gamma, Erich et al. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Education.

Gómez-Perez, Asunción (2004). "Ontology evaluation". In: *Handbook on ontologies*. Springer, pp. 251–273.

Google (2019). *GTFS general view*. url: `https://developers.google.com/transit/gtfs`.

Gruber, Thomas R (1993). "A translation approach to portable ontology specifications". In: *Knowledge acquisition* 5.2, pp. 199–220.

Guarino, Nicola (1997). "Semantic matching: Formal ontological distinctions for information organization, extraction, and integration". In: *International Summer School on Information Extraction*. Springer, pp. 139–170.

– (1998). *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*. Vol. 46. IOS press, pp. 3–15.

Gudivada, Venkat N et al. (1997). "Information retrieval on the world wide web". In: *IEEE Internet Computing* 1.5, pp. 58–68.

Heidegger, M. (1967). *Being and Time*. Blackwell. isbn: 9780631197706. url: `https://books.google.pt/books?id=S57m5gW0L-MC`.

Hlomani, Hlomani and Deborah Stacey (2014). "Approaches, methods, metrics, measures, and subjectivity in ontology evaluation: A survey". In: *Semantic Web Journal* 1.5, pp. 1–11.

Hong, Sungjun et al. (2016). "Analysis on the level of contribution to the national greenhouse gas reduction target in Korean transportation sector using LEAP model". In: *Renewable and Sustainable Energy Reviews* 60, pp. 549–559.

Horrocks, Ian et al. (2004). "SWRL: A semantic web rule language combining OWL and RuleML". In: *W3C Member submission* 21.79, pp. 1–31.

Hoyland, Christine A et al. (2014). "The RQ-Tech methodology: a new paradigm for conceptualizing strategic enterprise architectures". In: *Journal of Management Analytics* 1.1, pp. 55–77.

Hunt, Andrew and David Thomas (1999). *The pragmatic programmer*. Pearson Education.

ICEACSA-TEKIA, UTE (2019). *datex2 About*. url: `https://datex2.eu/datex2/about`.

Inkpen, Diana (2007). "Information retrieval on the internet". In: *Retrieved December* 29, p. 2013.

Kelly, Allan (2002). "The philosophy of extensible software". In: *The Internet's Coming Silent Spring*, p. 27.

Klyne, Graham, Jeremy J Carroll, and B McBride (2004). *Resource description framework (rdf): concepts and abstract syntax, 2004*.

Krisnadhi, Adila, Frederick Maier, and Pascal Hitzler (2011). "OWL and Rules". In: *Reasoning Web International Summer School*. Springer, pp. 382–415.

Kushmerick, Nicholas, Daniel S Weld, and Robert Doorenbos (1997). *Wrapper induction for information extraction*. University of Washington Washington.

Lamy, Jean-Baptiste (2017). "Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies". In: *Artificial intelligence in medicine* 80, pp. 11–28.

Lenzerini, Maurizio (2002). “Data integration: A theoretical perspective”. In: *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, pp. 233–246.

Maedche, Alexander, Günter Neumann, and Steffen Staab (2003). “Bootstrapping an ontology-based information extraction system”. In: *Intelligent exploration of the web*. Springer, pp. 345–359.

Mak, Ronald (2007). “A Highly Reliable Enterprise System for NASA’s Mars Rover Mission”. In: *Beautiful Code: Leading Programmers Explain How they Think (Oram, A., and Wilson, G.)* Pp. 319–338.

McBride, Brian (2004). “The resource description framework (RDF) and its vocabulary description language RDFS”. In: *Handbook on ontologies*. Springer, pp. 51–65.

McGuinness, Deborah L, Frank Van Harmelen, et al. (2004). “OWL web ontology language overview”. In: *W3C recommendation* 10.10.

Mehta, Vinayak (2020). *Camelot Documentation Release 0.8.2*. url: `https://readthedocs.org/projects/camelot-py/downloads/pdf/master/`.

Meyer, Bertrand (1997). *Object-oriented software construction*. Vol. 2. Prentice hall Englewood Cliffs.

Mitchell, Ryan (2018). *Web Scraping with Python: Collecting More Data from the Modern Web*. " O’Reilly Media, Inc."

Monroe, Robert T et al. (1997). “Architectural styles, design patterns, and objects”. In: *IEEE software* 14.1, pp. 43–52.

Mouncif, Hicham and Azeddine Boulmakoul (2014). “Application SIG et norme GTFS pour la conception d’un sys-tème d’information de transport multimodal: une approche SOA pour le calcul des itinéraires multimodaux viables”. In: *INTIS’2014*, p. 117.

Muslea, Ion, Steve Minton, and Craig Knoblock (1999). “A hierarchical approach to wrapper induction”. In: *Proceedings of the third annual conference on Autonomous Agents*. Citeseer, pp. 190–197.

Nekvasil, Marek (2007). “The use of ontologies in wrapper induction”. In: *Databases, Texts*, p. 132.

Noy, Natalya F (2004). “Semantic integration: a survey of ontology-based approaches”. In: *ACM Sigmod Record* 33.4, pp. 65–70.

Noy, Natasha et al. (2001). “Creating Semantic Web Contents with Protege-2000”. In: *Intelligent Systems, IEEE* 16, pp. 60–71. doi: `10.1109/5254.920601`.

Oliphant, Travis E (2007). “Python for scientific computing”. In: *Computing in Science & Engineering* 9.3, pp. 10–20.

OpenLisbon (2018). *Lisboa aberta missão*. url: `http://lisboaaberta.cm-lisboa.pt/index.php/pt/`.

Osterwalder, Alexander et al. (2014). *Value proposition design: How to create products and services customers want*. John Wiley & Sons.

Pinto, H. Sofia, A. Gomez-Perez, and J. P. Martins (1999). “Some Issues on Ontology Integration”. In: *In Proc. of IJCAI99’s Workshop on Ontologies and Problem Solving Methods: Lessons Learned and Future Trends* 18, pp. 7–1.

Rosén, Gustav (2019). *Analysis of Tabula: A PDF-Table extraction tool*.

Santos, Jorge (2008). “Integração de Conhecimento Temporal em Sistemas Inteligentes”. In: pp. 173–218.

Silva, Nuno (2004). “Multi-dimensional service-oriented ontology mapping”. PhD thesis. PhD thesis, Universidade de Trás-os-Montes e Alto Douro, Villa Real, Portugal.

Silva, Nuno and Joao Rocha (2003). “Ontology Mapping for Interoperability in Semantic Web.” In: *ICWI*, pp. 603–610.

Staab, Steffen and Rudi Studer (2010). *Handbook on ontologies*. Springer Science & Business Media.

Stephan, Grimm st, Hitzler st Pascal, and Abecker st Andreas (2007). "Knowledge representation and ontologies". In: *Semantic Web Services: Concepts, Technologies, and Applications*, pp. 51–105.

Van Heijst, Gertjan, A Th Schreiber, and Bob J Wielinga (1997). "Using explicit ontologies in KBS development". In: *International journal of human-computer studies* 46.2-3, pp. 183–292.

Van Rossum, Guido et al. (2007). "Python programming language." In: *USENIX annual technical conference*. Vol. 41, p. 36.

Walters, David and Geoff Lancaster (2000). "Implementing value strategy through the value chain". English. In: *Management Decision* 38.3, pp. 160–178. issn: 0025-1747. doi: 10.1108/EUM0000000005344.

Wei-Guo, Yi et al. (2010). "An ontology-based Web information extraction approach". In: *2010 2nd International Conference on Future Computer and Communication*. Vol. 1. IEEE, pp. V1–132.

White, Colin (2005). "Data integration: Using etl, eai, and eii tools to create an integrated enterprise". In: *Business Intelligence Journal* 10.I.

Zins, Chaim (2007). "Conceptual approaches for defining data, information, and knowledge". In: *Journal of the American society for information science and technology* 58.4, pp. 479–493.

Appendix A

Operator websites

Table A.1: Operator websites with respective method of timetable information

Operator	Website	Method of presentation
Metro do Porto	`https://www.metrodoporto.pt/frontoffice/pages/337`	A single PDF file with the information of every trip
Arriva	`http://www.arriva.pt/horarios/`	PDF file without embedded text and only images
Espírito Santo	`http://www.carreiras.espiritosanto.com.pt/horarios/`	PDF files
AV Pacense	`http://www.avpacense.pt/horarios.html`	PDF files
AV Landim	`http://www.avlandim.pt/horarios.html`	PDF files
AV Tâmega	`https://avtamega.com/Schedule`	Not working
AV Minho	`http://www.avminho.pt/horarios`	HTML
Rodonorte	`https://www.rodonorte.pt/pt/`	HTML
ETG	`http://www.gondomarense.pt/horarios-2006/index.htm`	PDF files
Valpibus	`https://www.valpibus.pt/partidas-chegadas`	PDF files and HTML for real time information
Maia Transportes	`https://www.maiatransportes.com/`	PDF files
MGC	`http://www.mgc.pt/pt/servico-regular/linhas-e-horarios/avintes/`	PDF files
Maré de matosinhos	`https://maredematosinhos.pt/linhas/`	PDF file without embedded text and only images
Transdev	`https://www.transdev.pt/`	Both HTML and PDF files are only available using route calculation
AV Souto	`http://www.avsouto.com/`	Flash application
TUST	`https://www.cm-stirso.pt/viver/mobilidade-e-transportes/transportes/rodoviario/transportes-urbanos-de-santo-tirso`	PDF file without embedded text and only images

Albano	`http://www.albanobus.pt/horario.php`	PDF file
Transportes urbanos de Braga	`https://www.tub.pt/percursos/`	AngularJS web application
Barquense	`https://barquense.viageos.com/wp-content/uploads/2018/06/`	PDF info-board without embedded text
GetBus	`https://www.getbus.eu/pt/braga-aeroporto-braga/`	HTML
Autna	`https://www.autna.com/es/horarios-y-tarifas/`	HTML and a single pdf file
Fertagus / Sulfertagus	`https://www.fertagus.pt/pt/horarios`	HTML
Transtejo / Soflusa	`https://ttsl.pt/passageiros/horarios-de-ligacoes-fluviais/`	HTML
Metro Lisboa	`https://www.metrolisboa.pt/viajar/horarios-e-frequencias/`	Does not have timetable information but has HTML frequencies and files using the GTFS format
CP	`https://www.cp.pt/passageiros/pt/consultar-horarios`	API available and files using the GTFS format

Appendix B

Service thesaurus

```
{
  " to ": [" to "," a "],
  " , ": [",", " e "," and "," ou "," or "],
  "monday": ["monday","segunda(s|-feira)?","2ª?"],
  "tuesday": ["tuesday","ter(ç|c)a(s|-feira)?", "3ª?"],
  "wednesday": ["wednesday","quarta(s|-feira)?","4ª?"],
  "thursday": ["thursday","quinta(s|-feira)?", "5ª?"],
  "friday": ["friday", "sexta(s|-feira)?", "6ª?"],
  "saturday": ["saturday","s(á|a)bados?"],
  "sunday": ["sunday","domingos?"],
  "holiday" : ["holiday", "feriados?"],
  "monday to friday" : ["working days", "dias úteis", "dias uteis"],
  "saturday , sunday" : ["week end", "fim de semana", "fim-de-
   semana"],
  "monday to sunday" : ["everyday", "todos os dias", "sempre"]
}
```

Listing B.1: Service thesaurus (Json).