INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Time-Sensitive Ontology Evolution Visualization

LEONARDO ANTÓNIO SARO DA SILVA ESTRELA
Outubro de 2023

POLITÉCNICO
DO PORTO

Time-Sensitive Ontology Evolution Visualization

Leonardo Estrela

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

Supervisor: Goreti Marreiros
Co-Supervisor: Alda Canito

Porto, October 14, 2023

Acknowledgement

Six years ago, I embarked on a journey that changed my life. Today, I conclude that journey. It feels bittersweet. Even though I am proud to finally conclude my academic journey, there is also a small part of me that genuinely wishes to start it all again.

First, I wish to thank my parents and my brother. You are the ones that made all of this possible, and I love you more than I will ever be able to explain.
I want to thank my colleagues and friends. We were not always the brightest nor the most productive, but we had fun!
I want to thank ISEP, and all of the professors that taught me throughout the last 6 years. You helped me become the engineer I am today.
Finally, I would like to thank my supervisors, Goreti Marreiros and Alda Canito, for the invaluable advice and help during the development of this project.

Abstract

With the integration of IoT in the industry, interest has risen regarding predictive approaches to maintenance, in hopes of predicting equipment or system anomalies before they take place. Ontologies, a crucial part of the semantic web, have found a use in these industries, as a way to semantically describe the data related to these equipments and systems. The ontology maintenance process, however, relies heavily on understanding the evolution of knowledge of the domain, over time. Visual representations of these ontologies and their evolution are very useful to ontology engineers (particularly in time-sensitive scenarios), even if tools that offer a service of this kind are fairly limited. Considering this, this dissertation presents a state of the art review of the existing **ontology evolution visualization** methods and tools, in the context of in time-sensitive scenarios, ending up by choosing to develop the solution using the WebVOWL viewer, React and Fuseki. It then describes the analysis of the requirements for the project, as well as a design proposal for a solution, taking into consideration the need to integrate with an already existing Java back-end application, and that the application must allow for the creation, loading and visualization of an ontology's evolution. The solution is described in detail, before concluding that it fulfills the requirements proposed at the beginning of the project, and successfully allows for the visualization of the evolutionary process of ontologies.

Keywords: ontology, semantic web, ontology evolution visualization, predictive maintenance, time related knowledge

Resumo

Com a integração da IoT na indústria, o interesse relacionado com manutenção preditiva subiu, com a intenção de prever anomalias em equipamentos ou sistemas antes que estas ocorram. Ontologias, um componente crucial da web semântica, encontraram um uso nestas indústrias, como uma forma de representar semanticamente os dados relacionados com estes equipamentos ou sistemas. Contudo, o processo de manutenção de ontologias depende muito do entendimento da evolução do conhecimento do domínio, ao longo do tempo. A representação visual destas ontologias e da sua evolução são muito úteis para engenheiros de ontologias (especialmente em cenários sensíveis ao tempo), embora ferramentas que ofereçam serviços deste género sejam um pouco limitadas. Considerando isso, esta dissertação apresenta uma análise do estado da arte dos métodos e ferramentas existentes para a visualização da evolução de ontologias, no contexto de cenários sensíveis ao tempo, acabando por selecionar para o desenvolvimento da solução o visualizador WebVOWL, React e Fuseki. Depois, descreve a análise dos requisitos do projeto, bem como a proposta de arquitetura para uma solução, considerando a necessidade de integrar uma aplicação *back-end* Java já existente, e que a aplicação tem que permitir a criação, carregamento e visualização da evolução de uma ontologia. A solução é descrita em detalhe, antes de concluir que a mesma cumpre os requisitos propostos na definição do projeto, e é bem sucedido em permitir a visualização do processo evolutivo de ontologias.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This document systematises the work developed during a Master's thesis in Software Engineering at Instituto Superior de Engenharia do Porto (ISEP), in the context of an already existing project developed at *Grupo de Investigação em Engenharia e Computação Inteligente para a Inovação e o Desenvolvimento* (GECAD).

In this chapter, the thesis is contextualized, by presenting the problems it aims to solve and the objectives to be reached - as well as the methodologies used in the process. It concludes by briefly explaining the structure of the report.

1.1 Problem Definition

With the advent of Internet of Things (IoT), and more recently, its integration in the industry, the interest regarding predictive and proactive approaches to equipment and system maintenance has risen. When it comes to predictive maintenance, its goal is to predict equipment anomalies and faulty statuses before they take place. This is done through the analysis and understanding of data related to the equipment and systems in question. Ontologies have found a use in IoT industries as a way to semantically describe this data, as well as help when it comes to decision making, thanks to its potential prediction and inference processes. They allow for a deeper understanding in a way that can be easily interpreted by both humans and machines [1], making them relevant in, among other things, interoperability between systems.

Ontology evolution is the process through which an ontology changes over time. The ontology maintenance process relies heavily on understanding the changes between two versions of an ontology, as well as being able to see its modification history [1] and visualization tools can make this a lot easier for humans to understand. Visual representations of ontologies are very useful to users who want to analyze and understand an ontology, in order to use it for data annotation or other tasks. They also allow for a intuitive way to consult, edit and share an ontology [2]. In time-sensitive scenarios, which is often the case with predictive maintenance, it is also important to understand the periods through which said changes were valid, i.e., to ascertain the "status" of a given part of the ontology at a specific date or interval. Ontology visualisation tools can be incredibly helpful for ontology engineers in this scenario, and are required to be both usable and intuitive.

In short, this project aims to understand the current state regarding ontology evolution visualization tools, and to design, implement and validate a solution that fits the requirements of the application context.

1.2 Project Context and Goals

The aim of this project is to develop an ontology evolution visualization tool that will help an ontologist understand the changes in a domain over time, and ultimately assist potential evolutionary processes. The application scenario involves an instance-driven ontology evolution process, in which the instances arrive via real-time streams of sensor data - in this case, provided by sensors in machines for predictive maintenance purposes - and the evolution process may happen continuously. The development will be composed of a front-end application, capable of consuming an already existing back-end solution developed at GECAD for this effect, which will need to be updated in order to fit the requirements of the project.

The following outcomes are expected:

- Review and analyse existing ontology evolution visualization methods;
- Development of a tool for ontology evolution visualization;
- Development of a user interface to guide the user during the ontology evolution process, assisting in decision-making;
- Result evaluation and analysis.

Regarding the approach taken for the project, a state of the art review was done, in order to understand the current landscape of technologies, as far as ontology evolution visualization goes. This is important for the project, since it seems to exist a lack of tools that allow for this type of ontology visualization. Besides understanding the existence of a tool that suits the project, the state of the art review takes into account tools or methods that could allow for a custom solution to be built (in the case that there is no existing solution that fits the requirements of this project).

Taking this into account, and considering the characteristics of the project, the **design and creation** methodology seems to fit perfectly. This strategy consists of designing a solution, implementing and validating it, before drawing conclusions from the final result [3].

Finally, the method with which the project will be tested and validated, after development, should be detailed, even if not in great detail. Taking into account the current state of the project, this description should not be intensive, but it should provide a comprehensive idea of what methods will be used, generally speaking.

1.3 Contributions

Since the project explores all the most relevant solutions for the problem - ontology evolution visualization methods - it will hopefully be useful for future investigation in this field when it comes to the state of the art of the topic.

Some of this research will be focused on the development of a front-end application to be used on time-sensitive ontology evolution visualization scenarios, which, as explained previously, is a field of study of great interest currently. The development of this type of applications is still rare, and so, the design and development of this tool will most likely be very useful in future research scenarios.

1.4 Structure

This document is structured as follows:

The first chapter introduces the problem at hands and the goals that are expected from the project. The context for this work is also be presented.

The second chapter presents some theoretical context regarding ontologies and other related semantic-web concepts. It then performs a review on the state of the art regarding ontologies, as well as ontology evolution visualization methods.

The third chapter presents the analysis and design done, for the project. it first analyzes the problem, before designing a solution.

The fourth chapter details the implemented solution, and explains all the major coding decisions that were made during development.

Finally, the fifth chapter serves as an overview of the report, and an appreciation of the project's success.

Chapter 2

State of the Art

This chapter provides some context regarding ontologies and related concepts, as well as a review on the state of the art regarding ontologies and, more specifically, ontology evolution visualization methods.

2.1 Context

Tim Berners-Lee [4] created the World Wide Web in 1989, while working at CERN. His original intention was to meet the demand for automated information-sharing between scientists in universities and institutes around the world, but in 1991, CERN made the technology available to the general public [5]. Berners-Lee vision to the web, however, was much more ambitious in comparison with the current version.

The World Wide Web is primarily composed of documents written in HTML (Hyper Text Markup Language), a language used for visual presentation of information. Most of the information on the Web is designed only for human consumption - web pages should be readable and understandable for humans - but their inherent meaning is not shown in a way that allows their interpretation by computers. These pages can serve a purely visual purpose, or be used for data integration and interoperability purposes.

Computers cannot understand the meaning of the information present in HTML pages. The visual Web prevents machines from automating information processing, integration, and interoperability, thus limiting its potential. HTML has certain limitations inherent to its design; it does not provide a rich vocabulary capable of capturing the meaning behind every piece of text, for instance. An example of that would be several current search engines, that find webpages containing certain keywords included in the user's search. They might take into account certain factors specific to the user, but there is no meaning associated to the keywords.

Considering this, **the semantic web** is a web where computers are able to interpret and rationalize information. Or, in other words, the objective of the semantic web is to make possible the interpretation of Web information by computers, and therefore, make information on the Web accessible and understandable by humans and machines. This web is not a new technology, but rather an extension of the existing syntactic web, as said by Tim Berners-Lee himself, who adds that, in the semantic web, "information is given a well-defined meaning, better enabling computers and people to work in cooperation" [6]. Both the current (syntatic) web and the semantic web can be described as a set of resources identified by a URI. The difference is that the syntatic web uses HTML to present its data, while the semantic web presents information in a machine interpretable way, semantically

representing data or information of resources [7] [8]. The current web has taken some steps towards semantization, but the fulfilment of Berners-Lee's vision is still a long way ahead.

Figure 2.1 shows the architecture of the semantic web, and its individual components.

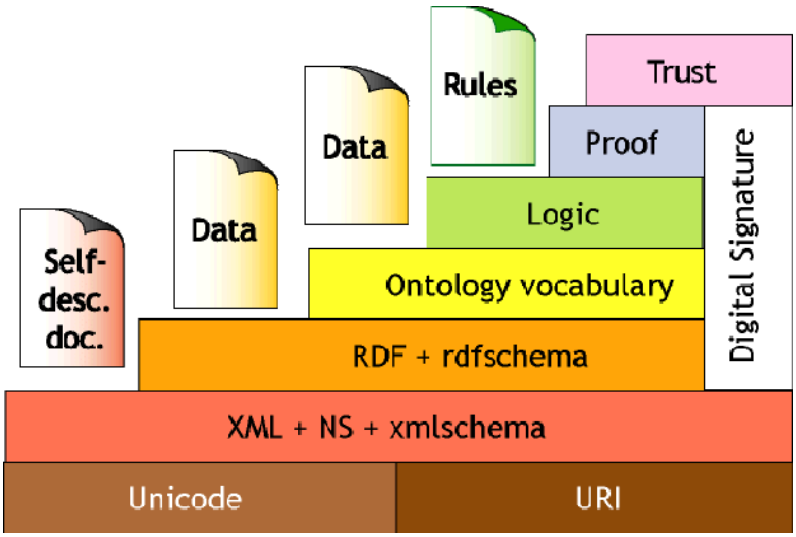

Figure 2.1: Architecture of the Semantic Web [9]

The component of the semantic web that section 2.2 focuses on is the ontology vocabulary, which is a common language between the published data and the semantic description of the information. And, following the same logic, an ontology is a way of representing the semantics of data, providing a uniform way by which different parties can communicate.

2.2 Ontologies

An ontology can be described as a shared conceptualization of a domain, which allows for said domain to be communicated between people, organizations and machines. This conceptualization typically includes categories, properties and relationships between different concepts or entities. Academic fields make use of ontologies to simplify and organize data - it is normal for each field to have a particular jargon or controlled vocabulary [10]. Examples of fields where ontologies are a primary concern are computer science, philosophy, medicine and certain fields of economics [11] [12].

As previously explained, in the context of the semantic web, an ontology is used to enable communication between different parties (or different views/levels of abstraction, for instances).

An ontology has four main components, those being classes, instances, axioms and relations. Each of these is explained below:

- Class - a concept, a type of object. The main formalized element of the domain [13] [14]. An example would be the concept of "Worker";
- Instance - a specific instance of an object. A representation of an object withing the domain [13] [14]. An example would be "Mary", which is an instance of the class "Worker";

- Axiom - represents a rule or a restriction, which must be accomplished on a relation [15]. An example would be the rule "All Workers have a Job";
- Relation - the connections between instances and classes. Or, in other words, relations are interactions between concepts in the same domain [15] [16]. An example would be the relation between "Mary", instance of "Worker", and "Accountant", instance of "Job".

Another concept within the context of ontologies is the attribute, which is a characteristic or feature of an object or class [17].

2.2.1 Ontology languages and standards

This section focuses on some of the main ontology languages and standards (which range from popular ontology tools, to related and recommended concepts or technologies).

Ontologies are represented through formal languages - both syntactically and semantically. These languages are necessary to ensure the validity of the statements and for computers to detected inconsistencies withing the provided vocabulary [18].

RDF

The Resource Description Framework (RDF) is a standard model for data interchange on the Web, introduced in 2004 by W3C. It represents knowledge in the form of a semantic graph, composed of nodes and links (which represent concepts and relations, respectively). RDF extends the Web's linking structure to name relationships and the two ends of a link (which is usually referred to as a "triple"). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications. [19]

A standard related to this language is **RDF Schema**, which provides a data-modelling vocabulary for RDF data. It is a semantic extension of (and written in) RDF, and allows to describe groups of related resources and the relationships between them. Its class and property system is similar to the type systems of object-oriented programming languages such as Java[20]

OWL

The Web Ontology Language (OWL) was first introduced in 2004 (while the latest version, OWL2, was introduced in 2009), and is currently the most used language to express ontologies.

It aims to:

- Define entities and properties (within a certain domain);
- Declare instances of these entities and properties;
- Enable reasoning concerning these entities and instances.

Contrary to RDF, OWL supports inter-operable vocabularies with semantics as ontologies. OWL allows for more expressive descriptions of properties and classes, by allowing the use of "relations between classes, cardinality, equality, richer typing of properties, characteristics of properties, and enumerated classes" [21]. It is part of a growing stack of technologies associated with the Semantic Web, of which RDF and RDF Schema are also part of.

SPARQL

Another important standard related to RDF is SPARQL, "a directed, labeled graph data format for representing information in the Web" [22]. It is generally considered to be the standard query language and protocol for Linked Open Data and RDF databases. In terms of its functionalities, it allows users to query information from NoSQL graph databases, or any other data sources that can be mapped to RDF [23]. An example of a type of data source that uses SPARQL as its standard query language is **RDF triplestores**. These data sources are designed to store entities composed of triplex collections of strings, representing subject-predicate-object relationships (or triples, as explained previously at 2.2.1).

It is not mandatory that SPARQL queries are constrained within one single database. Federated queries can access multiple data stores - unlike SQL queries. This is made possible due to the characteristics of SPARQL, that is not just a query language, but also an HTTP-based transport protocol [23].

SPARQL is able to perform the following types of queries:

- ASK - checks if there is at least one match of the query pattern in the RDF graph data, and then returns a boolean value representing the result;
- SELECT - returns any matches found in a tabular form (includes options for aggregation, sampling and pagination through OFFSET and LIMIT);
- CONSTRUCT - returns an RDF graph, based on match patterns in a WHERE clause;
- DESCRIBE - returns all triples that are associated with a specified resource.

Although SPARQL is mainly related to RDF, there are extensions that allow its conjugation with OWL as well [24].

2.3 Systematic Mapping Study

In order to understand the current state of the art regarding ontology evolution visualization, a systematic mapping study was performed. This methodology is used to structure a research area and present the known literature across a broad subject, in contrast to systematic reviews, which attempt to answer specific questions[25]. In the case of this project, because the question at hands was fairly complex and specific - the latter meaning that research specifically dedicated to the topic might be lacking - the systematic mapping review seemed to be the best choice.

The main research question of the project would be **"What are the existing ontology evolution visualization methods, in the context of time-sensitive scenarios?"**. However, a better way to perform the research is to divide this question into other research questions, smaller in scope. That said, the following questions were defined:

- What ontology evolution visualization methods exist?
- What are examples of ontology evolution visualization in time-sensitive scenarios, and how are they implemented?

The research was performed on the **ACM and ScienceDirect** platforms. These were chosen for the size of their collection and the prestige of both institutions, as well as the ease of use of the respective online platforms. The research queries can be seen at Table 2.1.

Table 2.1: Question queries for systematic mapping study

Question	Question query
First question	("ontology evolution" OR "ontology changes") AND (visualization) AND (methods OR state of the art OR review)
Second question	("ontology evolution" OR "ontology changes") AND visualization AND ("time sensitive" OR "time constrained" OR "emergency")

Table 2.2: Inclusion criteria for the systematic mapping study

Inclusion Criteria ID	Description
IC1	Article is peer reviewed
IC2	Article related to at least one of the research questions
IC3	Article is somehow relevant to the scope of the project

An initial research with the queries previously presented resulted in a total of **298** articles on ACM's digital library, and **153** articles on ScienceDirect's, which results in a total of 451 articles.

The criteria for inclusion and exclusion are detailed in 2.2 and 2.3, respectively The time frame defined for an article to be accepted for screening is **10 years**, as the landscape of front-end technologies is often changing, and research older than that could potentially suggest older, unusable technologies. Besides that, it was obviously important for the articles to be peer reviewed, since this suggests at least a certain level of validity, even before reading them. The articles should be related to at least one of the research questions, and should be somehow relevant to the scope of the project (an example of this would be an article that, while not directly related to ontology evolution visualization, concerns a new web ontology visualization tool, that could potentially be adapted to the project at hands).

After an initial filtering based on the inclusion and exclusion criteria previously explained, the number of articles remaining was **181**. The following step of the process consisted in reading the title and abstract of each article, in order to consider its usefulness to the project. The majority was excluded, since the articles in question were not relevant to the problem at hands, or did not target one of the research questions defined previously. Finally, this resulted in a total of **26** useful, valid articles, to be used in the state of the art review. Another **3** articles were added to the final set, despite not appearing on the systematic mapping study, as these were deemed to be relevant and important to the project.

The whole process is summarized in Figure 2.2.

Table 2.3: Exclusion criteria for the systematic mapping study

Exclusion Criteria ID	Description
EC1	Article was not written between 2013 and 2023
EC2	Article does not concern technology
EC3	Article is not written in either english or portuguese

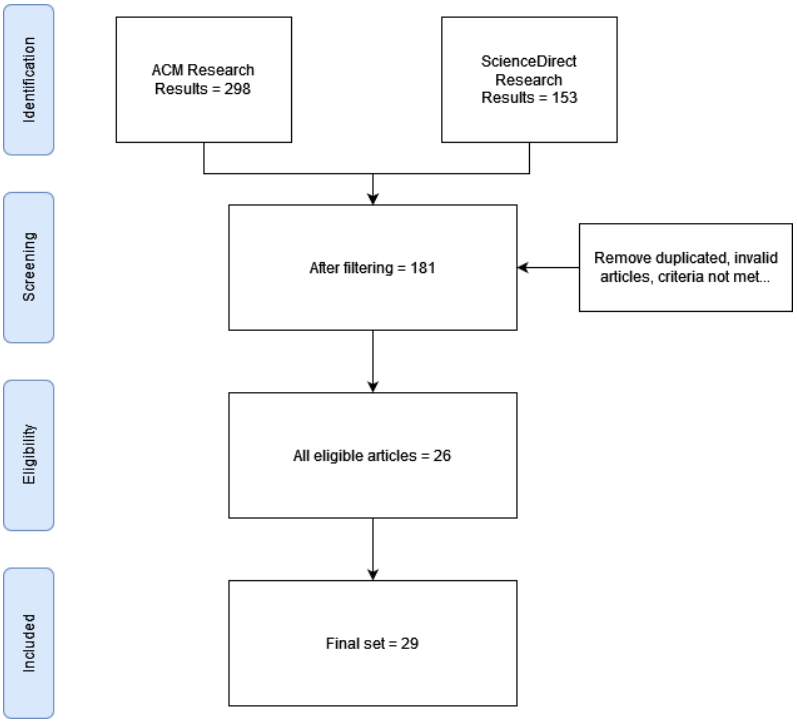

Figure 2.2: Summary of the Systematic Mapping study

2.4 Literature Review

The following literature review covers the data extracted from the systematic mapping study detailed in the previous section. It starts by giving an overview regarding the different ontology visualization methods identified. It then presents some of the main tools used for ontology visualization currently available. Several of the definitions and categorizations (including the classification used for visualization methods in subsection 2.3) from this section come from [2].

2.4.1 Ontology Visualization Methods

There is a vast number of ways to display an ontology. This subsection attempts to cover the ones that are most prominent ones, present in the most popular visualization tools.

These methods can be separated (according to [2]) using three different criteria:

- Number of dimensions (2D, 2.5D, 3D, ...)
- Graphical elements (nodes, circles, ...)
- The way these elements are layed out on the screen (tree, force-direct...)

While there are some alternative methods to display ontologies, most of them can be described as **directed graphs**. This is a graph where the edges have a direction - which is normally symbolized by using an arrow [26].

Figure 2.3 shows schematic illustrations of different visualization methods. All the examples depict the same ontology fragment, composed of five classes, an object property and a datatype property.

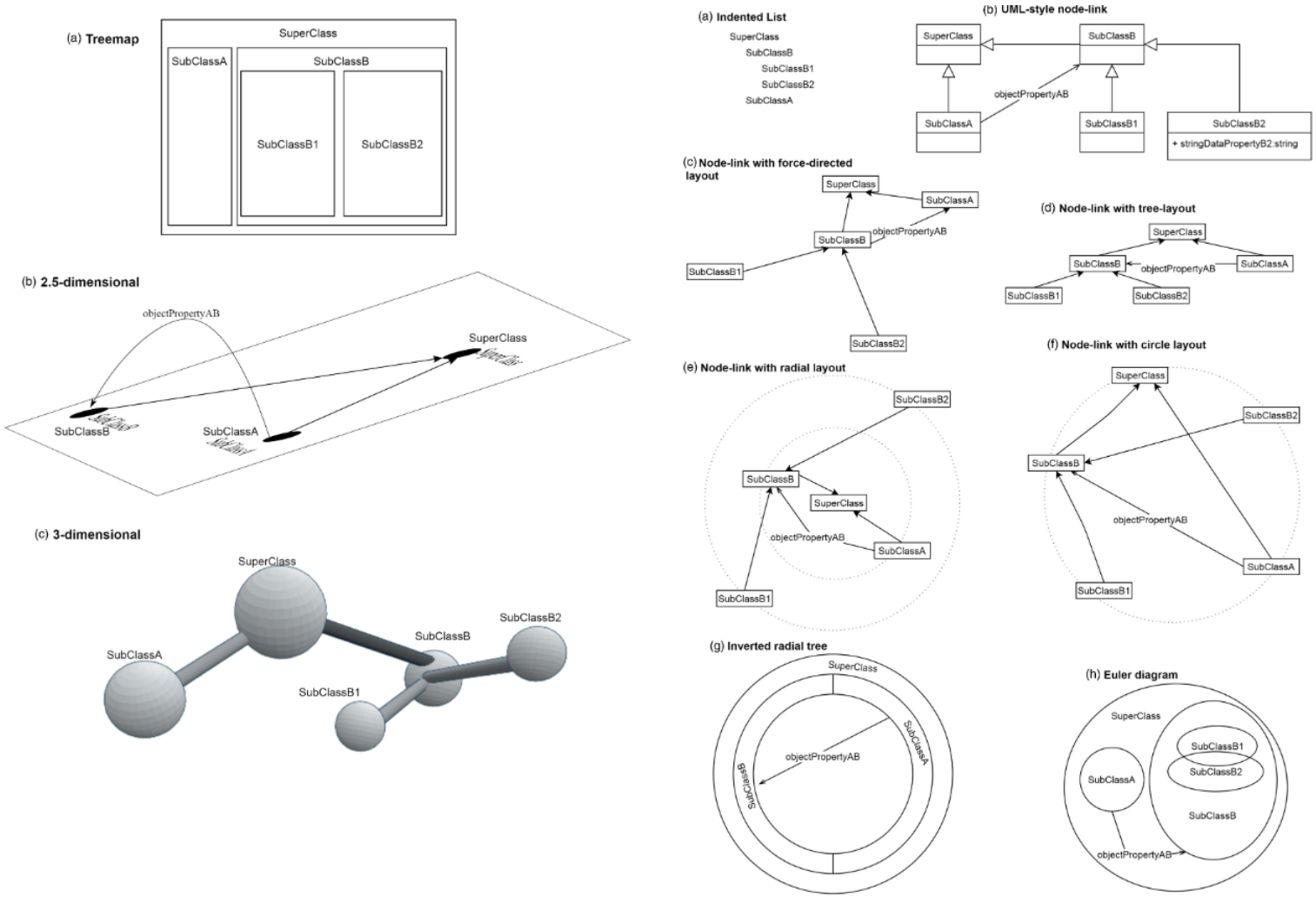

Figure 2.3: Schematic examples of visualization methods [2]

Indented List

The indented list is used for ontologies displaying a list of entities where entities lower in the hierarchy are shown indented under others, higher in the hierarchy. This method allows to visualize only one type of entities at a time and only the hierarchical relationships between them. The only graphical elements used are labels, often in combination with symbols or line fragments. An example can be seen at 2.3.

2D visualizations

The **node-link** (or graph) visualization method is the most frequently used in the most popular ontology visualization tools. It uses labeled nodes that are connected by links, which can also be labeled. Typically, these nodes represent entities like classes, and the links represent the relations between them. This type of visualization can be further distinguished into two subcategories: **label-based categorization** and **layout-based-categorization**. The former includes the following visualizations:

- **UML-inspired node** visualizations - in this layout, the node label consists of the name of the entity and other information (data properties, for instances)
- **name-label-only** visualizations - in this layout, the node label only consists of the name of the entity

Layout-based-categorization, on the other hand, includes the following visualizations:

- **Force-directed layout** - this layout is one of the most common layouts for node-link visualizations. It simulates a physical system where edges act as springs and nodes repel each other. While the layout is fairly easy to understand, because of the randomness of the algorithm, the disposition of the layout may change every time the algorithm is run [2]);
- **Tree layout** - the tree layout, in the context of ontology visualization, is often used to represent hierarchical relationships. The entity at the top of the hierarchy is displayed at the top of the layout, while its children are represented in the level directly below. The children of these children are then represented one level below, and so on. This layout remains consistent in each execution of the algorithm, but it is almost exclusively used for hierarchical relationships; [2]).
- **Radial layout** - this layout is also used for the representation of hierarchical relationship. The root entity is displayed at the center of the layout, while its children are represented on the orbit around it. The children of these children are then represented on the orbit around the previous orbit, and so on. One of the advantages of this layout is how space-efficient it is [2]);
- **Circle layout** - this layout forms a circle with all the entities of the ontology. Edge bundling techniques can sometimes be used, in combination with this layout, in order to visualize a large number of entities [2]);
- **Inverted radial tree layout** - in this layout, the nodes are place into a circle which consists of multiple concentric rings. Nodes near the root entity are displayed as parts of the outer ring. Their children are in the next ring, and so on [2]).

Other form of 2D ontology visualization is through a **Euler diagram**. This layout represents ontologies based on hierarchical relationships with circles. A child's circle is inside of its

Table 2.4: Table of use cases for each identified tool [29]

Case/Tool	CODEX	REX	OnEX	PromptDiff	OntoView	NeOn	KAON
1	x (partially)		x (partially)	x	x (partially)	x	x
2	x (partially)	x	x (partially)	x	x (partially)		

parent's circle, and so on. The position of the circles might represent other types of relationships, contrary to other hierarchy-based layouts. Some tools combine the Euler diagram with node-link visualizations - circles become nodes, with links between them being added.

Finally, there are **treemaps**. This layout also displays hierarchical relationships. Child rectangles are displayed inside of the parent rectangles, while siblings are shown next to each other, on the same level. This makes treemaps related to Euler diagrams.

Other visualizations

Some types of visualizations do not fit into any of the previous categories. Although less common, these are valid ways to represent ontologies.

The tool **Ontoviewer** [27] offers an experimental node-link visualization, enriched with a **third dimension for additional links**, making this method not yet a 3D visualization, but rather a 2.5D visualization.

OntoSphere [28] is a tool, however, that is able to display ontologies in 3D environments. While this method is impractical for most users, as most computer screens are 2D, this method is still worth mentioning, if for nothing else, its uniqueness.

2.4.2 Ontology Evolution Visualization

Ontology evolution visualization is a more specific form of ontology visualization. While it still makes use of the different forms of visualization previously described, it must also take into account the way an ontology changes over time. The graphical representation must allow the user to understand these changes. Understanding what changes were made between any two versions of an ontology and being able to see its modification history are important tasks in the ontology maintenance process and visualization tools can make these much easier for humans to understand.

A 2016 literature review concerning the topic [29] presented an overview of ontology evolution systems, while also discussing how these systems support ontology visualization. The main functionalities that these systems tend and should to cover are detailed at 2.4.

The article argues that ontology evolution is relevant in two different cases:

- When used in a semantically-enabled application, in which evolution plays an important role
- When the objective is to obtain knowledge about the evolving ontology

The article then gives an overview of several tools capable of covering, at least partially, one of both these use cases. This comparison is shown in table 2.4, which explains evidences that the only tool that fully performs both tasks is PromptDiff [30], while most of the other tools only fully perform one of them.

Step	Category	Functionality
step 1	I	1 show an ontology version
	I, E	2 show different ontology versions in evolution graph
	I, E	3 show changes/diff between ontology versions (with change types)
	I	4 show summary of changes
	I	5 show specialized view of changes
	I	6 show change history of a concept/relation
	I, E	7 show provenance information
	I	8 show information about/context of concept/relation
	I, E	9 compare different versions
	I	10 search and query ontology
	I	11 query old versions using terminology of new version
	I, E	12 discover trends
	I, E	13 discover volatile and stable regions
step 2:	-	1 identify and suggest changes
step 3:	-	1 identify conflicts
	I	2 show conflicts
	M	3 resolve conflicts
	M	4 accept and reject suggested changes
step 4:	-	1 evaluate influence on dependent artifacts
	I	2 show influence on dependent artifacts
	M	3 update of dependent artifacts
step 5:	M	1 execute changes
	E	2 identify and show implications of change in ontology
	M	3 add/edit change annotations
	M	4 roll-back mechanism
	-	5 save current state

Figure 2.4: Functionality for ontology evolution systems (Table) [29]

A 2021 systematic review on time-constrained ontology evolution [1] gives a broad overview regarding the topic of ontology evolution in time-sensitive scenarios. It also offers a brief overview on some methods used to visualize ontology evolution. According to the study, one of the tools currently exploring this issue is **EvoRDF** [31], a tool that allows the user to consult the evolution of ontologies over time, showing the changes between any two versions. The tool also allows the user to understand why certain changes have happened, by providing the evolutionary path of a given concept or property.

It also mentions several other studies that make use of graphs, in order to illustrate ontology evolution over time. This is the subject of a 2015 article [32], which covers the difficulties of fully translating an ontology into a graph. The author says that users rely on ontology evolution visualization tools to solve comparison tasks, in order to reliably derive trends or to find outliers and anomalies. It argues that there are two paradigms for time-oriented visualizations: "**time-to-time mappings** (animated diagrams) and **time-to-space mappings** (static displays, often enhanced by interaction techniques)".

Time-to-time mappings can be used for the exploration of evolving ontologies, but the cognitive load imposed on the users runs the risk of becoming too high. It also makes the implementation of interaction techniques more difficult because the elements are moving around the screen. Time-to-space mappings, on the other hand, can use a vertex-aligned representation that allows to statically attach a hierarchical organization. This allows the user to look wherever it wants, contrary to what happens in animated ontologies. They can perform comparison tasks visually, not mentally. In addition to this, time-to-time mappings tend to perform worse for time-oriented tasks. In conclusion, **time-to-space mappings tend to be the better solution when it comes to ontology evolution visualization**.

2.4.3 Identified Tools

This subsection focuses on the main ontology visualization tools identified in the state of the art review. These were chosen based on their popularity and ammount of documentation, as well as their capability to serve as ontology evolution visualization tools.

Protégé

Protégé is an open-source ontology editor, and possibly the most widely used ontology visualization tool. Its development started in the 1980s, and is still in progress. It offers a graphical UI called Entity Browsers, which offers, for instances, a basic indented list ontology visualization. It is available as a desktop application, as well as a web app, in the form of **WebProtégé** [33], a cloud-based framework that allows users to collaboratively edit OWL ontologies.

As a development platform, Protégé's architecture supports the development and use of plugins - many of these aimed at ontology visualization. An example of these would be **PromptDiff** [30], a tool identified present at table 2.4. It consists of a change management plugin and a ontology version comparison plugin. The former allows the user to consult the changes made on an ontology, with different views and levels of detail; while the latter allows the user to visualize indented trees of two ontology versions as a single tree [29].

Protégé however, in any of its forms, **does not support the consumption of a custom REST API**. There were plans, as of 2021, to be implement the possibility of consuming a custom REST API using WebProtégé, in order to allow for its integration with custom back-end solutions, but so far the development is yet to be made available by the maintainers [34].

WebVOWL

WebVOWL is a web application focused on user-oriented ontology visualization. It uses OWL in order to display all the graphical elements and a force-directed graph layout, that represents an ontology. The interaction techniques provided by the tool allow the user to explore the ontology and to customize its visualization. WebVOWL renderizes the visualizations based on JSON files, containing the ontologies. These can be converted for this effect using OWL2VOWL, a Java-based converter, which is provided alongside the main tool. WebVOWL renders all the graphical elements according to the VOWL specification. Its UI offers a number of interactive features that allow for rich manipulation and visualization, such as zooming, dragging, inspecting and editing. It supports nearly all language constructs of OWL 1 and OWL 2 [35].

WebVOWL's is suitable for learning and sharing, even if it lacks in editing compared to other tools currently available. The tool is almost entirely built using vanilla JavaScript, which, coupled with the fact that it is an open-source application, makes it ideal to adapt into new solutions developed in the context of the web.

WebVOWL **does not support time related knowledge or ontology evolution visualization out of the box**. It would only be possible to use WebVOWL in that context with the development of a custom solution.

Analysis of the tools

The two tools considered for the solution, so far, are Protégé and WebVOWL. It is important to consider that the following criteria must be met by the chosen tool:

- Requirement 1 - It must be able to display ontologies in a user-friendly way (whether it is through a desktop or web application is not a deciding factor)
- Requirement 2 - It must support the visualization of an ontology's evolution, or be adaptable into a solution that supports this
- Requirement 3 - It must allow for the consumption of a custom Java back-end, already developed, capable of manipulating and comparing ontologies

Both tools fit the first requirement, as both are capable of displaying ontologies in a user-friendly way (Protégé in desktop and web applications, while WebVOWL does it exclusively in a web application). However, only Protégé supports the visualization of an ontology's evolution, thanks to a few plugins developed for the effect. WebVOWL does not support this functionality out of the box, but it seems possible to adapt WebVOWL's code into a solution that supports this. Finally, neither Protégé nor WebVOWL support, in any of its forms, calls to a custom REST API.

Considering all of this, it seems there are two possible solutions for the project:

- A custom solution based on Protégé's open-source code (written in Java, and thus requiring a front-end technology compatible with Java, such as JavaFX [36])
- A custom solution based on WebVOWL's open-source code (written in JavaScript, and thus requiring a front-end technology compatible with JavaScript, such as React[37] or Vue.js[38])

Tools Chosen

It becomes evident, upon analyzing both solutions, that the second solution is the better one. WebVOWL's code base, upon inspection, looks considerable easier to adapt into a custom solution than Protégé's, and the latter requires a series of plugins in order for its user interface to offer a similar experience as WebVOWL's. Besides this, using JavaScript guarantees a wider array of choice when it comes to frameworks, documentation, library and so on.

In conclusion, the chosen solution for this project is **a ontology evolution visualization web application, based on WebVOWL's code base, using a JavaScript framework and a SPARQL storage**.

Considering that the original project already uses Apache Jena [39] in order to handle SPARQL queries and Ontology models, it makes sense that the storage technology is Apache Jena Fuseki [40], a sub-project of Jena that provides an HTTP interface for RDF data, and allows for SPARQL querying. By choosing Fuseki, the new developments can fit seamlessly into the original codebase.

Finally, there were no restrictions as far as which web framework to choose, besides it being a JavaScript framework. With that in mind, React [37] was chosen, simply due to popularity, simplicity and personal preference.

Chapter 3

Analysis and Design

This chapter aims to describe the analysis of the previously described problem, and the designed solution for it, as well as contextualise and describe the requirements of the project.

3.1 TICO

As said previously, this project exists in the context of **TICO** (TIme-Constrained Ontology evolution) [41], a back-end console application previously developed in GECAD, capable of generating new versions of an ontology, based on instances of said ontology.

After receiving both the ontology and instances files, TICO generates **Evolutionary Actions** (actions that change the original ontology in some way), executes them, and then returns the resulting ontology.

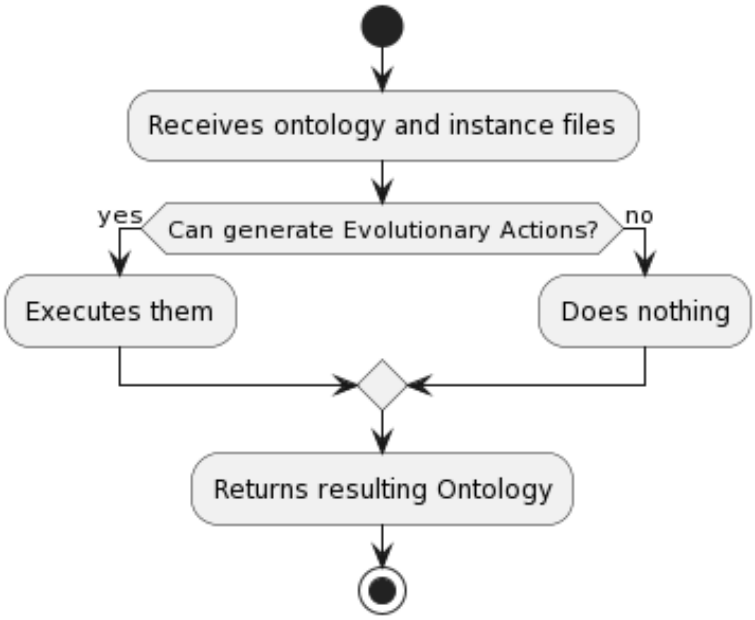

Figure 3.1: Activity Diagram of TICO

The main goal is to develop a front-end application, capable of helping the user visualize this evolutionary process, as well as store the results. In order to achieve this, several architectural decisions must be made, and the original project must be adapted. This process is described in bigger detail in section 3.6.

3.2 Methodologies for Analysis and Design

3.2.1 C4 View Model

The C4 model [42] is a methodology used to describe a software's architecture. It provides a way for software development teams to efficiently and effectively communicate their architecture, at different levels of detail, telling different stories to different types of audience. C4 encompasses 4 levels of abstraction, those being: **system, container, component** and **code**. A system diagram provides a starting point, showing how the software system in scope fits into the world around it. A container diagram focuses into the software system, showing the high-level technical building blocks. A component diagram focuses into an individual container, showing the components inside it. Finally, a code diagram can be used to focus into an individual component, showing how that component is implemented.

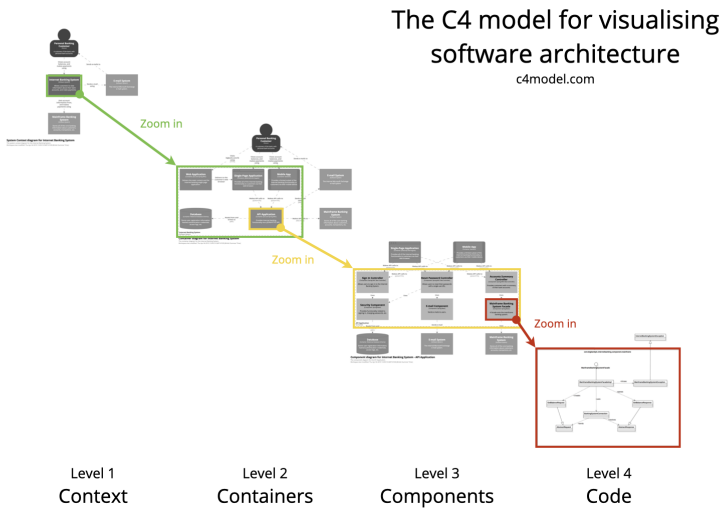

Figure 3.2: Overview of C4 Model

3.2.2 The 4+1 View Model

Software architecture deals with abstraction, decomposition and composition, and to describe it, one can use a model composed of multiple views or perspectives. The 4+1 View Model is made up of 5 different views, each referring to a different context or scope of an application:

- Logical View - refers to the services or the functionalities of the project. The system is decomposed into a set of key abstractions, each representing a different context of functionality. This view is usually backed by a Class Diagram or a Component Diagram.
- Process View - refers to the communication between the key abstraction referred to on the description of the Logical View. This view is usually backed by a Process Diagram.
- Physical View - refers to the mapping(s) of the software onto the hardware and reflects its distributed aspect. This view is usually backed by a Deployment Diagram.
- Development View - describes the static organization of the software in its development environment. This view is usually backed by a Package Diagram.

- Scenarios - the +1 view represents the multiple functionalities or capabilities of the system. These include both what the end user is capable of doing with it, as well as operations that solely concern the system itself. [43].

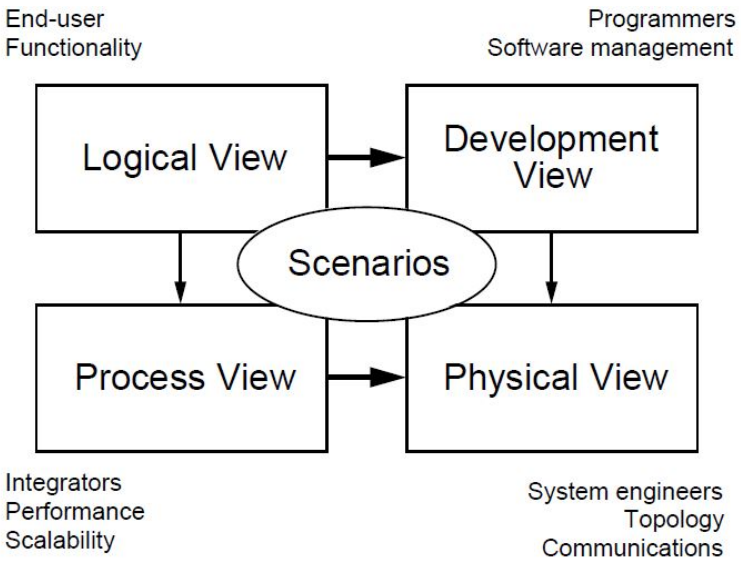

Figure 3.3: The 4+1 View Model

3.3 Functional Requirements

The project has the following functional requirements:

- The user should be able to import ontology and instance files into the system
- The user should be able to edit and delete the generated Evolutionary Actions, before executing
- The user should visualize the updated ontology, after executing the actions, and compare it to the older version
- After the process is completed, the user should be able to load the ontology and visualize the two versions.

Each of the functional requirements described above results into a use case. A use case diagram representing this can be seen at Figure 3.4.

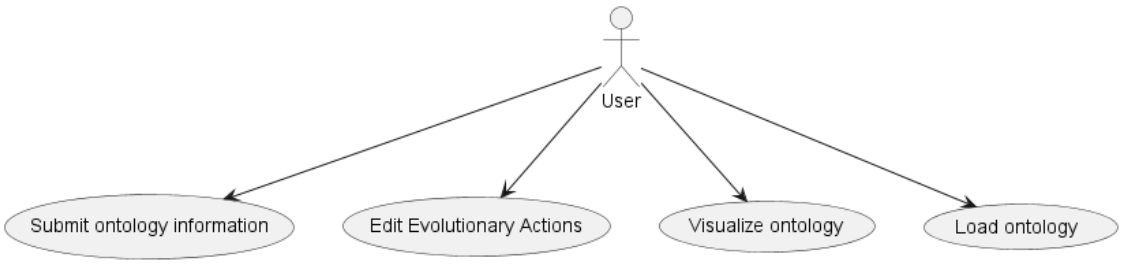

Figure 3.4: Use Case Diagram

3.4 Non-functional Requirements

A non-functional requirement defines system attributes. While the functional requirements translate into functionalities, the non-functional requirement imposes a restriction or a necessity on the design [44]. It is important to note that some of these attributes are not considered for the project at hand, because they do not fall within its scope (an example would be security, as there are no security attributes defined for the project, such as authentication, authorization...).

3.4.1 Supportability

Supportability requirements concern the testing of the system, as well as the platforms and environments on which the system should run. Although no requirements regarding testing were specified, the project should be tested using unit and integration tests, to ensure all the functionalities function properly.

The system should accept different types of file extensions (.json, .ttl, among others), although the main goal of the project regarding this is to fully function with Turtle [45] files. Considering that WebVOWL, the chosen viewer for the application, is able to convert the most popular ontology notations into its own syntax (VOWL), this does not constitute a problem as far as the front-end goes. The same level of supportability should be ensured in the back-end.

There are no requirements as far as what systems the project should run on. Considering the research described the subsection 2.4.3, the application is a web application, and should run on most modern browsers without major limitations.

3.4.2 Usability

Usability requirements concern the look and feel of the application. The user interface should be as simple as possible, in order to let the user focus on the evolutionary process of the Ontologies.

3.5 Analysis

This section focuses on the analysis of the problem. It contains the domain model and the use cases defined, in order to better understand the project.

3.5.1 Domain Model

The domain of the problem starts with the core of application: its user interface. This UI contains one or more viewers, which allow for the visualization of an Ontology and its different versions. The UI also allows for the submission of Files (ontology and instances), which will then allow the system to generate Evolutionary Actions. Once executed, these Actions result into an updated Ontology, which can now be displayed on the viewer.
The domain model is displayed on diagram 3.5.

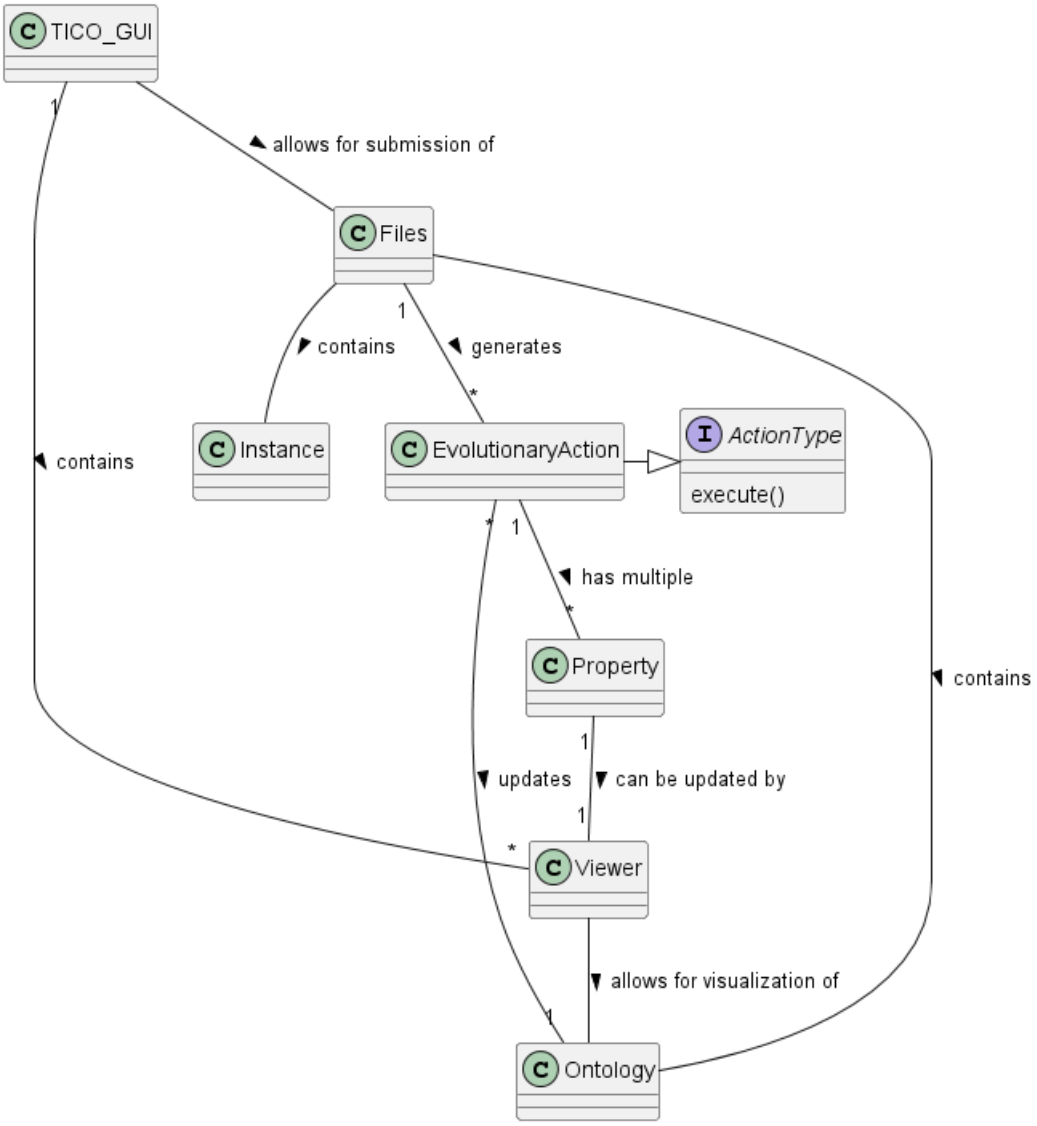

Figure 3.5: Domain Model

3.5.2 Use Cases

Use Case 1 - Submit ontology information

This is the first operation the user needs to perform in order to create a new version of an ontology. After inserting the ontology's name and the two mandatory files (ontology and instances files), the information is sent to the back-end. Here, the actions are created and returned to the frontend. This process is displayed on the diagram 3.6.

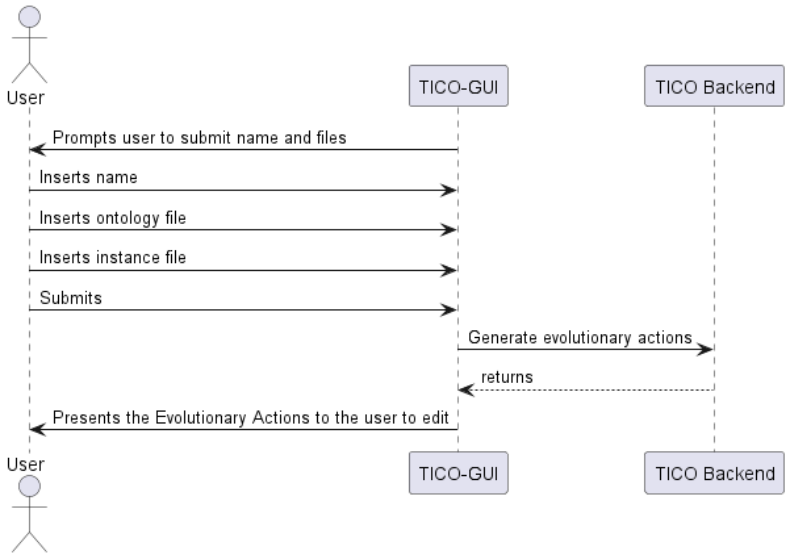

Figure 3.6: Use Case Diagram 1

Main Actor: System's End Users — Ontology Engineers
Stakeholders: Ontology Engineers - want to submit the ontology files in order to begin the evolution process
Preconditions: Internet access and all the projects necessary are available
Post-conditions: The files are submitted and the user can now edit the evolutionary actions.

Use Case 2 - Edit Evolutionary Actions

This phase of the process begins after the user submits the files, and these are processed by the back-end. Here, the user can edit and delete the generated Actions as desired, before submitting. Once this happens, the now updated Actions are sent to the back-end, which after executing the Actions, returns the resulting Ontologies.

Main Actor: System's End Users — Ontology Engineers
Stakeholders: Ontology Engineers - want to validate the Evolutionary Actions and edit if necessary, in order to customize the evolutionary process
Preconditions: Internet access and all the projects necessary are available
Post-conditions: The evolutionary actions are updated and the ontology visualization is now possible.

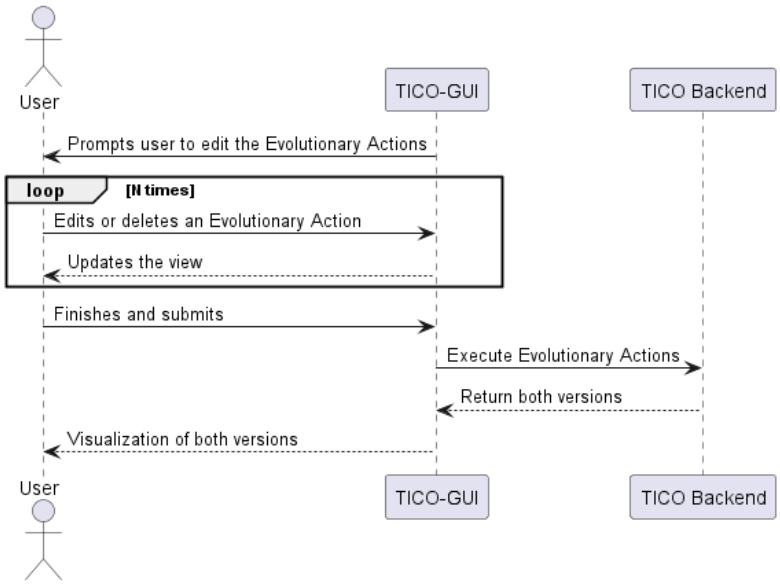

Figure 3.7: Use Case Diagram 2

Use Case 3 - Visualize ontology

Here, the user is able to visualize and analyse as desired the result of the evolutionary process. Both Ontology versions are displayed in adapted WebVOWL viewers, or, if the user decides to visualize one of them in a larger scale, only one of them is.

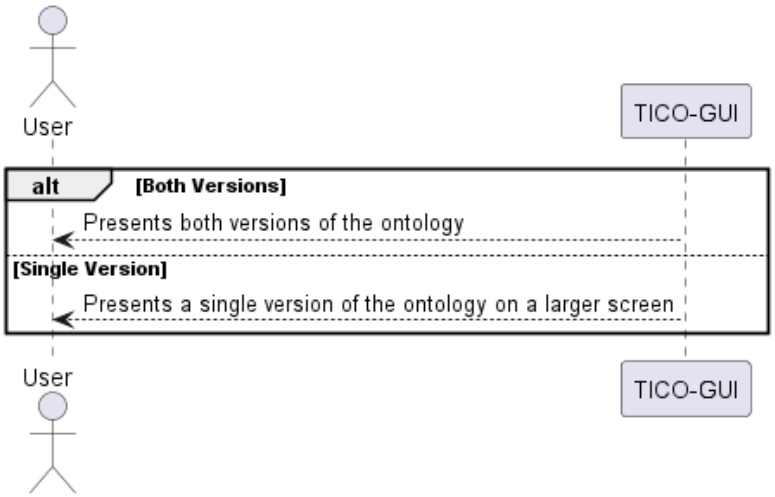

Figure 3.8: Use Case Diagram 3

Main Actor: System's End Users — Ontology Engineers
Stakeholders: Ontology Engineers - want to visualize the evolutionary process
Preconditions: Internet access and all the projects necessary are available
Post-conditions: The user visualizes the evolutionary process.

Use Case 4 - Load Ontology

Once the previous 3 processes are completed, the user can now access the visualization again by clicking on "Load Ontology", on the main menu. The frontend calls the back-end project, which returns all the saved Ontologies. After these are displayed on the frontend, the user can select one of them to visualize.

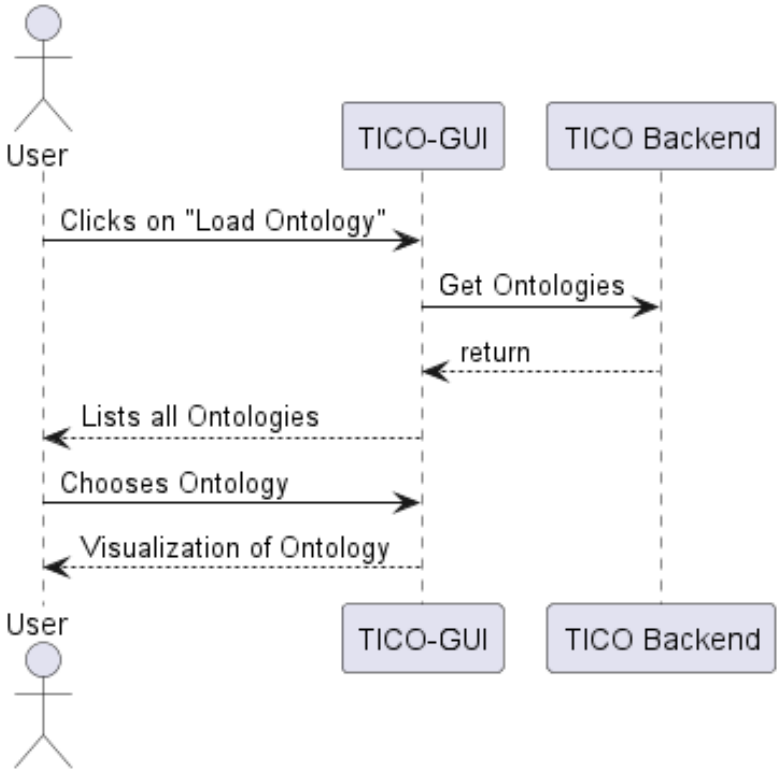

Figure 3.9: Use Case Diagram 4

Main Actor: System's End Users — Ontology Engineers
Stakeholders: Ontology Engineers - want to load an Ontology to visualize its evolutionary process
Preconditions: Internet access and all the projects necessary are available
Post-conditions: The ontology is loaded, and the ontology visualization is now possible.

3.6 Design

TICO is developed using Java, and uses Apache Jena [39] in order to manipulate the ontology and instances.

As the project consists of a front-end application that guides the user through the evolutionary process of an ontology using a user-interface, **the need for the development of a REST API becomes evident**.

3.6.1 Proposed Solution

This section attempts to summarise the proposed solution, at an early stage of development. It first contextualizes the solution by offering some details regarding the existing application.

It then presents an early architecture proposal, as well as the use cases of the application.

Existing Application

The existing application is a Java back-end application, capable of creating, updating, deleting and comparing ontologies. Although it does not offer a REST API, currently, this can be easily developed, thus allowing the application to receive calls from a web application. The files containing the generated ontologies are not stored on a triple store at the moment, but rather are stored on the local file system. A connection to a triple store, using SPARQL, is something to be considered for the proposed solution.

Proposed Architecture

The proposed solution consists of a web application that is able to communicate with the existing back-end application through a REST API. A diagram displaying the proposed architecture (level 2) can be seen at 3.10. The web application should be built with a JavaScript framework (like React [37], or Vue [38]) in order to adapt WebVOWL's [35] code into a custom solution for ontology evolution visualization. This is expected to result in many challenges, and so, it will only be possible to validate this solution after the development of a proof-of-concept application. Other methods (such as the use of a generic JavaScript graphical library) would have to be considered and investigated.

The existing back-end application would have to be adapted, in order to allow HTTP REST calls, and an API would have to be developed for this. Finally, this solution includes a SPARQL triple store, where the different ontologies, versions and instances would be stored.

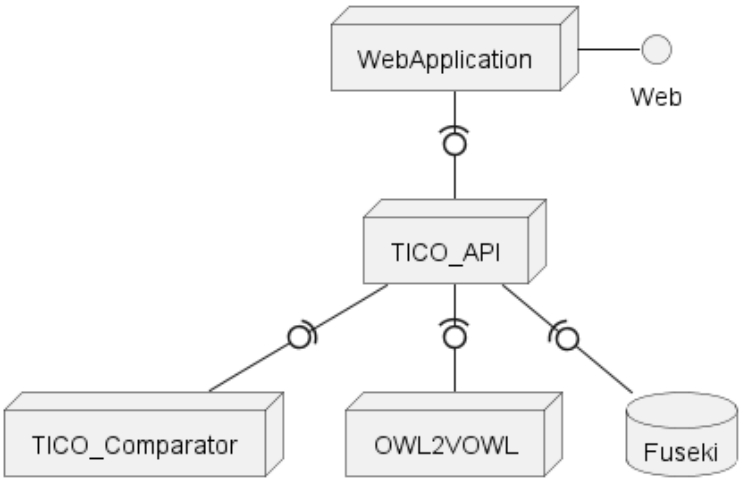

Figure 3.10: The proposed architecture (Level 2)

3.6.2 Use Cases

This subsection focuses on detailing the use cases described in 3.5.2, now that the architecture is clear and more detailed. The following diagrams represent Level 3 on the C4 model, the Component view of the system.

Use Case 1 - Submit ontology information

This is the first operation the user needs to perform in order to create a new version of an ontology. After inserting the ontology's name and the two mandatory files (ontology and instances files), the information is sent to the TICO-API project, which then calls the TICO-Main project. Here, the actions are created and returned to the frontend. This process is displayed on the diagram 3.11.

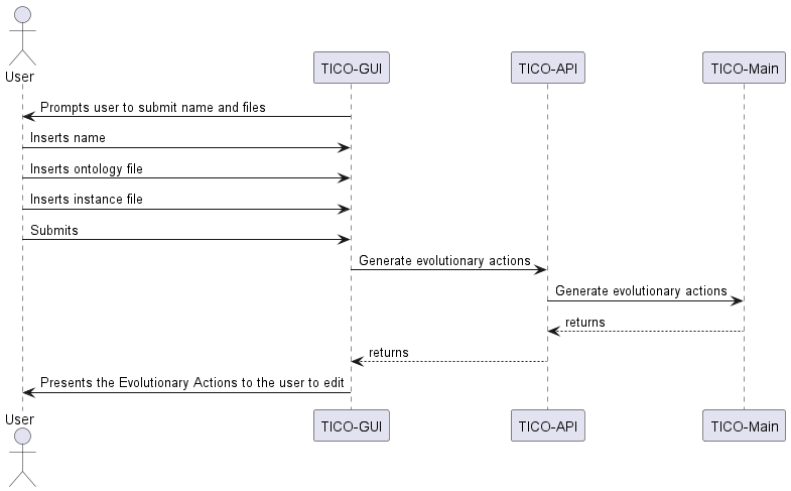

Figure 3.11: Use Case Diagram 1

Use Case 2 - Edit Evolutionary Actions

This phase of the process begins once the user submits the files, and these are processed by the TICO-Main project. Here, the user can edit and delete the generated Actions as desired, before submitting. Once this happens, the now updated Actions are sent to the TICO-API project, on which the following tasks are performed:

- The Actions are sent to the TICO-Main project and executed, returning the updated Ontology
- The Viewer versions of both the original and updated Ontologies are generated (by calling the OWL2VOWL project)
- The Ontology and its new version are stored in the Fuseki server

After this is completed, both versions are returned to the frontend, to be visualized.

Use Case 3 - Visualize ontology

Here, the user is able to visualize and analyse as desired the result of the evolutionary process. Both Ontology versions are displayed in adapted WebVOWL viewers, or, if the user decides to visualize one of them in a larger scale, only one of them is.

Use Case 4 - Load Ontology

Once the previous 3 processes are completed, the user can now access the visualization again by clicking on "Load Ontology", on the main menu. The frontend calls the Tico-API

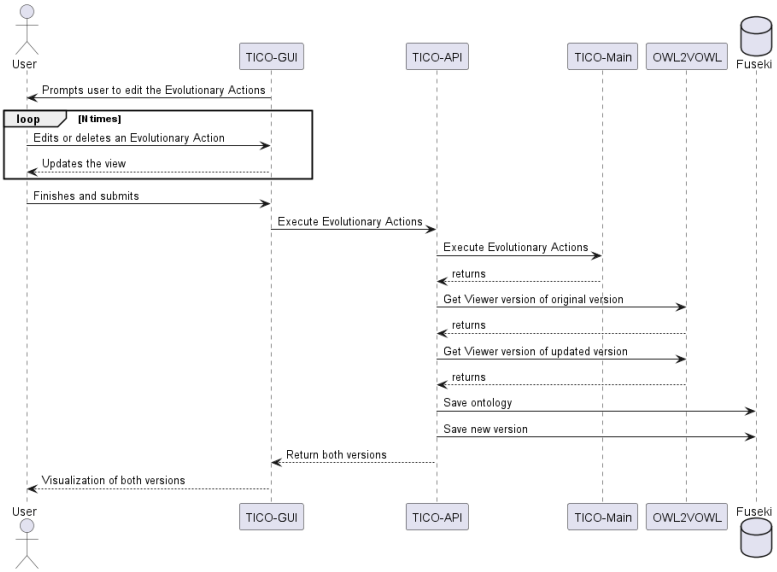

Figure 3.12: Use Case Diagram 2

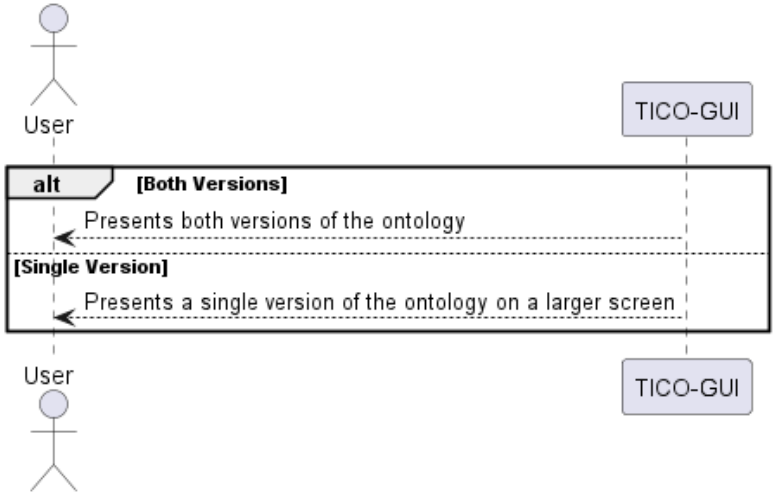

Figure 3.13: Use Case Diagram 3

project, which receives all the saved Ontologies from the Fuseki server. After these are displayed on the frontend, the user can select one of them to visualize.

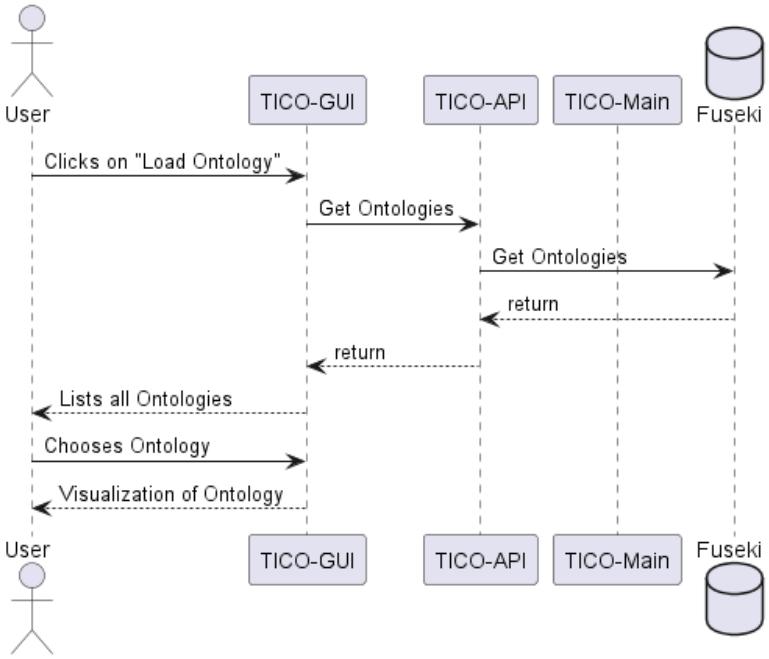

Figure 3.14: Use Case Diagram 4

Alternative Architecture

An alternative solution was designed, taking into account the other solution that was taken into account, during the state of the art review: develop a custom solution based on Protégé's open-source code (written in Java, and thus requiring a front-end technology compatible with Java, such as JavaFX). In this solution, instead of a web application, a Java application would be developed instead, using the already existing Java application, and developing graphical user interfaces (GUI) using JavaFX. A SPARQL triple store would be used like in the proposed architecture. The graphical user interface would have to be developed using an open-source Java based solution for ontology visualization. A diagram displaying the described architecture can be seen at 3.15.

This solution is valid, but was put aside in favour of the proposed architecture because of the reasons explained at section 2.4.3.

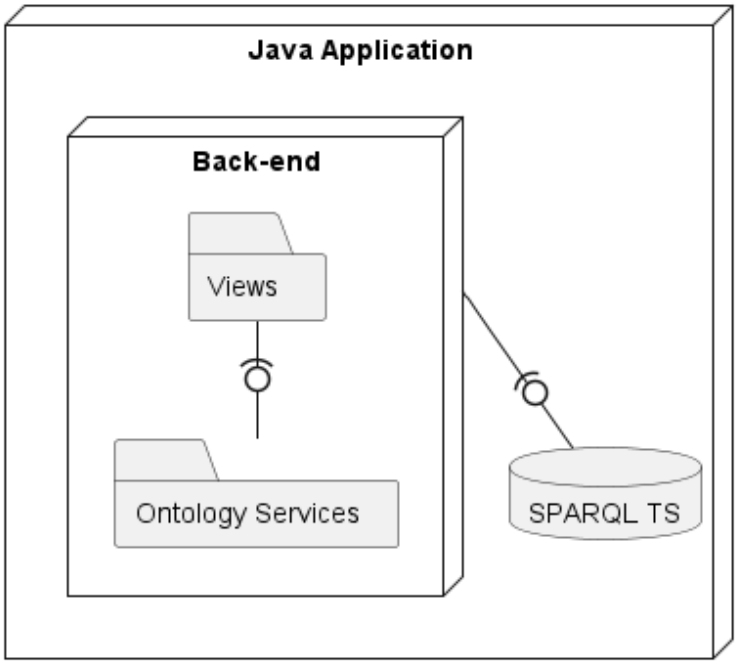

Figure 3.15: An alternative architecture

Chapter 4

Implementation

This chapter focuses on the details concerning the implementation of the application. It details the different applications developed in order to build the project, as well as the changes and decisions made throughout the development process.

The project relies on the integration of the following applications and storages:

- TICO-GUI: React application, containing the user interface;
- TICO-API: Spring Boot back-end application, the core of the project. Ensures communication between all applications;
- TICO-Main: Updated version of the original TICO back-end application;
- OWL2VOWL: Back-end application resposible for converting the ontologies to VOWL, in order to be visualized;
- Fuseki: SPARQL storage for the ontologies, versions and instances.

4.1 TICO-GUI

The TICO-GUI project is the front-end component of the system. It contains the graphical user interface that aids the user in visualizing the evolutionary process. Following what is described in the Analysis and Design chapter 3, the user interface allows the user to load or import an ontology. Each of the views, services and classes are detailed below, including the WebVOWL viewer used for the visualization.

The project has the following folder structure:

- WebVOWL (inside the "public" folder instead of "src" like the others);
- Views;
- Services;
- DTO classes.

It is also important to note that the front-end project uses the Chakra UI library [46] in order to build its views. This library was chosen because of its simplicity and lightweight characteristics.

4.1.1 WebVOWL

The WebVOWL viewer was the one chosen for the project, as said in Chapter 2, State of the Art 2. The viewer, however, is a vanilla JavaScript project, thus creating an obstacle when it came to integrating it in a way that was simple, clean and maintainable with a React project.
Two alternatives were considered:

- Converting the viewer's HTML file into a React component, while maintaing all the JavaScript logic untouched in separate files
- Including the viewer in the React views by using an **iframe**, and finding a way to reliably communicate with it.

The initial alternative was the one that, in theory, seemed to be the most promising, as it would result in a React version of WebVOWL - something that would surely be interesting not just for this project, but for other developers faced with similar scenarios (the need to visualize ontologies in web applications). Thus, when development began, this was the chosen solution.

However, as development progressed, it quickly became clear that this option brought unexpected challenges. The HTML portion of WebVOWL was much more complex than initially predicted, and while an initial version of the application contained a prototype of the viewer converted to a React component, there were still many problems to face, such as maintaining the same CSS styling and overall behaviour of the viewer. The biggest problem with this solution, however, was the following: because the JavaScript logic of the viewer was so intrinsically connected to the HTML view, this meant that, in order to achieve the React WebVOWL component that was desired, **development would not only have to include an adaption the HTML view, but of the JavaScript logic as well, which is by far the most complex part of WebVOWL**.
Considering this, and the fact that the other alternative was still unexplored, the original prototype was put on halt.
As said before, displaying the viewer inside of an iframe also provided a few challenges, with the main one being the communication between the React application and the project contained inside the iframe. This requires **the WebVOWL page to make available some functions that can be called** from one of the application's React components. In the case of this project, the one function that is callable from React is the **startWebVOWL** functions, which receives an IRI, an Ontology content (in VOWL notation, already converted by the OWL2VOWL project, as explained in the Analysis and Design section 3) and the version of this Ontology, in order to initialize the viewer. This function calls the original **initialize** function from WebVOWL, passing these same parameters. The code for this can be seen at Code Snippet 4.1.

```
<script>
        function startWebVOWL (ontologyContent, iri, version) {
            webvowl.app().initialize(ontologyContent, iri, version);
        }
</script>
```

Code Snippet 4.1: The startWebVOWL function

It important to note that the **initialize** function did not, initially, allow for these parameters to be passed. Usually, the WebVOWL viewer is only initialized through a file drop, a URL

parameter or by loading the default ontology file, stored in the project itself. As such, the function had to be adapted in order to receive these parameters. The updated function can be seen at Code Snippet 4.2.

```
app.initialize = function (ontologyContent, iri, version){
    globalOntologyContent = ontologyContent;
    globalIRI = iri;
    globalVersion = version;
        ...
        ontologyMenu.setup(loadOntologyFromText);
```

Code Snippet 4.2: The initialize function

As can be seen at Code Snippet 4.2, the ontology is loaded through the **loadOntologyFromText** function. The function definition is done during the setup, in the **parseOntologyContent** function. This function has also been updated, in order to support the new possible way of loading the ontology content. This can be seen at Code Snippet 4.3. If the **globalOntologyContent** variable has a value, it means WebVOWL must load the ontology content received through parameter on the **initialize** function.

```
    function parseOntologyContent( content ){
      ontologyMenu.append_bulletPoint("Reading ontology graph ... ");
      var _loader = ontologyMenu.getLoadingFunction();
    if(globalOntologyContent){
      _loader(globalOntologyContent, globalIRI, "noAlternativeNameYet");
    }else {
      _loader(content, ontologyIdentifierFromURL, "noAlternativeNameYet"
    );
    }
    }
```

Code Snippet 4.3: The parseOntologyContent function

From this point on, the logic remains unchanged. The **loadOntologyFromText** function does the following:

- Validates that the JSON ontology content is not empty;
- Validates if the JSON content is valid;
- Caches the ontology content, and loads it into the viewer.

The final result can be seen in Figure 4.1. It is important to note that the WebVOWL viewer **maintains all of its previously available functionalities**, such as:

- Zooming
- Editing
- Detail visualization
- Exporting

4.1.2 Views

The TICO-GUI project contains multiple views that allow the user to fulfill the intended functions of the project.
The first view the user sees, after the project is started, is the **Home menu**. It consists of

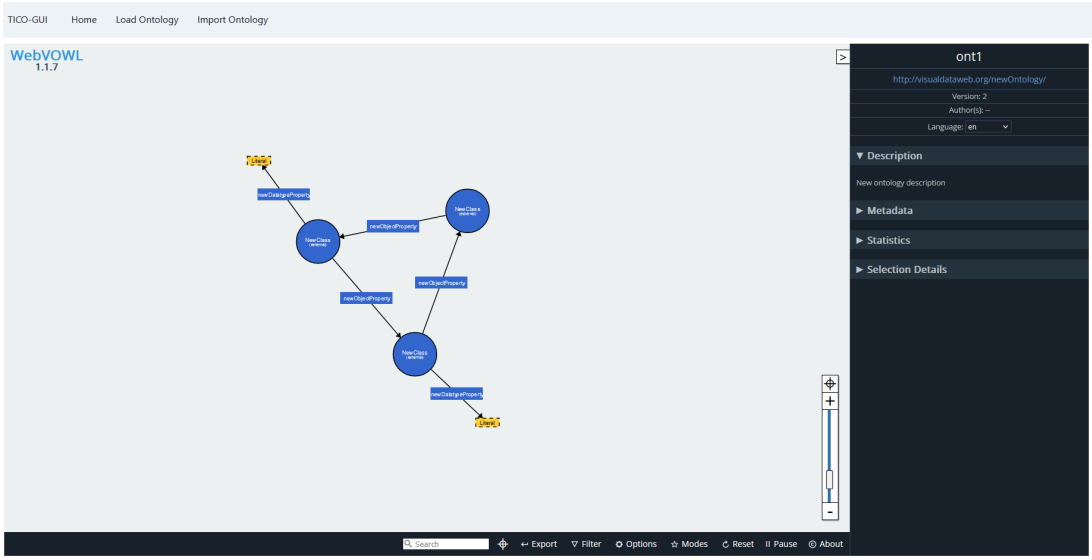

Figure 4.1: WebVOWL viewer inside iframe (Large Visualization page)

a simple menu containing the two possible paths to visualize an ontology's evolution - Load or Import - made available through two Link components. This view can be seen in Figure 4.2.

Figure 4.2: The Home menu

Load Ontology Page

The Load Ontology page allows the user to select one of the ontologies previously stored in the Fuseki storage, in order to visualize its evolutionary process.
The page is composed of a descriptive heading, and a select menu, containing all the Ontologies (fetched from the back-end during loading). The actual page can be seen at Figure 4.3.

Import Ontology Page

The Import Ontology page allows the user to begin the evolutionary process of an Ontology. The view contains fields for the user to insert all the necessary information: the Ontology's name/descriptor, the Ontology file and the Instances file. Once all the information is inserted

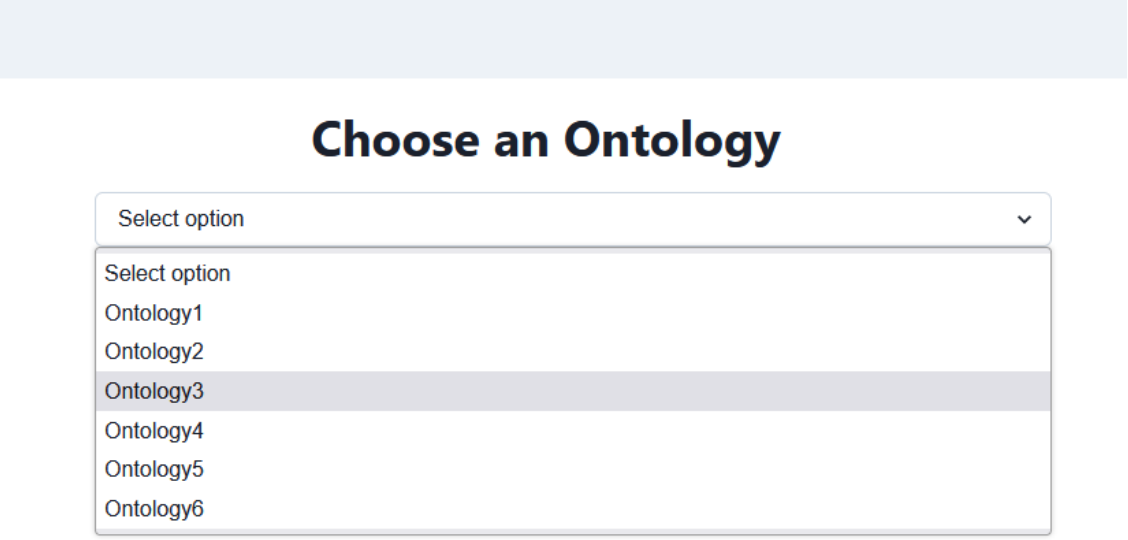

Figure 4.3: The Load Ontology page

(and if everything is valid), the user can then submit and move on to the next step of the process. The actual page can be seen at Figure 4.4.

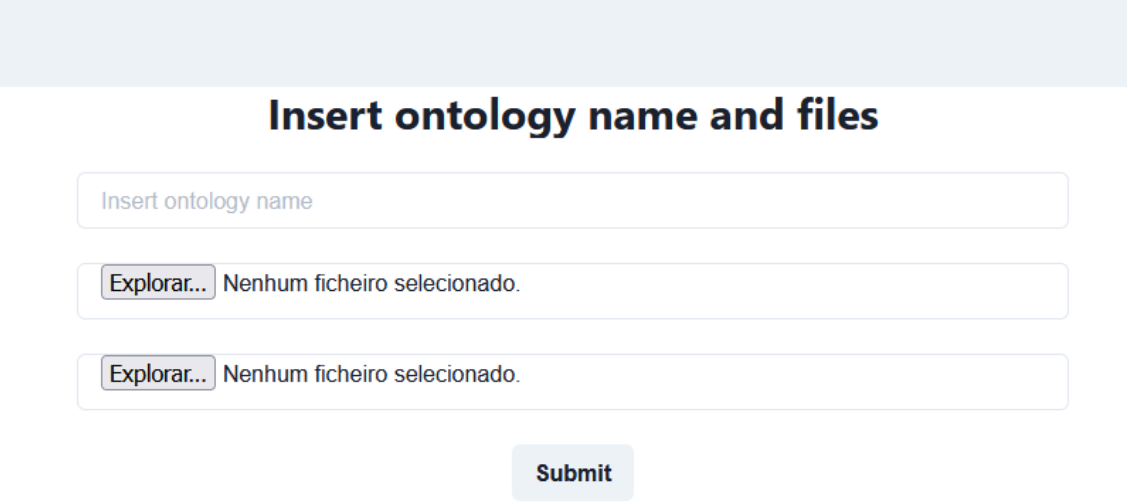

Figure 4.4: The Import Ontology page

Edit Evolutionary Actions Page

The Edit Evolutionary Actions page allows the user to edit or delete the generated Evolutionary Actions. Each of these Actions is displayed on screen, alongside each of its respective properties.

The TICO-Main project deals with multiple types of Evolutionary Actions, each containing different properties, of different types. As such, the front-end application needed not only to accept these different types of Actions, but also to be flexible, in this regard. In order

to achieve this, an effort was made to make the front-end **completely agnostic** as far as to which type each Action is, **treating each of them as a dynamic object**. How this was achieved is better explained further in the report, in the subsection concerning the Domain/DTO classes 4.1.4.

Considering this, the front-end application is ready to handle the following types of properties:

- String;
- Boolean;
- Number;
- Array.

These primitive types were all identified as possible property types during development. In the case of an Array property, its content can obviously be any number of properties of the other identified types. 4.4

```
1    actions.map((action, index) => {
2        const entries = Object.entries(action).filter(value => value[0]
   !== "className");
3        ...
4        return (
5            entries.length > 0 && entries.map(value => {
6                if(typeof(value[1]) === "number"){
7                    ...
8                } else if (...)
9        )
10   }
```

Code Snippet 4.4: Rendering logic for Edit Evolutionary Actions page

As mentioned before, all of these properties can be edited (with exception of the className property, which functions as a purely informational field), and each Evolutionary Action can be deleted if desired.

Once the process is finished, the user can submit the actions, and proceed to the visualization step. The actual page can be seen at Figure 4.5.

Evolution Visualization Page

This page presents the result of the evolutionary process to the user, by showing the changes performed on the Ontology.
The page contains two WebVOWL viewers on the right side of the page, alongside some extra information about each of the Ontology's versions on the left. If the user desires, it is possible to see each Ontology on a larger screen, by clicking the "magnifying glass" icon next to the info box (the Large Visualization page is described below, in 4.1.2). The actual page can be seen at Figure 4.6.

Large Visualization Page

The Large Visualization Page allows the user to see one of the Ontology's versions on a larger scale, for easier analysis and editing. The whole page is composed of a single WebVOWL viewer, containing the selected Ontology version. The actual page can be seen at Figure 4.7.

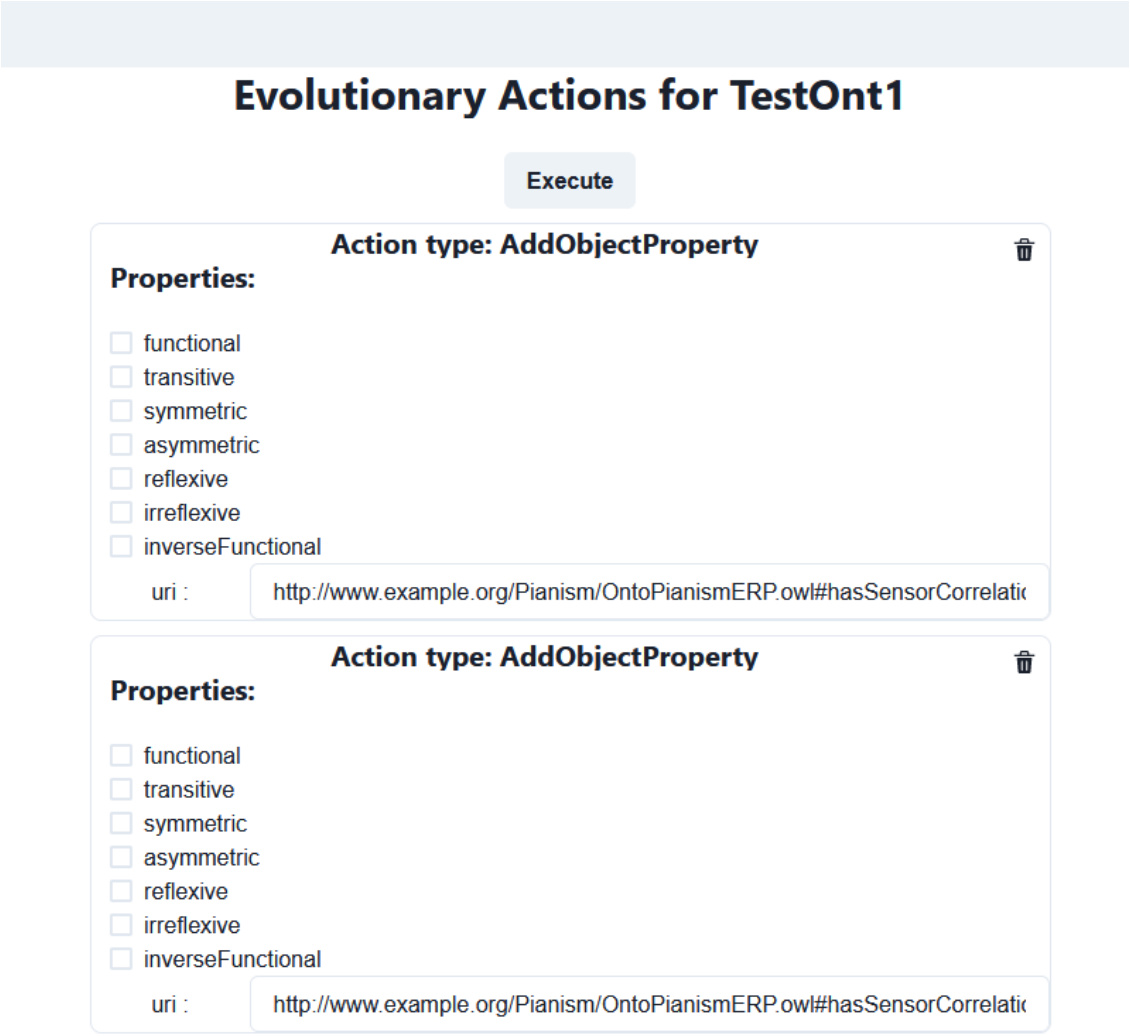

Figure 4.5: The Edit Evolutionary Actions Page

Figure 4.6: The Evolution Visualization Page

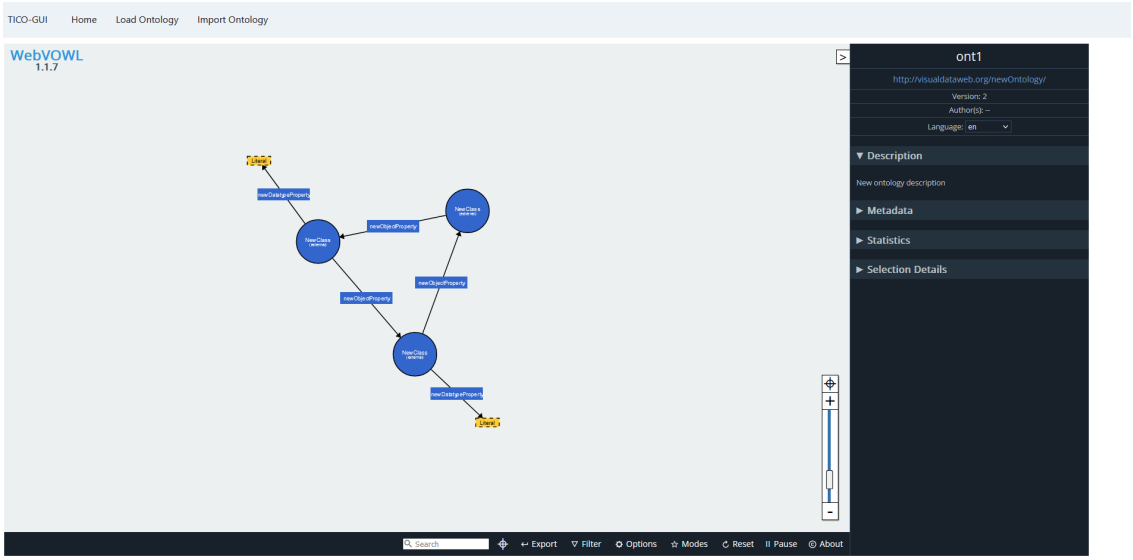

Figure 4.7: The Large Visualization Page

Modules

The following are modules that are re-used throughout the TICO-GUI project, and not in a single page.

LoadingPage.tsx This module allows the TICO-GUI project to inform the user that something is loading. It usually happens when something is being processed, or an API called is being made. The module is composed of a grey layer that covers the entire page, and a loading icon that conveys to the user that something is being done on the background. This layer also prevents the user from clicking on anything else on the page, and possibly disrupting the operation. The actual module can be seen at Figure 4.8.

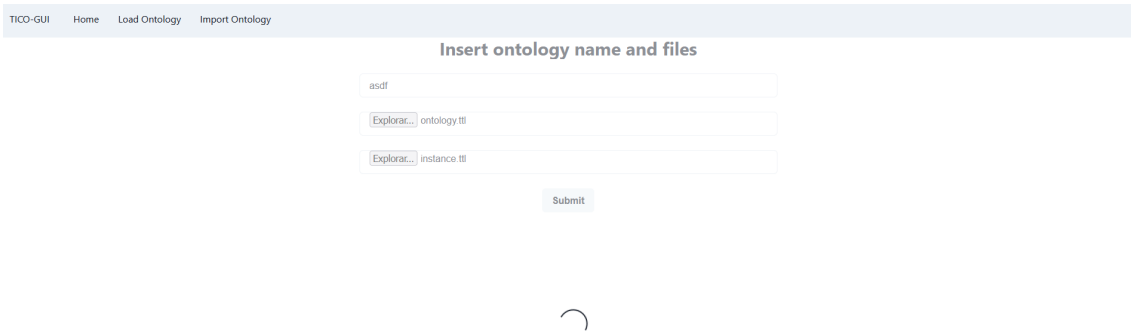

Figure 4.8: The Loading module

Navbar.tsx This module allows the user to navigate through the TICO-GUI project. The Navbar is a simple bar component, containing the name of the project, and all the links from which the user can begin its journey. The actual module can be seen at Figure 4.9.

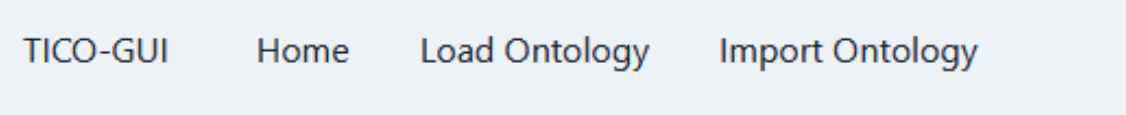

Figure 4.9: The Navbar module

WebVOWLIFrame.tsx This module allows the user to visualize an Ontology. It functions as a wrapper of the iframe used to show the WebVOWL application. It initializes the viewer (calls the "startWebVOWL" function on the HTML) by using the code shown on Code Snippet 4.5: it accesses the iframe reference, and then calls the function. The actual module can be seen at Figure 4.10.

```
1  useEffect(() => {
2        iframeRef.current!.contentWindow!.addEventListener('
   DOMContentLoaded', (event) =>
3            iframeRef.current!.contentWindow!.startWebVOWL(ontology, iri
   , version);
4          });
5   }, []);
```

Code Snippet 4.5: useEffect in WebVOWLIFrame

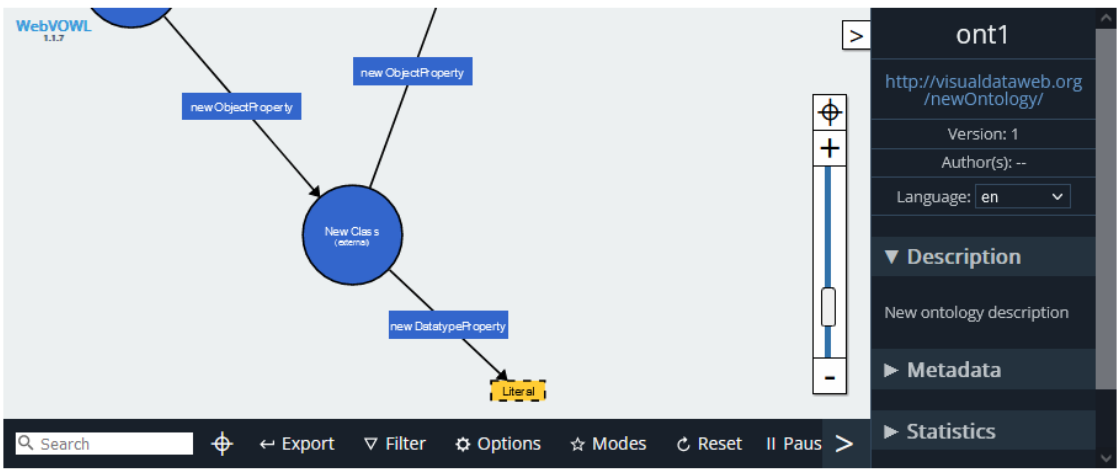

Figure 4.10: The WebVOWL IFrame module

4.1.3 Services

There is only one service class on the TICO-GUI project, and that is **OntologyService**. This service is responsible for communicating with the TICO-API project, and ensures the separation of responsibilities (React views are responsible for rendering, the service is responsible

for API calls).
The service has the following methods:

- getOntologies() - returns all the Ontologies from the Fuseki storage;
- getOntology(name: string) - returns a single Ontology from the Fuseki storage, by name;
- getEvolutionaryActions(ontology: string, instance: string) - returns all the generated Evolutionary Actions, given an ontology and an instance file content;
- executeActions(name: string, ontology: string, instance: string, actions: string) - returns the resulting Ontology of an execution of a set of Evolutionary Actions (passed as string for easier communication with the back-end).

4.1.4 DTO

A few DTO (Data Transfer Object) [47] classes are used, in order to ensure that the data being returned from the API has the expected composition. The classes are as follows:

- EvolutionaryActionDTO: this class contains is composed of multiple properties, composed of **a key of type string, and a value of type any**. This allows the front-end application to be agnostic regarding which Action types are being received. It allows for the Evolutionary Actions to be treated as dynamic objects. The class code can be seen at Code Snippet 4.6
- OntologyDTO: contains all the fields of a complete Ontology: name (string), version (string), Ontology content (string) and Viewer version content (string).
- SimpleOntologyDTO: only contains the name of the Ontology.

```
1  export default class EvolutionaryActionDTO {
2    [key: string]: any;
3  }
```

Code Snippet 4.6: EvolutionaryActionDTO

4.2 TICO-API

The TICO-API project is the core of the project, as it functions as the connection between all the other components of the system. It communicates with TICO-Main, the Fuseki storage and OWL2VOWL, orchestrating its actions, in order to return its results to the front-end. This section details the structure and the most important logic and actions performed by the project, as well as the Fuseki storage connection.

4.2.1 Fuseki

The Fuseki storage is the storage component of the system, and TICO-API is the only project that communicates directly with it.
TICO-API expects the Fuseki instance to **already be up, when the application starts**. This was an intentional design decision, as the application cannot function in any way without access to the storage. As such, the previously mentioned **AdminService** is responsible for not only ensuring that Fuseki is up, but also to ensure that it contains all the necessary datasets. The defined datasets for Fuseki are as follows:

- *ontologies* - stores all the Ontology contents
- *instances* - stores all the Instance contents
- *viewer* - stores all the Viewer version contents

The AdminService ensures that all of these datasets already exists, by trying to get each of them individually. If the return code is "404" (Not Found), TICO-API performs another request in order to create the missing dataset. This is performed upon launching the application, and before the API is available to be called. This way, it is ensured that if TICO-API is up and running, so is Fuseki. An example of this logic can be seen at Code Snippet 4.7.

```
1  URL postOntologiesDataset = new URL(fusekiURL + "/$/datasets?dbName="
     + ontologiesDataset + "&dbType=tdb");
2
3     URL getOntologiesDataset = new URL(fusekiURL + "/$/datasets/" +
     ontologiesDataset);
4
5     HttpURLConnection conGetOntologies = (HttpURLConnection)
     getOntologiesDataset.openConnection();
6     conGetOntologies.setRequestProperty("Authorization", authHeaderValue
     );
7     conGetOntologies.setRequestMethod("GET");
8     Integer codeOntologies = conGetOntologies.getResponseCode();
9
10    if(codeOntologies == 404){
11        HttpURLConnection conPostOntologies = (HttpURLConnection)
     postOntologiesDataset.openConnection();
12        conPostOntologies.setRequestProperty("Authorization",
     authHeaderValue);
13        conPostOntologies.setRequestMethod("POST");
14        Integer codeOntologiesPost = conPostOntologies.getResponseCode()
     ;
15        if(codeOntologiesPost == 200){
16            logger.error("Successfully created Ontologies dataset.");
17        }else {
18            logger.error("An internal Fuseki error has occurred, while
     trying to create Ontologies dataset. Check Fuseki's logs in order to
     better understand the problem.");
19        }
20    }
```

Code Snippet 4.7: Fuseki datasets setup

The other service that deals with Fuseki is FusekiService, which is covered more in depth in the subsection 4.2.3.

4.2.2 Controller

The Controller layer of the application is the channel through which external applications can communicate with the project, by making calls to its REST API. All the available REST methods of TICO-API are available on a single controller, **OntologyController**. This controller offers the following methods:

- getOntologies - Returns all the Ontologies stored on Fuseki;
- getOntologyByName - Returns the Ontology versions stored in Fuseki by name (e.g.: The two versions of an Ontology just created on the application);

- getEvolutionaryActions - Returns the Evolutionary Actions, based on an Ontology content and Instances content;
- createOntology - Returns the resulting Ontology, after all the actions were executed and the result was stored on Fuseki.

4.2.3 Services

The Service layer of the application is an intermediary between the Controller classes and the storage. It contains most of the logic existing in the project, while the Controller classes should mostly just call Service methods. This section details each of the service classes (with the exception of AdminService, which is already covered in section 4.2.1), and touches upon the most important pieces of logic in each of them.

FusekiService

The Fuseki service is the service responsible for making all the necessary calls to the Fuseki storage. It offers the following methods:

- createOntology;
- createOntologyVersion;
- getOntologies;
- getOntologyViewerVersionByIRI.

Each of the methods are described in detail below.

createOntology is the method responsible for creating the first version of an Ontology in the system. It requires the ontology's name, its content and the viewer version content. After connecting to the storage through an RDFConnection [48], it creates the first version of the Ontology's content on Fuseki, as well as the first version of the Viewer version. The first operation's code can be seen at Code Snippet 4.8. After the query string is created, the query is executed through an UpdateRequest, using the connection previously created. The exact same process is repeated for the Viewer version and the Instances content, but on different connections, this time connected to the "viewer" and "instances" dataset.

```
1  String queryText = "PREFIX dcterms: <prefix1> \n" +
2        "INSERT DATA { <" + IRI + "> <version0> \"" + ontologyContent +
    "\" }";
3
4     UpdateRequest query = UpdateFactory.create(queryText);
5
6     conn0.update(query);
```

Code Snippet 4.8: Creating Ontology in Fuseki

As can be seen in the previously mentioned Code Snippet, the SPARQL query consists of an "INSERT DATA", which introduces a line in the storage containing 3 fields: the ontology's IRI (identifier or name), its version (which in this case is the first one, or "version 0", and finally, its content.

createOntologyVersion is the method responsible for creating a new version of an already existing Ontology. The method creates an RDFConnection for each of the datasets, validates if the original Ontology already exists in the system, before creating a new version - with updated content and version, but the same IRI. The first operation's code can be seen at Code Snippet 4.9.

```
1   String queryCountText = "SELECT (COUNT(*) as ?TotalNumber) WHERE { ?
    s ?p ?o FILTER( CONTAINS(str(?s) , \"" + IRI + "\" )) }";
2
3   conn0.querySelect(queryCountText, (qs) -> {
4       String totalNumberString = qs.getLiteral("TotalNumber").
    getString() ;
5       totalNumber.set(Integer.parseInt(totalNumberString));
6   });
7
8   if (totalNumber.get() > 0){
9       \\ does exist
10      ...
11  }
```

Code Snippet 4.9: Creating Ontology in Fuseki

As can be seen in the previously mentioned Code Snippet, a COUNT query is performed on the storage, in order to find out if there are any Ontologies with this IRI already. If there are, the new version is created (in all different datasets).

getOntologyViewerVersionByIRI is the method responsible for returning a Viewer version of an Ontology. In the context of the front-end application, the original Ontology content does not matter: all that matters is its visualization. As such, it makes sense for a method that returns only the Viewer version to exist. After establishing a RDFConnection, it makes query in order to get the Viewer version content of an Ontology with a determined IRI and version, as can be seen on Code Snippet 4.10.

```
1   String queryTextViewerOriginal = "SELECT ?s ?p ?o WHERE { ?s ?p ?o
    FILTER( CONTAINS(str(?s) , \"" + IRI + "\" ) && CONTAINS(str(?p) , \"
    " + version1 + "\" ) ) }";
```

Code Snippet 4.10: Get Viewer content by ID and Version

getOntologies is the method responsible for returning all the Ontologies in the system. This method returns an array, in which each value contains the IRI of one of the Ontologies, as well as an array with all its different version numbers. This return is used in the front-end application, for instances, for the user to choose which Ontology to visualize. A snippet of this logic can be seen at Code Snippet 4.11.

```
String queryTextOriginal =
"SELECT DISTINCT (str(?s) as ?iri) WHERE { ?s ?p ?o }";

QueryExecution qExec = conn0.query(queryTextOriginal);
ResultSet rs = qExec.execSelect() ;
while(rs.hasNext()) {
    ...
    String queryVersions =
    "SELECT DISTINCT (str(?p) as ?version) WHERE { ?s ?p ?o FILTER (
CONTAINS (str(?s), \"" + actualContent + "\" ))}";
    ...
}
```

Code Snippet 4.11: Get Ontologies

TicoService

The TicoService is the service responsible for handling all the operations related to the TICO-Main project. More specifically, it is responsible for the process of handling the evolutionary process of an ontology. The service offers the following methods:

- getEvolutionaryActions - This method makes a call to the TICO-Main project, in order to receive the generated Evolutionary Actions (based on an Ontology content and an Instances content)
- createOntology - This method is responsible for creating an Ontology. It makes a call to the TICO-Main project, in order to execute the previously generated Evolutionary Calls. After receiving the result, it makes a call to the OWL2VOWL project in order to get the Viewer content of both versions, before saving everything to the Fuseki storage. A snippet of the logic can be seen at Code Snippet 4.12.

```
HttpRequest request = HttpRequest.newBuilder()
            .uri(new URI(ticoURL + "/execute"))
            .version(HttpClient.Version.HTTP_1_1)
            .header("Content-Type", "application/json")
            .POST(HttpRequest.BodyPublishers.ofString(inputString))
            .build();

HttpResponse<String> executeResponse = httpClient.send(request,
HttpResponse.BodyHandlers.ofString());
String executeResult = executeResponse.body(); //this is the new
version!
...
String viewerVersionOriginal = owl2VowlService.getViewerVersion(
ontologyContent.ontology);
String viewerVersion = owl2VowlService.getViewerVersion(
executeResult);
...
fusekiService.createOntology(IRI, ontologyContent.ontology,
viewerVersionOriginal);
fusekiService.createOntologyVersion(IRI, executeResult,
viewerVersion);
...
```

Code Snippet 4.12: Create Ontology

OWL2VOWLService

The OWL2VOWLService is the service responsible for communicating with the OWL2VOWL project. This service offers two methods:

- getServerTimeStamp - This method returns a valid timestamp, to be used when making calls to the OWL2VOWL server. This step is necessary, as a valid timestamp is **mandatory** for the rest of the calls to the project
- getViewerVersion - This method returns the Viewer (VOWL) version of an Ontology.

4.3 TICO-Main

The developments on TICO-Main consisted mainly of adapting the existing application, considering the requirements of the project. The core of the application (such as the Comparator logic, the classes and services) remained mostly unchanged. The main changes are:

- Convert it into a Spring Boot [49] application, in order to allow for the creation of a REST API
- The separation of the existing logic, in order to allow for the generation and execution of Evolutionary Actions at different points in time

The changes are further explained below.

4.3.1 REST API

In order to build the REST API, a new Spring Boot project was created, using Spring Initializr [50], an official generator of boilerplate Spring Boot projects (choosing the Maven and Java options).
After the project was created, all of the classes of the original TICO project were simply copied into the new Spring Boot application. All that was left was to create a controller class.

TicoController

The TicoController class offers two different methods:

- generateEvolutionaryActions - returns all the generated Evolutionary Actions
- executeEvolutionaryActions - returns the result of executing the Evolutionary Actions. Receives an array of EvolutionaryActions (in the form of a string, in order to ease communication).

In both cases, the Ontology and Instance contents are converted into OntModel [51], which is the class that the Comparator methods are expecting. A snippet of this code can be seen at Code Snippet 4.13. In the snippet, both the Ontology and Instances content are converted from the expected notation (Turtle) into instances of the OntModel class.

```
1     OntModel ontModel = ModelFactory.createOntologyModel();
2     ontModel.read(IOUtils.toInputStream(dto.ontology, "UTF-8"), "", "TTL
      ");
3     OntModel instanceModel = ModelFactory.createOntologyModel();
4     instanceModel.read(IOUtils.toInputStream(dto.instance, "UTF-8"), "",
       "TTL");
5
6     ModelManager.getManager().setup(ontModel, ontModel, instanceModel);
```

Code Snippet 4.13: TicoController logic

4.3.2 Logic changes

The original version of the project offers the method **run**, which is responsible for both generating and executing all the Evolutionary Actions. This did not serve the requirements of the project, and as such, the logic had to be separated. A copy of the original method is presented at Code Snippet 4.14. As can be seen in the Code Snippet, the method relied on multiple methods (checkNewClasses, checkForOntProperties...) to generate Evolutionary Actions, and execute them right away. There were also other methods that were unnecessary for the problem at hands (such as printEntityMetricStats).

```
1     public void run()
2     {
3         List<String> individuals_uris = SPARQLUtils.
      getIndividualsSPARQL(ModelManager.getManager().getInstanceModel());
4         int partitionSize = 10;
5         List<List<String>> partitions = Utilities.chopped(
      individuals_uris, partitionSize);
6
7         for(List<String> partition : partitions)
8         {
9             checkNewClasses(partition);
10            checkForOntProperties(partition);
11            copyIndividualsToEvolvedModel(partition);
12            getIndividualsMetrics(partition);
13            runComparatorOnIndividuals(partition);
14            updateTemporalRestrictions();
15            printEntityMetricsStats();
16        }
17    }
```

Code Snippet 4.14: The original "run" method

In order to achieve the intended result, each of these methods was analyzed and either removed or updated. This resulted into a new method, **getEvolutionaryActions**, which looks like what can be seen on Code Snippet 4.15. This method returns a list of all the Evolutionary Actions generated, for the given Ontology and Instances. The updated methods (such as **checkNewClassesReturn**) suffered small changes, like adding the Evolutionary Actions to an array and returning, instead of executing it right away.

```
1     public List<EvolutionaryAction> getEvolutionaryActions()
2     {
3         List <String> individuals_uris = SPARQLUtils.
      getIndividualsSPARQL(ModelManager.getManager().getInstanceModel());
4         List<EvolutionaryAction> toReturn = new ArrayList<>();
5
6         int partitionSize = 10;
7
8         List<List<String>> partitions = Utilities.chopped(
      individuals_uris, partitionSize);
9
10        for(List<String> partition : partitions)
11        {
12            List<EvolutionaryAction> newClasses = checkNewClassesReturn(
      partition);
13            List<EvolutionaryAction> ontProp =
      checkForOntPropertiesReturn(partition);
14            copyIndividualsToEvolvedModel(partition);
15            List<EvolutionaryAction> moreActions =
      runComparatorOnIndividualsReturn(partition);
16            List<EvolutionaryAction> aux = Stream.of(newClasses, ontProp
      , moreActions).flatMap(Collection::stream).collect(Collectors.toList
      ());
17            toReturn.addAll(aux);
18        }
19        return toReturn;
20    }
```

Code Snippet 4.15: The getEvolutionaryActions method

Finally, the execution of the Actions is performed on another new method, **executeActions**. This method receives an array of Evolutionary Actions, and executes each of them, before returning the resulting Ontology.
This, however, created a challenge: *EvolutionaryAction* is an abstract class [52], extended by multiple other classes (such as AddClass, AddProperty, among others). In short, an Evolutionary Action can have a number of different implementations - it can consist of different operations, such as adding a class, a property. Since the front-end application is unaware of the class of each Action, all the Evolutionary Actions arrive in a single array, **and each of the Action types has its own distinct execute method**, the only way to reliably execute each Action correctly, is for each of the classes to be identifiable through a property.
The chosen property name was "className", and it is a hardcoded value that each of the classes that implement *EvolutionaryAction* have (for instances, AddClass has a "className" property with value "AddClass").
It is then possible to execute each of the Evolutionary Actions. This is achieved through the method **findClassAndExecuteActions**. A snippet of this method can be seen in Code Snippet 4.16.

```
1    public void findClassAndExecuteActions(JSONObject action) throws
     JsonProcessingException {
2        String aux = action.get("className").toString();
3
4        switch (aux) {
5            case "AddAllValuesFromRestriction" -> {
6                AddAllValuesFromRestriction auxClass1 = new ObjectMapper
     ().readValue(action.toString(), AddAllValuesFromRestriction.class);
7                auxClass1.execute();
8            }
9            case "AddCardinalityRestriction" -> {
10               AddCardinalityRestriction auxClass2 = new ObjectMapper()
     .readValue(action.toString(), AddCardinalityRestriction.class);
11               auxClass2.execute();
12           }
13           ...
14       }
15   }
```

Code Snippet 4.16: The findClassAndExecuteActions method

4.3.3 Testing and Validation

The testing of the project was something that was taken into consideration from the very beginning of development, as it is a fundamental practice, in order to ensure the quality and functioning of the code. The tests performed included:

- Unit tests - Unit tests were made for the TICO-GUI project, in order to ensure the interfaces render correctly under the expected conditions;
- Integration tests - Integration tests were made for the TICO-API project, in order to ensure the different services and classes work correctly together.

While end-to-end tests were also considered for the project, they were eventually removed from scope, due to complexity and time constraints. They are, however, considered important future additions to the project.

The project fulfills every single functional and non-functional requirement proposed, in Chapter 3, Analysis and Design 3, as evidenced by the list below:

- The application allows the user to import ontology and instance files into the system;
- The application allows the user to edit the generated evolutionary actions;
- The application allows the user to visualize the evolutionary process of an ontology;
- The application allows the user to load the evolutionary process of an ontology after importing;
- The application allows for the submission of files, specifically Turtle files;
- The application offers a simple interface that allows for the visualization of the evolutionary process of ontologies.

Chapter 5

Conclusion

This final section of the report serves as summary and reflection on the project. It starts by giving a short overview on the report, as well as on the achieved objectives. It then concludes by making some considerations on the success of the project, this report included.

5.1 Report Overview

The first chapter 1 contextualizes the problem, and details the results that are expected from the project. This serves as a basis for chapter 2, in which the state of the art on ontologies and ontology evolution visualization methods is presented, in order to decide on the best technological solutions for the project. After some consideration, the WebVOWL viewer was chosen. Chapter 3 offers the analysis and design of the solution - presenting, also, an alternative architecture, if a different viewer had been chosen. Chapter 4 details the implementation process and attempts to explain all the major decisions taken.
Overall, the report is considered to cover all the requirements that were originally proposed. A list of the achieved goals can be seen in Table 5.1.

Table 5.1: Achieved Goals

Objectives	**Result**
Theoretical overview	Done
State of the Art	Done
Analysis of available technologies	Done
Design of a solution	Done
Use cases 1 - 4	Done
Testing	Partially Done

The project offered several challenges. The most obvious one comes from a lack of knowledge regarding some of the relevant concepts and technologies, before the start of the project - such as Jena, SPARQL and ontology engineering in general. A considerable time frame was dedicated, before development, to research and learning about these concepts, in order to better choose an approach for the problem.
The architecture of the solution brought challenges of its own, because of the number of individual projects that had to work in tandem for the solution to work. Not only that, but the fact that an existing project had to be adapted for the solution, required special attention during the analysis and design phase, before the start of development, in order to ensure that the implementation process happened as smoothly as possible.
Overall, the project can be considered a success, not only from a learning standpoint, but

also considering the end result. The few improvements that still could be made are detailed on the next section 5.2.

5.2 Future Work

Due to time constraints or oversights during the design process, the following developments should be considered on future iterations of the solution. Although these changes are not considered mandatory for the full completion of the project, they are still considered improvements over the current results.

- Implement a way to display Instances of each Class (Instances are currently being stored in Fuseki, but there is currently no way to visualize them in WebVOWL's viewer)
- Develop a way to highlight new elements, in the updated version of an Ontology (not supported by the WebVOWL viewer)
- Perform user testing on real life scenarios, to find new improvements or use cases for the solution
- Implement end-to-end testing, to ensure the system as a whole works as expected
- Implement support for Composite Evolutionary Actions (Actions that include multiple different actions).

5.3 Final appreciation

In conclusion, the project accomplished everything that was proposed. The implemented solution accomplishes the objective of building an application with an user interface, capable of aiding the user throughout the evolutionary process of an ontology. While there were challenges to overcome and a few improvements that could still be made, the final result accomplishes the requirements proposed at the beginning of development.

Bibliography

[1] A. Canito, J. Corchado, and G. Marreiros, "A Systematic Review on Time-Constrained Ontology Evolution in Predictive Maintenance," *Artif. Intell. Rev.*, vol. 55, no. 4, pp. 3183–3211, Apr. 2022, Place: USA Publisher: Kluwer Academic Publishers, issn: 0269-2821. doi: 10.1007/s10462-021-10079-z. [Online]. Available: https://doi.org/10.1007/s10462-021-10079-z.

[2] M. Dudáš, S. Lohmann, V. Svátek, and D. Pavlov, "Ontology visualization methods and tools: A survey of the state of the art," *The Knowledge Engineering Review*, vol. 33, Jul. 2018. doi: 10.1017/S0269888918000073.

[3] B. J. Oates, *Researching Information Systems and Computing*. Sage Publications Ltd., 2006, isbn: 1412902231.

[4] W3, *Tim berners-lee*, url=https://www.w3.org/People/Berners-Lee/, Last checked on: June 3, 2020, 2020. [Online]. Available: https://www.w3.org/People/Berners-Lee/, (accessed: 03.06.2020).

[5] CERN, *The birth of the web*, url=https://home.cern/science/computing/birth-web, Last checked on: June 10, 2020, 2020. [Online]. Available: https://home.cern/science/computing/birth-web, (accessed: 10.06.2020).

[6] T. Berners-Lee, *The semantic web*. [Online]. Available: https://www.w3.org/2000/Talks/0906-xmlweb-tbl/text.htm.

[7] M. M. Taye, "Understanding semantic web and ontologies: Theory and applications," 2010. doi: 10.48550/ARXIV.1006.4567. [Online]. Available: https://arxiv.org/abs/1006.4567.

[8] J. Cardoso, "The syntactic and the semantic web," in Jan. 2007. doi: 10.4018/9781599040455.ch001.

[9] T. Berners-Lee, *Semantic Web - XML2000 - Slide list*. [Online]. Available: https://www.w3.org/2000/Talks/1206-xml2k-tbl/ (visited on 02/24/2023).

[10] H. V. Dam, J. Engberg, and H. Gerzymisch-Arbogast, *Knowledge Systems and translation*. Mouton de Gruyter, 2012.

[11] G. Palermo, "The ontology of economic power in capitalism: mainstream economics and Marx," *Cambridge Journal of Economics*, vol. 31, no. 4, pp. 539–561, Jan. 2007, issn: 0309-166X. doi: 10.1093/cje/bel036. eprint: https://academic.oup.com/cje/article-pdf/31/4/539/4714980/bel036.pdf. [Online]. Available: https://doi.org/10.1093/cje/bel036.

[12] G. L. Zuniga, *An Ontology Of Economic Objects*, MPRA Paper, Apr. 1999. [Online]. Available: https://ideas.repec.org/p/pra/mprapa/5566.html.

[13] B.-J. van Putten, S. Wolfe, and V. Dignum, "An ontology for traffic flow management," Sep. 2008, isbn: 978-1-60086-997-6. doi: 10.2514/6.2008-8946.

[14] C. Reyes-Peña and M. Tovar-Vidal, "Ontology: Components and Evaluation, a Review," en, *Research in Computing Science*, vol. 148, no. 3, pp. 257–265, Dec. 2019, issn: 1870-4069. doi: 10.13053/rcs-148-3-21. [Online]. Available: http://rcs.cic.ipn.mx/2019_148_3/Ontology_%20Components%20and%20Evaluation_%20a%20Review.pdf (visited on 10/13/2023).

[15] T. Slimani, "A Study Investigating Typical Concepts and Guidelines for Ontology Building," en,

[16] A. Gómez-Pérez, M. Fernandez-Lopez, and O. Corcho, *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. First Edition* (Advanced Information and Knowledge Processing). Springer London, 2006, isbn: 9781852338404. [Online]. Available: https://books.google.pt/books?id=qR_3BwAAQBAJ.

[17] D. Taniar and J. Rahayu, *Web Semantics & Ontology* (IGI Global research collection). Idea Group Pub., 2006, isbn: 9781591409076. [Online]. Available: https://books.google.pt/books?id=uVu9AQAAQBAJ.

[18] B. Ulicny, J. Moskal, M. Kokar, K. Abe, and J. Smith, "Inference and Ontologies," *Advances in Information Security*, vol. 62, pp. 167–199, Oct. 2014, issn: 978-3-319-11390-6. doi: 10.1007/978-3-319-11391-3_9.

[19] *RDF - Semantic Web Standards*. [Online]. Available: https://www.w3.org/RDF/ (visited on 01/17/2023).

[20] *RDF Schema 1.1*. [Online]. Available: https://www.w3.org/TR/rdf-schema/ (visited on 02/15/2023).

[21] *OWL Web Ontology Language Guide*. [Online]. Available: https://www.w3.org/TR/owl-guide/ (visited on 01/17/2023).

[22] *SPARQL Query Language for RDF*. [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/ (visited on 02/02/2023).

[23] *What is SPARQL?* en-US. [Online]. Available: https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/ (visited on 02/20/2023).

[24] *Feature:SPARQL/OWL - SPARQL Working Group*. [Online]. Available: https://www.w3.org/2009/sparql/wiki/Feature:SPARQL/OWL (visited on 02/25/2023).

[25] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," en, *Information and Software Technology*, vol. 64, pp. 1–18, Aug. 2015, issn: 0950-5849. doi: 10.1016/j.infsof.2015.03.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584915000646 (visited on 02/19/2023).

[26] *5.11 Directed Graphs*. [Online]. Available: https://www.whitman.edu/mathematics/cgt_online/book/section05.11.html (visited on 02/19/2023).

[27] *Ontoviewer*. [Online]. Available: https://github.com/edmcouncil/onto-viewer/ (visited on 02/06/2023).

[28] *Ontosphere3d*. [Online]. Available: https://ontosphere3d.sourceforge.net/ (visited on 02/05/2023).

[29] P. Lambrix, Z. Dragisic, V. Ivanova, and C. Anslow, "Visualization for Ontology Evolution," Oct. 2016.

[30] N. Noy and M. A. Musen, "Promptdiff: A fixed-point algorithm for comparing ontology versions," in *AAAI/IAAI*, 2002.

[31] H. Kondylakis, M. Despoina, G. Glykokokalos, *et al.*, "Evordf: A framework for exploring ontology evolution," in *The Semantic Web: ESWC 2017 Satellite Events*, E. Blomqvist, K. Hose, H. Paulheim, A. Ławrynowicz, F. Ciravegna, and O. Hartig, Eds., Cham: Springer International Publishing, 2017, pp. 104–108, isbn: 978-3-319-70407-4.

[32] M. Burch and S. Lohmann, "Visualizing the evolution of ontologies: A dynamic graph perspective," *CEUR Workshop Proceedings*, vol. 1456, pp. 69–76, Jan. 2015.

[33] M. Horridge, R. S. Gonçalves, C. I. Nyulas, T. Tudorache, and M. A. Musen, "WebProtéGé: A Cloud-Based Ontology Editor," in *Companion Proceedings of The 2019 World*

Wide Web Conference, ser. WWW '19, event-place: San Francisco, USA, New York, NY, USA: Association for Computing Machinery, 2019, pp. 686–689, isbn: 978-1-4503-6675-5. doi: 10.1145/3308560.3317707. [Online]. Available: https://doi.org/10.1145/3308560.3317707.

[34] *Provide Web APIs to access ontologies in WebProtege · Issue #380 · protegeproject/webprotege*, en. [Online]. Available: https://github.com/protegeproject/webprotege/issues/380 (visited on 02/11/2023).

[35] *WebVOWL - Web-based Visualization of Ontologies*. [Online]. Available: http://vowl.visualdataweb.org/webvowl.html (visited on 02/25/2023).

[36] *JavaFX*, en-us. [Online]. Available: https://openjfx.io/localhost:1313/ (visited on 02/25/2023).

[37] *Getting Started – React*, en. [Online]. Available: https://reactjs.org/docs/getting-started.html (visited on 02/25/2023).

[38] *Introduction | Vue.js*. [Online]. Available: https://vuejs.org/guide/introduction.html (visited on 02/25/2023).

[39] *Apache Jena - Home*. [Online]. Available: https://jena.apache.org/ (visited on 10/01/2023).

[40] *Apache Jena - Apache Jena Fuseki*. [Online]. Available: https://jena.apache.org/documentation/fuseki2/ (visited on 10/14/2023).

[41] A. Canito, A. Nobre, J. Neves, J. Corchado Rodríguez, and G. Marreiros, "Using sensor data to detect time-constraints in ontology evolution," *Integrated Computer-Aided Engineering*, vol. 30, pp. 1–16, Feb. 2023. doi: 10.3233/ICA-230703.

[42] *The C4 model for visualising software architecture*. [Online]. Available: https://c4model.com/ (visited on 10/01/2023).

[43] P. Kruchten, "Architectural Blueprints—The "4+1" View Model of Software Architecture," en,

[44] S. Agile, *Nonfunctional requirements*, url=https://www.scaledagileframework.com/nonfunctional-requirements/, Last checked on: May 21, 2020, 2018. [Online]. Available: https://www.scaledagileframework.com/nonfunctional-requirements/, (accessed: 21.05.2020).

[45] *RDF 1.1 Turtle*. [Online]. Available: https://www.w3.org/TR/turtle/ (visited on 10/12/2023).

[46] *Chakra UI - A simple, modular and accessible component library that gives you the building blocks you need to build your React applications.* en. [Online]. Available: https://chakra-ui.com (visited on 10/10/2023).

[47] baeldung, *The DTO Pattern (Data Transfer Object) | Baeldung*, en-US, Aug. 2021. [Online]. Available: https://www.baeldung.com/java-dto-pattern (visited on 10/11/2023).

[48] *Apache Jena - RDF Connection : SPARQL operations API*. [Online]. Available: https://jena.apache.org/documentation/rdfconnection/ (visited on 10/11/2023).

[49] *Spring Boot*, en. [Online]. Available: https://spring.io/projects/spring-boot (visited on 10/11/2023).

[50] *Spring Initializr*, en. [Online]. Available: https://start.spring.io (visited on 10/11/2023).

[51] *OntModel (Apache Jena)*. [Online]. Available: http://loopasam.github.io/jena-doc/documentation/javadoc/jena/com/hp/hpl/jena/ontology/OntModel.html (visited on 10/11/2023).

[52] *Java Abstraction*, en-US. [Online]. Available: https://www.w3schools.com/java/java_abstract.asp (visited on 10/11/2023).