Ref. Ares(2022)4060104 - 01/06/2022

D4.5 Semantic services for AGRICORE

agricore

Deliverable Number	D4.5
Lead Beneficiary	STAM
Authors	STAM
Work package	WP4
Delivery Date	M33
Dissemination Level	Pubblic

www.agricore-project.eu


The Agricore project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No. 816078

Document Information

Project title	Agent-based support tool for the development of agriculture policies
Project acronym	AGRICORE
Project call	H2020-RUR-04-2018-2019
Grant number	816078
Project duration	1.09.2019-31.8.2023 (48 months)
Deliverable Authors	Mattia Repossi (STAM), Alessandro Fermi (STAM)
Deliverable Reviewers	Lisa Baldi (UNIPR), Álvaro Ojeda (IDE), Pablo Báez (IDE)

Version History

Version	Description	Organisation	Date
0.1	First template version	STAM	1 Mar 2022
0.2	ToC Draft	STAM	13 Apr 2022
0.3	ToC Review	IDE	15 Apr 2022
0.4	First Draft	STAM	23 May 2022
0.5	Comments and reviews	UNIPR, IDE	26 May 2022
0.6	Comments resolved	STAM	27 May 2022
1.0	Final version	STAM	31 May 2022

Disclaimer

All the contributors to this deliverable declare that they:

- Are aware that plagiarism and/or literal utilisation (copy) of materials and texts from other Projects, works and deliverables must be avoided and may be subject to disciplinary actions against the related partners and/or the Project consortium by the EU.
- Confirm that all their individual contributions to this deliverable are genuine and their own work or the work of their teams working in the Project, except where is explicitly indicated otherwise.
- Have followed the required conventions in referencing the thoughts, ideas and texts made outside the Project.

Executive Summary

This document explains the research and the development process for the Semantic Service Module for the AGRICORE project. Semantic Technology uses formal semantics to give meaning to the disparate data that surrounds us. Together with Linked Data technology, it builds relationships between data in various formats and sources, from one string to another, helping create context. Interlinked in this way, these pieces of raw data form a giant web of data or a knowledge graph, which connects a vast number of descriptions of entities and concepts of general importance (Ontology). Semantic Technology finally defines and links data by developing languages to express rich, self-describing interrelations of data in a form that machines can process. Thus, machines are not only able to process long strings of characters and index tons of data. They are also able to store, manage and retrieve information based on meaning and logical relationships. These concepts have been utilized to develop a dedicated module that enables users to query the ARDIT tool with natural language questions.

Abbreviations

Abbreviation	Full name
AI	Artificial Intelligence
ANSI	American National Standards Institute
AP	Application Profile
API	Application Programming Interface
ARDIT	Agricultural Research Data Index Tool
DCAT	Data Catalogue
DCAT-AP	Data Catalogue Application Profile
EC	European Commission
EP	European Parliament
EU	European Union
EUROSTAT	European Statistical Office
GeoDCAT-AP	A geospatial extension for the DCAT application profile for data portals in Europe
GeoPDF	Geo-Portable Document Format
GeoTIFF	Geo-Tag Image File Format
GRDDL	Gleaning Resource Descriptions from Dialects of Languages
GUI	Graphic User Interface
ICT	Information and Communication Technologies
KPIs	Key Performance Indicators
LUCAS	Land Use/Land Cover Area Frame Survey
LSTM	Long short-term memory
ML	Machine Learning
NLP	Natural language processing
OOV	Out Of Vocabulary
OWL	Web Ontology Language
POI	Point of Interest
RDF	Resource Description Framework
RNN	Recurrent Neural Network
SHACL	Shapes Constraint Language
SKOS	Simple Knowledge Organization System
UC(s)	Use Case(s)
UoA	Unit of Analysis
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTM	Urchin Tracking Module
VOCAB-DQV	Data Quality Vocabulary
XML	Extensible markup language
WP	Work Package
W3C	World Wide Web Consortium

List of Figures

List of Tables

Table of Contents

1 Introduction

The AGRICORE project proposes a novel tool for improving the current capacity to model policies dealing with agriculture by taking advantage of the latest progress in modelling approaches and Information and Communication Technologies (ICT). One of the underlying challenges that had to be addressed is the ability to extract value from the many available data sources in the Agricultural domain. To tackle this, Work Package 1 (WP1) of the AGRICORE project has dealt with the characterisation and retrieval of several databases that can provide the information instrumental in the development of the AGRICORE tool. The characterisation of the datasets allows model developers and quantitative researchers to plan their data requirements, methodological choices and the actual specification of the variables employed ahead of starting to acquire the datasets, which may be a time-consuming activity.

Secondly, Work Package 2 (WP2), leveraging the work done in WP1, provides the technological architecture to store and access this knowledge. The ARDIT tool described in "D1.9 Agricultural Research Data Index Tool (ARDIT)" plays a primary role in storing and accessing data and knowledge making use of its indexing, storage and search functionalities. Taking into consideration this last functionality (search), ARDIT, with its own capabilities, allows users to search for data leveraging standard search algorithms based on indexing. This requires prior knowledge of the data structures, which creates a barrier in the search for data. To overcome these barriers ARDIT will integrate a natural language processing (NLP) module to enable users to search for data simply by expressing a question in the English language. Semantic Technology uses formal semantics to give meaning to the disparate data that surrounds us. Together with Linked Data technology, it builds relationships between data in various formats and sources, from one string to another, helping create context. Interlinked in this way, these pieces of raw data form a giant web of data or a knowledge graph, which connects a vast amount of descriptions of entities and concepts of general importance. Semantic Technology defines and links data by developing languages to express rich, self-describing interrelations of data in a form that machines can process. Thus, machines are not only able to process long strings of characters and index tons of data. They are also able to store, manage and retrieve information based on meaning and logical relationships. Semantic services, thus, add another layer to ARDIT's search functionalities enabling users to search for related concepts instead of just matching words.

2 Ontologies and data services

An ontology can be defined as a model that represents a set of concepts and their relationships within a knowledge domain such that it can be seen as a model of knowledge and the tool for its management. Within an organisation, an ontology represents a structure created for users to answer complex questions that they address to the information systems. A detailed description can be found in "D1.1 Standardized Methodology and Set of Ontologies for the Characterization of Data Sources", but for the purpose of this document, the main concept is restated. Ontologies provide the cornerstones for developing structured data models setting the ground rules and granting interoperability in complex data architectures. The following sections will introduce all the main concepts that allow the understanding of the benefits of developing data services based on structured ontologies.

2.1 Ontologies in brief

2.1.1 Definition

An ontology is an organisational system designed to categorise and help explain the relationships between various concepts in the same area of knowledge and research. It is a way of organising concepts, information and ideas that means to be universal within the field, and allows for a common language to be spoken. It provides a common background and understanding of a particular domain, or field, of study, and ensures a common ground among those who study the information. Ontology for AGRICORE is defined with the help of some predefined vocabularies which help to make the agricultural specific ontology used for the project.

"Ontology = schema + instance"

The ontology consists of various entities and their subclasses.

- **Schema** -The ontology describes the entity and the relationship associated with every class and sub-class. A representation of a plan or theory in the form of an outline or model.
- **Instance** - The ontology has instances related to agriculture like forest, soil, food crops, land coverage, etc. Instances are the `things' represented by a concept - a human cytochrome C is an instance of the concept of Protein. The combination of an ontology with associated instances is what is known as a knowledge base.

Reusing existing ontologies may be a requirement if a system needs to interact with other applications that have already committed to particular ontologies or controlled vocabularies. Many ontologies are already available in electronic form and can be imported into an ontology-development environment. There are libraries of reusable ontologies on the Web, among these some of the most developed and used are:

- SAREF [1], the Smart Applications REFerence (SAREF) ontology is a shared model of consensus that facilitates the matching of existing assets in the smart applications domain. SAREF provides building blocks that allow the separation and recombination of different parts of the ontology depending on specific needs.
- ifcOWL [2] provides a Web Ontology Language (OWL) representation of the Industry Foundation Classes (IFC) schema. IFC is the open standard for representing building and construction data.
- The DCAT-AP [3] Application profile for data portals in Europe (DCAT-AP) is a specification based on W3C's Data Catalogue vocabulary (DCAT) for describing public sector datasets in

Europe. Its basic use case is to enable a cross-data portal search for data sets and make public sector data better searchable across borders and sectors. This can be achieved by the exchange of descriptions of data sets among data portals.

- The Open Data and Ontologies for Cultural Heritage (ODOCH) [4] aims at bringing together researchers in Semantic Web and Digital Humanities to discuss results and experiences on the design, development and use of ontology-based information systems for Cultural Heritage (Linked) Open Data.
- PANTHER Classification System [5] is a comprehensive system that combines gene function, ontology, pathways and statistical analysis tools to enable biologists to analyse large-scale genome-wide experimental data.
- MarineTLO [6] is a top-level ontology for the marine domain (also applicable to the terrestrial domain), developed to tackle the need for having integrated sets of facts about marine species, and thus to assist research about species and biodiversity. It provides a unified and coherent core model for schema mapping which enables formulating and answering queries which cannot be answered by any individual source.
- The Music Ontology [7] provides a model for publishing structured music-related data.

Ontologies, on a practical note, are specific vocabularies that are created for a particular domain or subject. For example, let us take into consideration the application of ontologies in the field of healthcare. Medical professionals use them to represent knowledge about symptoms, diseases, and treatments. Pharmaceutical companies use them to represent information about drugs, dosages, and allergies. Combining this knowledge from the medical and pharmaceutical communities with patient data enables a whole range of intelligent applications such as decision support tools that search for possible treatments; systems that monitor drug efficacy and possible side effects; and tools that support epidemiological research.

Ontology, in its purest form, is formally a field of Philosophy concerned with the "nature of existence". Ontologies are "*The study of that which is common to all things which exist" Aristotle, Metaphysics.*

Let us now consider an ontology describing different types of agriculture, agriculture-related terminologies, concepts related to land, soil, types of crops, prices related to crops, pollution, etc. The AGRICORE ontology consists of these concepts which are related to agriculture which helps to understand better the concept and in future will help policymakers take decisions for growing better crops, yielding more and better-quality crops, good prices, etc.

Some of the possible competency questions could be:

- What are the datasets containing data on soil pollution?
- What are the datasets collecting informations about crop's taxonomies?
- Which are the datasets involving energy in Italy from 2003 to 2018?

Now let us consider a list of terms needed either to make statements about AGRICORE ontology or to explain its main characteristics to a user.

- Theme of the dataset
- Type of dataset
- Purpose of the dataset
- What are units of analysis used for the measurement of temperature, land coverage, etc.

It is important to get a comprehensive list of terms without worrying about overlaps between concepts they represent, relations among the terms, any properties that the concepts may have,

or whether the concepts are classes or slots. The terms selected are those that describe objects having independent existence rather than terms that describe these objects and then the classes are organised into a hierarchical taxonomy (Figure 1).

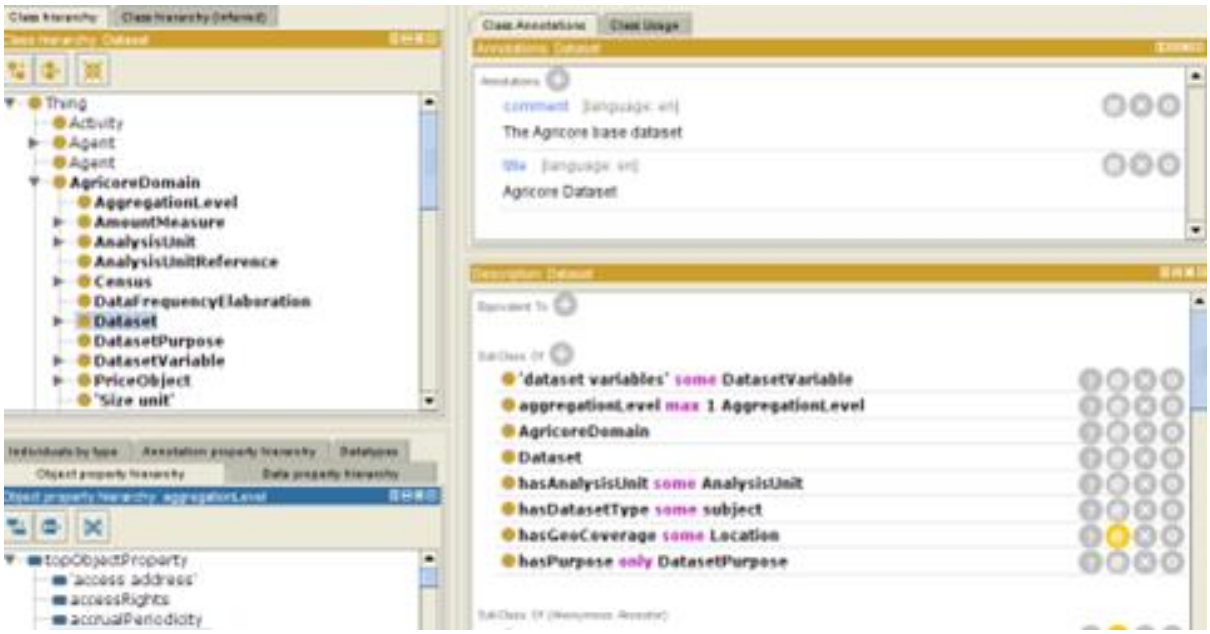

Figure 1: AGRICORE ontology example

In general, there are several types of object properties that can become slots in an ontology:

- "intrinsic" properties such as analysis unit, aggregation levels, etc.
- "extrinsic" properties such as geo-coverage location, dataset purpose, etc.
- parts, if the object is structured; these can be both physical and abstract "parts";
- relationships to other individuals; these are the relationships between individual members of the class and other items.

One of the most important things to remember is the following: there is no single correct ontology for any domain. Ontology design is a creative process and no two ontologies designed by different people would be the same. The potential applications of the ontology and the designer's understanding and view of the domain will undoubtedly affect ontology design choices. Another type of example is to use vocabularies to organise knowledge. Libraries, museums, newspapers, government portals, enterprises, social networking applications, and other communities that manage large collections of books, historical artefacts, news reports, business glossaries, blog entries, and other items can now use vocabularies, using standard formalisms, to leverage the power of Linked Data is about communities agreeing on the meaning of their data and sharing it in a massively networked information space. This vision is taking shape in many sectors, including e-commerce, medicine, scientific research, and government services. OCLC Research [8] is a leader in driving this transformation in the library community.

For every term in our language, humans assign some meaning. It would be possible to go on forever in discussing how that meaning is ascribed; it may include images, feelings, events, other words, sounds, etc. This means that humans ascribe to a set of terms in our *conceptualization.* Most importantly, conceptualizations:

- Are Hidden

- Differ from person to person

Within a certain community, this person-to-person difference may be less than between people in different communities. For example, if an individual of a certain community says "a fast horse", everyone around him will most likely know exactly what he is talking about. But an expert in horses would probably be more specific, after all, a fast thoroughbred horse (a racehorse) is different from a fast draft horse (the kind that pulls the Budweiser wagon). This "being more specific" is a *specification.*

- A ***lexicon*** is the simplest form of system that could be called an ontology. It is just a set of words, with no particular structure, and their meanings.
- A ***taxonomy*** is a very simple ontology that is one step better than a lexicon in that it provides some structure in the form of links to more specific and more general terms. For "horse" you might have: more general: ungulate (large mammal with hooves). More specific: thoroughbred, draft (various breeds of horses).
- A ***thesaurus*** is a very specific thing in computer science - there is an ANSI standard for electronic thesaurus. It is also a very simplistic form of ontology, one step better than taxonomy in that it adds a "related-to" link. For horses, you might have the general and specific links mentioned above, and then "related-to" racing and pony.

2.2 The Semantic web

The Semantic Web is a network of data that is interconnected in such a way that it can be easily processed by machines instead of humans. It can be understood as an enhanced version of the existing World Wide Web, providing an effective means of data representation in the form of a globally linked database. By supporting the incorporation of semantic content into Web pages, the Semantic Web aims to transform the currently existing Web of unstructured documents into a Web of information/data. The term Semantic Web was coined by Tim Berners-Lee [9]. The Semantic Web is driven by the World Wide Web Consortium (W3C). It is based on the W3C's Resource Description Framework (RDF) and is typically developed with syntaxes that use Uniform Resource Identifiers (URIs) to represent data. These syntaxes are referred to as RDF syntaxes. The inclusion of data in RDF files allows computer programs or web spiders to search, discover, collect, evaluate, and process the data on the web. The main goal of the Semantic Web is to trigger the evolution of the existing Web so that users can search, discover, share, and merge information with less effort. Humans can use the Web to perform various tasks, such as booking online tickets, searching for various information, using online dictionaries, etc. Nevertheless, machines are not able to perform any of these tasks without human intervention, because Web pages are made to be read by humans, not machines. The Semantic Web can be seen as a vision of the future in which data can be quickly interpreted by machines so that they can take over many tedious tasks related to finding, linking, and using the information available on the Web. The Semantic Web is a process that enables machines to quickly understand complicated human queries and respond according to their meaning. This type of understanding requires that the relevant information sources be semantically structured, which is a difficult task. Semantic Web technologies enable people to create data stores on the Web, build vocabularies, and write rules for handling data. Linked data is supported by technologies such as RDF, SPARQL, OWL, and SKOS.

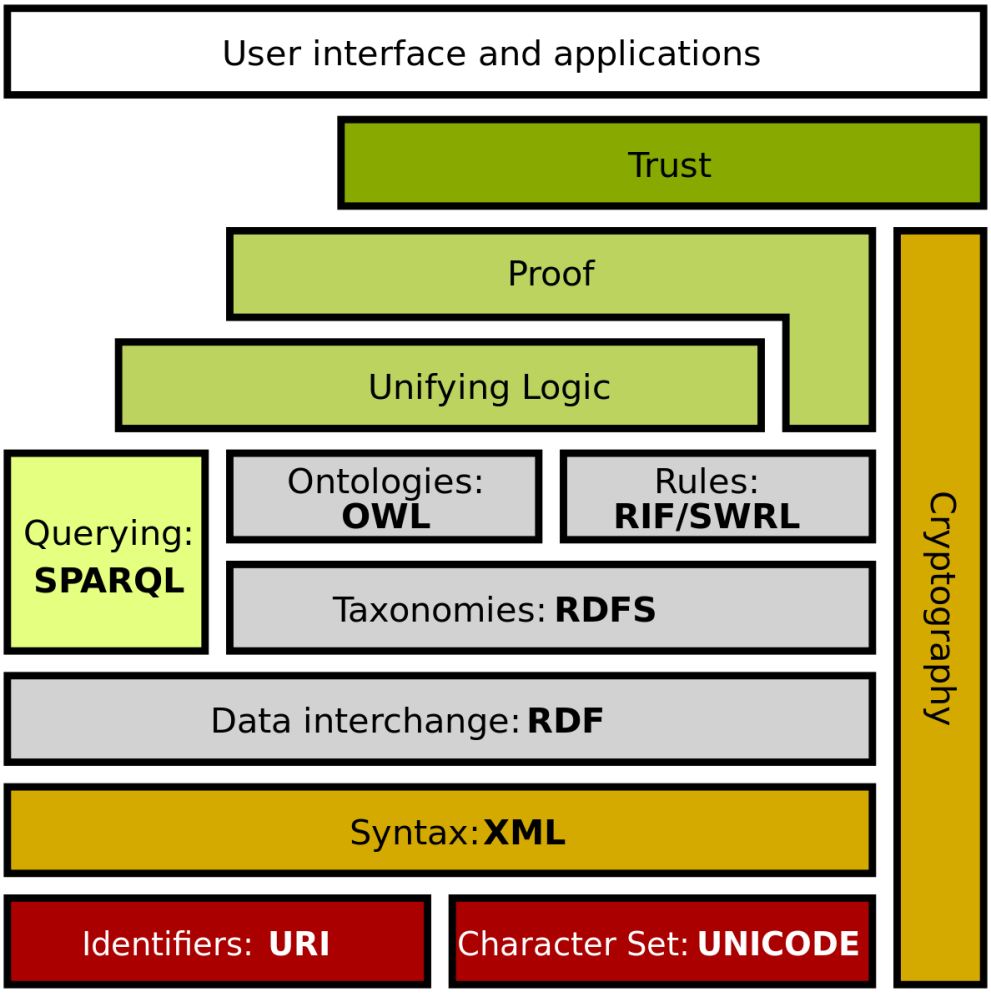

Figure 2: Semantic Web

(source: https://it.m.wikipedia.org/wiki/File:Semantic_web_stack.svg)

The main elements of the Semantic Web can thus be summarised:

- **Linked Data -** The Semantic Web is a Web of data — of dates and titles and part numbers and chemical properties and any other data one might conceive of. RDF provides the foundation for publishing and linking your data. Various technologies allow you to embed data in documents (RDFa, GRDDL) or expose what you have in SQL databases, or make it available as RDF files.
- **Vocabularies -** At times it may be important or valuable to organise data. Using OWL (to build vocabularies, or "ontologies") and SKOS (for designing knowledge organisation systems) it is possible to enrich data with additional meaning, which allows more people (and more machines) to do more with the data.
- **Query -** Query languages go hand-in-hand with databases. If the Semantic Web is viewed as a global database, then it is easy to understand why one would need a query language for that data. SPARQL is the query language for the Semantic Web.
- **Inference -** Near the top of the Semantic Web stack one finds inference — reasoning over data through rules. W3C work on rules, primarily through RIF and OWL, is focused on translating between rule languages and exchanging rules among different systems.

- **Vertical Applications -** W3C is working with different industries — for example in Health Care and Life Sciences, eGovernment, and Energy — to improve collaboration, research and development, and innovation adoption through Semantic Web technology. For instance, by aiding decision-making in clinical research, Semantic Web technologies will bridge many forms of biological and medical information across institutions

2.2.1 Linked Data

The concept of linked data is probably the backbone of the Semantic Web. It allows to model data on the Web in such a way possible related resources are all available in a standard format and are reachable and manageable by Semantic Web tools. Linked data not only map the resources we need to retrieve but also relationships among these data. These relationships are built to create a Web of Data (as opposed to a sheer collection of datasets) or, to be more suggestive, a "graph of knowledge" about a domain.

As mentioned above, the Resource Description Framework (RDF) is one of three standards of the Semantic Web that allows for representing and modelling Linked data. In particular, RDF is the data model of the Semantic Web. That means that all data in Semantic Web technologies is represented as RDF. If you store Semantic Web data, it is in RDF as well as if you query Semantic Web data (typically using SPARQL), it is RDF data. Every piece of information expressed in RDF is defined by a "Triple" with the following characteristics:

- **Subject:** any resource which may unequivocally identified by a URI.
- **Predicate:** a URI-identified specification of a relationship (typically defined and contained in some ontology).
- **Object:** a resource or a literal to which the Subject is related by the predicate.

Example: this example is a particular triple that relates a unit of measure concept, defined in the AGRICORE Ontology, with a specific value, i.e. squared kilometres, defined in the public vocabulary http://dd.eionet.europa.eu/vocabulary/eurostat/unit. The predicate is represented by a property again defined in the AGRICORE Ontology.

[https://agricore-project.eu/resource/MeasureUnit/SquareKM | agricore:measureReference | http://dd.eionet.europa.eu/vocabulary/eurostat/unit/KM2]

Following the RDF/XML syntax, the triple above is given by:

```
<agricore:MeasureUnit rdf:about="https://agricore-project.eu/resource/MeasureUnit/SquareKM">
  <agricore:measureReference rdf:resource="http://dd.eionet.europa.eu/vocabulary/eurostat/unit/KM2 />
</agricore:MeasureUnit>
```

Example: this example defines an RDF triple with a constant value.

[https://agricore-project.eu/resource/DataFrequency/triennal | agricore:mathRepresentation | "MEAN"]

Following the RDF/XML syntax the above triple can be written as:

```
<agricore:DataFrequencyElaboration rdf:about="https://agricore-project.eu/resource/DataFrequency/triennal">
  <agricore:mathRepresentation>MEAN</agricore:mathRepresentation>
</agricore:DataFrequencyElaboration>
```

The two examples above highlight some of the main characteristics of the RDF language. In particular, they made evident the importance of having available reusable collections of classes and properties that specify and characterise a domain of interest. The RDF data, indeed, make use of those classes, properties and relationships to represent data in this domain. Such collections of classes and properties are gathered and formalised in Vocabularies and Ontologies.

2.3 Data Services powered by ontologies

To this day organisations are dealing with a massive and expanding amount of data in research, health, industry, and other fields in the digital era. Data processing and interpretation are aided by information systems. This is a difficult undertaking since data utilised for a particular purpose sometimes come from disparate sources and is frequently unstructured and incomplete. To address these issues, ontologies include taxonomies and background information. Data services (or Data-as-a-Service) are collections of small, independent, and loosely coupled functions that enhance, organise, share, or calculate information collected and saved in data storage volumes. Data services in IT is a term for a third-party service that help to manage data for clients. Many uses of this term involve services that are also called “data as a service” (DaaS) – these are Web-delivered services that perform various functions on data.

In the AGRICORE project, the Semantic Service Module can be thought of as a standalone data service that based on the AGRICORE DCAT-AP 2 ontology enables the user to find datasets by performing natural language searches. One of the major characteristics of ontologies is that they ensure a common understanding of information and that they make explicit domain assumptions. As a result, the interconnectedness and interoperability of the model make it invaluable for addressing the challenges of accessing and querying data in large data structures. Also, by improving metadata and provenance, and thus allowing organisations to make better sense of their data, ontologies enhance data quality.

Ontologies, moreover, by integrating essential relationships between concepts, enable automatic inference about data. Such reasoning is easily implemented in semantic graph databases that use ontology as the semantic schema. Furthermore, ontology acts like a "brain". They "work and reason" with concepts and relationships in a way that is close to the way humans see interrelated concepts. In addition to the argument function, the ontology provides easier and more consistent navigation as the user moves from one concept to another within the ontological structure. Another interesting feature is that ontologies are easy to extend because the relationships and correspondences of concepts are easy to add to existing ontologies. Thus, this model scales with data growth without impacting dependent processes and systems if something goes wrong or needs to be changed. Ontologies also provide the means to represent all data formats, including unstructured, semi-structured, or structured data, allowing for smoother data integration, concept discovery, and more and easier text and data-driven analysis.

While ontologies provide a rich set of tools for modelling data, their usability comes with certain limitations. One such limitation is the available property constructs. For example, while providing powerful class constructs, the most recent version of the Web Ontology Language – OWL2 has a somewhat limited set of property constructs [10]. Another limitation comes from the way OWL employs constraints. They serve to specify how data should be structured and prevent adding data inconsistent with these constraints. This, however, is not always beneficial. Often, data imported from a new source into the RDF triplestore would be structurally inconsistent with the constraints set using OWL. Consequently, this new data would have to be modified before being integrated with what is already loaded in the triplestore. A novel alternative to using ontologies to model data is using the Shapes Constraint Language (SHACL) for validating RDF graphs against a set of constraints. A shape specifies metadata about a type of resource – how it is used, how it should be used and how it must be used. As such, similarly to OWL, SHACL can be applied to validate data. However, unlike OWL, SHACL can be applied to validate data already in the triplestore.

2.4 Ontologies in AGRICORE

2.4.1 Objectives

The objective of AGRICORE ontology is to cater to the need of providing a domain-specific ontology for the AGRICORE semantic search engine. The ontology must cover all the relations and types of entities with respect to the RDF data available so that it is used to create the knowledge base. This knowledge base is then used for traversing the queries that are entered by the user. The knowledge base is a highly connected graph which contains all the entities and their relationships (see deliverable D1.1 "Standardised Methodology and Set of Ontologies for the Characterisation of Data Sources" for further details).

Implementation

The word “semantic” implies meaning or understanding. As such, the fundamental difference between Semantic Web technologies and other technologies related to data (such as relational databases or the World Wide Web itself) is that the Semantic Web is concerned with the meaning and not the structure of data. This fundamental difference engenders a completely different outlook on how storing, querying, and displaying information might be approached. Some applications, such as those that refer to a large amount of data from many different sources, benefit enormously from this feature. Others, such as the storage of high volumes of highly structured transactional data, do not.

There are a number of deep concepts and methodologies that characterise the Semantic Web, some of which are listed as follows

- **Linked Data model:** a data model able not only to capture the resource but also the relationship, i.e. the link, among data in order to create a graph of interconnected resources or a graph of "knowledge".
- **Ontology:** An ontology is a formal, explicit and shared representation of a domain of interest; practically, an ontology allows a programmer to specify, in a clear, meaningful way, the concepts and relationships that characterise that domain. Ontologies are based on reusable Vocabularies that provide the terms (concepts and relationships) used in the ontology itself.
- **Semantic Search:** a data searching technique in which a search query aims to not only find keywords but to determine the intent and contextual meaning of the words a person is using for search.

For the implementation of the ontology various services were used which are mentioned below:

- **ARDIT** is used to characterise the public and private datasets which are available for the AGRICORE project.
- **DCAT-AP** is a model along with some additional vocabularies used to make the AGRICORE ontology.
- **Protégé** is an ontology development tool that allows users to create ontologies according to the Web Ontology Language (OWL).

2.4.1.1 DCAT-AP

DCAT is an RDF vocabulary designed to facilitate interoperability between data catalogues published on the Web. This document defines the schema and provides examples for its use. DCAT enables a publisher to describe datasets and data services in a catalogue using a standard model and vocabulary that facilitates the consumption and aggregation of metadata from multiple catalogues. This can increase the discoverability of datasets and data services. The DCAT Application Profile for data portals in Europe (DCAT-AP) is a specification based on W3C’s Data Catalogue vocabulary (DCAT) for describing public sector datasets in Europe. The DCAT-AP

was developed by a working group of experts following an open collaborative process. Since its initial development in 2013, this process has continued to further develop the specification, leading to the publication of DCAT-AP v1.1 in October 2015. The basic use case for DCAT-AP is to enable cross-data portal search for data sets and make public sector data better searchable across borders and sectors. This can be achieved by the exchange of descriptions of datasets among data portals. From the start, DCAT Application Profile had the purpose of adapting DCAT to facilitate the reuse of data.

2.4.1.2 Vocabularies

In computer science, vocabulary is a theoretical concept that represents the set of all possible words in a language. In NLP, though, it often means just the set of all unique tokens which appeared in a dataset. It is simply impossible to know all the possible words in a language nor is it necessary for an NLP application. The role of vocabularies on the Semantic Web is to help data integration when, for example, ambiguities may exist on the terms used in the different data sets, or when a bit of extra knowledge may lead to the discovery of new relationships. The Semantic Web is a vision of an extension of the existing World Wide Web, which provides software programs with machine-interpretable metadata of the published information and data. In other words, further data descriptors are added to otherwise existing content and data on the Web. As a result, computers are able to make meaningful interpretations similar to the way humans process information to achieve their goals.

2.4.1.3 Agricore Knowledge Graph

The core of the knowledge graph is a knowledge model: a collection of interlinked descriptions of concepts, entities, relationships, and events. Knowledge graphs put data in context via linking and semantic metadata and this way provide a framework for data integration, unification, analytics and sharing. An ontology-based approach to Semantic Search relies heavily on the RDF standard, where the data are represented by triples labelled by URI (see section above about the linked data). In other words, the data form a so-called "knowledge graph". Search engines following this approach provide often a tool for annotating Web pages with "Semantic Annotations", i.e. with pieces of meta-information coming from the underlying ontologies. Once this knowledge graph, or knowledge base, is available, ontology-based search engines use a structured query language that is most of the time based on the SPARQL language. This language, or a suitable extension thereof, allows complex graph queries with regular expressions over relationships on edge labels, which represent the relationships and axioms modelled in the underlying ontologies. Answers to the query are subgraphs of the knowledge graph matching the query (itself a graph) and are ranked by specific scoring models (see deliverable D1.1 "Standardised Methodology and Set of Ontologies for the Characterisation of Data Sources for further details").

3 Semantic Services Module Design

3.1 Agile approach

Agile refers to the ability to create and respond to change. It is a way of dealing with, and ultimately succeeding in, an uncertain and turbulent environment. The authors of the Agile Manifesto [11] chose "Agile" as the label for this whole idea because that word represented the adaptiveness and response to change which was so important to their approach. It is really about thinking through how you can understand what is going on in the environment that you are in today, identify what issues you are facing, figure out how you can adapt to them and find a solution as you go along. Agile software development refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organising cross-functional teams. Agile methods generally promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organisation and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals.

Agile software development is more than frameworks such as Scrum, Extreme Programming, or Feature-Driven Development. Agile software development is more than practices, such as pair programming, test-driven development, stand-ups, planning sessions, and sprints. Agile software development is an umbrella term for a set of frameworks and practices based on the values and principles expressed in the Manifesto for Agile Software Development and the 12 Principles behind it. One thing that separates Agile from other approaches to software development is the focus on the people doing the work and how they work together. Solutions evolve through collaboration between self-organising cross-functional teams utilising the appropriate practices for their context. There is a big focus in the Agile software development community on collaboration and the self-organising team. That does not mean that there are no managers. It means that teams have the ability to figure out how they are going to approach things on their own. It means that those teams are cross-functional. Those teams do not have to have specific roles involved so much as that when you get the team together, you make sure that you have all the right skill sets on the team. There still is a place for managers. Managers make sure team members have, or obtain, the right skill sets. Managers provide the environment that allows the team to be successful. Managers mostly step back and let their team figure out how they are going to deliver products, but they step in when the teams try but are unable to resolve issues.

3.1.1 Scrum framework

Scrum is a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value. Scrum is a lightweight framework that helps people, teams and organisations generate value through adaptive solutions for complex problems. Scrum co-creators Ken Schwaber and Jeff Sutherland have written The Scrum Guide to explain Scrum clearly and succinctly. This Guide contains the definition of Scrum. This definition consists of Scrum's accountabilities, events, artefacts, and the rules that bind them together. In a nutshell, Scrum requires a Scrum Master to foster an environment where:

1. A Product Owner orders the work for a complex problem into a Product Backlog.
2. The Scrum Team turns a selection of the work into an Increment of value during a Sprint.
3. The Scrum Team and its stakeholders inspect the results and adjust for the next Sprint.

4. Repeat

Scrum is simple. It is the opposite of a big collection of interwoven mandatory components. Scrum is not a methodology. Scrum implements the scientific method of empiricism. Scrum replaces a programmed algorithmic approach with a heuristic one, with respect for people and self-organisation to deal with unpredictability and solve complex problems. The below Figure 3 represents Scrum in Action.

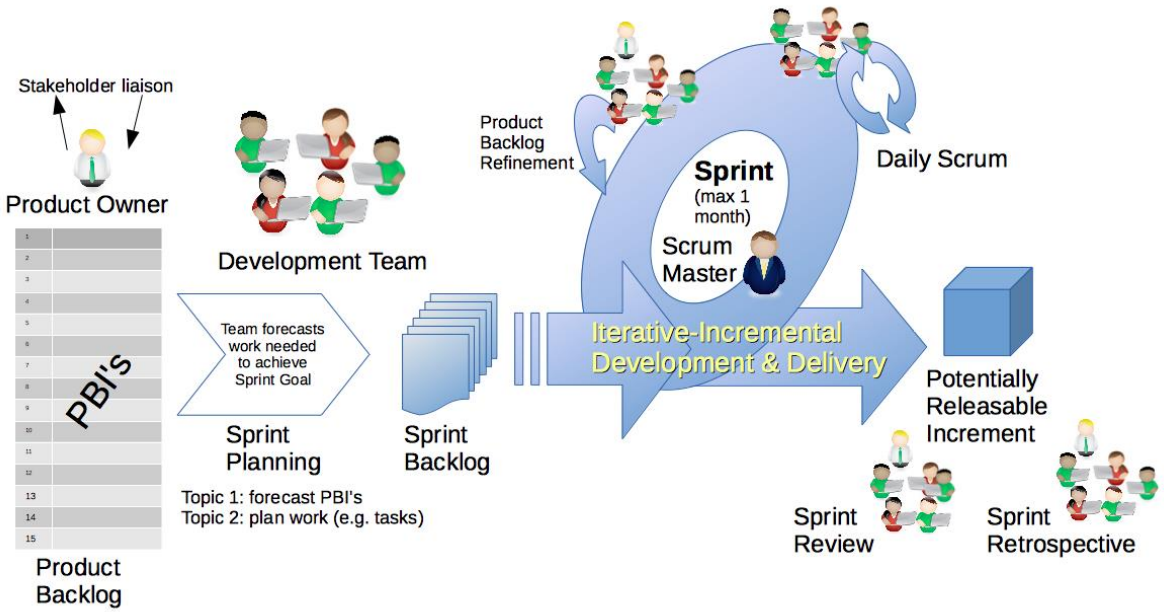

Figure 3: SCRUM Framework

(source: https://commons.wikimedia.org/wiki/File:Scrum_Framework.png)

3.2 Design of the functionalities

The design process starts from the understanding of the high level functional and technical needs of the relevant stakeholders for a given project or product. In this particular case, considering the highly technical nature of the implementation, the functional needs were overruled by the technical ones. The SCRUM framework deals with this process by defining Epics. An Epic can be defined as a unit of work which can be divided into smaller user stories for simplification. An Epic can be spread across sprints and even across agile teams and be a high-level description of what the client wants, and accordingly, it has some value attached to it. As we mentioned, an Epic is a high-level requirement, hence its scope can change over the course of time. Epics are a helpful way to organise the work and create a hierarchy. The idea is to break down the work into shippable pieces so that large projects can actually get done and can continue to ship value to the customers on a regular basis. Epics help teams break their work down while continuing to work towards a bigger goal.

For example, throwing a New Year party at home can be an Epic requirement. To do so, it is necessary to organise the effort: from the biggest objectives down to the smallest details. You should be able to respond to change, report progress, and stick to a plan. Once you are aware of

the Epic, it can be drilled down to create smaller tasks like creating a guest list, deciding on the menu, shopping grocery, decorating at home, shopping for the new year, etc.

Another fundamental process is identifying the state of the art of the solution that needs to be implemented by scouting for tools, research papers and methodologies. This helps to build an efficient implementation strategy that leverages what is already available (and verified) in order to reach the project goals. The next chapters deep dive into the definition of the project Epics and into the analysis of the methodological and technological frameworks available, focusing, firstly, on natural language processing.

3.2.1 Semantic Services Module EPICS

Epics are a theme of work that contains Issues (stories) needed to complete that larger goal. They keep product backlogs coherent and organised while providing greater control end-to-end over the release process.

Use epics:

- When the team is working on a large feature that involves multiple discussions on different issues in different projects in a group.
- To track when the work for the group of issues is targeted to begin and end.
- To discuss and collaborate on feature ideas and scope at a high level.

Considering the concepts above, 5 Epics (Table 1) were created for the development of the Semantic service module of AGRICORE.

Table 1: EPICS

ID	Brief description	For	Who	the	Is a	that
<n>	*<description>*	*<customers, stakeholders>*	*<do something>*	*<solution>*	*<something - the "how">*	*<provides this value>*
1	scouting for possible solutions	developers	need to scout for possible components, libraries and frameworks related to the implementation of semantic services based on ontologies	documentation produced and the knowledge acquired	work document	identify possible solutions to address the overall challenge
2	training of the AI model	developers	need to train a data-driven AI model	AI model	neural network	converts a natural language query into a SPARQL query
3	design of the architecture	developers	need to develop the semantic service module	documentation produced and the knowledge acquired	work document	describes the architectural scheme of the components, defines the communication protocols and designs a possible sequence diagram of a couple of use cases (to be defined)

4	development of the semantic service module	AGRICORE users	need to be able to search for datasets through natural language queries	semantic Service Module	application	that is able to integrate with third parties providing search functionalities based on natural language processing and ontologies
5	interfaces for third-party applications	ARDIT users	needs to integrate the semantic service module	interface	a set of REST APIs	allow users of the ARDIT platform to query the system with natural language questions)

3.2.2 Methods for natural language processing

3.2.2.1 Statistical methods

Statistical methods are mathematical formulae, models, and strategies used to analyse raw research data statistically. Statistical approaches collect information from raw data and give several tools for evaluating the robustness of research outcomes. Much NLP research has depended largely on machine learning since the so-called "statistical revolution" [12] in the late 1980s and mid-1990s. Instead of utilising statistical inference, the machine-learning paradigm allows for the automatic learning of such rules through the examination of huge corpora (a collection of documents, sometimes with human or computer annotations) of typical real-world cases. NLP challenges have been used to test a variety of machine-learning techniques. From the input data, these algorithms create a huge number of "features." However, statistical models, which make soft, probabilistic judgements by assigning real-valued weights to each input attribute, have become increasingly popular (complex-valued embeddings, and neural networks, in general, have also been proposed, for e.g. speech). When such models are used as part of a broader system, they have the advantage of being able to describe the relative certainty of many distinct possible responses rather than just one, which leads to more trustworthy conclusions. Some of the first machine learning algorithms, such as decision trees, generated systems of hard "if-then" rules that were comparable to handwritten rules. Part-of-speech tagging, on the other hand, brought hidden Markov models to natural language processing, and research has increasingly concentrated on statistical models, which make probabilistic judgments and assign real-valued weights to the input data elements. Cache language models, which are now used by many speech recognition systems, are an example of statistical models. When given new input, especially information that contains mistakes (as is quite typical for real-world data), such models are more resilient, and they yield more accurate results when incorporated into a bigger system with several subtasks. Statistical approaches in NLP research have been mostly superseded by neural networks since the neural revolution. They are, nonetheless, still significant in situations where statistical interpretability and openness are necessary.

3.2.2.2 Neural networks

Statistical approaches have the disadvantage of requiring extensive feature engineering. The area has mainly abandoned statistical methodologies in favour of machine learning using neural networks. Using word embeddings to capture semantic features of words and increasing end-to-end learning of a higher-level task (e.g., question responding) rather than depending on a pipeline of discrete intermediary tasks are two popular strategies (e.g., part-of-speech tagging and dependency parsing). This transition has resulted in significant modifications in the design of NLP systems, to the point that deep neural network-based techniques may be considered a new paradigm separate from statistical natural language processing. For example, the term neural

machine translation emphasizes that deep learning-based approaches to machine translation directly learn sequence-to-sequence transformations, obviating the need for intermediate steps like word alignment and language modelling that were previously required in statistical machine translation.

3.2.3 Tokenization

3.2.3.1 Introduction

Tokenization is a way of separating a piece of text into smaller units called tokens. This procedure is used to feed the neural network model a set of numerical inputs it can understand (a deep learning model cannot recognise simple word strings); indeed, after the tokenization phase, the tokens are translated into numbers using a predefined lookup table that maps each token to a numerical ID. Tokenization can be broadly classified into 3 types: word, character, and subword (n-gram characters) tokenization. Consider the sentence: "Never give up". The most common way of forming tokens is based on space. Assuming space as a delimiter, the tokenization of the sentence results in 3 tokens – Never-give-up. As each token is a word, it becomes an example of Word tokenization. Similarly, tokens can be either characters or subwords. If we take, for example, the word "smarter", the corresponding character tokens extracted from it are "s-m-a-r-t-e-r", while the subword tokens would possibly be "smart-er".

3.2.3.2 Word Tokenization

Word Tokenization is the most commonly used tokenization algorithm. It splits a piece of text into individual words based on a certain delimiter. Depending upon delimiters, different word-level tokens are formed. Pretrained Word Embeddings such as Word2Vec and GloVe come under word tokenization. One of the major issues with word tokens is dealing with Out Of Vocabulary (OOV) words. OOV words refer to the new words which are encountered at testing. These new words do not exist in the vocabulary. Hence, these methods fail in handling OOV words. Other than this, it can also create a very big vocabulary because of the use of both space and punctuation tokenization procedures; this can increase memory consumption and time complexity.

3.2.3.3 Character Tokenization

Character Tokenization splits the raw text into individual characters. The logic behind this tokenization is that a language has many different words but has a fixed number of characters. This results in a very small vocabulary. It overcomes the aforementioned drawbacks of Word Tokenization. Character Tokenizers handle OOV words coherently by preserving the information of the word. It breaks down the OOV word into characters and represents the word in terms of these characters. Character tokens solve the OOV problem, but the length of the input and output sentences increases rapidly as a sentence is represented as a sequence of characters. As a result, the models that use these types of tokenizers. fail to learn meaningful input representations, therefore resulting in a performance reduction; indeed, a context-independent representation of the letter "t" is much harder to learn than a context-independent representation of the word "today". This brings us to another tokenization known as Subword Tokenization or Hybrid which is in between a Word and Character tokenization.

3.2.3.4 Subword Tokenization

The main idea of Subword (or Hybrid) Tokenization is to solve the issues faced by word-based tokenization (very large vocabulary size, large number of OOV tokens, and different meanings of very similar words) and character-based tokenization (very long sequences and less meaningful individual tokens).

The subword-based tokenization algorithms use the following principles:

- Do not split the frequently used words into smaller subwords.

- Split the rare words into smaller meaningful subwords.

For example, “boy” should not be split but “boys” should be split into “boy” and “s”. This will help the model learn that the word “boys” is formed using the word “boy” with slightly different meanings but the same root word. Subword tokenization allows the model to have a reasonable vocabulary size while being able to learn meaningful context-independent representations; it also makes it easier to process new words because the model can use decomposition to split them into known subwords. There are different types of subword tokenization algorithms. All of them rely on some form of training, typically done on the corpus the model will be trained on.

3.2.3.4.1 Byte-Pair Encoding (BPE)

Introduced in 2015 this methodology [13] deeply relies on a pre-tokenization phase to split the input data into words, which can be as simple as space tokenization or even a more complex procedure like rule-based tokenization. In this phase, it is also counted the frequency of each word in the input data. After this preliminary phase, BPE creates a base vocabulary with the symbols in the words and then proceeds to iteratively learn merge rules to form new symbols from two already existing symbols, until the vocabulary reaches the desired size (a value that must be given as a parameter before the tokenizer training). To learn the new symbols, BPE counts the frequency of each possible symbol pair and picks the one that occurs more frequently; that pair is merged and then added to the vocabulary. The GPT tokenizer is an example of a BPE tokenizer, including 40.478 entries in its vocabulary, 40.000 of which come from the merging phases.

3.2.3.4.2 Byte-level BPE

Considering all possible Unicode base characters as entries for the base vocabulary can result in a very large final vocabulary. A solution used by the GPT-2 tokenizer is to use bytes as base vocabulary; this way it will always be of size 256, but it will also contain every base character. With some additional rules to deal with punctuation, this tokenizer can tokenize every text without facing unknown symbols.

3.2.3.4.3 WordPiece

This algorithm [14] is used by models like BERT and Electra. It is mostly like the BPE algorithm, but it branches from it on the pair-choosing phase; in fact, WordPiece chooses the symbol pair that maximises the likelihood of the training data once it is added to the vocabulary. This means that each pair has a certain likelihood value, which is calculated by dividing the pair probability by the probabilities of the first symbol followed by the second symbol. WordPiece searches for the pair with the highest value of likelihood and adds it to the vocabulary. In an intuitive way, it evaluates what the vocabulary loses if a pair is inserted into it.

3.2.3.4.4 Unigram

This algorithm [15] is different from both BPE and WordPiece algorithms as it initialises its base vocabulary to a large number of symbols and progressively trims down each one of them to obtain a smaller vocabulary. It is not typically used alone as a tokenizer, but it is instead used in conjunction with SentencePiece. At each step, the Unigram algorithm defines a loss over the training data based on the current vocabulary and given language model; then, for each symbol in the vocabulary, it computes how much the overall loss would increase if the symbol was to be removed from the vocabulary.

Unigram then removes a certain percentage of symbols whose loss increase is the lowest, meaning that those symbols least affect the overall loss. This procedure is repeated until the vocabulary reaches the desired size.

3.2.3.4.5 SentencePiece

This algorithm [16] tries to solve a problem all the previously described algorithms suffer from, that is, it is always assumed that the input text uses spaces to separate words. That does not work so well for every language; in fact, languages like Japanese or Chinese do not use them at all. One solution would be to use specific pre-trained tokenizers (like the XML tokenizer does), but if a more general answer is desired, a SentencePiece algorithm can be used. SentencePiece treats the input as a raw input stream, therefore including the space in the set of characters to be used; then it makes use of the BPE or, more likely, the unigram algorithm to construct the appropriate algorithm. To make some examples, the XLNet model, the T5 model and the ALBERT model use SentencePiece. The decoding phase with these types of algorithms is very easy since all tokens can be concatenated, and the underscore symbol is replaced by a space.

3.2.4 Word Embeddings Algorithms

In Natural Language Processing (NLP), word embedding is a term used for the representation of words in real-valued vectors defined in a suitable high-dimensional vector space. These vectors encode the meaning of the word or the sentence in such a way that words that are closer in the vector space are expected to be semantically similar. Word embeddings can be obtained using a set of language modelling and feature learning techniques where words or phrases from a vocabulary are mapped to such vectors.

3.2.4.1 Description

Roughly speaking, a word embedding is a learned representation for text where words that have a similar meaning have as well a similar numerical representation. Word embeddings are in fact a class of techniques and algorithms where individual words, or some parts of them, are represented as real-valued vectors in a predefined vector space. More precisely, a possible definition of a vector embedding reads as follows: "An embedding is a mapping from a discrete object, e.g., the words of a sentence, to vectors of real numbers. The individual dimensions of these vectors have no inherent meaning. Instead, it is the overall patterns of positions and distance among vectors, i.e., some of their geometrical properties, that a machine learning algorithm takes advantage of. Word embedding algorithms, indeed, provide a dense vector representation of words that encodes information about the meaning and the context of the word."

The resulting vector space representation, given by the embedding algorithms, provides an environment where words with similar meanings are locally clustered within the space. Word Embedding algorithms are a modern approach to NLP and have played an important role in some of the latest breakthroughs in this field. The most common word embedding techniques may be listed below.

- TF – IDF (Term Frequency – Inverse Document Frequency) is an algorithm based on a statistical approach to determine the mathematical significance of words in a collection of documents
- Word2Vec uses a neural network model to learn word associations from a large corpus of text. This algorithm can detect synonymous words or suggest additional words for a partial sentence.
- GloVe, i.e., Global Vectors for word representation, is an unsupervised learning algorithm aiming to generate word embeddings by aggregating global word co-occurrence matrices from a given corpus.
- Bert is a bidirectional encoder from transformers, i.e., it is a transformer-based Deep Learning architecture for NLP.

3.2.4.2 Use cases

3.2.4.2.1 Word2vec

The Word2Vec algorithm [17] relies on a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence. As the name implies, Word2Vec represents each distinct word with a vector embedding. The vectors are defined in a common vector space so that a simple mathematical function (the cosine similarity between the vectors) may indicate the level of semantic proximity between the words represented by those vectors. In general terms, Word2Vec can be applied to any kind of input (not just words) that has intangible topical relationships. Moreover, when applied to words it can iterate over a large corpus of text to learn associations or dependencies among words. As mentioned above, Word2Vec models are good at capturing semantic relationships among words. For example, the relationship between a country and its capital, like Paris is the capital of France and Berlin is the capital of Germany. Indeed, such models are trained so that close vector embeddings in the vector space representation are assigned to semantically similar words. Word2Vec finds such semantic similarities by using the cosine similarity metric. If the cosine angle is equal to 1, then the corresponding words are overlapping. If the cosine angle is 90, the corresponding words are independent or hold no contextual similarity. For these reasons, it is best suited for performing semantic analysis, which has application in recommendation systems and knowledge discovery. Word2Vec offers two neural network-based variants: Continuous Bag of Words (CBOW) and Skip-gram. In CBOW, the neural network model takes various words as input and predicts the target word that is closely related to the context of the input words. On the other hand, the Skip-gram architecture takes one word as input and predicts its closely related context words. CBOW is quicker and finds better numerical representations for frequent words, while Skip Gram can efficiently represent rare words.

3.2.4.2.2 GloVe

GloVe stands for Global Vectors for Word Representation [18] and it is an alternative method to generate word embeddings. GloVe is an unsupervised learning algorithm for obtaining vector representations for words and extends the work of Word2Vec to capture global contextual information in a text corpus. It does this by calculating a global word-word co-occurrence matrix, whereas Word2Vec only captures the local context of words. In fact, as for design Word2Vec only considers neighbouring words to capture the context. GloVe instead considers the entire corpus and creates a large matrix that can capture the co-occurrence of words within the corpus. The GloVe technique has a simpler least square cost or error function that reduces the computational cost of training the model. The resulting word embeddings are different and improved.

3.2.4.2.3 BERT — Bidirectional Encoder Representations from Transformers

Introduced by Google in 2019, BERT belongs to a class of NLP-based language algorithms known as transformers. BERT is a massive pre-trained deeply bidirectional encoder-based transformer model. In order to alleviate the unidirectionality constraint present in other encoder models, BERT uses a “masked language model” (MLM) pre-training objective. The masked language model randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked word based only on its context. More details may be found in the original paper "Devlin, J et al. - 2018. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" [19]. The tokenizer used by BERT is the “WordPiece” tokenizer (see section above for more details). Roughly speaking, WordPiece relies on a fixed-size vocab containing all the letters of the English alphabet plus 30,000 of the most common English words and subwords; it works by decomposing the words that occur in a sentence into those words and subwords and any words that fail to decompose according to these rules will decompose into individual characters. For each separate token, BERT outputs 12 768-dimensional vectors. Each of these vectors represents a different “kind” of encoding of the words.

Although the vectors are all broadly "good", the best utilisation of BERT in terms of what to do with them (pool them all, pick one, average a subset, etcetera) depends on the application and the downstream task.

3.2.5 Seq2Seq Models

3.2.5.1 Description

Heuristically, a Sequence-to-Sequence model (often abbreviated as Seq2Seq model) is a model that takes a sequence of items (words, letters, time series, etc.) and outputs another sequence of items. This model can be used as a solution to any sequence-based problem, especially ones where the inputs and outputs have different sizes and categories. More formally, Seq2Seq models constitute a special class of Recurrent Neural Network (RNN) architectures that are typically used (but not restricted) to solve complex language problems like Machine Translation, Question Answering, Text Summarization, etc [20]. The typical architecture used to build Seq2Seq models is the so-called encoder-decoder architecture. As outlined by its name, this architecture is composed of two main components: an encoder and a decoder. The encoder may be some kind of RNN or transformer-based model, for instance, a BERT-like transformer, and its purpose is to capture the context of the input sequence in the form of a hidden state vector. The decoder component is also an RNN or a Transformer model whose initial state is initialised by the hidden state vector computed by the encoder. This initial state is then processed and used internally by the decoder to produce the output sequence in accordance with the downstream task.

3.2.5.2 Encoder/Decoder architecture

Encoder :

As already mentioned, according to the latest advancements in the NLP field, an encoder can rely on complex recurrent or convolutional neural network or on a Transformer architecture that is based solely on suitable attention mechanisms (see for instance the original paper of Vaswani A. et al. - 2017 "Attention is all you need [21]", where the Transformer architecture was first introduced). The main purpose of the encoder is to read the input sequence and summarise the semantic information in a so-called internal state vector or context vector (in the case of Long short-term memory (LSTM) networks, these are called the hidden state and cell state vectors). Typically, only these state vectors are preserved whereas the outputs of the encoder are discarded. This context vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.

Decoder :

As the encoder, the decoder may be some kind of recurrent or convolutional network, e.g., an LSTM, or a Transformer-based model, like GPT. In both cases, the initial states of the decoder are initialised to the final states of the encoder component, i.e., the context vector of the encoder's final cell is input to the first cell of the decoder network. Using these initial states, the decoder starts generating the output sequence, and these outputs are also taken into consideration for future outputs.

Overall Encoder-Decoder Architecture

- During inference, one word is generated at a time.
- The initial states of the decoder are set to the final states of the encoder.
- The initial input to the decoder is always the START token.
- At each time step, the states of the decoder are preserved and set them as initial states for the next time step.
- At each time step, the predicted output is fed as input in the next time step.

- The loop is broken when the decoder predicts the END token.

3.2.5.3 Fine-tuning of a pre-trained model

A pre-trained model is a model created by someone else to solve a problem like the one the user wants to solve. Instead of building a model from scratch, one can make use of the already trained model as a starting point. To make a real-life example, a teacher has years of experience in the particular topic s/he teaches. With all this accumulated information, the lectures that students get are a concise and brief overview of the topic. It can be seen then as a "transfer" of information from the learned to a novice. A neural network is trained on some data. This network gains knowledge from this data, which is compiled as "weights" of the network. These weights can be extracted and then transferred to any other neural network. Instead of training the other neural network from scratch, the learned features are "transferred". Fine-tuning is seen as a procedure consisting of making small adjustments to achieve the desired output. In deep learning, it involves using weights of a previous algorithm for programming another similar learning process. In a neural network, weights connect each neuron in one layer to every other neuron in the next layer. Frameworks like Keras, Torch and TensorFlow allow the user to fine-tune pre-trained models in a simple and clear way. The tweaks that fine-tuning executes are only helpful if the dataset of the new problem is, in some way, similar to one of the existing models. For example, a network that has been used to recognise cars can be fine-tuned to also recognise trucks, as it already knows concepts like doors, lights, windshields, etc. Fine-tuning is mostly helpful in reducing or even totally removing both time complexity and resource usage of training a model from scratch since the pre-trained model already has knowledge of the common topic the two models are based. Another advantage emerges when the data available for a new model is limited and training it can prove to be a problem. With fine-tuning, most of the missing data can be incorporated from previous models, making the training process much easier.

In neural network terms, fine-tuning requires the following steps:

- Retrieve the pre-trained model.
- Remove its output layer, as it was programmed for tasks specific to that model.
- Depending on the degree of similarity of the two models, it can be useful to add or remove particular network layers.
- Freeze the layers, so that their weights will not update anymore during the following training phase.
- Train the model on the new data, changing the input layer.

The output layer learns by itself, during the whole training phase, to display the result intended for the new neural network.

3.2.5.4 Models for inference

Model inference (also called *running the model in production*) is a process in which a machine learning algorithm is given some data and calculates and then outputs some value, like a numerical score. A model used like this is also usually referred to as artificial intelligence since it is performing functions similar to human thinking. A model lifecycle can be broken into two main parts:

- A training phase, in which the model is created (or, in ML terms, trained) by running data into it.
- An action phase, which is the inference described above. Here, the model is given live data and it has to produce a meaningful output.

The processing the model performs is known as "scoring"; consequently, the output is typically called "score". It has to be remembered that the model only understands numerical values,

therefore there has to be a middleware both on the input and the output side of the model to transform these values into a human-readable form (and vice versa). Therefore, model inference lets the user accomplish many different tasks:

- Text related tasks like classification, data extraction, summarisation and translation.
- Image related tasks like segmentation and object detection.
- Audio related tasks like speech recognition.
- Multimodal tasks like optical character recognition, visual question answering, and table question answering.

3.2.5.5 Use cases

3.2.5.6 Google T5 Model

T5 is an extremely large new neural network model that is trained on a mixture of unlabelled text and labelled data from popular NLP tasks, then fine-tuned individually for each of the tasks that the authors aim to solve. It works quite well, setting the state of the art on many of the most prominent text classification tasks for English, and several additional question-answering and summarisation tasks as well. The most obvious new idea behind this work is that it is a text-to-text model: during training, the model is asked to produce new text as an output, even for training tasks that would normally be modelled as classification and regression tasks with much simpler kinds of output. However, this idea seems to have been chosen out of engineering convenience, and there is no evidence that it is necessary for the results they have seen. Instead, what makes this work successful (and impressive) is that the authors took many of the best ideas from a number of recent works in NLP and did an extremely good job at rigorously testing and refining each idea as they added it. This yielded both a very well-tuned model and a lot of insights into the fine-grained design decisions that go into training a large general-purpose neural network on language.

3.2.5.6.1 Facebook BART model

Facebook's BART Transformer-based model is a Seq2Seq model that uses standard Transformer-based neural network architectures which can be seen as a generalisation of the BERT model, due to the deep bidirectional encoder it defines, and GPT, considering its autoregressive left-to-right decoder component. The encoder's attention mask is fully visible, like BERT, and the decoder's attention mask is causal, like GPT2.

Formally, BART is a denoising autoencoder built with a sequence-to-sequence model [22]. Pre-Training consists of two steps.

- Text is corrupted with an arbitrary noising function.
- The Seq2Seq architecture learns how to reconstruct the original text.

BART has shown to be very effective when fine-tuned for text generation downstream tasks, but it yields good results also for text comprehension tasks.

3.3 Technologies for semantic and natural language processing

3.3.1 Apache Jena

Apache Jena is an open-source Semantic Web framework for Java. This technology provides an Application Programming Interface (API) to extract data from and write to Resource Description Framework (RDF) graphs. The graphs are represented as an abstract model, which can be sourced with data from files, databases, Uniform Resource Locators (URLs), or a combination of

these. A model can also be queried through SPARQL Protocol and RDF Query Language (SPARQL), a semantic query language for databases, capable of retrieving and manipulating data stored in RDF format. It provides a vast set of tools and libraries to develop semantic web and linked-data apps. The framework includes:

- An API for reading, processing and writing RDF (Resource Description Framework) data in different formats (XML and Turtle are the main ones).
- An API for handling ontologies in OWL (Ontology Web Language) and RDFS (Resource Description Framework Schema) formats.
- An engine for reasoning with RDF and OWL data.
- Tools for storing RDFS on disk.
- A query engine based on SPARQL, which is a semantic query language capable of retrieving and manipulating data stored in RDF (or similar) formats.
- Servers to upload data to other applications through different protocols (being SPARQL the main one).

3.3.2 Haystack

Haystack is an open-source framework for building search systems that work intelligently over large document collections. Recent advances in NLP have enabled the application of question answering, retrieval and summarisation to real-world settings and Haystack is designed to be the bridge between research and industry. The main use cases of Haystack are:

- Question Answering: ask questions in natural language and find granular answers in your documents.
- Semantic Search: retrieve documents according to the meaning of the query, not its keywords.
- Summarisation: ask a generic question and get summaries of the most relevant documents retrieved.
- Question Generation: takes a document as input and returns generated questions that the document can answer.

3.3.3 Huggingface

HuggingFace Transformer models provide an easy-to-use implementation of some of the best performing models in natural language processing. Transformer models are the current state-of-the-art (SOTA) in several NLP tasks such as text classification, text generation, text summarisation, and question answering. The original Transformer is based on an encoder-decoder architecture and is a classic sequence-to-sequence model. The model's input and output are in the form of a sequence (text), and the encoder learns a high-dimensional representation of the input, which is then mapped to the output by the decoder. This architecture introduced a new form of learning for language-related tasks and, thus, the models spawned from it achieve outstanding results overtaking the existing deep neural network-based methods. Since the inception of the vanilla Transformer, several recent models inspired by the Transformer used the architecture to improve the benchmark of NLP tasks. Transformer models are first pre-trained on a large text corpus (such as BookCorpus or Wikipedia). This pretraining makes sure that the model "understands language" and has a decent starting point to learn how to perform further tasks. Hence, after this step, only a language model is obtained. The ability of the model to understand language is highly significant since it will determine how well the model can be further trained for something like text classification or text summarisation.

3.3.4 Graph DB

Ontotext GraphDB is a graph database and knowledge discovery tool compliant with RDF and SPARQL and available as a high-availability cluster. As for a typical graph DB, ontologies are an important input for the databases. The underlying idea is a semantic repository. GraphDB is built on top of RDF4J architecture implemented through RDF4J's Storage and Inference Layer. The architecture is made of three main components:

- Workbench is a web-based administration tool. The user interface is based on RDF4J Workbench Web Application
- Engine consists of a query optimizer, reasoner, storage and plugin manager. The reasoner in GraphDB is Forward chaining with the goal of total materialisation. The plugin manager supports user-defined indexes and can be configured dynamically during run-time. These include:
    - RDF Rank, which is an algorithm that identifies the most relevant entities
    - GeoSPARQL, which is the standard for geographically linked data. The plugin is able to convert between coordinate reference systems into the default, which OGC specifies as CRS84 format
    - Lucene, which supports full-text search capabilities. This provides a variety of indexing options and the ability to simultaneously use multiple, differently configured indexes in the same query using Apache Lucene, a high-performance, full-featured text search engine
- Connectors: the performance of searches such as full-text search and faceted search can be vastly improved via the connectors by enabling the implementation by an external component or service. GraphDB has a connector for both well-known open-source search engines, Solr and Elasticsearch. There is also a connector enabling MongoDB integration, providing scalability and performance advantages.

According to the Ontotext, Graph DB supports:

- GraphDB uses RDF4J as a library, utilising its APIs for storage and querying.
- It supports the GraphQL, SPARQL and SeRQL languages and RDF (e.g., RDF/XML, N3, Turtle) serialisation formats.
- OWL 2 RL profile is fully supported and QL partially.
- It integrates OpenRefine for the ingestion of tabular data and provides semantic similarity search at the document level.

Interfaces

An interface is how a person or thing interacts with another thing. It has an initiator that requests an action and the responder – a device, application or appliance – executes the response. It could be as simple as the control panel on a piece of equipment – changing settings on the washing machine and starting it. At a high level there are 3 types of interfaces that can be implemented:

- **Hardware interface -** It refers to those that include all those devices that we use to enter and process data on the computer. As an example of this type of interface, there are the mouse, keyboard and screen of a PC.
- **Software interface -** They are those interfaces developed to be able to deliver information to the user, about the processes and control tools, that is to say, that the user can see this data reflected on the screen. The type of software interface is the one used by programs and the operating system so that actions and processes are understandable by the user.

- **HSI or mixed interface** -Software interface types: It is a type of interface made up of a software part and a hardware part, and the term HSI is used to refer to both the configuration and functionality of SoC (system-on-chip) peripherals and how they interact with CPUs. For this reason, this type of interface generates adequate communication between users and computers. You could say that it is the layer in which the software communicates with the hardware. An example of this type of interface would be those processors or different programmable peripherals, which are linked through a software interface and an interconnection network of chips.

In the development of the Semantic Service Module dedicated software interfaces will be developed.

3.3.5 APIs

A connection between computers or computer programs is known as an application programming interface (API). It is a form of software interface that provides a service to other programs. An API specification is a document or standard that explains how to create or utilize a connection or interface. An API is implemented or exposed by a computer system that fulfils this standard. The word API can be used to refer to either the specification or the implementation. An application programming interface, unlike a user interface, connects computers or pieces of software to one other. It is not intended for usage by anybody other than a computer programmer who is incorporating it into the software. An API is made up of many pieces that serve as tools or services for programmers. A program or programmer who utilises one of these components is said to invoke that section of the API. Subroutines, methods, requests, and endpoints are all terms used to describe the API calls. These calls are defined by an API standard, which details how to use or implement them. One of the goals of APIs is to conceal the internal workings of a system, exposing just the pieces that a programmer would use and keeping them constant even if the internal workings change. The word API is typically used to refer to web APIs, which allow communication between computers connected by the internet. An API might be custom-built for a specific pair of systems or it can be a shared standard allowing interoperability among multiple systems. Programming languages, software libraries, computer operating systems, and computer hardware all have APIs. APIs date back to the 1940s, although the phrase did not become popular until the 1960s and 1970s.

3.3.6 REST Pattern

REST (representational state transfer) is a software architectural style that was developed to assist the design and development of the World Wide Web's architecture. REST establishes a set of guidelines for how an Internet-scale distributed hypermedia system, such as the Web, should be designed. The REST architectural style emphasises scalability of component interactions, standardised interfaces, component deployment independence, and the establishment of a layered architecture to enable caching components to minimise user-perceived latency, enforce security, and encapsulate old systems. REST is a generally established set of rules for establishing stateless, dependable online APIs that have been used throughout the IT industry. In fact, RESTful is an informal term for a web API that follows the REST requirements. RESTful online APIs are primarily built on HTTP protocols for accessing resources via URL-encoded parameters and data transmission using JSON or XML. On the World Wide Web, "web resources" were first defined as URL-identified documents or files. Today's definition is far more broad and abstract, encompassing everything, entity, or activity that can be recognised, named, addressed, handled, or performed in any way on the Internet. Requests to a resource's URI in a RESTful Web service elicit a response with a payload structured in HTML, XML, JSON, or another format. The answer might, for example, certify that the resource status has changed. Hypertext links to similar sites can also be included in the response. HTTP is the most often used protocol for these requests and

answers. It supports GET, POST, PUT, and DELETE actions (HTTP methods). RESTful systems (Figure 4) strive for quick speed, dependability, and the potential to grow by reusing components that can be controlled and updated without impacting the system as a whole, even while it is operating, utilising a stateless protocol and standard procedures. REST aims to improve scalability, simplicity, modifiability, visibility, portability, and dependability. This is accomplished by adhering to REST concepts including client–server design, statelessness, cacheability, layered systems, code-on-demand support, and a unified interface. For a system to be labelled as RESTful, certain principles must be followed.

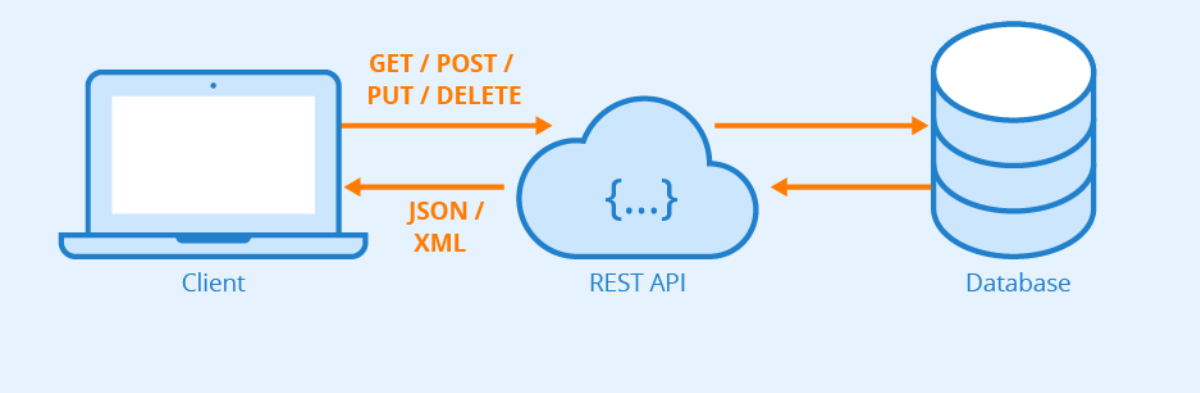

Figure 4: REST Patter

(source: https://i0.wp.com/mycodetips.com/wp-content/uploads/2019/12/Rest-API.png?resize=800%2C604&ssl=1)

3.4 Microservices

Monolithic architecture is a huge container that holds all of an application's software components: user interface, business layer, and data interface. This has various drawbacks, including inflexibility, lack of dependability, scalability difficulties, and delayed growth. Microservices architecture was intended to avoid these problems. Microservices are a sort of cloud application architecture. Because of the development of the internet and the pervasiveness of mobile computing in recent years, application developers have been forced to focus on designing lightweight, self-contained components. Developers must be able to deploy apps rapidly and make changes to them without having to redeploy them completely.

As a result, a new development model known as "microservices" has emerged. Microservices architecture, often known as microservices, is an architectural strategy or style of application development that entails breaking down big programs into smaller, functional pieces that may operate and communicate independently via APIs. Although they are autonomous components, the application can use any number of these microservices to operate together and achieve the necessary goals. Microservices-based applications are deconstructed and divided into smaller parts (microservices) that the client may access over an API gateway. Microservices provide faster development, simpler issue discovery and resolution, easier maintenance, flexibility, and improved availability and scalability by breaking down a program into smaller pieces.

Microservices design focuses on categorising programs that are otherwise huge and cumbersome. Each microservice is tailored to a certain component or function of an application, such as logging, data search, and so on. Several of these microservices are combined to make a single, efficient application. This user-friendly, functional split of a program has various advantages. The user interface can be used by the client to make requests. At the same time, one or more microservices are commissioned to accomplish the desired operation via the API

gateway. As a result, ever more complicated problems requiring the use of several microservices may be handled quite quickly.

Using microservices architecture in application development can be highly beneficial. Below are some of its key benefits.

- **Requires less development effort to scale up** - Due to the independent nature of microservices, smaller development teams can parallelly work on different components to update existing functionalities. This makes it significantly easier to not only identify hot services and scale independently from the rest of the application but also improve the application as a whole.
- **Can be deployed independently** - Each microservice constituting an application needs to be full-stack. This enables microservices to be deployed independently at any point. Since the microservices are granular in nature, it is easy for development teams to work on one microservice to fix errors and then redeploy it without redeploying the entire application.
- **Microservices offer improved fault isolation** - In monolithic applications, the failure of even a small component of the overall application can make it inaccessible as a whole. In some cases, determining the error could also be tedious. With microservices, isolating the problem-causing component is easy since the entire application is divided into standalone, fully functional software units. If errors occur, other non-related units will still continue to function.
- **No dependence on one Tech Stack** - A technology stack refers to a set of programming languages, databases, front-end and back-end tools, frameworks, and other such components used by the developers to build an application. With microservices, developers have the freedom to pick a technology stack that is best suited for one particular microservice and its functions. They have absolute control instead of having to opt for one standardised tech stack that encompasses all of an application's functions.

4 Application Architecture

4.1 Overview of the architecture

Following the approach based on machine learning, the Semantic Service module is implemented using a Seq2Seq model to translate natural language questions into SPARQL queries. SPARQL queries will be run on semantic graph databases (also called RDF triplestores).

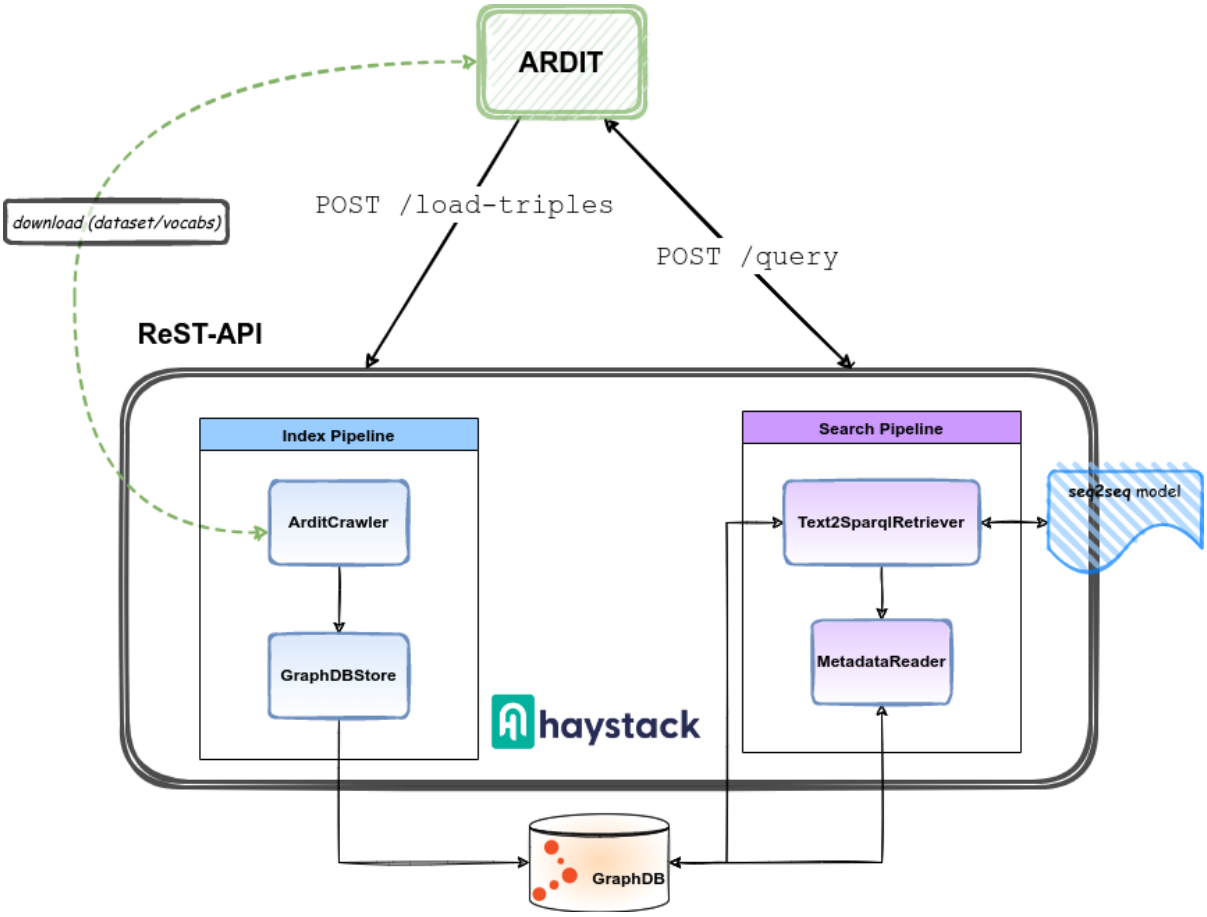

Figure 5: ARDIT Semantic Services High level architecture

The main components of the Semantic Service module are:

- **GraphDB**: the storage level, a Semantic Graph Database compliant with W3C Standards.
- **Haystack**: an open-source framework for building search systems based on NLP.
- **A Transformer model**: a trained Seq2Seq model for converting natural language questions to SPARQL queries.

4.2 Modules

4.2.1 GraphDB

GraphDB is a Semantic Graph Database (also called RDF triplestores), compliant with W3C Standards. GraphDB is a family of highly efficient, robust, and scalable RDF databases. It streamlines the load and use of linked data cloud datasets, as well as your own resources. For easy use and compatibility with the industry standards, GraphDB implements the RDF4J framework interfaces and the W3C SPARQL Protocol specification and supports all RDF serialization formats. The database is the preferred choice of both small independent developers and big enterprise organisations because of its community and commercial support, as well as excellent enterprise features such as cluster support and integration with external high-performance search applications such as Lucene, Solr, and Elasticsearch.

4.2.1.1 Installation

To use the free version of GraphDB for AGRICORE, the Docker image deployed on this project has been used. The image comes with a repository that contains the default graph:

- Triples related to the ontology.
- Triples related to AGRICORE datasets.

To run the GraphDB instance, execute the command: `docker run --name agricore-graphdb -p 8200:8200`

Then, the instance can be accessed through a browser at http://localhost:8200.

4.2.1.2 Import graphs for vocabularies

Triples related to vocabularies (public and private) could be manually loaded using the web application:

- Navigate to http://localhost:8200/import#user.
- Click on `Upload RDF file` and select a file that contains triples.
- The file imported is shown in the list below.
- Click on the Import button next to the file.
- Add the `Base IRI`, the `Target graphs` and select the radio `Named graph`.

Alternatively, the `GraphDBAPIWrapper` python class that wraps GraphDB API can be used to create named graphs in the GraphDB repository. The *GraphDBAPIWrapper* is in the project repository.

To use this facility this is the snippet of code.

```
graphapi = GraphDBAPIWrapper(index="ardit", port=8200)
graphapi.upload_triples(index="ardit", path=Path("./ardit-vocabs/dataset-types.ttl"),
graph_name="https://agricore-project.eu/ontology/agricore-dcatap/subjects")
```

4.2.2 Haystack

Haystack is an open-source framework for building search systems that work over large document collections. Recently, NLP has been applied to the framework for implementing question answering, retrieval and summarisation. Haystack is geared towards creating customisable and production-ready research pipelines. There are three different levels where you can interact with components in Haystack:

- **Nodes**, which are the core components that process and route incoming text. Nodes can be chained together using a Pipeline and they function like building blocks that can be easily switched out for each other. A Node takes the output of the previous Node (or Nodes) as input. When nodes are added to a Pipeline and call Pipeline.run(), it calls each Node's run() method in the predefined sequence.
- A **Pipeline** is a way to stack components (nodes) to achieve a result. When adding Nodes to a Pipeline, the user can define how data flows through the system and which Nodes perform their processing step when. On top of simplifying data flow logic, this also allows for complex routing options such as those involving decision nodes. Pipelines could be defined through python scripts or through YAML files.
- Haystack can be deployed as a **REST API**. Using Haystack through an API can benefit developers who are looking to deploy the Question Answering functionality in their projects, which could be on web or mobile apps. The API uses a web server to receive HTTP requests and pass them on to a running instance of Haystack for processing, before returning Haystack results as an HTTP response. Existing API endpoints are defined using FastAPI route methods [23]. Custom endpoints can be added to the Haystack API by defining new API endpoints using the FastAPI methods.

4.2.2.1 Index pipeline

The indexing pipeline is responsible for downloading vocabularies and datasets from ARDIT and inserting them into GraphDB in such a way as to comply with the rules of the SPARQL query models prepared to train the BART model. Two new classes (Nodes) and a new ReST API will be developed to handle this use case.

- `ArditCrawler`: the class is responsible for downloading datasets and vocabularies from ARDIT and storing them on the local file system;
- `GraphDBStore`: the class stores on GraphDB the datasets in the main graph of the repository and the vocabularies in a separate graph referenced by the URI of the vocabulary.

To use the indexing pipeline, Haystack's ReST API will be extended via the `LoadTriples` interface.

4.2.2.2 Search pipeline

The search pipeline is responsible for converting a query expressed in natural language into a SPARQL query and using it to search the datasets contained in GraphDB. Haystack has already a Node (`Text2SparqlRetriever`) that implement this behaviour and it simply needs a trained BART model. The next Node (`MetadataRetriever`) in the pipeline is simply a retriever that can search for metadata to associate with the found datasets before returning the results to ARDIT. Haystack already has a ReST API for using search pipelines.

4.2.3 AI Transformer model

4.2.3.1 Creating the dataset for training

In order to build the dataset for training our machine learning model, some questions were identified. However, in order to obtain the critical mass of data for the training of the model, the dataset was extended by the means of interpolation and paraphrasis methods. At this point, the questions had to be translated into SPARQL queries (see chapter "SPARQL queries examples" of this document). As a result, three datasets were generated:

- The **training** dataset, for the training of the AI transformer model.
- The **validation** dataset, which is used during the training for the validation of partial and final results and monitoring of the loss function.

- The **test** dataset, which is used as an unbiased data source for final testing purposes to ensure that the model is able to generalise.

4.2.3.2 Paraphrasis

Using an AI model developed by Google, the questions were paraphrased [24]. These are some examples:

Ex.1

Original Question: Find all datasets that cover the theme environment

Paraphrased Questions :

- Find all datasets that cover the theme environment for desktops. This is the largest dataset of all time.
- Find datasets that cover the theme environment. This dataset has been created in MATLAB.
- Find all datasets that cover the theme environment. What is the theme environment?
- Find all datasets covering the theme environment.
- How do I find the datasets of the theme environment for Free!
- What are the datasets that cover the theme environment.?
- Find all datasets that cover the theme environment in the theme environment in csv.php. Find all datasets that cover the theme environment in csv.php.
- Find all datasets that cover the theme environment. Theme environment.
- Find the datasets that cover the theme environment. and are used to develop and validate the theme environment.
- Find datasets that cover the theme environment.
- What datasets cover theme environment in Linux?
- What are the datasets that cover the theme environment?
- Get all datasets covering theme environment.
- Where can I find all datasets that cover the theme environment. Specifically, if I cite the theme environment as the most readable edgware, I can find all the datasets that cover it too.
- Find all datasets that cover the theme environment.

Ex.2

Original Question: Which are the datasets that cover the subject of soil pollution?

Paraphrased Questions :

- What are the datasets that cover soil pollution?
- What are the datasets that cover subject soil pollution?
- Which datasets cover soil pollution?
- What are the datasets that cover the subject soil pollution?
- Which are the datasets that cover subject soil pollution?
- Which is the dataset that covers the subject soil pollution?
- Which are the datasets that cover soil pollution?

Ex.3

Original Question: Find all datasets that deal with the subject of soil pollution in the time span from 2000 to 2020

Paraphrased Questions :

- What is the dataset for soil pollution data and how does it compare with other data sets in the time span from 2000 to 2020?
- Find all datasets that deal with the subject soil pollution in the time span from 2000 to 2020.
- Find all datasets that deal with subject soil pollution in the time span from 2000 to 2020 in the datasets of the DNR.
- Where can I find a huge dataset for soil pollution in the time span of 2000 to 2020?
- How to find all datasets that deal with the subject soil pollution in the time span from 2000 to 2020 in a broader way?
- Find all datasets that deal with the subject soil pollution in the time span from 2000 to 2020 in the US and Canada.
- How many datasets are available to us for soil pollution data in the time span from 2000 to 2020?
- What is the scope of the data collected about soil pollution in the time span from 2000 to 2020?
- In order to find all datasets that deal with the subject soil pollution in the time span from 2000 to 2020. Use the following datasets to find all datasets.
- What is the best dataset about soil pollution in the time span from 2000 to 2020?
- What datasets deal with soil pollution in the time span from 2000 to 2020?
- How can I get all datasets for soil pollution in the time span 2000 to 2020 in the year 2000?

Ex.4

Original Question: Which are all environmental datasets that deals with subject soil pollution?

Paraphrased Questions :

- What are the environmental datasets that deal with subject soil pollution?
- What are environmental datasets that deals with subject soil pollution?
- Which are the environmental datasets that deals with subject soil pollution?
- What is the environmental dataset that deals with subject soil pollution?
- Which is a symbiosis of soil pollution?
- Which are the environmental datasets that deal with subject soil pollution?
- Which are all environmental datasets that deal with subject soil pollution?
- What are all environmental datasets that deals with subject soil pollution?
- What are the all environmental datasets that deals with subject soil pollution?

Ex.5

Original Question: Which are the datasets with geographical coverage in France and time span 2000 - 2020?

Paraphrased Questions :

- Which datasets have the geographical coverage France and time span 2000 - 2020?
- What are the datasets with geographic coverage France and time span 2000 - 2020?
- Which datasets with geographical coverage France and time span 2000 - 2020?
- Which datasets with geographical coverage France and time span 2000-2020?
- Which datasets have the geographic coverage of France and time span 2000 - 2020?
- Which datasets have the geographical coverage France and time span 2000-2020?
- What are the datasets with geographical coverage France and time span 2000 - 2020?

Ex.6

Original Question: Find all datasets with Europe geographical coverage

Paraphrased Questions :

- Find all datasets with Europe geographical coverage.
- What are all datasets with Europe geographic coverage?
- In this page you can find all datasets with Europe geographical coverage.
- Find all datasets with European geographical coverage.
- Where is the dataset containing data for the Europe Geocaching area. Get full datasets with Europe geocaching.
- Find all datasets with Europe geographic coverage.
- Where can I find datasets with European coverage and their respective regions in Google Earth?
- Find all datasets with European geographic coverage.
- What datasets have European geographical coverage?
- Is there any datasets with geographic coverage for Europe?
- In this section find all datasets with Europe geographical coverage (Singles, Grid, Maps, Tables, Maps and More).
- Which datasets have European geographical coverage?
- What is the best dataset with European geographical coverage

4.2.3.3 Interpolation

Another method utilized to extend the dataset is interpolation as stated in the previous chapter. This methodology required the development of a data retrieval script for the extraction of the labels of different concepts on which the given question was based from the AGRICORE knowledge graph.

```
PREFIX ogcgs: <http://www.opengis.net/ont/geosparql#>
SELECT distinct ?label
WHERE {
    SERVICE <http://publications.europa.eu/webapi/rdf/sparql>
    {
       SELECT  ?label
       WHERE
       {
          graph <http://publications.europa.eu/resource/authority/data-theme> {
                ?subject  ?predicate  ?object .
          }
          ?object a skos:Concept .
          ?subject skos:prefLabel ?label .
          FILTER (lang(?label) = 'en')
       }
    }
}
```

The above script extracts this list of labels:

	label
1	"Agriculture, fisheries, forestry and food"@en
2	"Economy and finance"@en
3	"Education, culture and sport"@en
4	"Energy"@en
5	"Environment"@en
6	"Government and public sector"@en
7	"Health"@en
8	"International issues"@en
9	"Justice, legal system and public safety"@en
10	"Regions and cities"@en
11	"Provisional data"@en
12	"Population and society"@en
13	"Science and technology"@en
14	"Transport"@en

Figure 6: Labels extracted from AGRICORE knowledge graph

Next, a specific Python script was developed to generate multiple questions from a given template by replacing placeholders with the extracted labels.

```
def interpolate(self, template: str, label: dict) -> str:
        pattern = re.compile("|".join(label.keys()))
        q = pattern.sub(lambda m: label[re.escape(m.group(0))], template)
        return q

def get_random_label(self, labels: dict) -> dict:
        idx = -1
        random_values = {}
        for k, values in labels.items():
            if k == '<end_date>': continue
            elif k == '<start_date>':
                idx = choice(list(range(len(values))))
                random_values[k] = values[idx]
                random_values['<end_date>'] = labels['<end_date>'][idx]
            else:
                random_values[k] = choice(labels[k])
        return random_values

def label_interpolator(self, labels: Dict, template: str, n_iter = 5000) -> List:
        return [self.interpolate(template, self.get_random_label(labels)) for _ in range(n_iter)]
```

4.2.3.4 Fine-tuning of HuggingFace pre-trained models

The AI model used relies on the google T5 transformer model, one implementation of which is available on the HuggingFace repository. As for the training environment, Google Vertex AI Workbench was used. Vertex AI is a single development environment for the entire data science workflow. It can set up an end-to-end notebook-based production environment. Notebooks, in this case, are web applications for creating and sharing computational documents based on Python interpreters. Notebooks are particularly suitable for machine learning pipelines since they allow the compartmentation of the different steps of a typical machine learning flow.

For the AGRICORE project the following steps have been implemented:

1. Download one of the T5 based model available on the HuggingFace repository that was already pre-trained for the downstream task of translating text general-purpose SPARQL queries.
2. First tuning of the model with the lc_quad dataset, a dataset build from Wikidata and DBpedia knowledge graphs that links text (in question form) to SPARQL queries.
3. Second tuning of the model resulted in step 2 with our **training** dataset based on the AGRICORE knowledge graph (code below)

```
model_checkpoint = "./agricore_models/test_model_T5_lcquad"print_gpu_utilization()# Get the T5 tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)# Get basic T5 config and Bart model
architecture model = AutoModelForSeq2SeqLM.from_pretrained(model_checkpoint).to("cuda")

data_files = {"train": "agricore_train_text2sparql.json", "test": "agricore_eval_text2sparql.json"}
agricore_dataset = load_dataset("STAM/agricore", use_auth_token=True, data_files=data_files)

ts_train_dataset = agricore_dataset['train']
ts_eval_dataset = agricore_dataset['test']

args = Seq2SeqTrainingArguments(
   f"TEST",
   evaluation_strategy = "epoch",
   learning_rate=2e-5,
   per_device_train_batch_size=2,
   per_device_eval_batch_size=2,
   gradient_accumulation_steps=4,
   fp16=True,
   weight_decay=0.01,
   save_total_limit=3,
   num_train_epochs=3,
   predict_with_generate=True)

trainer = Seq2SeqTrainer(
   model=model,
   args=args,
   train_dataset=ts_train_dataset,
   eval_dataset=ts_eval_dataset, # cross validation
   tokenizer=tokenizer,
   data_collator=data_collator)

trainer.train()
```

4.2.3.5 Results

As a result of the training phase, the AI machine learning model is able to translate a text to SPARQL queries based on the AGRICORE knowledge graph. For example:

The question: **"Which datasets contain the dataset variable u111 - agriculture (excluding fallow land and kitchen gardens) and have temporal extent 2006 - 2018?"** is translated by the model into the following SPARQL query:

```
select distinct ?dataset where [ ?dataset a agricore:Dataset . ?dataset
agricore:hasDatasetVariables ?datasetVariable . ?datasetVariable agricore:variableName
?variableName . [ FILTER ( regex(str(?variableName), 'u111 - agriculture (excluding
fallow land and kitchen gardens)', 'i') ) ] union [ ?datasetVariable
agricore:referenceValues ?refValues . FILTER ( regex(str(?refValues), 'u111 - agriculture
(excluding fallow land and kitchen gardens)', 'i') ) ] BIND (IF(STRLEN(str(2006)) > 0,
2006, 'NaN') AS ?startParam) BIND (IF(STRLEN(str(2018)) > 0, 2018, 'NaN') AS ?endParam)
OPTIONAL [ ?dataset terms:temporal ?dateRange . ?dateRange a terms:PeriodOfTime .
?dateRange dcat:startDate ?startDate . ?dateRange dcat:endDate ?endDate . BIND
(IF(?startParam <= year(?startDate), year(?startDate), ?startParam) AS ?max_start_date)
BIND (IF(?endParam <= year(?endDate), ?endParam, year(?endDate)) AS ?min_end_date) FILTER
(isNumeric(?startParam) && ?max_start_date <= year(?endDate) && isNumeric(?endParam) &&
?min_end_date >= year(?startDate)) . ] FILTER ((bound(?startDate) || ?startParam ='NaN')
&& (bound(?endDate) || ?endParam = 'NaN'))]
```

The above query is completely coherent with the true values in the labelled record and the actual result is one of the datasets present in the AGRICORE knowledge graph: **https://agricore-project.eu/ontology/agricore-dcatap/StatEnvironmentDataset/LUCAS2018**

4.3 SPARQL queries examples

Some SPARQL hints for constructs used in queries are below:

- `?something` is a variable
- `FILTER` a keyword to make conditions
- `UNION` a keyword useful for concatenating solutions from two or more possibilities
- `SERVICE` a keyword to include an external SPARQL endpoint to run (sub)queries on for triples not stored locally
- `graph` a keyword to include an external named graph as triples dataset

4.3.1 Datasets by dcat:theme

To search all Agricore `?dataset` in the default graph where `?theme` is contained in `?subject` returned by subqueries.

- The first subquery uses the SPARQL endpoint of `publications.europa.eu` to retrieve the graph of the triples about data-theme. Triples were filtered using the `skos:prefLabel` of `?subject` retrieved.
- Second subquery uses a graph contained in the GraphDB instance `http://inspire.ec.europa.eu/theme` and uses only triples which type is `http://purl.org/net/provenance/ns#DataItem`. Then, they were filtered using `<http://purl.org/dc/terms/title>` property.

Note: the values used for filtering are an example of possible user input.

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX agricore: <https://agricore-project.eu/ontology/agricore-dcatap#>
PREFIX ns11: <http://publications.europa.eu/resource/authority/>
select distinct ?dataset ?theme
where {
    ?dataset a agricore:Dataset .
    ?dataset dcat:theme ?theme .
    FILTER ( ?theme IN (?subject)) .
    {
        SERVICE <http://publications.europa.eu/webapi/rdf/sparql>
```

```
        {
            SELECT  ?subject
            WHERE
            {
                graph <http://publications.europa.eu/resource/authority/data-theme> {
                    ?subject  ?predicate  ?object .
                }
                ?object a skos:Concept .
                ?subject skos:prefLabel ?label .
                FILTER ( regex(?label, "environment"@en, "i") ) .
            }
        }
    } union
    {
        select ?subject
        where
        {
            graph <http://inspire.ec.europa.eu/theme> {
                ?subject  ?predicate  ?object .
            }
            ?object a <http://purl.org/net/provenance/ns#DataItem> .
            ?object <http://purl.org/dc/terms/title> ?title .
            FILTER ( lang(?title) = "en")
            FILTER ( regex(?title, "land cover", "i") ) .
        }
    }
}
```

4.3.2 Datasets by dcat:theme or hasDatasetType or hasPurpose

Search all AGRICORE `?dataset` in the default graph where

- `?theme` could be contained in `?subject` returned by subqueries.
- `?datasetType` could be contained in `type` returned by the subquery.
- `?datasetPurpose` could be contained in `purpose` returned by the subquery.

Details:

- The first block is the same to retrieve the dataset by theme.
- Second subquery searches *subjects* in the graph `https://agricore-project.eu/ontology/agricore-dcatap/subjects` which is a private vocabulary defined by users in ARDIT. Triples retrieved were filtered by `skos:prefLabel`
- Third subquery search *dataset purposes* in the graph `https://agricore-project.eu/ontology/agricore-dcatap/DatasetPurpose` which is a private vocabulary defined by users in ARDIT. Triples retrieved were filtered by `skos:prefLabel`

Note: the values used for filtering are an example of possible user input.

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX agricore: <https://agricore-project.eu/ontology/agricore-dcatap#>
select distinct  ?dataset ?theme ?datasetPurpose ?datasetType
where {
    ?dataset a agricore:Dataset .
    # theme
    {
        ?dataset dcat:theme ?theme .
        FILTER ( ?theme IN (?subject)) .
        {
            SERVICE <http://publications.europa.eu/webapi/rdf/sparql>
            {
                SELECT  ?subject
                WHERE
                {
                    graph <http://publications.europa.eu/resource/authority/data-theme> {
                        ?subject  ?predicate  ?object .
                    }
                    ?object a skos:Concept .
                    ?subject skos:prefLabel ?label .
                    FILTER ( regex(?label, "environment"@en, "i") ) .
                }
            }
```

```
        }
        union
        {
            select ?subject
            where
            {
                graph <http://inspire.ec.europa.eu/theme> {
                    ?subject  ?predicate  ?object .
                }
                ?object a <http://purl.org/net/provenance/ns#DataItem> .
                ?object <http://purl.org/dc/terms/title> ?title .
                FILTER ( lang(?title) = "en")
                FILTER ( regex(?title, "land cover", "i") ) .
            }
        }
    }
    union
    # types
    {
        ?dataset agricore:hasDatasetType ?datasetType .
        FILTER ( ?datasetType IN (?type)) .
        {
            select ?type
            where
            {
                graph <https://agricore-project.eu/ontology/agricore-dcatap/subjects> {
                    ?subject ?predicate  ?type .
                }
                ?type a skos:Concept .
                ?type skos:prefLabel ?label .
            #    FILTER ( lang(?title) = "en")
                FILTER ( regex(?label, "land", "i") ) .
            }
        }
    }
    union
    # purpose
    {
        ?dataset agricore:hasPurpose ?datasetPurpose .
        FILTER ( ?datasetPurpose IN (?purpose)) .
        {
            select ?purpose
            where
            {
                graph <https://agricore-project.eu/ontology/agricore-dcatap/DatasetPurpose> {
                    ?subject  ?predicate  ?purpose .
                }
                ?purpose a agricore:DatasetPurpose .
                ?purpose skos:prefLabel ?label .
          #      FILTER ( lang(?title) = "en")
                FILTER ( regex(?label, "climate", "i") ) .
            }
        }
    }
}
```

4.3.3 Datasets by hasGeoCoverage or the aboves

Search all Agricore `?dataset` in the default graph where

- `?theme` could be contained in `?subject` returned by subqueries
- `?datasetType` could be contained in `type` returned by subquery
- `?datasetPurpose` could be contained in `purpose` returned by subquery
- `?geoCoverage` could be contained in `?country_subject` or `?cont_subject` returned by subquery

Details:

- The first block is the same as the previous query.
- Subquery for countries merges two graphs (countries and continents) and filters by country name or related country's continent.

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX agricore: <https://agricore-project.eu/ontology/agricore-dcatap#>
PREFIX ogcgs: <http://www.opengis.net/ont/geosparql#>
select distinct  ?dataset ?theme ?datasetPurpose ?datasetType ?geozone
where {
    ?dataset a agricore:Dataset .
    # theme
    {
        ?dataset dcat:theme ?theme .
        FILTER ( ?theme IN (?subject)) .
        {
            SERVICE <http://publications.europa.eu/webapi/rdf/sparql>
            {
                SELECT  ?subject
                WHERE
                {
                    graph <http://publications.europa.eu/resource/authority/data-theme> {
                        ?subject  ?predicate  ?object .
                    }
                    ?object a skos:Concept .
                    ?subject skos:prefLabel ?label .
                    FILTER ( regex(?label, "theme"@en, "i") ) .
                }
            }
        }
        union
        {
            select ?subject
            where
            {
                graph <http://inspire.ec.europa.eu/theme> {
                    ?subject  ?predicate  ?object .
                }
                ?object a <http://purl.org/net/provenance/ns#DataItem> .
                ?object <http://purl.org/dc/terms/title> ?title .
                FILTER ( lang(?title) = "en")
                FILTER ( regex(?title, "theme", "i") ) .
            }
        }
    }
    union
    # types
    {
        ?dataset agricore:hasDatasetType ?datasetType .
        FILTER ( ?datasetType IN (?type)) .
        {
            select ?type
            where
            {
                graph <https://agricore-project.eu/ontology/agricore-dcatap/subjects> {
                    ?subject ?predicate  ?type .
                }
                ?type a skos:Concept .
                ?type skos:prefLabel ?label .
                #    FILTER ( lang(?title) = "en")
                FILTER ( regex(?label, "subject", "i") ) .
            }
        }
    }
    union
    # purpose
    {
        ?dataset agricore:hasPurpose ?datasetPurpose .
        FILTER ( ?datasetPurpose IN (?purpose)) .
        {
            select ?purpose
            where
            {
                graph <https://agricore-project.eu/ontology/agricore-dcatap/DatasetPurpose> {
                    ?subject  ?predicate  ?purpose .
                }
                ?purpose a agricore:DatasetPurpose .
                ?purpose skos:prefLabel ?label .
                #      FILTER ( lang(?title) = "en")
                FILTER ( regex(?label, "purpose", "i") ) .
            }
        }
    }
    # geo coverage
    union {
        ?dataset agricore:hasGeoCoverage ?geozone .
        FILTER ( ?geozone IN (?country_subject) || ?geozone IN (?cont_subject)) .
        SERVICE <http://publications.europa.eu/webapi/rdf/sparql>
        {
```

```
            SELECT  ?country_subject ?cont_subject
            WHERE
            {
                graph <http://publications.europa.eu/resource/authority/country> {
                    ?country_subject  ?country_predicate  skos:Concept .
                }
                graph <http://publications.europa.eu/resource/authority/continent> {
                    ?cont_subject  ?cont_predicate skos:Concept .
                }
                ?country_subject skos:prefLabel ?country_name .
                ?country_subject ogcgs:sfWithin ?continent .
                ?continent skos:prefLabel ?continent_name
                FILTER( lang(?country_name) = "en")
                FILTER( lang(?continent_name) = "en")
                FILTER( regex(?country_name, "italy"@en, "i") || regex(?continent_name, "america"@en, "i"))
             }
         }
     }
}
```

4.3.4 Variables and reference values

- The given query helps us to extract all the distinct datasets from the AGRICORE vocabulary corresponding to: -dataset variable -variable name -refValues.
- FILTER is added to refine our result as per the requirement, and UNION is also used to concatenate the results of various conditions.
- In the given query, the filters are variableName as 'phosphorous' and refvalue as 'forestry'.

```
      ## Search in variables and in reference values
   PREFIX agricore: <https://agricore-project.eu/ontology/agricore-dcatap#>
   select distinct ?dataset ?datasetVariable ?variableName ?refValues
   where {
     ?dataset a agricore:Dataset .
     ?dataset agricore:hasDatasetVariables ?datasetVariable .
     ?datasetVariable agricore:variableName ?variableName .
     {
         FILTER ( regex(str(?variableName), "Phosphorous", "i") )
     }
     union
     {
         ?datasetVariable agricore:referenceValues ?refValues .
         FILTER ( regex(str(?refValues), "forestry", "i") )
     }
   }
```

4.3.5 Unit of analysis reference

- The given query helps us to extract all the distinct datasets from the AGRICORE vocabulary corresponding to -analysis unit -unitReference.
- And here a filter is added to unitReference, which is the unit in the main query and label 'topsoil' in the subquery.

```
   ## Search by unit of analysis reference
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX agricore: <https://agricore-project.eu/ontology/agricore-dcatap#>
select distinct ?dataset ?analysisUnit ?unitReference
where {
     ?dataset a agricore:Dataset .
     ?dataset agricore:hasAnalysisUnit ?analysisUnit .
     ?analysisUnit agricore:hasUnitReference ?unitReference .
     FILTER ( ?unitReference IN (?unit)) .
     {
         select ?unit
         where
         {
             graph <https://agricore-project.eu/ontology/agricore-dcatap/AnalysisUnitReference> {
                 ?unit ?predicate ?object .
             }
             ?unit a agricore:AnalysisUnitReference .
             ?unit skos:prefLabel ?label .
             FILTER ( regex(?label, "topsoil", "i") ) .
```

```
            }
        }
}
```

4.4 Deployment

The whole application has been released on Google Cloud Platform. Docker images of the backend and frontend are used through a Docker-Compose. Docker is a software platform that allows to quickly build, test, and deploy applications. Docker compacts software into standardised units called containers that offer everything that is needed for proper execution, including libraries, system tools, code, and runtimes. With Docker, it Is possible to deploy and scale resources for an application in any environment, while always keeping tabs on the code. Docker provides a standard way to run code. Just as the virtual machine virtualized hardware servers (i.e. eliminates the need to manage them directly), containers virtualise a server's operating system. Docker is installed on every server and provides simple commands with which to create, start, or stop containers.

Therefore, Docker containers have a number of advantages. Some of these are:

- **Isolation -** Docker allows you to create a system independent of the environment in which it is installed; knowing that your application will work smoothly on different platforms takes a lot of weight off those who design the architecture.
- **Portability -** Since all the dependencies of the application itself are in the same block (or, in technical terms, containers), it is easy to be able to transport the software from one place to another, entrusting Docker with its portability.
- **Lightness -** Unlike many systems that are installed through virtual machines, Docker offers the possibility of creating a system that is structurally placed above the basic architecture necessary to make it work; in this way, there will be no need to create several virtual machines that consume the resources of your system, but docker will take care of it.
- **Robustness -**Docker is much less demanding in terms of hardware and needs very little memory than virtual machines, thus providing efficient levels of isolation that help save not only costs but also time.

This technology allows you to deploy code faster, standardise the operation of applications, transfer code in an optimised way, and save money by improving usage

5 Extension of use in other modules of the project

The implementation of Semantic Services based on ontologies could potentially be extended to other modules of the AGRICORE project. It must be noted that the extension of such services must be supported by the extension of the relative ontology and by the training of the dedicated AI model. One of the most direct applications could be tied to the DWH developed to store the data contained in the datasets, in particular, it would be possible to generate specific queries, from natural language, on the content of the datasets themselves in order to retrieve a certain piece of information, either being a specific record of a dataset or a function applied to a range of records and datasets. On a more ambitious note, it could be even possible to suggest possible graphical representations of the query results. As an example, a user could ask the system "How much land has been covered by farms through the last 10 years in France?". In this case, the system could translate the question into a query comprehending keywords such as "GROUP", "SUM", "FILTER", "JOIN", etc. Finally, considering the syntax of the query and through clustering algorithms, the system could suggest a histogram or a line chart to visualise the data.

6 Conclusions

The deliverable 4.5 "Semantic Services for AGRICORE" provides a description of the methodology and the frameworks utilised for the development of the tool other than providing the actual details of the implementation itself. Behind the development of the Semantic Service Module of the AGRICORE project, complex and extensive research has been conducted in order to identify the most suitable and feasible approach and the tools to put it into practice. The choice of an AI-based tool, powered by the Haystack framework, appears to be, at this time, the best all-around solution. This is also confirmed by the promising results obtained.

7 References

[1] ^ SAREF. [Online]. Available: https://saref.etsi.org/

[2] ^ IFC. [Online]. Available: https://technical.buildingsmart.org/standards/ifc/ifc-formats/ifcowl/

[3] ^ DCAT-AP. [Online]. Available: https://joinup.ec.europa.eu/collection/semantic-interoperability-community-semic/solution/dcat-application-profile-data-portals-europe/release/11

[4] ^ ODOCH. [Online]. Available: http://odoch19.uniroma1.it/odoch19/

[5] ^ PANTHER. [Online]. Available: http://www.pantherdb.org/

[6] ^ MarineTLO. [Online]. Available: https://projects.ics.forth.gr/isl/MarineTLO/

[7] ^ MUSIC. [Online]. Available: http://musicontology.com/

[8] ^ OCLC. [Online]. Available: (https://www.oclc.org)

[9] ^ G. Kuck, "Tim Berners-Lee's Semantic Web," South African Journal of Information Management, vol. 6, Dec. 2004, doi: 10.4102/sajim.v6i1.297.

[10] ^ B. Motik et al., "OWL 2 Web Ontology Language: Structural Specification and Functional-Style," Journal of Pragmatics - J PRAGMATICS, vol. 27, Jan. 2008.

[11] ^ K. Beck et al., Manifesto for Agile Software Development. 2001. [Online]. Available: http://www.agilemanifesto.org/

[12] ^ M. Johnson, "How the Statistical Revolution Changes (Computational) Linguistics," in Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics: Virtuous, Vicious or Vacuous?, Mar. 2009, pp. 3–11. [Online]. Available: https://aclanthology.org/W09-0103

[13] ^ R. Sennrich, B. Haddow, and A. Birch, Neural Machine Translation of Rare Words with Subword Units. arXiv, 2015. doi: 10.48550/ARXIV.1508.07909.

[14] ^ M. Schuster and K. Nakajima, "Japanese and Korean Voice Search," in International Conference on Acoustics, Speech and Signal Processing, 2012, pp. 5149–5152.

[15] ^ T. Kudo, Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. arXiv, 2018. doi: 10.48550/ARXIV.1804.10959.

[16] ^ T. Kudo and J. Richardson, SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. arXiv, 2018. doi: 10.48550/ARXIV.1808.06226.

[17] ^ T. Mikolov, K. Chen, G. Corrado, and J. Dean, Efficient Estimation of Word Representations in Vector Space. arXiv, 2013. doi: 10.48550/ARXIV.1301.3781.

[18] ^ J. Pennington, R. Socher, and C. Manning, "GloVe: Global Vectors for Word Representation," in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Oct. 2014, pp. 1532–1543. doi: 10.3115/v1/D14-1162.

[19] ^ A. Vaswani et al., Attention Is All You Need. arXiv, 2017. doi: 10.48550/ARXIV.1706.03762.

[20] ^ I. Sutskever, O. Vinyals, and Q. V. Le, Sequence to Sequence Learning with Neural Networks. arXiv, 2014. doi: 10.48550/ARXIV.1409.3215.

[21] ^ C. Raffel et al., Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. arXiv, 2019. doi: 10.48550/ARXIV.1910.10683.

[22] ^ M. Lewis et al., BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. arXiv, 2019. doi: 10.48550/ARXIV.1910.13461.

[23] ^ FastAPI. [Online]. Available: (https://fastapi.tiangolo.com/)

[24] ^ Paraphraser model. [Online]. Available: (https://towardsdatascience.com/paraphrase-any-question-with-t5-text-to-text-transfer-transformer-pretrained-model-and-cbb9e35f1555)

For preparing this report, the following deliverables have also been taken into consideration:

Deliverable Number	Deliverable Title	Lead beneficiary	Type	Dissemination Level	Due date
D1.1	Standardised Methodology and Set of Ontologies for the Characterisation of Data Sources	UNIPR	Report	Public	M09
D1.3	EU statistics datasets	STAM	Report	Public	M29
D2.1	Data Warehouse (DWH) for agriculture policy impact assessment data management	AAT	Report	Public	M24
D6.1	AGRICORE Architecture and Interfaces	IDE	Report	Public	M24
D10.1	Project Management Handbook	IDE	Report	Confidential	M01