



6CCS3PRJ Individual Project

**Knowledge Graph generation from
diverse datasets using SPARQL Anything**

Final Project Report

Author: Alvin Lam

Supervisor: Dr Albert Meroño-Peñuela

Student ID: K20045786

April 5, 2023

Abstract

Increasing internet use has resulted in vast amounts of online resources on the World Wide Web (WWW), but structural representation of this data is currently limited. This project aims to fill these gaps by applying Artificial Intelligence (AI) techniques to represent musical heritage data on the WWW. In particular, this project will use the tool SPARQL Anything to generate a knowledge graph based on an ontology. The knowledge graph will represent data from an organ dataset and be expanded upon by incorporating external resources. Furthermore, this report details the process of knowledge graph generation, which can be applied in other contexts. Upon evaluation, the produced knowledge graph was found to be high quality, easily interpretable and scalable, but there was still room for improvement. Extensions of this project could include the use of external datasets and data sources to broaden the knowledge graph.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary.
I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Alvin Lam

April 5, 2023

Acknowledgements

I would like to take this opportunity to thank my supervisor, Dr Albert Meroño-Peñuela, for his time and effort dedicated to guiding me with this project. I would also like to thank my family for their moral support throughout.

Contents

1	Introduction	3
1.1	Rationale	3
1.2	Aims	4
1.3	Scope	5
2	Background	7
2.1	The Semantic Web	7
2.2	Ontology	10
2.3	Knowledge Graphs	11
2.4	SPARQL	12
3	Context	14
3.1	Organs	14
3.2	Polifonia Project	17
3.3	SPARQL Anything	19
3.4	Other Tools	20
4	Literature Review	24
4.1	Background and Context Information Sources	24
4.2	Knowledge Graph Generation Techniques	25
4.3	Knowledge Graph Evaluation Techniques	26
4.4	Additional Complementary Literature	27
5	Requirements	28
5.1	Preliminaries	28
5.2	Software Requirements	28
5.3	User Requirements	29
6	Specification	30
6.1	Software Requirement Specification	30
6.2	User Requirement Specification	33
7	Design	34
7.1	UML Sequence Diagram	34
7.2	Ontology Structure	35
7.3	Query Flow Diagram	36
7.4	Query Skeleton Structure	38

8	Implementation	41
8.1	Command Line Command	41
8.2	Prefixes	42
8.3	Ontology Reconstruction	43
8.4	Knowledge Graph Generation	47
8.5	Changes During Implementation	51
8.6	Software Maintenance	52
8.7	Results	53
9	Evaluation	54
9.1	Requirements-Based Assessment	54
9.2	Knowledge Graph Quality Assessment	56
9.3	Query Scalability Assessment	63
10	Legal, Social, Ethical & Professional Issues	66
10.1	Code of Conduct	66
10.2	FAIR Principles	67
11	Conclusion	69
11.1	Concluding Remarks	69
11.2	Limitations	70
11.3	Future Work & Extensions	71
	Bibliography	74
A	Appendix	80
A.1	Evaluated Knowledge Graphs	80
A.2	Knowledge Graph Understandability Survey	98
A.3	Knowledge Graph Understandability Example	98
B	User Guide	103
B.1	Instructions	103
C	Source Code	105
C.1	Declaration	105
C.2	GitHub Repository Link	105
C.3	Command Line	105
C.4	Knowledge Graph Generation Query	106

1

Introduction

Knowledge engineering involves designing, creating and maintaining knowledge-based systems [57] within the field of the semantic web, which structures information on web pages [8]. Moreover, a knowledge graph is an application of knowledge engineering and a graphical representation of real-world data. Ontologies, which are specifications of a concept [11], provide frameworks for knowledge graphs to input real-world data. This first chapter will introduce the rationale, aims and scope of this project.

1.1 Rationale

There are many information sources today that are still not structurally represented or fully explored on the World Wide Web (WWW). In an article [10], the authors (including creator of the WWW: Tim Berners-Lee) discuss substantial amounts of structured data (i.e. data having a structure or a schema [3]) published on the WWW but show there is ample opportunity for further exploration of other topics. Therefore, there is demand for a means of representing

information, potentially through the use of knowledge graphs, to make vast amounts of web data available for computational and human use. Finding an automated method of generating knowledge graphs from structured data or text found on the WWW is vital to addressing this demand. Current solutions for generating knowledge graphs include SPARQL-Generate [30], RML [53] and Large Language Models (LLMs). LLMs are machine-learning models trained by enormous amounts of data to mimic the patterns of natural language with ChatGPT [29] being an example of such implementations. Nonetheless, the capability of these tools is restricted by the number of input formats they can accommodate and their level of complexity in terms of learnability [19]. Consequently, selecting SPARQL Anything [27] for implementation was most favourable.

Generating knowledge graphs directly from datasets is not trivial as the process is complex and requires an understanding of knowledge engineering. Knowledge engineering, itself, requires experience and research in areas such as: what ontologies to create or use, what vocabularies to use and what classes to use. Thus, knowledge graph generation from a dataset is not instantaneous and requires extensive research. In the case of smaller knowledge graphs, manual creation without the use of a tool is feasible. Nonetheless, physically building the knowledge graph triple by triple would be time-consuming and is prone to human error, especially if the knowledge graphs and datasets expand in size. This approach is, therefore, not maintainable.

Motivation for organ representation, in particular, revolves around the lack of detail surrounding them. Current musical culture representation is generally insufficient [42], thus specific information regarding organs is necessary to enable the empowerment of such a complex structure. The organ is a fundamental part of music history and a distinguished composer- Mozart lauded the instrument as “the king of instruments”. Furthermore, Johann Sebastian Bach, a renowned composer widely regarded as one of the greatest composers of all time, had a strong preference for the organ [63]. Enrichment of existing representations with vast amounts of detail about organs is a powerful method of honouring such a venerable instrument. The wealth of valuable information in this emerging field of organs will, therefore, enable more effective querying and strengthen its symbolic representation.

1.2 Aims

Integration of various data sources into a single, encapsulated knowledge graph can enable the sharing and understandability of this information by both humans and machines. Knowledge graphs may provide economic advantages by reducing the time and effort required to

analyse vast amounts of data. Consumers may use the knowledge graph for their own needs or follow the process outlined in the report to create their own. Thus, providing an effective and quick procedure for knowledge graph generation was one of the primary aims.

To the extent of our research, creating knowledge graphs for this particular subject (organs) had not yet been fully explored in this level of detail. For instance, knowledge graphs were previously developed for music recommendation systems [51] by other researchers. So knowledge graph generation techniques had been done previously, but the employed methodology is of a different context to our problem. Existing organ knowledge graphs on Wikidata [18], MusicBrainz [31] and DBpedia [17] are sufficient for basic familiarity, but do not go into the same amount of depth as the knowledge graph created in this project. This project, therefore, aims to address this particular knowledge representation gap.

1.3 Scope

The project is part of the Polifonia project, which seeks to create an ecosystem of computational tools and methodologies to spread knowledge about musical history on the internet [42]. Therefore, this report assists the Polifonia project in achieving its objectives and facilitates its progression. This designated project involves a dataset centred around organs so the report includes a breakdown of the various organ components as well as background information regarding technical knowledge.

This report will detail the process of knowledge graph generation using an organ-orientated dataset and ontology [14], both of which are contextualised to fully understand the dataset and to refine the provided ontology. In order to commence, knowledge of the semantic web and its capabilities are required so as to understand the scope of the project. Researching and being able to interpret RDF (a standard model for data interchange on the web [35] used for representing and exchanging metadata), ontologies and knowledge graphs are also required to reach the solution as well as interpret it.

The following chapters provide a comprehensive view of the design and implementation process for generating knowledge graphs. The *Design* chapter complements the implementation phase by using visual tools and careful planning through software development techniques. Subsequently, the *Implementation* chapter meticulously explains the steps required to reach the final solution. A thorough evaluation of the produced knowledge graph is performed with both qualitative and quantitative forms of assessment being carried out. The legal, social, ethical and professional issues surrounding this project are also reviewed to ensure adherence

to the Code of Conduct & Code of Good Practice issued by the British Computer Society [6] as well as the FAIR Principles [62]. This is also intended to assist in identifying any potential issues that may arise during the project.

Finally, the conclusion summarises work completed during the course of this project and includes a discussion on future work and limitations.

2

Background

First, background of this project is outlined and suitable explanations provide context for the entire project. The broad scope of the project is presented before going into project-specific details in the *Context* chapter.

2.1 The Semantic Web

The Semantic Web is an extension of the World Wide Web (WWW) that allows information on it to be machine-readable [24]. The concept was founded by Tim Berners-Lee (founder of the WWW) with the idea that machines can easily interpret information available on the WWW. Tim Berners-Lee initially thought of this idea in 1999 stating “I have a dream for the Web [in which computers] become capable of analysing all the data on the Web – the content, links, and transactions between people and computers. A ‘Semantic Web’” [7]. Ultimately, the semantic web aims to empower the knowledge embedded online so that it can be parsed by machines [24].

An example of these developments can be seen in a data storage project that contains structures interpretable by both humans and machines- Wikidata. Wikidata is a large open knowledge base that is part of the Wikimedia family (including Wikipedia) and is connected to other datasets as well [28]. Data is stored in WikiData using various web data storage methods such as JavaScript Object Notation (JSON), Extensible Markup Language (XML) and Resource Description Framework (RDF). The latter is described in more detail in the next section.

Another example within the project scope is MusicBrainz. MusicBrainz also provides information that is stored using RDF [58]. In this case, MusicBrainz's focus is around music so it may be more relevant and provide more specific data for the project.

2.1.1 Resource Description Framework

Resource Description Framework (RDF) is a method in which data can be stored, similar to how data can be stored in JSON or XML formats. However, the structure of RDF is very different. The general idea of RDF revolves around the concept of linked and connected data so it stores data in triples with relationships between data. Data stored in this format is both machine and human-readable, making the reuse and extension of data semantics possible [47]. Each triple follows this structure:

subject predicate object

Subject:

The subject is the entity that the triple is about or refers to, which is typically a resource. A resource can be a Uniform Resource Identifier (URI- a string that identifies a name or resource on the Internet [56]), a physical thing or an abstract concept.

Predicate:

The predicate conveys the relationship between subject and object (i.e. how they are related).

Object:

The object is an entity that the triple describes in relation to the subject. This can also be blank.

In this project, we will look closer at the Turtle (TTL) format for RDF. A TTL file represents RDF triples with a subject, object and relationship. This format is specifically relevant when representing multiple related RDF triples (RDF Graph) and allows them to be readable by both humans and machines [21].

An example of an RDF statement (in TTL) can be seen below:

```
<http://www.example.com/Alvin>  
<http://www.example.com/vocab#studiesAt>  
<http://www.example.com/KCL>
```

In this example,

- **The subject is:**

“http://www.example.com/Alvin”

- **The predicate is:**

“http://www.example.com/vocab#studiesAt”

- **The object is:**

“http://www.example.com/KCL”

This statement indicates that the entity *“http://www.example.com/Alvin”* studies at the entity *“http://www.example.com/KCL”*, according to the vocabulary defined by *“http://www.example.com/vocab”*. However, writing all the various URI links separately is tedious and can look confusing for human readers so using prefixes as shown below can make it more concise:

```
@prefix ex: <http://www.example.com/>.  
@prefix exVocab: <http://www.example.com/vocab#>.  
  
ex:Alvin exVocab:studiesAt ex:KCL
```

This example uses two abbreviations of URIs and has named the aliases for them to be ‘ex’ and ‘exVocab’ respectively, simplifying the triple.

An English translation of the RDF triple is written below:

“Alvin studies at KCL”.

As mentioned previously, RDFs can also be visualised in graph format with numerous relationships between subjects and objects with the predicate specifying their relationships. In particular, this project’s produced knowledge graph will be in RDF format and represented in a TTL file. RDFs can be used to describe ontologies, which are detailed in the next section.

2.2 Ontology

An ontology is a conceptual framework for modelling domain knowledge [48]. Ontologies make use of RDF triples by having two nodes (the subject and object) and an edge (the predicate). They act as models in which real data can be input, so these ontologies can be reused in different contexts and shared between users. These generalised models are ideal for knowledge graph generation because data from different contexts can be displayed in the same way using the same ontology.

The main components of an ontology are similar to the RDF triples in the previous section but have different names. They involve:

- **Classes:**

A collection of objects or things that are related to each other in some way.

- **Relationship:**

A link between a class and an attribute that describe how they are related.

- **Attributes:**

A characteristic or property that a class may have, which is described by the relationship between them.

In relation to RDF statements, a connection between two nodes in an ontology will have subject and object nodes with a connecting edge specified by the predicate. *Figure 2.1*, is an example of a graphical visualisation in the context of a football team:

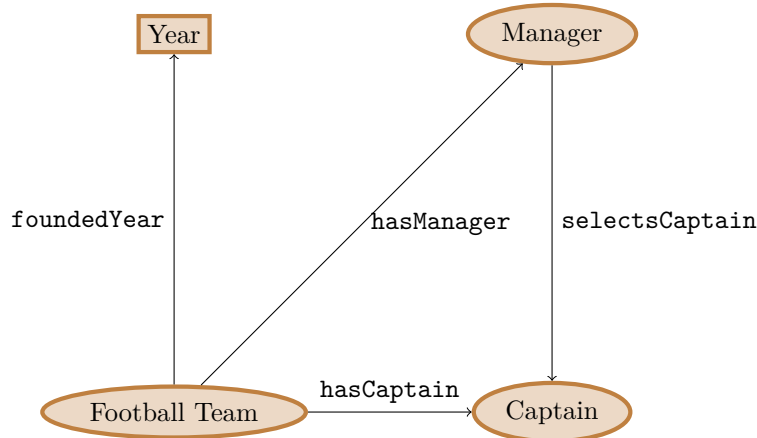


Figure 2.1: Example Arsenal Ontology

Figure 2.1 depicts a Football Team with a Manager and a Captain. The Manager and Captain (of the football team) also have a relationship because the manager selects a specific

captain for the football team. The attribute ‘Year’ only describes the ‘Football Team’ class so is denoted as such.

Ontologies are the framework in which knowledge graphs can be constructed and guide the generation of knowledge graphs.

2.3 Knowledge Graphs

A knowledge graph is a graph (i.e. data graph) that depicts real-world knowledge or data [39]. Similar to ontologies, knowledge graphs are often represented as a network of interconnected nodes and edges but instead, nodes represent real-world entities (such as people or places) and edges represent the real-world relationships between them (such as ‘owns’ or ‘discovered’).

As mentioned, knowledge graphs apply real data from an ontology framework. With a dataset and relevant ontology, we can create specific instances of each ontological relationship.

Knowledge graphs are used in a variety of real-world applications including search engines, recommendation systems and natural language processors. More detail on these examples are below:

- **Search Engines:**

Knowledge graphs are used by search engines such as Google to find appropriate results given a search query. Results are interpreted and relevant information is displayed based on relationships between the URIs and meaning of the search term. The most relevant information with regard to the search query is displayed to the user. The method in which a knowledge graph aids a specific search query is called a “semantic search” and provides more accurate and relevant results. This mirrors how humans tend to think by emulating our natural ability to find associations between data [37].

- **Recommendation Systems:**

Knowledge graphs can be used in such systems because there are often many associations between different entities. Recommendation systems use relationships in the knowledge graph to display the most relevant data to the user.

- **Natural Language Processing:**

In this context, knowledge graphs are used to establish connections between distinct terms linked within text. A system employing Natural Language Processing may want to understand the meaning of a sentence and be able to interpret and input it into a search

query so that relevant results can be found.

An example of a knowledge graph can be in *Figure 2.2*, following the ontology in *Figure 2.1*:

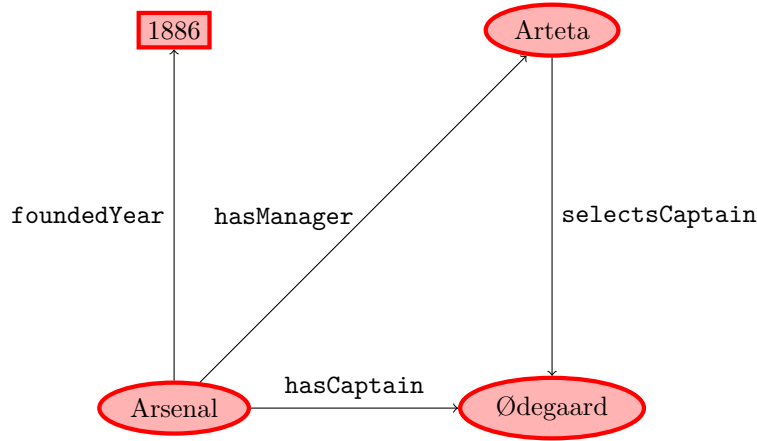


Figure 2.2: Example Arsenal Knowledge Graph

Figure 2.2 describes Arsenal’s men’s football team. Arteta is the manager of Arsenal and Arsenal’s captain is Ødegaard. Arteta (the manager) also selects the captain. The year Arsenal was founded is displayed in the graph as 1886.

The more linked data we add from the dataset, the larger the knowledge graph will grow.

2.4 SPARQL

SPARQL enables us to query RDF data formats and retrieve specific data from large RDF files. SPARQL queries often have the same structure as standard SQL queries. Specific data can be specified and filtered by stating the subject, object and relationship triple in the query’s SELECT and WHERE clauses respectively [26]. An example of a SPARQL Query is shown with an explanation below:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?first_name
       ?surname
WHERE
{
    ?person a foaf:Person.
    ?person foaf:name ?first_name.
    ?person foaf:surname ?surname.
```


}

Example from: [20]

Here, the shortcut ‘foaf’ (friend of a friend) is used, which is often used to describe people, their characteristics, relationships and other information about them [20].

The **SELECT** part of the query extracts all the first and last names of people in the queried TTL file with RDF type *foaf:Person*. The selected people must also have a name and surname specified in the file.

The **WHERE** clause of the query specifies that the queried variable *?person* belongs to the RDF class *foaf:Person*. The *?person* should also possess both a name and a surname represented by the property *foaf:name* and *foaf:surname* respectively.

In conclusion, if the query were to be executed in an appropriate RDF document, the result would display the first and last names of all individuals in the TTL file of type *foaf:Person*. Moreover, the individuals outputted will both have a name and surname property.

3

Context

This chapter will contextualise the dataset and project, providing an understanding of all relevant information included in the organ dataset. Also, the chapter provides a brief introduction to the tool being used: SPARQL Anything as well as a justification for using this tool over others.

3.1 Organs

The dataset provided revolves around the musical instrument: Organs. Mozart once described organs as:

“The king of instruments.”

An organ can come in various different sizes ranging from a small upright piano to a large structure consisting of many sub-systems. This chapter will provide a description of the instrument’s components.

3.1.1 A Brief History

The inventor of the organ is accredited by many sources to an engineer in Alexandria during the third century BC called Ctesibius. In the first century AD, organs used water and hydraulic engineering to generate sound [9].

During the 14th century, organs evolved from the use of just pedals and saw the development of a keyboard so larger ranges of sounds could be produced. This was accommodated by the creation of new pipes so having access to more of these unique sounds was beneficial. These organs laid the building blocks for the organs we recognise today [59].

Due to technological advancements, creation of organs became a lot easier allowing them to become more complex. Recent organ builders have had access to a plethora of materials and tools required to create these large structures with complex systems. Being able to study and explore different types of organs with ease, also aided the evolution of such structures [1].

3.1.2 How Organs Work:

Operation of the organ involves generating and directing airflow to the requested pipe causing them to vibrate and produce sound. The organist uses the keyboards and pedals to control the flow of air to pipes and draws stops to combine different sets of pipes to produce unique, blended sounds [12].

Generally, the organ system works as follows:

1. Note is played on the keyboard.
2. Pressurised air is passed through the organ system.
3. When a particular stop is pulled, an internal slider is moved to allow air to pass through pipes.
4. Sound for that particular key and stop is produced.

Relevant components of the organ and organ dataset are described below.

Pipe Types:

The particular type of pipe in an organ can significantly affect the produced sound and tone. Below, are the two main types of pipes used in an organ.

- Reed Pipes: Passes air through a reed (similar to a clarinet).
- Flue Pipes: Forces air through a pipe (similar to a recorder).

[12]

Rank:

A rank is a row of pipes (controlled by a single stop) that produces the same musical sound but does so at different pitches.

For example, a rank of spire flue pipes all produce the same wind instrument sound, but each key pressed will produce a different flue pitch [12].

Stops:

Drawing/selecting a stop gives the organist access to that stop's set of pipes available (for a particular key on the keyboard). Types of stops include:

- Draw Stops
- Tabs

Multiple stops can be drawn at once in the same division and multiple pipes will produce sound (pressurised air will be passed through them) from the press of one key [12].

Divisions:

An organ division is a set of pipes found within an organ controlled by the keyboard or pedals. Stops of an organ are arranged into divisions, which are usually given unique names. For example, some divisions include:

- Swell (smaller pipes).
- Great (larger pipes).
- Pedal.
- Choir.
- Positiv.

All of these divisions usually accompany whatever is deemed appropriate for the given context. For example, when accompanying a choir, using the choir division would be suitable so as to not drown out the voices of the choir [12].

Coupler:

A coupler enables the keyboard of one division to play the stops of another division. (i.e. the combining of divisions). For example, the "Great" division is usually played on the manual (keyboard) of the organ, but pulling the "Great to Pedal" coupler can allow for this division to be played on the pedals [12].

Wind System:

Wind systems are responsible for:

- Producing

- Storing
- Delivering

air within the organ system.

The organ wind system is a subsystem of the organ, which is responsible for delivering air to the organ’s different pipes. Wind is generated within the system and then directed through a series of channels to the organ’s pipes. Upon delivery of air to each individual pipe, a sound is made based on the key or pedal pressed [12].

3.2 Polifonia Project

The Polifonia project is a European project funded by the EU Horizon 2020 Programme that will manifest the connections between musical heritage from the 16th century to present day [40]. The project involves many different types of experts from Musicologists to Computer Scientists attempting to find links between different parts of musical cultural heritage.

The main aim of this project is to combine computational tools and methods to access and manipulate musical cultural heritage on the WWW. Another aim is to use these computational methods in a different context (independent of the Computer Science field) and test whether such techniques can be adequately applied elsewhere. Hence, the involvement of experts from all over Europe to facilitate the project’s progression [42].

The objective of this report is to create a knowledge graph in the context of organs as part of the Polifonia project. The provided dataset, which was derived from an organ encyclopedia, was organised and computationally formatted by a musicologist as part of this project. The provided organ ontology [14] was created by an ontology engineer but is vast and requires adjustment to match the scope of this report. Due to the project being European-based, data stored in the dataset is in Dutch and the knowledge graph created will also produce Dutch text or terms. More detail on the provided dataset and ontology are provided in the following subsections.

3.2.1 Organ Dataset

The organ dataset provided describes various details regarding a given organ. The dataset, itself, is split into many different files and groups specific details about an organ into one JSON file. Each JSON file contains information about many different organs, which can be uniquely identified by their id. The structure of each JSON file follows the same format: the organ

and the corresponding details concerning that organ. An example of two files “base.json” and “history_base.json” can be seen below.

Listing 3.1: base.json extract

```
{
  "Part01_001MIDDE": {
    "building": "Koorkerk",
    "monumentnumber": "28671",
    "name": "Middelburg, Koorkerk",
    "organnumber": "971",
    "place": "Middelburg",
    "whichorgan": "",
    "year": "1479"
  },
  "Part01_002UTREJ": {
    "building": "Jacobikerk",
    "monumentnumber": "36148",
    "name": "Utrecht, Jacobikerk",
    "organnumber": "1514",
    "place": "Utrecht",
    "whichorgan": "",
    "year": "1509"
  },
  ... # Other organs
}
```

Listing 3.2: history_base.json extract

```
{
  "Part01_001MIDDE": {
    "builder": "Peter Gerritsz",
    "originallocation": "Utrecht, Nicolaikerk",
    "year": "1479"
  },
  "Part01_002UTREJ": {
    "builder": "Gerrit Petersz",
    "originallocation": "",
    "year": "1509"
  },
  ... # Other organs
}
```

The data is split into multiple different files containing the unique organ identifier with its relevant grouped data. The dataset contains data in Dutch since the Polifonia Project is a European project.

3.2.2 Organ Ontology

The provided organ ontology [14] contains all the relevant information relating to an organ, ranging from its location to the parts within it.

The classes included in the ontology for the organ system are:

- Organ Console
- Organ Wind System
- Organ Case
- Organ Action
- Organ Division

But the ontology also includes classes about a particular organ’s owner and offers background on a given organ. For example:

- Description
- Agent and the role of this agent (owner, organist, builder etc.)

The ontology is extremely vast and may contain nodes or relationships that do not appear within the dataset. Therefore, refinement and adjustment of the ontology, to fit the needs of the provided dataset, is required. Carefully selecting parts of the ontology to act as the framework for the resulting knowledge graph is vital to ensure all relevant data in the dataset is accurately represented.

3.3 SPARQL Anything

The tool selected for knowledge graph generation is SPARQL Anything [27]. The main purpose of this tool was to assist Semantic Web practitioners in dealing with heterogeneous data sources [4]. The façade design pattern was used to wrap non-RDF resources such as JSON and make them queryable as if they were RDF [19]. In general, this design pattern improves the usability of a complex system by providing a simpler and more intuitive interface to it. In this case, SPARQL Anything masks its complex components by allowing the user to continue to use SPARQL 1.1 query language, so familiar functions such as BIND() or CONCAT() can be applied, simplifying the knowledge graph generation procedure. Therefore, the time required to learn how to use such a tool would be short as there is an abundance of resources available related to learning SPARQL 1.1 query language, in contrast to other tools which may require learning new syntax or a new language.

SPARQL Anything includes several useful attributes. For example, the SERVICE operator allows access to SPARQL Anything’s features within a standard SPARQL query by overriding SPARQL 1.1’s SERVICE operator with a virtual endpoint [4]. Additionally, the CONSTRUCT operator can be used to generate knowledge graphs based on a specified ontology.

3.4 Other Tools

This section will detail alternate methods of knowledge graphs generation describing the approach as well as providing a comparison with SPARQL Anything. These existing methods were considered as an approach to implementation but eventually decided against them.

3.4.1 SPARQL-Generate and RML

Other solutions similar to SPARQL Anything’s approach and implementation include SPARQL-Generate and RML.

SPARQL-Generate

SPARQL-Generate is an extension of SPARQL 1.1 and generates RDF from an RDF formatted dataset or documents in arbitrary formats. Expansion from SPARQL 1.1’s syntax enables the use of a new clause called GENERATE, which allows for RDF generation. [45]

Compared with SPARQL Anything, SPARQL-Generate does have some minor disadvantages. Primarily, SPARQL-Generate does not support metadata or embedded data formats, so SPARQL Anything is the only tool covering such structures [19], allowing for more flexibility during implementation. Although SPARQL-Generate is an extension of SPARQL 1.1 and has lower learning demands, it still requires research of a newly created extension. Key terms such as GENERATE will have to be explored whereas SPARQL Anything, because it uses Facade-X, is solely reliant on SPARQL 1.1 query language knowledge [19]. As a result, adaptability is also improved since functions in SPARQL 1.1’s language should suffice. SPARQL-Generate, in comparison, also has an ITERATOR clause allowing for iterator functions that enable the duplication of a BIND, for example [45]. However, an understanding of SPARQL-Generate’s extensions can not be expected [19]. For those looking to continue work on this project, having a typical Semantic Web engineering workflow with SPARQL Anything will be beneficial as we can assume preexisting experience with SPARQL 1.1.

In terms of complexity, SPARQL Anything performs better in experiments using the number

of distinct items or variables as a measurement for complexity. Tests run based on input size have shown similar performance times with both SPARQL-Generate and SPARQL Anything so there are no differences in that regard[19]. Nonetheless, this was important when considering possible expansion of the knowledge graph and when assessing the speed of query execution.

RML

RML is a mapping language based on R2RML: W3C’s standard for mapping relational databases into RDF [25]. Similar to SPARQL Anything and SPARQL-Generate, RML allows for the mapping of heterogeneous data sources.

In contrast to SPARQL Anything, RML does not support metadata, embedded data or plain text formats [19], thus constraining its versatility. In RML, the steps required to transform data formats into RDF are tedious as it requires data to be transformed before being uploaded into a SPARQL triple store, whereas SPARQL Anything allows direct resource querying [19].

Due to RML’s extension of the R2RML vocabulary, a new mapping language will have to be learnt in order to use this tool [19], making the studying overhead far greater than SPARQL Anything’s. Those looking to extend the project may require more time to understand RML than SPARQL Anything due to (assumed) prior knowledge of SPARQL 1.1 query language. Additional methods, on top of R2RML, such as *rml:iterator* and *rml:referenceFormulation* [25], facilitate the transformation process to RDF. Nevertheless, SPARQL Anything’s exclusive reliance on SPARQL 1.1 results in improved adaptability [19].

3.4.2 Large Language Models

A large language model (LLM) is a type of artificial intelligence (AI) that can carry out a variety of natural language processing tasks [54]. These are trained on millions upon millions of input data, ensuring high levels of accuracy and precision, so creation of tools such as artificial chatbots could be as accurate and broad as possible. Knowledge graph generation using LLMs has not yet been fully explored in research. A specific example of an LLM chatbot that may offer a viable solution to knowledge graph generation, due to its extensive knowledge and expertise, is a recent tool ChatGPT.

ChatGPT is a machine-learning-based tool developed by OpenAI and is an interactive chatbot. Answers provided by ChatGPT are conversational so are easy to understand and is suggested to be the best chatbot to date [43]. OpenAi released a prototype chatbot on 30th November 2022 and is currently free to use. Uses of ChatGPT include:

- Solving programming questions.
- Fixing bugs in code.

When asked to create a knowledge graph regarding organs, ChatGPT produced:

Nodes: Keyboard, Pedals, Pipes, Blower, Stops

Edges:

- The keyboard is connected to the pipes, indicating that pressing keys on the keyboard causes the corresponding pipes to produce sound.
- The pedals are connected to the pipes, indicating that pressing the pedals can also cause certain pipes to produce sound.
- The blower is connected to the pipes, indicating that the blower supplies air to the pipes, allowing them to produce sound.
- The stops are connected to the pipes, indicating that the stops control which pipes are active and can produce sound.

This response represents a small knowledge graph mainly comprised of organ parts and how it produces sound. The output of the knowledge graph is in plain text, which requires manual reading and interpretation, rendering it unusable for machines.

This approach is a very straightforward technique for generating an organ-related knowledge graph and makes use of the strengths of ChatGPT. The knowledge graph, although in prose, is very useful and easy to interpret due to ChatGPT's ability to communicate conversationally. The information provided is based on the chatbot's general understanding of the organ and is consistent with the instrument's details. The nodes and edges stated in the knowledge graph are also consistent with the elements of an organ.

However, this approach generates a small knowledge graph and has to be interpreted correctly in prose. The validity of the produced knowledge graph is limited as it only includes general topics related to organs, without any real-world data. As a result, the generated graph is more akin to an ontology than a knowledge graph. The limitations of ChatGPT include that it is restricted to its input domain [29], so real-world data can not be found and passed into this ontology to create an accurate knowledge graph.

In conclusion, the knowledge graph produced by ChatGPT is closer to an organ ontology. The limitations surrounding ChatGPT do confine it to providing a generic solution. SPARQL

Anything offers a more reliable approach to knowledge graph generation because real-world data can be displayed on the knowledge graph. However, if ChatGPT can evolve to find a way to gather real-world data online using the Semantic Web, for instance, the knowledge graph created would be a viable solution.

4

Literature Review

In this chapter, resources (i.e. articles, websites, videos or books) being studied are discussed. The literature reviewed encompassed a broad range of topics to facilitate understanding of the context and ensures successful completion of the project. All references are mentioned for readers to carry out more research if they wish.

4.1 Background and Context Information Sources

Research surrounding the semantic web and its technologies was required in order to provide a broad context for the project and its implementation. Articles [24] and [8] offered gentle introductions to the topic although the former was not directly relevant to the project scope. Limited resources specifically relating to the project's scope meant that resources of different contexts had to be explored. The concept of the semantic web originated from the founder of the WWW: Tim Berners-Lee, so understanding his perspective through a book [7] co-authored by him could provide valuable insights. Other articles [10] were explored to understand the

current status of the semantic web and identify areas for further research.

Targeting articles that offered an introduction to the topic were favoured to help build a basic understanding. However, they were limited by their introductory nature and lack of in-depth detail. Using articles such as [48], [39], [56], [35], [3], [57] and [47] helped the understanding of relevant background information. All articles introduced their topics sufficiently and for further research into the practical uses of knowledge graphs, [37] and [51] were used. Fully grasping [51], in particular, was vital due to its relevance in the field of music. Websites from W3C [21], [36] and [60] also facilitated research by filling gaps in understanding. Being a reputable international standard organisation and involving a wide variety of members displays the source’s credibility. However, from a lay reader’s perspective, these websites could be challenging to understand due to the use of technical language.

Research regarding organs was necessary to gain an extensive understanding of the provided dataset. Starting with the history of organs required the use of [9], [1] and [59]. This involved a book as well as some online material to help provide a short historical summary of organs and their evolution. Extraction of relevant information from each source was necessary to provide a brief historical context. Specific sources regarding the history of organs were scarce and others found, [2] and [50], relied heavily on previous understanding of music history so the selected sources of information were best from a lay reader’s perspective. Different mediums were used particularly to understand the different parts of the organ such as videos [12] and [55]. This aided visualisation of the components in the dataset from experts in the field. Alternate introductory sources were limited so using videos was sufficient.

Details regarding the Polifonia project’s purpose and contextual information were acquired from its specification [42] and the official website [40]. These websites provided nuanced details regarding the Polifonia project, which were the only few resources available regarding the project. Bias, however, may be introduced as all resources were written by participants of this project.

4.2 Knowledge Graph Generation Techniques

Techniques for knowledge graph generation were heavily researched to identify the approach for this project. Tools such as SPARQL-Generate [30] were assessed using articles [44] and [45], but found the overhead required to learn new syntax was too high. RML [53] was also considered using [25] but was more complex than existing solutions. Both tools were adequate knowledge graph generation implements with sufficient guidance on how to proceed in articles

and documentation.

Research on Language Models for knowledge graph generation discovered insufficient resources surrounding knowledge graph generation, suggesting its limited use. Large Language Models (LLMs) [54] could, however, provide a solution to knowledge graph generation given their recent rise in complexity and popularity. Nevertheless, articles surrounding knowledge graph generation using LLMs were also limited, possibly due to its novelty and general lack of awareness around the topic. The tool ChatGPT [29], specifically, was explored for knowledge graph generation with introductory resources [43] being studied for basic knowledge of the tool and investigation of its capabilities.

SPARQL Anything [27], however, was the chosen direction for knowledge graph generation due to its simplicity and similarity to SPARQL 1.1 query language. Articles [19] and [4] provided stark rationale for selecting SPARQL Anything over the aforementioned tools. This allowed use of SPARQL 1.1 query language resources such as a book [26] with an abundance of information regarding the language. Furthermore, drawbacks of RML and SPARQL-Generate included the lack of specific documentation surrounding the tools, which could pose a problem during debugging. Specific examples of knowledge graph generation with SPARQL Anything could also be seen in the documentation [27] and various other articles [19], [4] and [52]. In terms of execution time and input size, all tools are relatively similar so any of them could have been used without major scalability challenges as discussed in [19].

Existing organ knowledge graphs were also researched. Resources surrounding this topic were sparse and the only relevant knowledge graphs were found on linked open dataset sites such as Wikidata [18], DBpedia [17] and MusicBrainz [31]. These resources provided insight into the current scope of available organ knowledge graphs, revealing basic material that needed to be fully explored. Using all available resources in our knowledge graph would be important in creating an expansive knowledge graph.

4.3 Knowledge Graph Evaluation Techniques

Resources for critiquing and evaluating the produced knowledge graph included a book [38] and a paper [13], which provided general information on how to evaluate the quality of any knowledge graph. Using two different mediums of evaluation guidance provided ample information on assessing knowledge graph quality. In particular, [13], provided reasoning and follow-up articles for quality assessors. Alternate in-depth methods of evaluation were explored in [33]. However, the aims of this article did not align with [38] and [13], which mainly focused

on the qualitative assessment of the produced knowledge graph.

4.4 Additional Complementary Literature

Any additional literature supported the report by offering more detail on said topics. For example, during implementation, JSON path was used so providing the consulted documentation [61] would be helpful for those looking to extend the project or seek more detail.

5

Requirements

In this chapter, the requirements for this project's success will be listed.

5.1 Preliminaries

The required tools for commencing implementation:

1. Installation of SPARQL Anything from [27] by downloading .jar file.
2. Installation of Java Virtual Machine on the computer.
3. Access to a Command Line Interface.
4. An Integrated Development Environment.

5.2 Software Requirements

These requirements are centred towards what needs to be implemented and produced upon

project completion. Assessing and validating the solution is also necessary to ensure correctness.

1. Ontology must be correctly configured and input into the query's CONSTRUCT clause.
2. Knowledge graph created must correctly reflect the organ dataset and follow ontology structure.
3. Dataset must be refined to ensure correct data is being input into the knowledge graph.
4. Knowledge graph must expand on the current dataset using external links.
5. Knowledge graph must be evaluated to prove correctness and validity.
6. Scalability of knowledge graph generation must be assessed.

5.3 User Requirements

These requirements specify what an external user should be able to do when using the solution.

1. User must be able to execute written query.
2. User must be able to select an organ to view information on.
3. User must be able to see the knowledge graph following query execution.
4. User must be able to see relevant data in the knowledge graph.
5. User must be able to view other relevant external data in the knowledge graph.

6

Specification

This chapter will go into more detail regarding each of the requirements specified in the previous chapter. More specifically, this chapter will provide an explanation of how each requirement can be achieved.

6.1 Software Requirement Specification

Requirement No.	Requirement	Specification
--------------------	-------------	---------------

1	Ontology must be correctly configured and input into the query's CONSTRUCT clause.	Organ ontology created will match data from the given dataset and only relevant segments of the provided ontology will be included. To ensure the generation of a valid knowledge graph, it is important to extract relevant parts of the ontology and define the correct types. This guarantees that all classes are properly converted and ensures all relationships are accounted for with no redundant details.
2	The knowledge graph created must correctly reflect the organ dataset and follow ontology structure.	Identify data from the dataset in the knowledge graph and ensure relationships are consistent. Compare ontology with produced knowledge graph and ensure the framework is being followed. Override the SERVICE keyword in the query to use services of SPARQL Anything.
3	The dataset must be refined to ensure correct data is being input into the knowledge graph.	Query correctly navigates through the dataset to find the right data to add to the knowledge graph. This can be done through string manipulation in the query to generate the correct path, thus obtaining the right data. In the WHERE clause of the query, ensure data being input into the knowledge graph is understandable and concise (if not already).

4	The knowledge graph must expand on the current dataset using external links.	Search for external links to expand the knowledge graph beyond the dataset and present additional relevant information. Use web resources such as WikiData, MusicBrainz and DBpedia to find potential expansion points of the knowledge graph derived from the organ dataset. Use multiple SERVICE calls (within the WHERE clause) to obtain external data and add it to the current knowledge graph. String manipulation and string-to-link conversion (IRI) may be necessary to add external links.
5	Knowledge graphs must be evaluated to prove correctness and validity.	Validate correctness of the knowledge graph produced. Look into accuracy, coverage and coherency of the produced knowledge graph and analyse the different aspects of those areas in more detail. [38] and [13], identified in the <i>Literature Review</i> Chapter, will be used to help during evaluation.
6	Scalability of knowledge graph generation must be assessed.	Assess speed of the query for knowledge graph generation to ensure expansion of the knowledge graph is possible in the future without encountering severe time constraints. This will measure scalability based on many different factors (more detail in the <i>Evaluation</i> chapter).

Table 6.1: Software Requirement Specification Table

6.2 User Requirement Specification

Requirement No.	Requirement	Specification
1	User must be able to execute written query.	Upon downloading the GitHub repository and the SPARQL Anything executable, a user should be able to execute the query using SPARQL Anything and have it produce a knowledge graph.
2	User must be able to select an organ to view information on.	When writing the command on the command line to execute the query, the user can be able to input an organ that they want to view specific information on by passing it as a parameter. The resulting knowledge graph is then specific to the requested organ.
3	User must be able to see the knowledge graph following query execution.	Upon executing the query, a knowledge graph should be visible in the form of a TTL file. The user can also see the knowledge graph on command line if they wish.
4	User must be able to see relevant data in the knowledge graph.	The knowledge graph must display the correct data and relationships between them. When viewing the final solution, the user can compare data from the dataset and knowledge graph. Organ-specific data can be seen for the requested organ.
5	User must be able to view other relevant external data in the knowledge graph.	The user should be able to view external links to data in the knowledge graph and acknowledge the relationship between data from the dataset and external data. External links unique to the requested organ can also be observed.

Table 6.2: User Requirement Specification Table

7

Design

This design chapter provides a high-level overview of the software implementation. With the aid of visual tools such as diagrams, a plan for implementation will be outlined.

7.1 UML Sequence Diagram

A high-level overview from user input to output is important to illustrate so it may be followed in the latter sections of this chapter. Providing a visual representation also makes it easier to understand how all relevant components of the project interact with each other. Most SPARQL Anything calls follow the same format as depicted in *Figure 4* of [4] and is strictly followed in *Figure 7.1*. A contextualised UML Sequence diagram is shown in *Figure 7.1*:

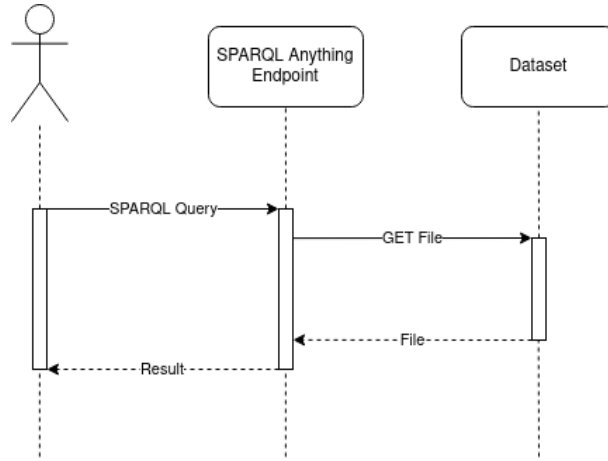


Figure 7.1: UML Sequence Diagram

7.2 Ontology Structure

In this section, a plan for the ontology's general structure will be outlined based on: the dataset, provided ontology and any external websites. Relevant relationships to be used in the real ontology will also be noted.

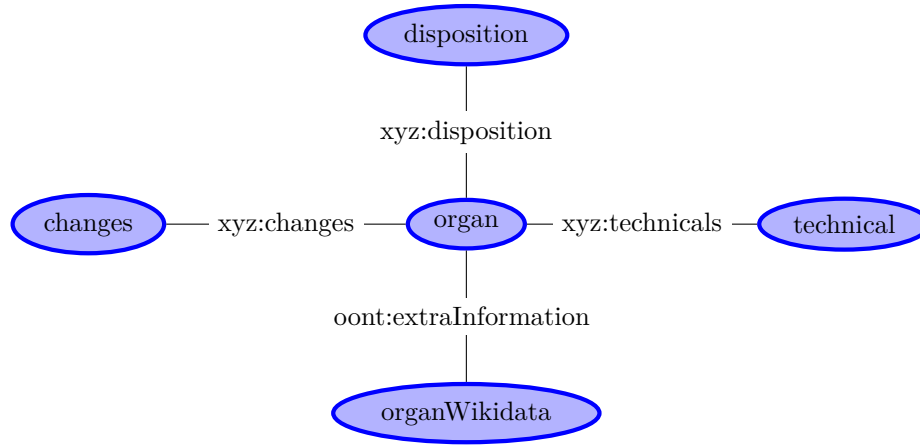


Figure 7.2: Core Organ Ontology

Figure 7.2 illustrates the main components to be expanded upon in the ontology based on data in the dataset. More details of the main nodes are detailed below:

- **organ:**
The primary organ component that will connect with other many other nodes.
- **changes:**
This component specifies the organ adjustment details and maintenance changes for an organ.

- **disposition:**

This component refers to the organ’s current components and details relating to them.

- **technicals:**

This component is responsible for the specific musical details of the organ.

- **organWikidata:**

This provides an expansion point from Wikidata for the existing knowledge graph.

The section of the ontology specifically for external Wikidata links can be structured in the same format as displayed on its website. Using *Figure 7.2*, extending the *organWikidata* node is most plausible. For instance, the organ page on Wikidata [18] contains organ-related triples. An extract from [18] is shown below:

```
organ subclassOf keyboardInstrument
organ subclassOf buildingComponent
organ studiedBy organology
...
```

In this case, ‘organ’ can be replaced with our *organWikidata* node and the corresponding relationships and objects will be added to the ontology. All the extra nodes and edges can be added as external links to the knowledge graph using their unique Wikidata URI.

7.3 Query Flow Diagram

A flow diagram is a graphical representation of the sequence of steps or actions that need to be taken to complete a process [5].

In this section, the logic required to build the SPARQL Anything Query will be illustrated in the form of a flow diagram. This will provide the structure required to build the query and produce a knowledge graph.

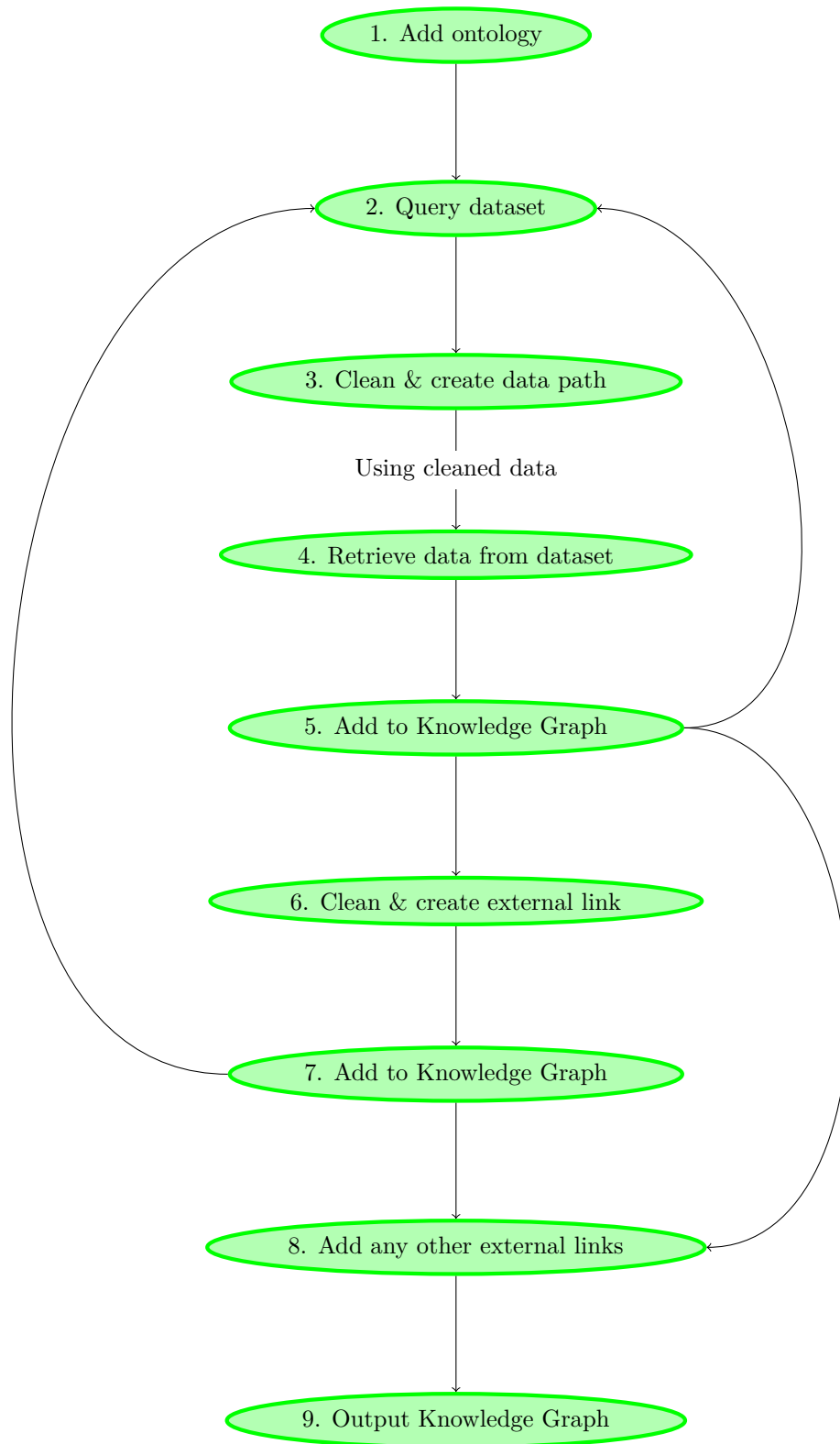


Figure 7.3: Query Flow Diagram

The nodes in *Figure 7.3* represent necessary actions and one of the edges is explicitly stated

for clarity. Below, an explanation for each node is provided:

1. **‘Add ontology’ node**

Adds ontology to the CONSTRUCT clause of the query.

2. **‘Query dataset’ node**

Uses SPARQL Anything to specify the file containing data of interest.

3. **‘Clean & create data path’ node**

Uses string manipulation to create a valid path to desired data in the dataset.

4. **‘Retrieve data from dataset’ node**

Uses created path above to retrieve correct data.

5. **‘Add to knowledge graph’ node**

Adds retrieved data to the knowledge graph. Then, it either loops to step 2 using a different JSON file, continues to step 6 by adding custom external links or skips to step 8 to add other independent external links. The latter pertains to the last SPARQL Anything call in the query.

6. **‘Create & clean external link’ node**

Creates an external link using retrieved data within the same SPARQL Anything call.

7. **‘Add to knowledge graph’ node**

Adds the custom external link to the knowledge graph. Then, it either loops to step 2 using a different JSON file or continues to step 8 to add other independent external links. The latter pertains to the last SPARQL Anything call in the query.

8. **‘Add any other external links’ node**

Adds any other links that do not require data from the dataset such as Wikidata and MusicBrainz.

9. **‘Output knowledge graph’ node**

Produces the resulting knowledge graph.

7.4 Query Skeleton Structure

In this section, the flow diagram illustrated in *section 7.3* will be used to structure the query and aid in generating a knowledge graph. This can be observed in the comments next

to each line, which correspond to a step in the flow diagram created. It also outlines the basic structure that will be used to create the query.

```
PREFIX ... # Add relevant links

CONSTRUCT {
    ... # Add organ ontology
}
WHERE
{
    SERVICE <x-sparql-anything:file:./___.json> # Query
    dataset
    {
        ... # Clean and create data path
        ... # Retrieve data from dataset
        ... # Add to Knowledge Graph
        ... # Clean and create external link (if applicable)
        ... # Add to Knowledge Graph
    }
    SERVICE <x-sparql-anything:file:./___'.json> # Query
    dataset
    {
        ... # Clean and create data path
        ... # Retrieve data from dataset
        ... # Add to Knowledge Graph
        ... # Clean and create external link (if applicable)
        ... # Add to Knowledge Graph
    }
    .
    .
    .
    ... # Add any other external links
    # Output Knowledge Graph
}
}
```

With regard to *Figure 7.3*, the query skeleton structure abides by the plan set out. Below, a small description mapping each node in *Figure 7.3* will be provided.

- **Add relevant links**

Will add any PREFIXES necessary during implementation.

- **Add organ ontology**

Refers to node 1, where the ontology is adjusted if necessary and placed into the query's

CONSTRUCT clause.

- **Query Dataset**

Refers to node 2 by selecting the file in the dataset where desired data resides by overriding SPARQL's SERVICE operator.

- **Clean & create data path**

Refers to node 3 and uses SPARQL 1.1 functions such as CONCAT and BIND to create the appropriate JSON path.

- **Retrieve data from dataset**

Refers to node 4 whereby data is located using fx:properties and the recently created JSON path.

- **Add to knowledge graph**

Refers to node 5 and adds located data to the knowledge graph using RDF functioning. Possible directions for continuation have been detailed in *Section 7.3*.

- *Optional:* **Clean & create external link**

Refers to node 6 and uses SPARQL 1.1 functions such as CONCAT, IRI, REPLACE and BIND to create the external link.

- *Optional:* **Add to knowledge graph**

Refers to node 7 where located data is added to the knowledge graph by assigning it to the corresponding variable in the ontology. Possible directions for continuation have been detailed in *Section 7.3*.

- **Add any other external links**

Refers to node 8 which ceases SPARQL Anything calls and adds any relevant external links using SPARQL's BIND function. Explicitly stating the links and adding them to the knowledge graph may be a viable method.

- **Output Knowledge Graph**

Refers to node 9 where the final knowledge graph is produced.

8

Implementation

This chapter will detail the implementation phase of the project and how the solution was reached using the *Design* and *Specification* chapters.

8.1 Command Line Command

In order to begin implementation, it was necessary to become familiar with command line commands beforehand. Since the executable .jar file enables anyone to use SPARQL Anything, starting the command with execution of the .jar file is needed. The version of SPARQL Anything used during implementation will be the latest at the time: v0.8.1. For example:

```
java -jar sparql-anything-0.8.1.jar
```

Then, expand it specifying the file path and file that contains the query. For example:

```
-q filepath/filename.sparql
```

Because the query involves generating a knowledge graph based on what organ the user wants to view, passing a parameter via the command is vital. This can be done as follows:

```
--values variable=variablevalue
```

The assigned variable can be used in the SPARQL query by specifying the variable name and concatenating a question mark and underscore at the front. So passing the parameter ‘variable’, following the previous example, can be accessed through:

```
?_variable
```

If necessary, the resulting knowledge graph can be output into a file given a location. For example:

```
--output filepath/outputfile.ttl
```

Altogether, the command will look like this:

```
java -jar sparql-anything-0.8.1.jar  
-q filepath/filename.sparql  
--values variable=variablevalue  
--output filepath/outputfile.ttl
```

In English, this command states:

“Using the executable `sparql-anything-0.8.1.jar`, run the query in *filename.sparql* from the *filepath* folder and pass in *variable* into the query with value *variablevalue*. Finally, output the knowledge graph in *outputfile.ttl* from the same file location”.

8.2 Prefixes

The prefixes in the ontology are mainly used in the relationship part of the triples. Below are the PREFIX that will be used:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>  
PREFIX fx: <http://sparql.xyz/facade-x/ns/>  
PREFIX xyz: <http://sparql.xyz/facade-x/data/>  
PREFIX oont: <http://w3id.org/polifonia/ontology/organs/>  
PREFIX wd: <https://www.wikidata.org/wiki/>
```

8.3 Ontology Reconstruction

As mentioned in the *Design* chapter, scope of the provided ontology was too broad for the envisioned knowledge graph. Therefore, refinement of the provided ontology and exploration with external links was necessary to produce a relevant ontology. As mentioned in the *Context* chapter, the ontology serves as the framework for the knowledge graph and will be placed in the CONSTRUCT segment of the query. Knowledge of ontology engineering was not required as this was merely a reorganisation and refinement of the provided ontology. In the following subsections, the process of creating the ontology using the dataset and external data will be detailed.

8.3.1 Dataset Ontology

Identifying the main points of expansion was accomplished in *Figure 7.2*. In this subsection, the provided ontology and *Context* chapter will be used to identify data relevant to these groups in *Figure 7.2*.

Organ

This node represents the requested organ and acts as the main node in which all data would branch from.

Upon selection of relevant information from the dataset's JSON files, they were appended to the organ node. The provided ontology was also used to correctly identify relationships to the organ node. Relevant nodes and relationships to the organ are listed below:

- *technicals* - xyz:technicals
- *disposition* - xyz:disposition
- *change* - xyz:change
- *organWikidata* - oont:extraInformation
- *builder* - oont:builder
- *originalLocation* - oont:consolelocation
- *dateOfBirth* - oont:dateOfBirth
- *building* - oont:monument
- *monumentNumber* - oont:monumentNumber
- *organName* - oont:name
- *organNumber* - oont:organNumber

- state - oont:state
- particularity - oont:particularities
- history - oont:history
- creator - oont:creator
- moreInformation - oont:moreInformation

These nodes are directly relevant to the organ as illustrated in the provided organ ontology. Nodes such as *originalLocation*, *dateOfbirth* and *state* were added as they were part of the dataset and provided extra relevant detail. Because *technicals*, *disposition*, *change* and *organ-Wikidata* were identified as nodes for further expansion in *Figure 7.2*, they were also connected to the main *organ* node. Relationships used to describe the association between the organ and its objects were provided and usually had intuitive names based on the object.

Technicals

This branch of technical nodes regarding an organ was mainly derived from one JSON file and a segment of the provided ontology that referred to its ‘parthood’. All relationships branching from this ‘technicals’ node were provided. Relevant nodes and their identified relationships are below:

- systemPlayingAids - oont:systemPlayingAids
- pitch - oont:pitch
- range1 - oont:keyboardRange
- range2 - oont:pedalRange
- temperature - oont:temperature
- windPressure - oont:windPressure
- windSystem - oont:windSystem

range1 and *range2* corresponded to ranges of the keyboard and pedal respectively so their relationship noted it as such.

Disposition

This branch of nodes referred to current qualities of the selected organ. The provided ontology covered this segment but consisted of nodes that did not appear in the dataset so needed to be refined. Relationships were also provided. The readjusted ontology would include:

- divisionName - xyz:divisionName
- partition - oont:partition
- specification - oont:AdditionalSpecification

Change

This branch stated changes made to the organ during its lifetime. The provided ontology did not mention changes of a given organ so it was added for more data representation. Relationships used were provided. The new sections added to the ontology were:

- dateChange - oont:date
- description - oont:AdditionalSpecification
- maintainer - oont:Builder

8.3.2 External Ontology

The method in which external data was added to the knowledge graph involved following the flow in *Figure 7.3*.

External Data Sources

As mentioned in the *Specification* chapter in *Table 6.1*, Wikidata and MusicBrainz can be used to expand the existing knowledge graph. Following *Figure 7.2*, a node branching out of the main *organ* node was made to represent Wikidata’s organ page [18]. The relationships used were the same as those on Wikidata. Data branching off this *organWikidata* node and their corresponding relationship are:

- keyboardInstrument - wd:Property:P279
- buildingComponent - wd:Property:P279
- organology - wd:Property:P2579
- westernClassicalMusic - wd:Property:P366
- musicTradition - wd:Property:P366
- organExpert - wd:Property:P3095
- organist - wd:Property:P1535
- catholicEncyclopedia - wd:Property:P1343
- metropolitanMuseumOfArtTaggingVocabulary - wd:Property:P1343

- dbpedia - wd:Property:P1709
- organCase - wd:Property:P527
- organPipe - wd:Property:P527
- musicalKeyboard - wd:Property:P527
- pedalKeyboard - wd:Property:P527
- organStop - wd:Property:P527
- organConsole - wd:Property:P527
- swellBox - wd:Property:P527
- pipeOrgan - wd:Property:P1889

External data from MusicBrainz’s organ page [31] was not as vast but extended the knowledge graph in a different manner than Wikidata with overlapping data being connected. Expanding from the main organ node using relationships defined on MusicBrainz, provided more context with relationships mirroring that of MusicBrainz as well. This can be seen below:

- barrelOrgan - rdfs:subclassOf
- electricOrgan - rdfs:subclassOf
- pipeOrgan - rdfs:subclassOf
 - pipeOrganInfo - oont:extraInformation
- reedOrgan - rdfs:subclassOf
 - reedOrganImage - oont:locationImage
- windInstrument - rdf:type
- organWikidata - oont:extraInformation

Extra expansions such as *pipeOrganInfo* and *reedOrganImage* expanded on the nodes above them to provide additional contextual information.

Custom Links

These custom links served as points of expansion for the knowledge graph as they used data from the dataset and string manipulation. Identifying data from the dataset that was able to be expanded upon was challenging as data was very specific to the organ and external links found online were too broad. However, data that was more general such as locations could be expanded upon. Identified points of extension and the name of the expanded node are below:

- building → buildingInfo
- state → stateInfo
- maintainer → maintainerInfo

The relationship for all additional nodes was “oont:extraInformation”.

Generic Links

These links were not organ-specific but provided more detail and general information regarding some aspects of the knowledge graph. Data in the dataset being so specific restricted the amount of external generic links that could be added, nevertheless, there were some points of expansion such as:

- pitch → pitchInfo
- windPressure → windPressureInfo
- division → divisionInfo

The relationship for all additional nodes was “oont:extraInformation”.

8.3.3 Final Ontology

Combining both the dataset and external ontologies forms one ontology that can be seen in RDF. This ontology can be placed in the CONSTRUCT section of the query.

The newly improved ontology is displayed in the CONSTRUCT section of the final query located in the *Appendix Listing C1*.

8.4 Knowledge Graph Generation

The process of creating the knowledge graph generation query for the ontology will be detailed below:

8.4.1 Dataset Knowledge Graph Query

This subsection will describe the process of creating a query that is responsible for extracting data from the dataset, as planned in *Figure 7.3*. All code in this subsection will be part of the query’s WHERE clause.

The snippet below follows *Figure 7.3* and provides the template for most SPARQL Anything SERVICE calls that extract data from a JSON file. JSON Path is also used to navigate through

JSON files as shown in the SPARQL Anything documentation [27]. Specific details regarding JSON path were found on its relevant GitHub [61] to assist path generation.

```
SERVICE <x-sparql-anything:file:./output/history_base.json>
{
  BIND(CONCAT("$. ", ?_organ, ".originallocation") AS
    ?organOriginalLocation) .

  fx:properties
    fx:json.path.1 ?organOriginalLocation ; .

  [] a fx:root;
    rdf:_1 ?originalLocation ;
}
```

The query's sequence of events will be chronologically listed and explained below:

1. SERVICE call queries the *history_base.json* file in the *output* folder using SPARQL Anything.
2. CONCAT creates the JSON path using the parameter *?_organ*, which is passed in through command line, so correct data can be found for the right organ. BIND assigns the JSON path to the variable *?organOriginalLocation*.
3. *fx:properties* then uses the JSON path stored in *?organOriginalLocation* to locate the data.
4. Data is found and assigned to the variable *?originalLocation*, which is added to the knowledge graph.

This SPARQL Anything call is followed by many others to complete the knowledge graph. Although not explicitly stated in the *Design* chapter, the parameter *?_organ* is bound to the ontology variable *?organ* inside the first SPARQL Anything call as seen below:

```
BIND (?_organ AS ?organ) .
```

While viewing the produced knowledge graphs during implementation, *?dispositions* was frequently found to be an empty string. To reduce confusion, a condition was stated within that specific SPARQL Anything call. *?disposition1* acted as a temporary variable while IF() was used to identify whether it was an empty string. If so, it was assigned a new string in order to improve understandability since it was one of the main nodes depicted in *Figure 7.2*.

```
SERVICE <x-sparql-anything:file:./output/dispositions.json>
```

```

{
    BIND(CONCAT("$.", ?_organ,
        ".dispositions[?(@.current==true)].description") AS
        ?organDescription) .

    fx:properties
        fx:json.path.1 ?organDescription ; .

    [] a fx:root;
        rdf:_1 ?disposition1 ; .

    BIND(IF(?disposition1 = "", "disposition" ,
        ?disposition1) AS ?disposition)
}

```

8.4.2 External Knowledge Graph Query

This subsection will describe the process in which external data was added to the knowledge graph and how it was implemented.

Custom Data

For links that required data from the dataset, the query skeleton structure (in *Section 7.4*) and *Figure 7.3* were followed, providing the framework to continue query creation. Custom links were created from Wikipedia as it was one of the few websites that covered the scope of what could be added. It also gave adequate extra information for the data and sufficiently extended breadth of the resulting knowledge graph. However, there is still a possibility that a Wikipedia page on a particular topic had not been created yet. To match the context of the dataset, the link created was from the Dutch version of Wikipedia. As mentioned in the ontology creation *Section 8.3*, custom links were difficult to identify as data was very specific to a particular organ. Nonetheless, an example of a custom link addition can be seen below:

```

SERVICE <x-sparql-anything:file:./output/base.json>
{
    BIND(CONCAT("$.", ?_organ, ".building") AS ?organBuilding
        ) .

    fx:properties
        fx:json.path.1 ?organBuilding ; .

    [] a fx:root;

```

```

        rdf:_1 ?building ;

        BIND( IRI( REPLACE( CONCAT("https://nl.wikipedia.org/wiki/",
        ?building), " ", "_")) AS ?buildingInfo) .

    }

```

The snippet above also follows *Figure 7.3* and is used whenever a custom external link is added to the knowledge graph. The beginning of the query follows the same format as the dataset knowledge graph query but has an extra line at the end.

Using string manipulation and other SPARQL functions, the custom link is created as follows:

1. **CONCAT** combines the building name retrieved from the dataset (stored in the *?building* variable) with a generic Dutch Wikipedia link.
2. **REPLACE** removes spaces in the *?building* variable (if any) and replaces them with underscores to create a valid link.
3. **IRI** takes this new string and converts it into a URI.
4. **BIND** takes this newly created link and assigns it to the variable *?buildingInfo* to be added to the knowledge graph.

This is an example of where SPARQL Anything's Facade-X approach is useful because features of SPARQL within a SPARQL Anything query can be used without having to learn something completely new.

General Data

Links specifically from Wikidata and MusicBrainz were added at the end of the WHERE clause and outside of the SPARQL Anything calls so as to be consistent with *Figure 7.3*. On the other hand, externally identified links were added to the knowledge graph within the relevant SPARQL Anything call. Below is an example of an externally identified link that is placed within its related SPARQL Anything SERVICE call.

```

SERVICE <x-sparql-anything:file:./output/tech.json>
{
    BIND(CONCAT("$.", ?_organ, ".pitch") AS ?organPitch) .

    fx:properties
        fx:json.path.1 ?organPitch ; .

```

```

[] a fx:root;
   rdf:_1 ?pitch ;

BIND(URI("https://organhistoricalsociety.org/OrganHistory/
works/works04.htm") AS ?pitchInfo) .
}

```

This snippet follows the same structure as most SPARQL Anything calls, but the last line binds the associated URI to *?pitchInfo*. This website, in particular, found more information describing the pitch of an organ.

External data regarding MusicBrainz and Wikidata were added by following the format on the website and bound to a variable in the ontology. Most data on these websites used links from Wikidata so the PREFIX *wd* for Wikidata was employed, but some data contained explicit links. Specific SERVICE calls querying Wikidata or MusicBrainz were not necessary as all the data added to the knowledge graph came from the same organ Wikidata website [18] or MusicBrainz organ site [31]. The websites, themselves, provided translations of the page to a selection of different languages so can be understood by a variety of different users. Physical extraction, directly from the websites into the knowledge graph, was a more suitable method of expansion than using more SERVICE calls. Below, is an example of two objects extracted from Wikidata and incorporated into the knowledge graph:

```

BIND(wd:Q752638 AS ?barrelOrgan) .
BIND(IRI("https://dbpedia.org/ontology/Organ") AS ?dbpedia) .

```

8.4.3 Final Knowledge Graph Query

The final query to generate the knowledge graph involves:

1. CONSTRUCT: Ontology
2. WHERE:
 - (a) SERVICE: calls to SPARQL Anything
 - (b) External Data

Full query can be seen in the *Appendix Listing C.1* section.

8.5 Changes During Implementation

Several hindrances to progress were encountered during implementation and will be mentioned so those looking to extend this work are aware of these stumbling blocks. These may

include changes to the process outlined in the *Design* chapter or noteworthy adjustments made during the implementation phase.

1. IF() was used to replace any empty string “” for *?disposition* in order to reduce confusion for users. Being a prominent node, it was integral for it to be well understood.
2. Manual input of Wikidata and MusicBrainz organ data was favoured over using SPARQL Anything to query the websites directly because the only sites accessed were the organ links and the overhead required to add more SERVICE calls rendered it futile.
3. Use of an adjusted ontology, as opposed to the provided ontology, was preferred and detailed in this chapter. This was to ensure a concise and understandable knowledge graph was produced that fit the scope of this project.
4. In the *Design* chapter, *Figure 7.3* depicts a flow diagram, but the implementation does not strictly adhere to this template in some aspects. For example, the “Clean & create data path” node does not specifically involve any ‘cleaning’ of data. However, the diagram is flexible given it is created with the purpose of guiding implementation.

8.6 Software Maintenance

Long-term maintenance of the query needs to be discussed to assess its longevity. Consideration of query maintenance is important for identifying any possible future software changes and for potential continuation of this work. Below are examples of some dependencies that may affect the maintenance of software:

- **Dependence on SPARQL 1.1 query language:**

This implementation depends on SPARQL 1.1 syntax as the query uses built-in methods such as *BIND* and *URI*. Deprecation or significant changes to the relevant SPARQL 1.1 features may render the query unusable or inhibit execution.

- **Dependence on SPARQL Anything:**

The implementation also depends on SPARQL Anything and its maintenance. Cessation or discontinuation of this tool may affect the usability of the query. Repurposing or significant changes to the tool may also impact usability.

- **Dataset maintenance:**

Dataset needs to be updated and maintained in a real-world context so the query does not produce an obsolete or incorrect knowledge graph.

- **JSON format redundancy:**

If technological advancements were to make the JSON format outdated, repurposing the query would be challenging since the dataset is in JSON format. The query, itself, is also tailored to JSON by using JSON path as a means of navigating through the file.

- **JSON Path redundancy:**

Alternate means of navigating through JSON files will be required if JSON Path were to become redundant as the query relies on navigation through the dataset's JSON files.

- **Command Line redundancy:**

If use of the command line were to become redundant, execution of the query would not be possible and a knowledge graph could not be automatically generated.

8.7 Results

An example of a generated knowledge graph can be seen in the *Appendix Listing A.1* section for the organ *Part14_000Brouwershaven*.

9

Evaluation

This chapter details how the knowledge graph is evaluated upon completion and measures implementation success. Both qualitative and quantitative methods were used during this section to provide a comprehensive evaluation.

9.1 Requirements-Based Assessment

Assessing whether requirements listed in the *Requirements* chapter were achieved is important to measure success of implementation within the scope of the project. For each requirement, a description stating whether it was achieved will be provided.

9.1.1 Software Requirement Assessment

Requirement No.	Software Requirement	Assessment
--------------------	----------------------	------------

1	Ontology must be correctly configured and input into the query's CONSTRUCT clause.	Ontology has been refined and adapted based on needs of the project. All relationships or triples have either been derived from the provided ontology or appended to the ontology based on the dataset.
2	The knowledge graph created must correctly reflect the organ dataset and follow ontology structure.	Knowledge graph produced does reflect data from the organ dataset. More detail on the quality of the knowledge graph is in <i>section 9.2</i> .
3	The dataset must be refined to ensure correct data is being input into the knowledge graph.	Query correctly extracts the right data from the dataset and adds it to the knowledge graph. Data is usually mapped onto the ontology one-to-one, without further adjustment to ensure correct data is input into the knowledge graph.
4	The knowledge graph must expand on the current dataset using external links.	Using the organ Wikidata site [18] and organ MusicBrainz site [31], data was added to the existing knowledge graph. Custom links based on data in the dataset as well as other external links were also added as extensions to the knowledge graph. However, DBpedia was omitted as an external source due to its simplicity and overlap with the chosen sites.
5	Knowledge graphs must be evaluated to prove correctness and validity.	This will be evaluated in <i>Section 9.2</i>

6	Scalability of knowledge graph generation must be assessed.	This will be assessed and evaluated in detail in section 9.3.
---	-------------------------------------------------------------	---------------------------------------------------------------

Table 9.1: Software Requirement Assessment Table

9.1.2 User Requirement Assessment

Requirement No.	User Requirement	Assessment
1	User must be able to execute written query.	User can download SPARQL Anything [27] and execute a given query on command line.
2	User must be able to select an organ to view information on.	User can view the codes of all available organs in the dataset using the organids.json file. This organ code can then be passed in through command line, which generates a custom knowledge graph upon execution.
3	User must be able to see the knowledge graph following query execution.	User can either view the knowledge graph on command line or in a TTL file with a specified location and file name.
4	User must be able to see relevant data in the knowledge graph.	User can view the knowledge graph and see relevant data corresponding to the requested organ.
5	User must be able to view other relevant external data in the knowledge graph.	User can view the knowledge graph and see external links as well as custom links based on data from the requested organ.

Table 9.2: User Requirement Assessment Table

9.2 Knowledge Graph Quality Assessment

Assessment of knowledge graph quality is important to ensure the produced knowledge graph is valid and correct. [38] was used as guidance to assess knowledge graph quality as well as the 18 qualitative requirements detailed in [13]. For each attribute, specific details are assessed with explanations and evaluation of knowledge graph quality.

The three main areas used to assess knowledge graph quality are:

- Accuracy
- Coverage
- Succinctness

For simplicity, five randomly selected knowledge graphs will be used to measure overall quality and will be representative of all organs. Organs to be tested from the dataset are: *Part14_000Brouwershaven*, *Part14_000Niezijl*, *Part14_000Folsgare*, *Part14_000GravenhageNoorderkerk* and *Part14_000Groede*. All produced knowledge graphs are in the *Appendix Listing A.1 to A.5*. Generating the knowledge graphs and assessing quality from the output files was the method employed.

9.2.1 Accuracy

“Accuracy refers to the extent to which entities and relations- encoded by nodes and edges in the graph- correctly represent real-life phenomena” [38]. In this context, accuracy means the knowledge graph generated (both nodes and edges) needs to correctly reflect the relationships between organ topics.

There are three types of accuracy:

- Syntactic Accuracy.
- Semantic Accuracy.
- Timeliness.

Syntactic Accuracy

Syntactic accuracy refers to whether the data presented is valid for its given type [38].

An example of a potential violation:

“This is an organ” would be incompatible with xsd:integer.

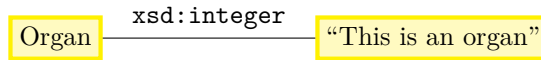


Figure 9.1: Syntactic Accuracy Violation

Upon assessing the produced knowledge graphs for each of the five organs, no syntactic inaccuracies were found. Since the ontology was created using the provided ontology and external links, the resulting knowledge graphs are, theoretically, free from any syntactic inaccuracies.

Semantic Accuracy

Semantic accuracy refers to whether data values are correctly represented in a real-world context [38].

An example of a potential violation:

Date of build (for an organ) coming after today's date.

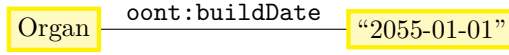


Figure 9.2: Semantic Accuracy Violation

In regards to semantic accuracy, none of the five organs contained semantic inaccuracies upon assessment. This can be explained for similar reasons as above regarding the absence of syntactic inaccuracies.

Timeliness

Timeliness refers to how up-to-date or relevant the knowledge graph is with the current state of the modern world [38]. This is also in line with [13] stating that “Knowledge graphs should contain the latest resources to guarantee freshness”.

An example of a potential violation:

*Organ's location stated as “Netherlands”, but has recently been moved to “Portugal”
and the knowledge graph has not been updated.*

In the produced knowledge graphs, violations of timeliness can occur due to the static nature of the dataset. An example of a violation may come about if the *?disposition* is no longer the organ's current disposition in real life, but data in the dataset is not up-to-date. Therefore, the knowledge graph produced, in this case, will output an untimely solution. This particular violation will occur progressively as the organ's divisions can change over time. However, at the current moment in time, no timeliness violations occur in the five produced knowledge graphs due to the recentness of the used dataset.

9.2.2 Coverage

“Coverage refers to avoiding the omission of domain-relevant elements, which may yield incomplete results” [38]. In this context, the knowledge graph must completely fill the ontology framework in all of its nodes and edges. This is to accurately represent the entire dataset given. There are two types of coverage:

- Completeness.
- Representativeness.

Completeness

Completeness ensures the knowledge graph is correctly filled with all the relevant information present in a particular dataset [38]. Adding contextual information about entities and from different resources [13] is also important to ensure completeness.

Some examples of potential violations:

- *Organ classes lacking information.*
- *Important values missing for a specific organ property.*

Reasoning for the omission of some data is related to how it can be represented in the knowledge graph without causing confusion. For example, the specific ranges of a given organ for each keyboard or pedal are omitted due to the possible confusion it may cause when representing their ranges (*?range1* and *?range2*). In addition, representing all the organ's ranges would be challenging and it may be seen as unnecessary. However, this is only relevant to some data.

Contextual information is added in the form of descriptions and custom links in all assessed knowledge graphs. The relationship *oont:extraInformation*, for instance, provides context and extra detail for a specific node. Different resources can be seen in the five knowledge graphs such as Wikidata links, data from the dataset and various other website links.

To adequately measure dataset coverage and completeness, one of the tested knowledge graphs was selected, namely organ: *Part14_000Brouwershaven* displayed in *Appendix Listing A.1*. Using the JSON files in the dataset, its coverage was calculated. Out of 31 pieces of data available for a given organ throughout the dataset, 28 were covered by the resulting knowledge graph. This demonstrates the completeness of the existing knowledge graph but also shows room for possible improvement if the ontology were to be expanded upon.

In terms of completeness of Wikidata and MusicBrainz, all relevant details on both pages were appended to the knowledge graph. Details on Wikidata such as identifiers (i.e. Bibliothèque nationale de France ID, GND ID, National Library of Israel J9U ID etc.), however, were of little significance so were omitted from the knowledge graph to reduce confusion.

Representativeness

Representativeness ensures the knowledge graph does not involve any bias and does not exclude anything relevant [38].

Some examples of potential violations:

- *Under-represent organ data from other languages.*
- *Under-represent people (e.g. organist, owner) from a particular gender, race etc.*

Violations regarding under-representation stem from the dataset provided. For example, the dataset is written in Dutch so organ data is limited to the Dutch language. With respect to the under-representation of people, anyone involved with the organ is mentioned in all five knowledge graphs such as: *?maintainer* and *?builder* so no violations occur in that regard.

9.2.3 Succinctness

“Succinctness refers to the inclusion of relevant content that is represented in a concise and intelligible manner” [38]. In this context, it means the knowledge graph created should be easily understandable and to the point.

There are two types of succinctness:

- Representational Conciseness.
- Understandability.

Representational Conciseness

Representational conciseness refers to the extent in which knowledge graph content is compactly represented [38]. This is consistent with requirements from [13], particularly: “Triples should be concise” and “Knowledge graph does not contain redundant triples”.

An example of a potential violation:

Containing two properties that serve the same purpose i.e.

- *Organist*
- *Organ Player*



Figure 9.3: Representational Conciseness Violation

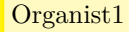
Each knowledge graph is created with the intent of making them concise. Creation of the ontology structure was derived and refined from the provided ontology. *Figure 7.2* in the design was made specifically to represent the data in a concise manner and avoid the inclusion of redundant data. Implementation also kept this in mind when refining the ontology. Upon assessing each of the five organs, the produced knowledge graphs are representationally concise.

Understandability

Understandability pertains to the requirement for the knowledge graph produced to be easily understandable and readable [38].

An example of a potential violation:

Using property names such as `Organist1`, when asked to state the name of an organist (Actual name should be used instead).



Organist1

Figure 9.4: Understandability Violation

Data in the dataset is understandable from the perspective of a Dutch reader. Given the relationships, the meaning of data can be easily inferred. However, there are some cases in the produced knowledge graphs where nodes produce an empty string “” due to vacant data in the dataset. For example, the variable *?partition* in the ontology is often empty and, out of the five tested organs, four of the *partition*’s data is missing. However, important nodes such as *?disposition*, which were integral to the understanding of other data in the knowledge graph, were replaced with other comprehensible strings to reduce confusion.

From a lay user’s perspective, the information from Wikidata may appear perplexing at first glance since only codes (i.e. wd:Q1327327, wd:Q752638 etc.) of the unique Wikidata link are displayed. However, this can be mitigated by physically clicking the URI link while viewing these external Wikidata links.

To accurately measure understandability, a survey was carried out consisting of various competency questions for a given knowledge graph in *Appendix Listing A.6*. Assessing participants’ ability to answer these questions gave an accurate representation of how understandable the knowledge graph was. However, this may be hindered by the knowledge graph presented to participants being translated from Dutch to English, which was necessary as most people were predominantly English speakers. The competency questions, themselves, were derived from the provided organ ontology’s GitHub page. Not all questions on this page were used to reduce time completion and encourage participation in the survey. Questions used in the survey were:

1. Who built the organ?
2. Where are the original parts of the organ?
3. Where is the organ originally located?
4. When is the organ moved to another location?

5. Why is the organ moved to another location?
6. Any additional comments?

All ten participants consisted of those with a Computer Science background and were knowledgeable of Semantic Web’s technologies, which consisted mainly of second or third-year Computer Science University students at King’s College London. One participant was a native Dutch speaker and completed the task using the original knowledge graph as they were able to understand it. Users of this knowledge graph are part of the Polifonia project so are knowledgeable in this field. Therefore, sampling users from a Computer Science background is a fair representation of a potential user’s capability. Ensuring this particular set of users could understand the knowledge graph, was vital.

Surveys were conducted using Google Forms and shared with all eligible participants in *Appendix Figure A.1*. Answers to each question on the survey were usually either correct or incorrect, as they were readily available on the knowledge graph. Specifically, the form consisted of competency questions as well as the possibility to add any additional comments. The knowledge graph, itself, was sent as part of the survey invitation in a separate file. Number of participants with correct answers for each question is plotted below:

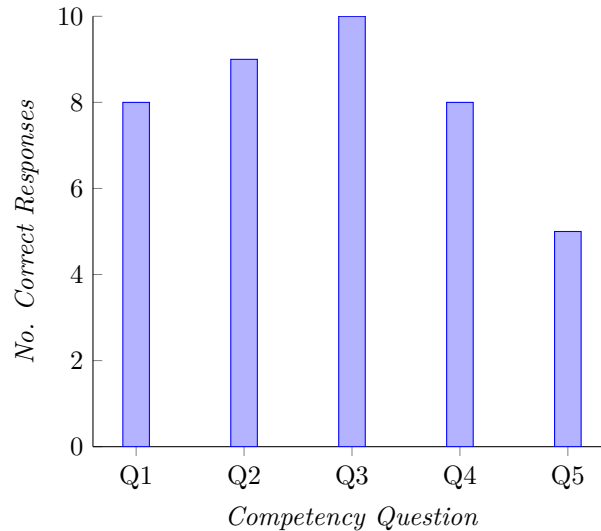


Figure 9.5: Correct Competency Question Responses

The bar chart *Figure 9.5* plots the number of correct responses per question. In general, the produced knowledge graph was well understood by participants and were mostly able to get the correct answer to the competency question. This provides evidence of understandability although the last competency question received equivocal responses, proving the produced knowledge graph is not perfect.

When asking for comments regarding the knowledge graph (Q6), they showed a common theme of confusion surrounding the musical aspects, which may have impacted results.

9.3 Query Scalability Assessment

In this section, scalability of the knowledge graph will be tested. In particular, the areas being measured to assist quantification of knowledge graph scalability are:

- Number of SERVICE Calls.
- Size of Files.

For each one, measurement of scalability will be computed using the time required to execute the query on command line, which is how long it takes to generate the knowledge graph. This will make use of a command on Windows OS called: *Measure-Command* [46], which measures the time required for a command to execute. For instance, the command used to measure time it takes to generate a knowledge graph for organ: *Part01_001MIDDE* is:

```
Measure-Command{java -jar sparql-anything-0.8.1.jar -q
queries/organ-details.sparql --values
organ=Part01_001MIDDE -o output/output.ttl}
```

The same sample of five organs will be selected for knowledge graph generation and tests will be run on a 64-bit OS with Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz and 8GB RAM. Regarding time calculation, the procedure will involve running each of the five sampled organs ten times using *Measure-Command*, followed by calculation of the average time per organ. Finally, mean times for all five organs will be noted.

9.3.1 Size of Dataset

For this test, knowledge graph generation time was calculated with respect to dataset size, which was accomplished through the removal of data in the JSON files. The five tested organs, however, will always remain in the files. This is possible since the structure of each JSON file follows the same format as shown in the *Context* chapter. External links were left in the query as their addition did not significantly affect results. After systematically removing organs from each JSON file to measure size, the actual size of a file will be noted using a VS Code extension: *filesize* and the dataset's total size will be calculated. A graph plotted with cumulative file sizes against time can be seen in *Figure 9.6*.

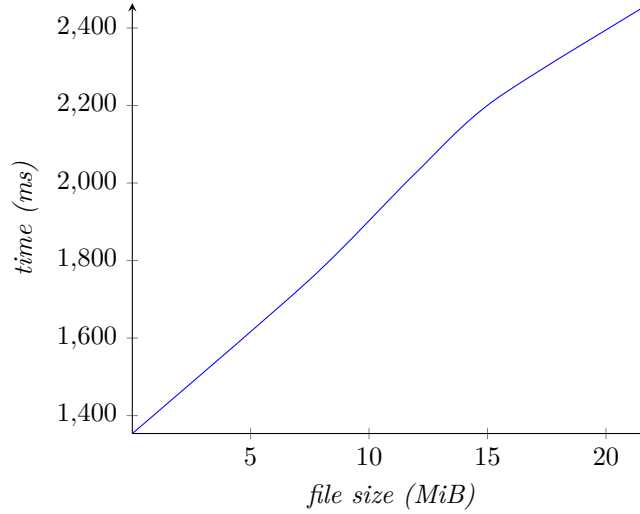


Figure 9.6: Analysis of Dataset Size

Figure 9.6 shows an initial linear relationship between the dataset’s size and time. This shows that the knowledge graph-generating query produced is scalable to a certain extent. It is unknown, however, how well the query will perform for much larger dataset sizes.

Measurement of much larger input values was done in [19] and showed an exponential relationship between execution time and input size. However, the findings in relation to our input size are consistent as the paper does show a linear relationship in the smaller input sizes. Therefore, it is not a problem for the scope of our generated knowledge graph. Nevertheless, extremely large input sizes may pose a potential issue scalability-wise in the future.

9.3.2 Number of SERVICE Calls

In this test, number of SERVICE calls within the query are tested to observe their effect on the speed of knowledge graph generation. In the implemented query, removal of SERVICE calls will be done incrementally and plotted against execution time. Measuring the number of SERVICE calls will indicate scalability of the knowledge graph, providing insight into its capacity for expansion.

For this project, knowledge graph size will increase proportionally with the number of SERVICE calls. A mock knowledge graph was created using additional SERVICE calls that were not in the project scope and added for testing purposes to measure expandability. Data gathered from the SERVICE call was added to the mock knowledge graph. This was to account for the genuine overhead when adding data onto the knowledge graph during command execution. In this particular test, all external links in the knowledge graph were removed so as to not affect

execution time. The plotted graph for number of SERVICE calls against time can be seen in *Figure 9.7*.

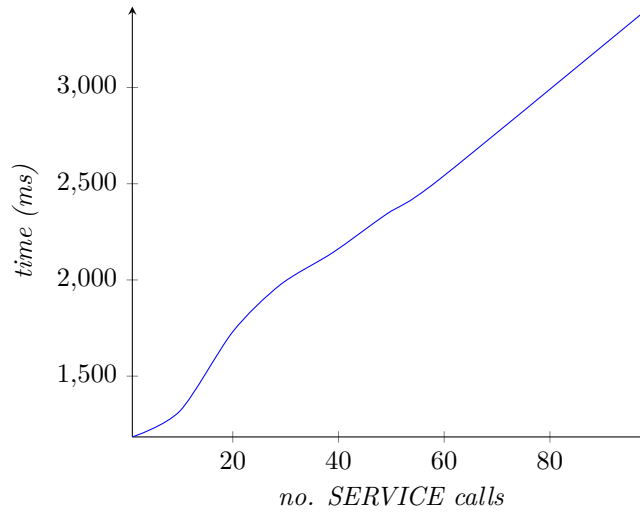


Figure 9.7: Analysis of SERVICE Calls

After plotting the graph (*Figure 9.7*) of up to 100 SERVICE calls, an initial linear relationship can be seen between variables. The expandability of the knowledge graph is possible within reason. However, this graph only plots values for up to 100 SERVICE calls so time required for extremely large amounts of SERVICE calls may differ.

10

Legal, Social, Ethical & Professional Issues

In this chapter, legal, social, ethical and professional concerns surrounding the project will be discussed. The project adhered to both the Code of Conduct issued by the British Computer Society [6] and the FAIR Principles [62] outlined in an article discussing the reuse of scholarly data.

10.1 Code of Conduct

The project acts in public interest as it enhances knowledge discovery through the creation of a knowledge graph, providing a broader understanding of organs. All external resources used are open-source and allow for the use of data. In particular, Wikidata's license [16] states that it is dedicated to the public domain and has no copyright. MusicBrainz license [15] allows for adaption provided credit is given. Consent has been obtained from external sources and licenses for both websites were strictly followed. SPARQL Anything's license [32] was also adhered to as the tool was used for this project, which is permitted in its Apache 2.0 license. The project aims

to provide anyone from any culture or background with a means of viewing musical heritage data on the WWW without discrimination. The Polifonia project, itself, involves people from many different fields so it enables multi-disciplinary collaboration.

The project complies with professional competence and integrity by incorporating knowledge from various sources of information such as those stated in the *Literature Review* chapter. Throughout the report, documentation and reasoning for actions are explained with appropriate citations where necessary to ensure academic integrity. Feedback throughout the project from the supervisor Dr Albert Meroño-Peñuela was kindly accepted whether it be constructive criticism or project advice, enabling consideration of alternate viewpoints and seeking valuable support for the work.

Relevant authorities understand the project being undertaken as part of the Polifonia project. Communication with the supervisor has allowed for a smooth process and guaranteed complete transparency for the project's duration. Definition of intellectual data such as the dataset was made explicit throughout the project to ensure that data ownership was clearly defined.

Duty to the profession was also considered throughout the project. *Requirement*, *Specification* and *Design* chapters describing development of the project used similar software development techniques. A professional relationship was kept with all those who assisted during the project's evolution to ensure respect and integrity. Discussion of alternative tools demonstrated a commitment to continuous professional development by considering all available technologies and selecting the most appropriate option.

However, it is possible to create a malicious knowledge graph by following the same process used to generate the knowledge graph in this project. As a result, it is important to limit the distribution of this report to trusted parties only. Malevolent actors may also attempt to use this process to create knowledge graphs that disseminate false or fabricated information. Data in the knowledge graph being in Dutch may make some users feel excluded or may introduce bias from a Dutch perspective. However, the knowledge graph can be translated into the appropriate language if necessary as it is publicly available. Overall, the dataset revolves around organs so the produced knowledge graph should not contain anything that may cause offence.

10.2 FAIR Principles

The acronym FAIR stands for: **F**indable, **A**ccessible, **I**nteroperable and **R**eusable. The article [62] was written by a diverse set of publishers to promote the sharing and reuse of

research data.

10.2.1 Findable

This ensures resources used are easy to find. The project's provided resources are all publicly searchable as part of the Polifonia project, which includes the dataset and ontology. The knowledge graph generation query, itself, is in a public GitHub repository and can be found by anyone.

10.2.2 Accessible

This ensures resources are available and can be accessed with ease. The project is stored in a public GitHub repository with no restraints as are the provided ontology and dataset. A lay user of any culture or background can use the resources as they wish, given they have internet access, so is not discriminatory. A user guide in the Appendix B.1 has also been provided to help any user execute the project.

10.2.3 Interoperable

This ensures resources are flexible and can integrate with others. Alternative external resources have been incorporated into the project so further expansion is entirely achievable. Standard formats such as RDF are also used to facilitate integration with other resources. The tool SPARQL Anything was used as it is publicly available so others may also use it.

10.2.4 Reusable

This ensures resources can be used in different contexts. Although the created ontology has been refined for the project's goals, it may be reused along with the query. This report also provides ample detail for project replication if anyone wishes to do so and involves a reference list with relevant sources for further details. The source code in the GitHub repository is made available under the MIT license, permitting unrestricted use and reuse by anyone.

11

Conclusion

In this final chapter, the project will be concluded with directions for future work or possible extensions. Limitations of the project will also be discussed.

11.1 Concluding Remarks

In general, knowledge graph generation using SPARQL Anything was successful with the resulting knowledge graph displaying adequate scalability and qualitative attributes such as completeness and understandability. Generating a knowledge graph using SPARQL Anything to display data from the WWW was a quick and simple solution to address the lack of structurally represented data. The process for generating knowledge graphs has been an extensive procedure, but compared to other tools, was an effective and uncomplicated method for converting any dataset into a knowledge graph.

Nonetheless, using SPARQL Anything as a tool for knowledge graph generation was challenging. Having no prior experience with SPARQL or using the tool SPARQL Anything proved

difficult, so careful reading of documentation [27] and studying SPARQL 1.1, itself, using [26] was required. Limited documentation surrounding SPARQL Anything was also problematic as resources surrounding the tool were scarce. Nevertheless, SPARQL Anything’s Facade-X approach compensates for this problem by allowing users to use SPARQL 1.1 syntax, where resources were plentiful. SPARQL Anything’s use of existing technologies such as JSON path was also helpful as there was sufficient information surrounding it. Compared to other knowledge graph generation tools such as RML [53], SPARQL-Generate [30] and LLMs like ChatGPT [29], SPARQL Anything provided a simple and accurate approach.

Extensive background and context research were also essential to understanding the project scope and provided resources. This research facilitated the implementation and evaluation phases when generating the knowledge graph.

The knowledge graph that was produced underwent qualitative and quantitative evaluations to verify its quality and scalability. Critiquing the solution was vital to provide an unbiased review, so improvements could be made in the future. Evaluation revealed that knowledge graph quality was adequate but improvements, as always, could still be made and quantitative tests showed scalability to a certain extent.

If additional time was permitted during the course of this project, more sections of the provided ontology would be used- possibly with external organ datasets or other valid resources.

11.2 Limitations

Limitations of the project have been considered for the duration of this project. Being aware of the project’s limitations is vital to ensure readers intending to extend this work are knowledgeable of its bounds. These limitations are split into subsections and explicitly listed below:

11.2.1 General Limitations

- **Software reliability on external factors.**

Maintenance factors identified in *Implementation 8.6* section may limit the query’s ability to execute in the future. Deprecation or significant changes in tools used may render the query ineffective and extension of the project may no longer be feasible.

- **Bias being introduced from the provided resources.**

The provided dataset and ontology may include pre-existing biases as they may only

include data that has been specially curated for the Polifonia project.

- **Invalid Wikipedia pages for some external custom links.**

Due to the particularity of some data in the dataset, custom links created may not always produce populated Wikipedia pages. However, relevant recommended Wikipedia pages can be seen if one has not been created yet.

11.2.2 Evaluation Identified Limitations

- To ensure accuracy of the knowledge graph produced, it will be necessary to maintain and update the dataset on a regular basis.
- Dataset is written in Dutch so is limited to external links surrounding Dutch organs. Therefore, the produced knowledge graph mostly contains Dutch data and is limited to Dutch readers.
- The presence of some empty strings in the dataset indicates missing data, which may limit the knowledge graph to a certain extent based on the completeness of the provided dataset.
- The evaluation, based on file size and number of service calls, is limited in scope and may not entirely reflect relationships between these factors and time for larger datasets. As a result, this may impact scalability of the query used for generating the knowledge graph.

11.3 Future Work & Extensions

This section discusses future work and possible extensions of the project, which may also provide solutions to the limitations identified above. Anyone willing to continue or extend work completed in this project is welcome to take the directions detailed below, but should also be wary of the limitations noted above.

- **Extension:**

Use of external datasets from websites such as Kaggle [41], Google Dataset Search [34] and many others, could further extend the knowledge graph and potentially provide a more detailed solution. Finding external datasets, however, may prove challenging due to the specificity of the current dataset.

- **Extension:**

Use of DBpedia [17] as an external data source, similar to how Wikidata and MusicBrainz were used. However, when researching its potential use, external links were not as relevant as the selected options.

- **Extensions:**

Use the provided ontology to its fullest capability by creating external data sources. Creation of relevant data files by extracting information from the WWW may also supplement the existing dataset and leverage the full potential of the provided ontology.

- **Future Work:**

Application of Natural Language Processing techniques can be used to extract and integrate relevant information from the WWW into the knowledge graph. Gaps in the dataset can also be addressed by identifying relevant information to that organ on the WWW. This may also present the opportunity to explore datasets in languages other than Dutch, which can broaden the scope of datasets that can be used. A recent article [23] details this process of generating knowledge graphs using Natural Language Processing, which may prove useful.

- **Future Work:**

Develop a user interface or visualisation tool that enables user interaction with the generated knowledge graph by providing a visual display. This will aid understandability of the knowledge graph as well as make it more intuitive from a user’s perspective. It will also help lay users who want to use or interpret the knowledge graphs for themselves. An example of data visualisation can be seen in an existing tool ‘QueDI’ that allows lay users to easily query knowledge graphs. More detail on this tool can be found in an article [22]. Alternatively, knowledge graph visualisation frameworks already exist and may be used to aid data visualisation, for example, [49].

- **Future Work:**

Assess the knowledge graph’s performance by testing it with significantly larger values than those used to evaluate scalability in the *Evaluation 9.3* section. Alternate methods of evaluation can also be explored with other metrics such as estimation of accuracy [33].

Those looking to extend or replicate this project should pay close attention to SPARQL Anything [27] documentation and gain a comprehensive understanding of all available examples in other formats such as XML, HTML [4] and CSV [52]. Familiarity with the dataset and a

broad understanding of the semantic web, similar to this project’s *Context* and *Background* chapters, would also be advised. Adequate planning such as ontology definition and data preprocessing may be necessary depending on the project. Software maintenance issues in *Implementation 8.6* section must also be considered for long-term use. They should also be aware of knowledge graph evaluation metrics discussed in *Evaluation 9.3* section and continue to adhere to the Code of Conduct [6] and FAIR Principles [62] throughout development.

References

- [1] Thomas Acreman. Organ early history. <http://www.classichistory.net/archives/organ>, 2018. Accessed: 2022-12-07.
- [2] Willi Apel. Early history of the organ. *Speculum*, 23(2):191–216, 1948.
- [3] Arvind Arasu, Hector Garcia-Molina, and Stanford University. Extracting structured data from web pages. *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data - SIGMOD '03*, 2003.
- [4] Luigi Asprino, Enrico Daga, Aldo Gangemi, and Paul Mulholland. Knowledge graph construction with a façade: a unified method to access heterogeneous data sources on the web. *ACM Transactions on Internet Technology*, 23(1):1–31, 2023.
- [5] ASQ. Flowcharts. <https://asq.org/quality-resources/flowchart>. Accessed: 2022-12-12.
- [6] BCS. British computer society. <https://www.bcs.org/>. Accessed: 2023-03-10.
- [7] Tim Berners-Lee and Mark Fischetti. *Weaving the web : the original design and ultimate destiny of the World Wide Web by its inventor*. Harperbusiness, 2011.
- [8] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.
- [9] Stephen Bicknell. *The history of the English organ*. Cambridge University Press, 1996.
- [10] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data: The story so far. In *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227. IGI global, 2011.

- [11] K. et al Breitman. *Ontology in Computer Science*, pages 17–34. Springer London, London, 2007.
- [12] Andrew Cantrill. Royal college of organists [video]. <https://youtu.be/kAlj3CE-7mM>, 2020. Accessed: 2022-12-09.
- [13] Haihua Chen, Gaohui Cao, Jiangping Chen, and Junhua Ding. A practical framework for evaluating the quality of knowledge graph. In *Knowledge Graph and Semantic Computing: Knowledge Computing and Language Understanding: 4th China Conference, CCKS 2019, Hangzhou, China, August 24–27, 2019, Revised Selected Papers 4*, pages 111–122. Springer, 2019.
- [14] Fiorela Ciroku. Organ ontology. <https://github.com/polifonia-project/organs-ontology>. Accessed: 2022-12-02.
- [15] Creative Commons. Creative commons: Musicbrainz license. <https://creativecommons.org/licenses/by-sa/3.0/>, 2019. Accessed: 2023-03-11.
- [16] Creative Commons. Creative commons: Wikidata license. <https://creativecommons.org/publicdomain/zero/1.0/>, 2019. Accessed: 2023-03-11.
- [17] DBpedia Contributors. Organ dbpedia. <https://dbpedia.org/ontology/Organ>. Accessed: 2022-12-09.
- [18] Wikidata Contributors. Organ wikidata. <https://www.wikidata.org/wiki/Q1444>. Accessed: 2022-12-09.
- [19] Enrico Daga, Luigi Asprino, Paul Mulholland, and Aldo Gangemi. Facade-x: an opinionated approach to sparql anything. *Studies on the Semantic Web*, 2021.
- [20] Libby Miller Dan Brickley. Foaf. <http://xmlns.com/foaf/0.1/>, 2004. Accessed: 2022-12-02.
- [21] Tim Berners-Lee David Beckett. W3c ttl. <https://www.w3.org/TeamSubmission/turtle/>, 2011. Accessed: 2022-11-27.
- [22] Renato De Donato, Martina Garofalo, Delfina Malandrino, Maria Angela Pellegrino, Andrea Petta, and Vittorio Scarano. Quedi: From knowledge graph querying to data visualization. In *SEMANTiCS*, pages 70–86, 2020.

- [23] Danilo Dessì, Francesco Osborne, Diego Reforgiato Recupero, Davide Buscaldi, and Enrico Motta. Generating knowledge graphs by employing natural language processing and machine learning techniques within the scholarly domain. *Future Generation Computer Systems*, 116:253–264, 2021.
- [24] Vladan Devedzic. Education and the semantic web. *International Journal of Artificial Intelligence in Education*, 14(2):165–191, 2004.
- [25] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. Rml: A generic language for integrated rdf mappings of heterogeneous data. *Ldow*, 1184, 2014.
- [26] Bob Ducharme. *Learning SPARQL : querying and updating with SPARQL 1.1*. O’reilly, 2013.
- [27] Justin Dowdy Enrico Daga, Luigi Asprino. Sparql anything github. <https://github.com/SPARQL-Anything/sparql.anything>. Accessed: 2022-12-10.
- [28] Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandečić. Introducing wikidata to the linked data web. In *The Semantic Web–ISWC 2014: 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I 13*, pages 50–65. Springer, 2014.
- [29] John Schulman et al. Chatgpt website. <https://openai.com/blog/chatgpt/>. Accessed: 2022-12-10.
- [30] Maxime Lefrancois et al. Sparql-generate github. <https://github.com/sparql-generate/sparql-generate>. Accessed: 2023-01-06.
- [31] MetaBrainz Foundation. Organ musicbrainz. <https://musicbrainz.org/instrument/55a37f4f-39a4-45a7-851d-586569985519>. Accessed: 2022-12-09.
- [32] The Apache Software Foundation. Apache 2.0 license: Sparql anything license. <https://www.apache.org/licenses/LICENSE-2.0>, 2019. Accessed: 2023-03-11.
- [33] Junyang Gao, Xian Li, Yifan Ethan Xu, Bunyamin Sisman, Xin Luna Dong, and Jun Yang. Efficient knowledge graph accuracy evaluation. *arXiv preprint arXiv:1907.09657*, 2019.
- [34] Google. Google dataset search. <https://datasetsearch.research.google.com/>. Accessed: 2023-03-03.

- [35] Simon Gottschalk. Creation, enrichment and application of knowledge graphs. 2021.
- [36] RDF Working Group. W3c rdf. <https://www.w3.org/RDF/>, 2014. Accessed: 2022-11-27.
- [37] Chris Hall. Building a search engine using knowledge graphs. <https://www.stardog.com/blog/how-to-build-a-semantic-search-engine-using-a-knowledge-graph/>, 2021. Accessed: 2022-11-28.
- [38] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D’amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, Jose Emilio, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M Rashid, Anisa Rula, Juan Sequeda, Lukas Schmelzeisen, Steffen Staab, and Antoine Zimmerman. *Knowledge Graphs*. Morgan I& Claypool Publishers, 2021.
- [39] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d’Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. Knowledge graphs. *ACM Computing Surveys (CSUR)*, 54(4):1–37, 2021.
- [40] Knowledge Media institute. Polifonia website. <https://kmi.open.ac.uk/projects/name/polifonia>. Accessed: 2023-01-24.
- [41] Kaggle. Kaggle datasets. <https://www.kaggle.com/datasets>. Accessed: 2023-03-03.
- [42] UNIBO KCL, IReMus NISV, Albert Meroño-Peñuela KCL, Jacopo de Berardinis, Anita Carriero, Mari Wigham, Andrea Poltronieri, Fiorela Ciroku, Christophe Guillotel-Nothmann, and Philippe Rigaux. D2. 1: Ontology-based knowledge graphs for music objects (v1. 0). *Musical Heritage Knowledge Graphs*, 2021.
- [43] Ahmad R Kirmani. Artificial intelligence-enabled science poetry. *ACS Energy Letters*, 8:574–576, 2022.
- [44] Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. Flexible rdf generation from rdf and heterogeneous data sources with sparql-generate. In *Knowledge Engineering and Knowledge Management: EKAW 2016 Satellite Events, EKM and Drift-an-LOD, Bologna, Italy, November 19–23, 2016, Revised Selected Papers*, pages 131–135. Springer, 2017.
- [45] Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. A sparql extension for generating rdf from heterogeneous formats. In *The Semantic Web: 14th International*

Conference, ESWC 2017, Portorož, Slovenia, May 28–June 1, 2017, Proceedings, Part I 14, pages 35–50. Springer, 2017.

- [46] Microsoft. Windows measure command. <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/measure-command?view=powershell-7.3>. Accessed: 2023-02-25.
- [47] Eric Miller. An introduction to the resource description framework. *D-lib Magazine*, 1998.
- [48] Riichiro Mizoguchi. Part 1: Introduction to ontological engineering. *New Generation Computing*, 21(4):365–384, 2003.
- [49] Rungsiman Nararatwong, Natthawut Kertkeidkachorn, and Ryutaro Ichise. Knowledge graph visualization: Challenges, framework, and implementation. In *2020 IEEE Third International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 174–178, 2020.
- [50] Orpha Ochse. *The history of the organ in the United States*. Indiana University Press, 1988.
- [51] Sergio Oramas, Vito Claudio Ostuni, Tommaso Di Noia, Xavier Serra, and Eugenio Di Sciascio. Sound and music recommendation with knowledge graphs. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):1–21, 2016.
- [52] Marco Ratta and Enrico Daga. Knowledge graph construction from musicxml: An empirical investigation with sparql anything. *Musical Heritage Knowledge Graphs*, 2022.
- [53] RML.io. Rml github. <https://github.com/RMLio>. Accessed: 2023-01-06.
- [54] Margaret Rouse. What is large language model (llm)? - definition from techopedia, Mar 2023.
- [55] John P. Santoianni. How an organ makes music [video]. <https://youtu.be/4S6BErQs-HE>, 2015. Accessed: 2022-12-09.
- [56] Leslie F Sikos. *Mastering structured data on the semantic web : from HTML5 microdata to linked open data*. Apress, Berkeley, Calif., 2015.
- [57] Rudi Studer, V.Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data I& Knowledge Engineering*, 25(1-2):161–197, Mar 1998.

- [58] A. Swartz. Musicbrainz: a semantic web service. *IEEE Intelligent Systems*, 17(1):76–77, 2002.
- [59] Case Western Reserve University. Organ medieval history. <https://caslabs.case.edu/medren/medieval-instruments/organ-medieval/#:~:text=Organs%20came%20in%20a%20wide,1404>). Accessed: 2022-12-07.
- [60] W3C. W3c vocabularies. <https://www.w3.org/standards/semanticweb/ontology>, 2105. Accessed: 2022-11-27.
- [61] Leo Wang. Json path github. <https://github.com/wanglingsong/JsonSurfer>. Accessed: 2022-12-02.
- [62] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, and Alejandra Gonzalez-Beltran. The fair guiding principles for scientific data management and stewardship. *Scientific Data*, 3(1), Mar 2016.
- [63] Christoph Wolff and Markus Zepf. *The Organs of JS Bach: A Handbook*. University of Illinois Press, 2011.