First Component Release

Document Due Date: 31/08/2020
Document Submission Date: 31/08/2020

Work Packages 3, 4, 5, 6, 7, 8

Type: Other (Software)
Document Dissemination Level: Public

INODE
Intelligent Open Data Exploration
is funded by the Horizon 2020 Framework Programme of the EU for Research and Innovation.
Grant Agreement number: 863410— INODE — H2020-EU.1.4.1.3.

(This page has been intentionally left blank)

Executive Summary

This deliverable provides the first software component release of the INODE project. The deliverable contains the following 6 sub-deliverables:

- D3.1 Integrated Query Processing Services
- D4.1 Data Linking and Modeling Services
- D5.1 Data Access & Exploration Services
- D6.1 User Assistance Services
- D7.1 Multi-Modal Discovery Services
- D8.1 Evaluation Service

In Section 1 we give a brief overview of the INODE system architecture. In Section 2 we provide screenshots of a working prototype of INODE 1.0. In Section 3 we list the API specification of the currently implemented services. In Section 4, we give a snapshot of the current data models of the three use cases. Note that a detailed description of these components is part of the next deliverables D3.2 to D8.2.

Project Information

Project Name	Intelligent Open Data Exploration
Project Acronym	INODE
Project Coordinator	Zurich University of Applied Sciences (ZHAW), CH
Project Funded by	European Commission
Under the Programme	H2020-EU.1.4.1.3. - Development, deployment and operation of ICT-based e-infrastructures
Call	H2020-INFRAEOSC-2019-1
Topic	INFRAEOSC-02-2019 - Prototyping new innovative services
Funding Instrument	Research and Innovation action
Grant Agreement No.	863410

Document Information

Document reference	D3.1, D4.1, D5.1, D6.1, D7.1, D8.1
Document Title	First Component Release
Work Package reference	WP3, WP4, WP5, WP6, WP7, WP8
Delivery due date	31/08/2020
Actual submission date	31/08/2020
Dissemination Level	Public
Authors(s)	Belmpas Theofilos, Orest Gkini, Koutrika Georgia, Skoutas Dimitris, Stavroula Eleftheraki (ATHENA) Amer-Yahia Sihem, Boumaout Mourad, Personnaz Aurélien (CNRS) Lücke-Tieke Hendrik, May Thorsten (Fraunhofer) Litke Antonis, Papadakis Nikolaos, Papadopoulos Dimitris (Infili) Fabricius Maximilian, Subramanian Srividya (MPE) Bastian Frederic, Mendes de Farias Tarcisio (SIB) Massucci Francesco, Multari Francesco, Rull Guillem (SIRIS) Calvanese Diego, Lanti Davide, Mosca Alesandro, Guohui Xiao (UNIBZ) Braschler Martin, Kosten Catherine, Sima Ana, Smith Ellery, Stockinger Kurt (ZHAW)

Table of Contents

1 INODE System Architecture

In this section we give a brief overview of the INODE's system architecture. The high-level architecture is illustrated in Figure 1. In a nutshell, INODE brings together the following main services that we discuss in detail in Sections 1.1, 1.2 and 1.3.

- *Data Access & Exploration* services enable the user to communicate with the system.
- *User Assistance* services allow the system to be reactive as well as anticipative of the user needs.
- *Multi-Modal Discovery* services enable visual interaction and exploration.
- *Data Linking & Modeling* services enable working with diverse datasets.
- *Integrated Query Processing* services are responsible for the execution of the requests coming from the user-facing services.

We refer to all the services shown in green as **OpenDataDialog** and to the services shown in orange as **OpenDataLinking**. The services shown in blue are **BackendServices**.

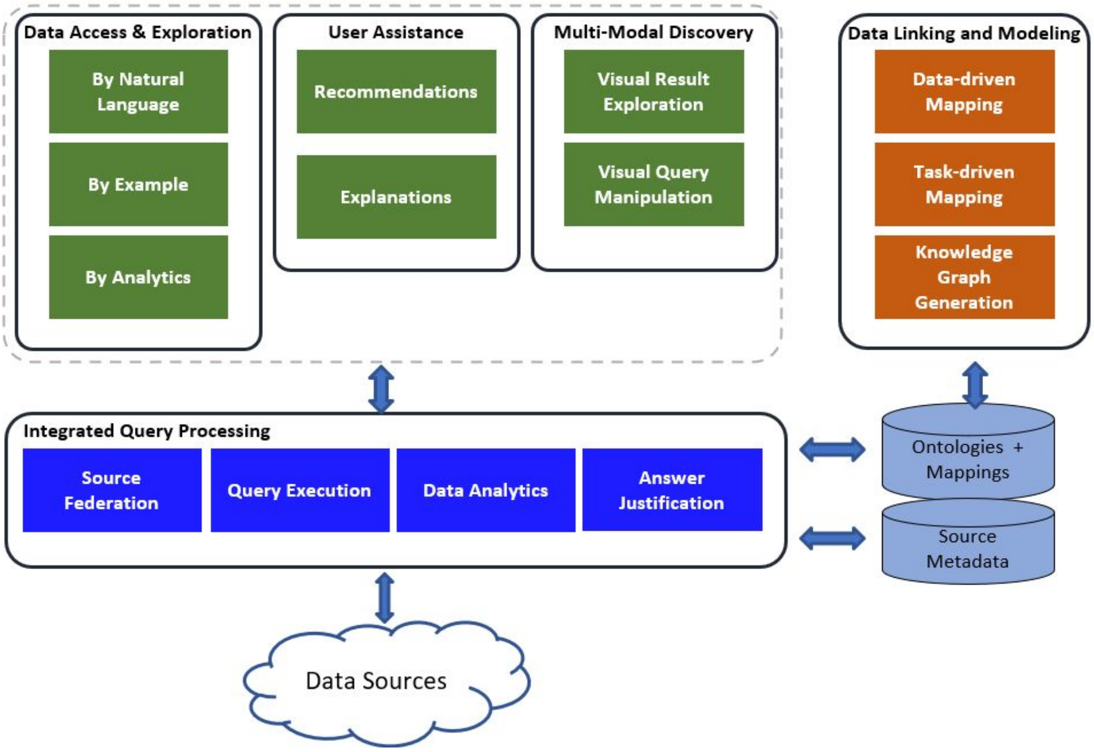


Figure 1: Major components of the INODE architecture.

1.1. OpenDataDialog

OpenDataDialog is the synergy of Data Access & Exploration, User Assistance, and Multi-Modal Discovery Services. We will now describe these services in more detail.

Data Access & Exploration. INODE introduces operators for the user to access and explore the data. For instance, in *by example*, the user inputs examples of data and expects the system to return similar ones in the underlying dataset. For *by analytics*, the user inputs analytics in the form of histograms, data distributions, aggregates (such as variance and counts), etc., and expects to receive data results that exhibit those analytics. For *by natural language*, the user provides a query in natural language and INODE translates the query into SQL or SPARQL. INODE instantiates sets of items to allow operator composition.

User Assistance. INODE guides the user in data exploration by offering recommendations (i.e., queries that could be asked) and explanations, i.e., natural language descriptions of queries to further help the user understand system responses and the underlying data.

Multi-Modal Discovery. This layer implements means to explore the results of each exploration step and to manipulate operators. In doing so, it helps users understand the options they have for finding the data they need through *visual exploration* of results at each exploration step and *interactive manipulation and optimization* of exploration operators.

Visual exploration of intermediate results aims at enabling users to visually manage the actual content. When necessary, users can revise their exploration steps through interactive manipulation and optimization of exploration operators.

1.2 OpenDataLinking

OpenDataLinking is the synergy of Data Linking and Modeling Services that we will now describe in more detail.

Data Linking and Modeling. This layer enables linking of loosely coupled collections of datasets to support queries across them. INODE supports two different forms of mapping construction, namely *data-driven mapping*, which is triggered when new data sources are added to the system, and *task-driven mapping*, which is triggered when the execution of new analytic tasks is requested. In data-driven mapping, the structure and content of new data sources are analyzed and correlated to the ontology, so as to generate new mappings and propose them to the system designer for validation. In task-driven mapping, requested tasks are matched against available but not yet integrated data sources, and candidate sources with generated mappings are proposed, again for validation.

Moreover, when integrating text data, INODE enables *automatic knowledge graph generation*, by identifying entities and relationships in unstructured documents and integrating them into a queryable ontology. As a consequence, both structured and unstructured data can be linked and queried in a uniform way.

1.3 BackendServices

Integrated Query Processing. This service is responsible for the execution of queries and can be considered as the back-end service for OpenDataDialog and OpenDataLinking. *Source federation* provides an integrated coherent view of the heterogeneous data sources (e.g. SQL, SPARQL, text) accessible in INODE to enable ontology-based data access. *Query execution* provides on-the-fly query rewriting by exploiting different forms of reasoning taking into account various data dimensions (such as temporal, spatial, etc). *Data analytics* focuses on efficient query transformation and execution to compute complex analytical functions. *Answer justification* generates compact and easy to understand explanations for query results.

2 INODE in Action

We have implemented a preliminary version of INODE 1.0. The following section describes how the OpenDataDialog and the OpenDataLinking services can be used.

Referring to our system architecture shown in Figure 1, we use the following systems:

OpenDataDialog:

- Data Access and Exploration:
  - By Natural Language:
    - SODA+[1]
    - NALIR+[2]
  - By Example:
    - CNRS-Pipelines[3]
- User Guidance:
  - Explanations:
    - Logos[4]
- Multi-Modal Discovery:
  - Visual Result Exploration:
    - FHG MultiTableExplorer
    - FHG executor-processor library integration

OpenDataLinking:

- Data Linking and Modeling:
  - Ontop-Bootstrapper (MPBoot)
  - Noima[5]: Infili-Extraction Engine
  - ZHAW-Extraction Engine

[1] We added NLP extensions to the original SODA source code. Blunschi, L., Jossen, C., Kossmann, D., Mori, M., & Stockinger, K. (2012). SODA: Generating SQL for business users. *Proceedings of the VLDB Endowment*, *5*(10), 932-943.

[2] A modified version of the NaLIR system: Li, F., & Jagadish, H. V. (2014). Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, *8*(1), 73-84.

[3] Data Exploration Pipelines: http://www.inode-project.eu/blog/data-exploration-pipelines/

[4] Kokkalis, A., Vagenas, P., Zervakis, A., Simitsis, A., Koutrika, G., & Ioannidis, Y. (2012, May). Logos: a system for translating queries into narratives. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (pp. 673-676).

[5] Noima: http://www.inode-project.eu/blog/noima/

2.1 OpenDataDialog

We will now explain the functionality of OpenDataDialog by walking through the following three different usage scenarios of increasing complexity. These scenarios demonstrate the operability and successful integration of various INODE-services.

- Scenario 1: NL-to-SQL, SQL-to-NL and simple data model visualization
- Scenario 2: Adding more advanced result visualization to Scenario 1
- Scenario 3: Adding pipeline operators to Scenario 2

Scenario 1: NL-to-SQL, SQL-to-NL and Simple Data Model Visualization

Figure 2 shows the INODE pilot landing page where users will begin their data exploration/query journey.

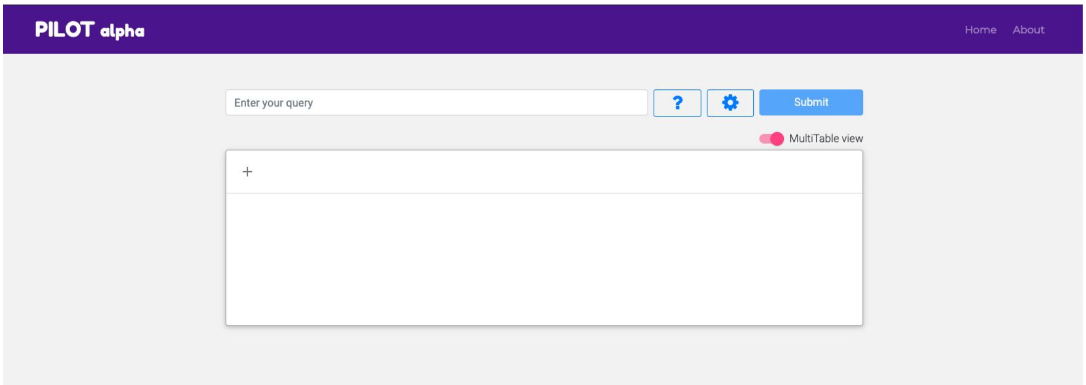


Figure 2: Landing page of INODE pilot.

.

1. *Translating natural language to SQL*: Assume that a user wants to issue the query "Find the topics of projects that ended in 2014" (see Figure 3) against the CORDIS database. Clicking the gear button opens a menu for the user to select which systems to use, in this case Nalir+ or SODA and which database to search, in this case CORDIS. The users need to choose how many interpretations (i.e. different resulting SQL statements) they would like to have from each search system as well as how many results (rows of data from the database) from each interpretation they would like to see.

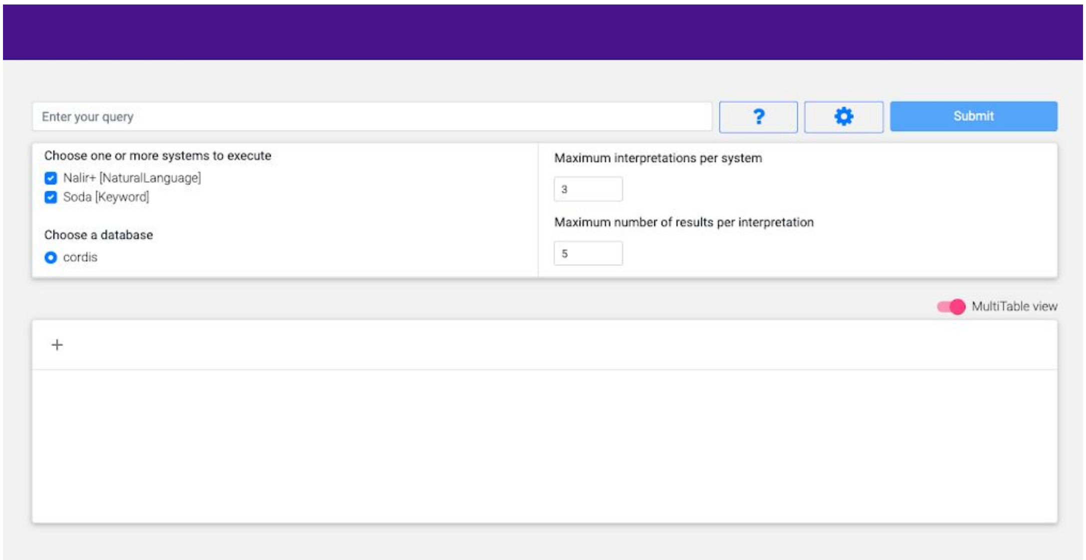
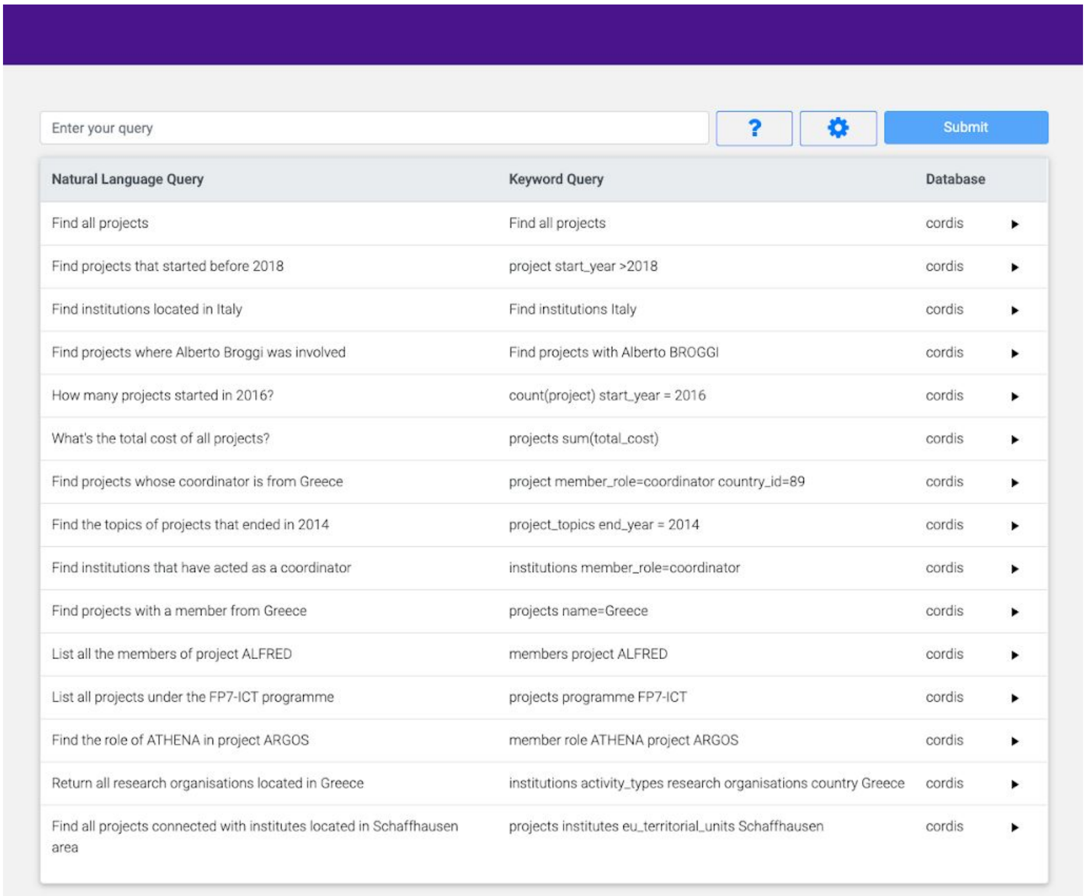


Figure 3: Choosing the NL-to-SQL systems and the database.

2. Clicking the question mark button opens a menu (Figure 4) with example queries intended to guide the users in their data exploration/query journey. These example queries show both the natural language question that a user would ask, as well as, how to formulate this query in a way that both query engines can understand and return the expected results from the database. Clicking the play button executes the query. These query examples are intended to help guide the user in their own query formulations.



Natural Language Query	Keyword Query	Database
Find all projects	Find all projects	cordis
Find projects that started before 2018	project start_year >2018	cordis
Find institutions located in Italy	Find institutions Italy	cordis
Find projects where Alberto Broggi was involved	Find projects with Alberto BROGGI	cordis
How many projects started in 2016?	count(project) start_year = 2016	cordis
What's the total cost of all projects?	projects sum(total_cost)	cordis
Find projects whose coordinator is from Greece	project member_role=coordinator country_id=89	cordis
Find the topics of projects that ended in 2014	project_topics end_year = 2014	cordis
Find institutions that have acted as a coordinator	institutions member_role=coordinator	cordis
Find projects with a member from Greece	projects name=Greece	cordis
List all the members of project ALFRED	members project ALFRED	cordis
List all projects under the FP7-ICT programme	projects programme FP7-ICT	cordis
Find the role of ATHENA in project ARGOS	member role ATHENA project ARGOS	cordis
Return all research organisations located in Greece	institutions activity_types research organisations country Greece	cordis
Find all projects connected with institutes located in Schaffhausen area	projects institutes eu_territorial_units Schaffhausen	cordis

Figure 4: Example queries help the users to get a sense for what kind of queries are supported.

After executing the query “Find the topics of projects that ended in 2014”, the user is shown the following results from the query. As shown in Figure 5, the user sees several different interpretations of their query from the query engine they selected (seen in Figure 3). Each interpretation shows 5 results, or rows from the database.

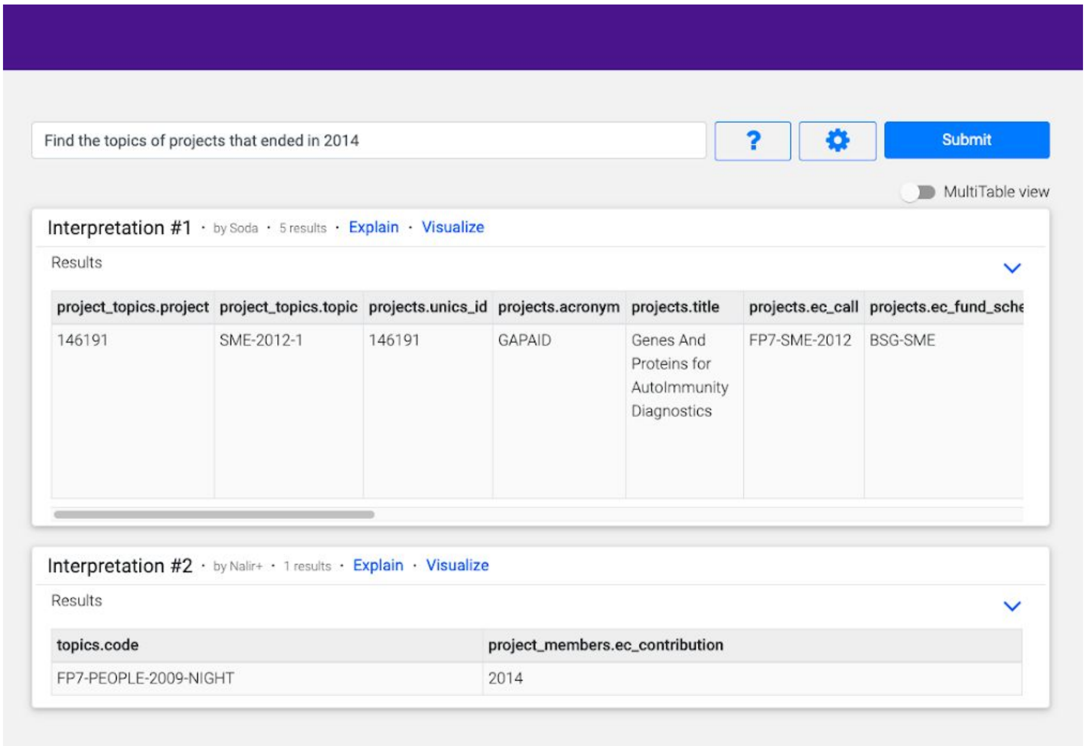


*Figure 5: Two different interpretations of the query "*Find the topics of projects that ended in 2014".

The following features help the user better understand the data which has been returned by their query.

1. Clicking on "Explain" for each interpretation gives the user a natural language explanation of what data was returned in their query as seen in Figure 6.

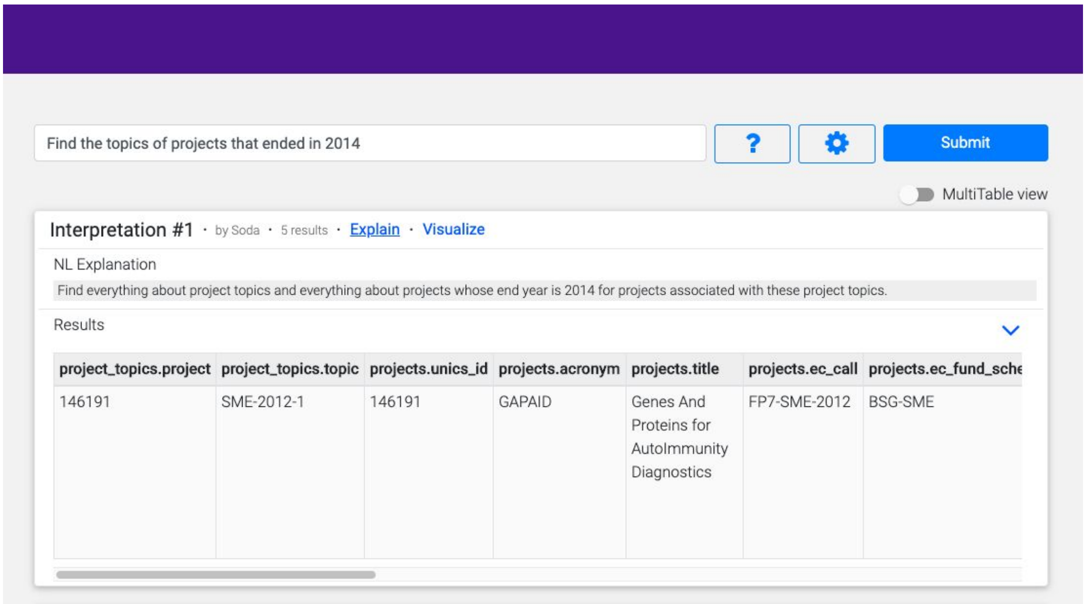


Figure 6: NL explain shows the natural language interpretation of the resulting SQL query.

2. Clicking on "Visualize" for each interpretation gives the user a visualization of the database model as seen in Figure 7. The tables queried by the user are highlighted in pink, and the executed SQL statement is also shown in order to provide the user with additional context for understanding their data and determining how to proceed in their exploration.

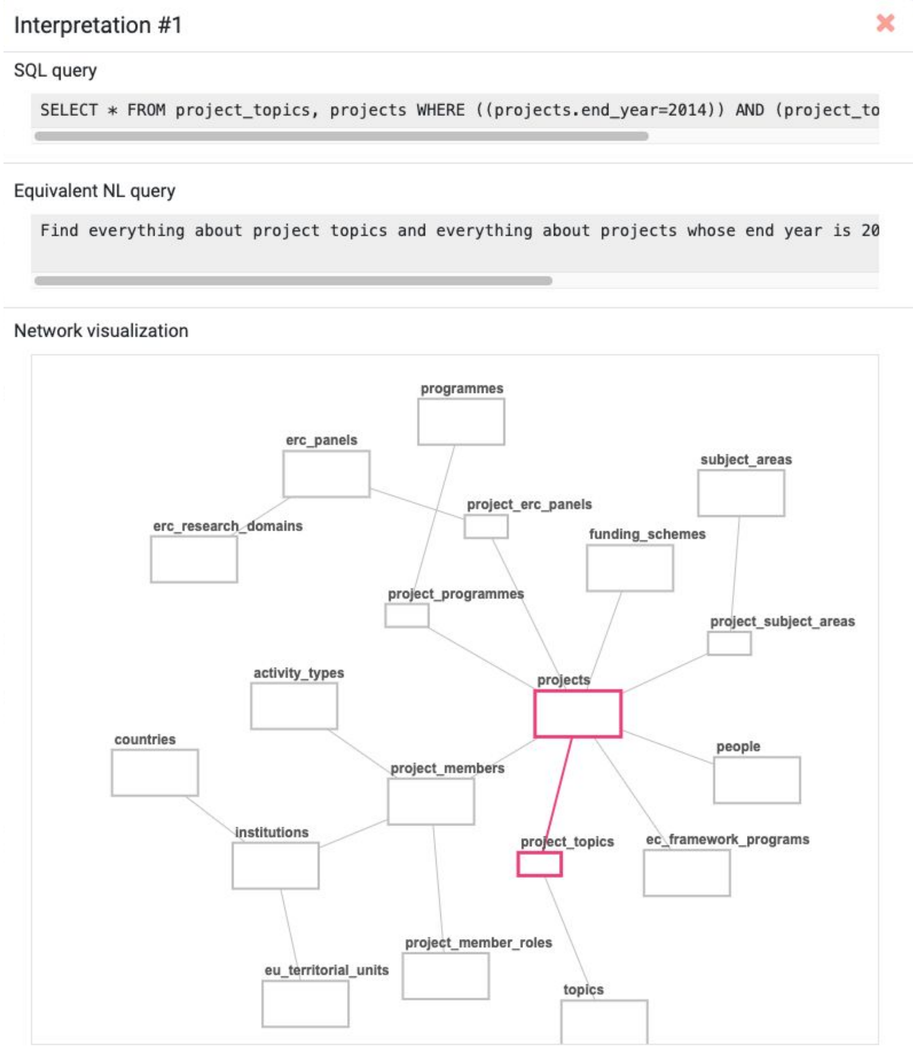


Figure 7: For a given natural language query, the resulting SQL statements, the NL-explanation and the respective tables of the data model are shown that are used for answering the NL query.

Scenario 2: Adding More Advanced Results Visualization

Turning on the MultiTable view adds additional ways for the user to explore the data such as charts (tailored to the data type of the column) and data aggregations for each interpretation of the given Natural Language query. Initially all of the columns from the resulting interpretations are visualized in the MultiTable view (as seen in Figure 8). The query shown in Figure 8 requires a join of two tables, which means that a significant number of columns are visualized with the MultiTable view. All of the different tables can be viewed by scrolling to the right. The user has the option to hand select which columns are the most interesting for their data exploration. Users can choose which columns and tables they want to visualize by clicking on the plus sign in the first row as seen in Figure 9. Figure 10 shows a pared down version of the tables and columns a viewer might wish to see.

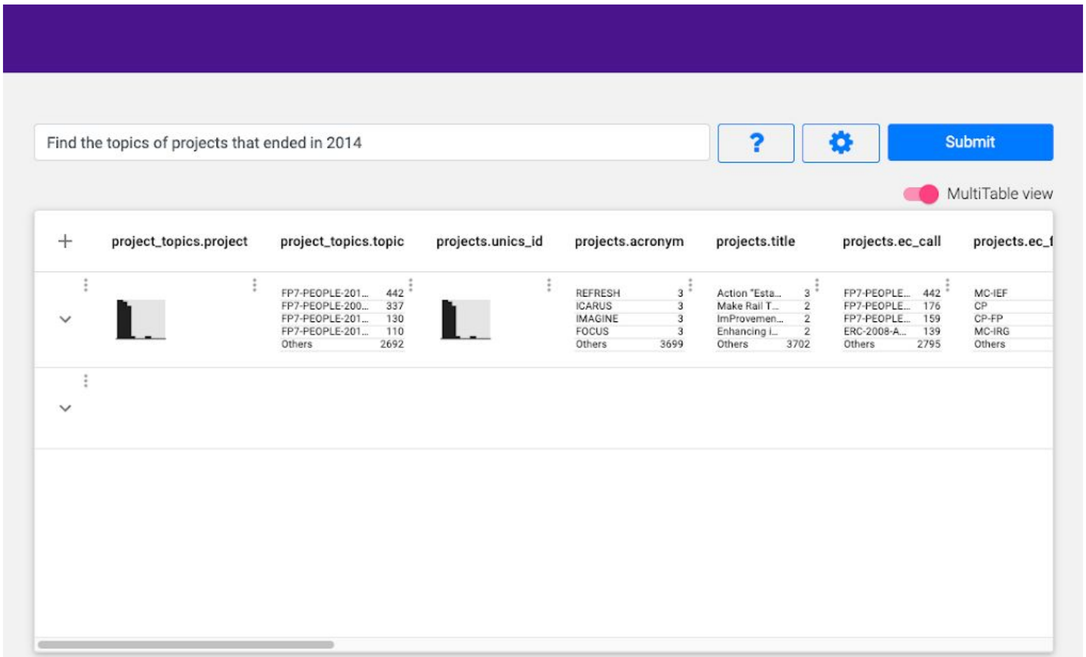


Figure 8: Executing the query "Find the topics of projects that ended in 2014" with the MultiTable view on.

Figure 9: Clicking on the + opens a window, where the user can hand select which columns to display in the MultiTable view.

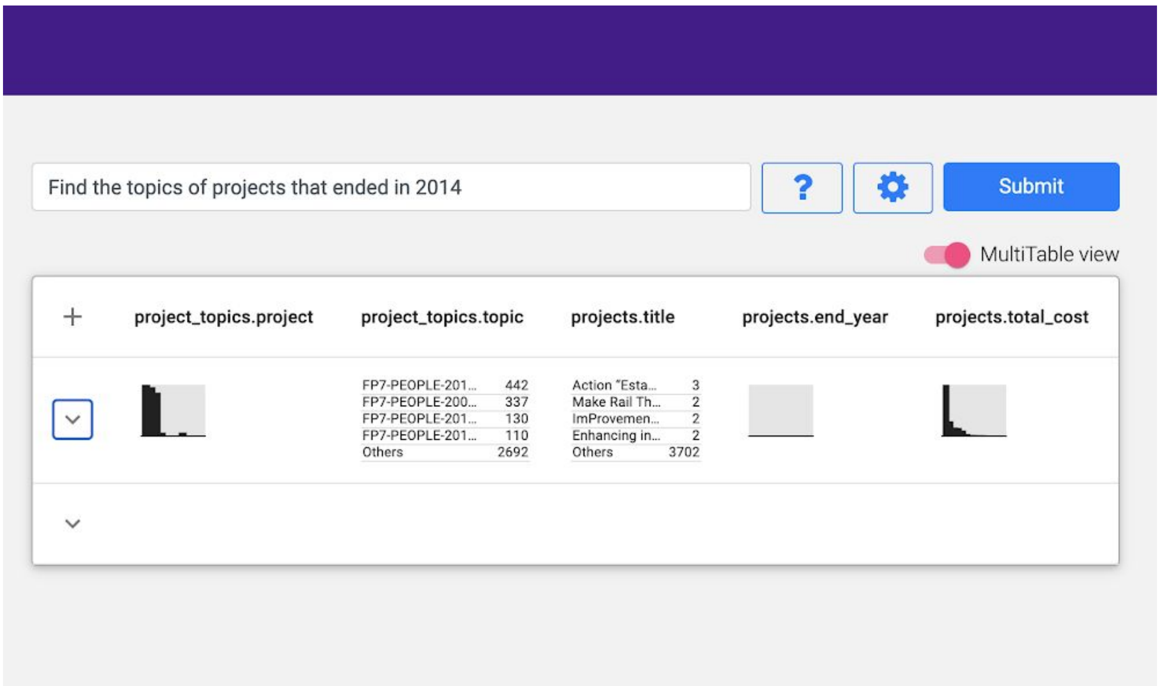


Figure 10: A smaller selection of columns and tables, hand selected by the user.

The user still has the option to view rows of data individually in this view by clicking on the drop down symbol as seen in Figure 11.

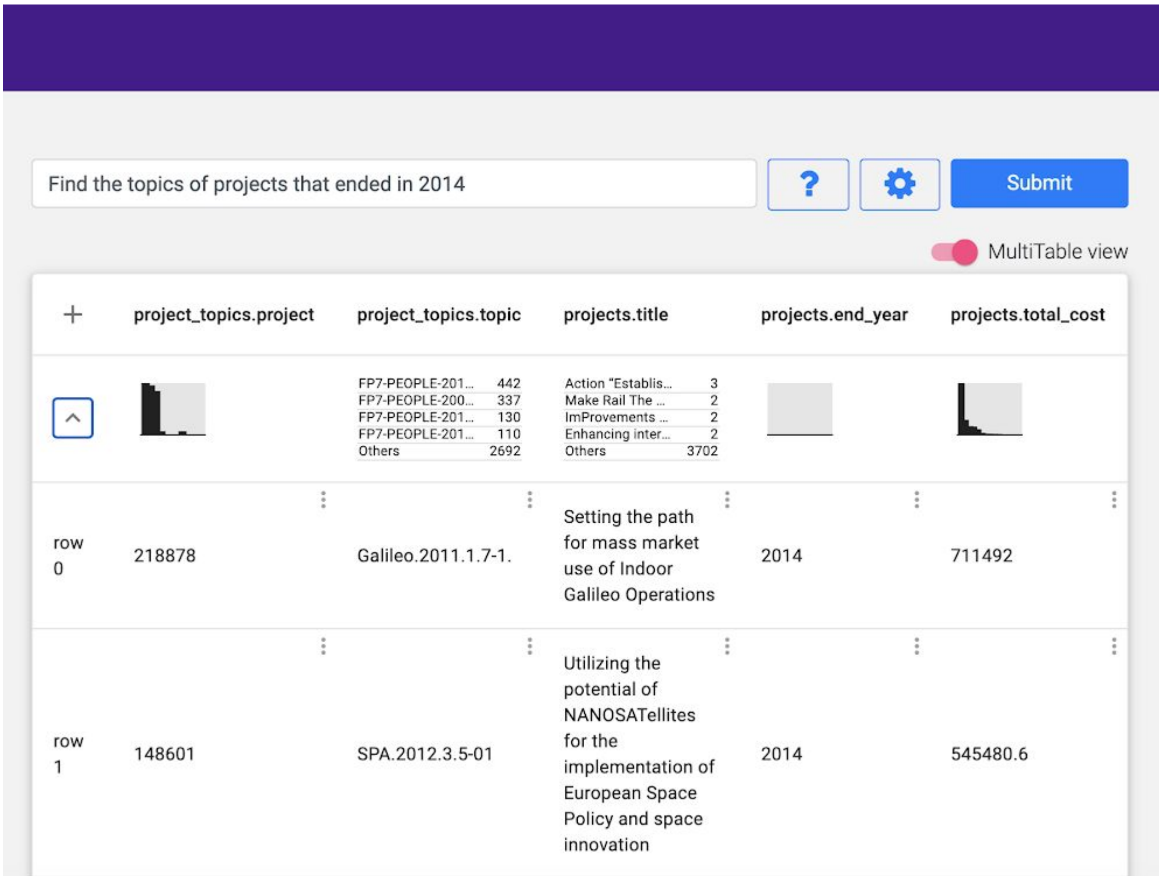


Figure 11: The user is able to view and verify the rows of visualized data by clicking on the down tick symbol.

Scenario 3: Adding Pipeline Operators

The third scenario allows the user to build data pipelines to further refine and explore their data. The user can begin to build the data pipelines with the "MultiTable view" on. Currently, there are 4 different pipeline operators: by filter, by superset, by overlap and by facet.

Each column of each query interpretation has 3 dots in the upper right hand corner that display a window showing the available operators that can be added to the data exploration pipeline.

by filter

The user can begin a "by filter" operation by first clicking on the drop down symbol to display the rows and then on the three small dots, which display the window "explore by filter" as seen in Figure 12 below. The "by filter" operator enables the user to search for data according to a certain attribute in a column.

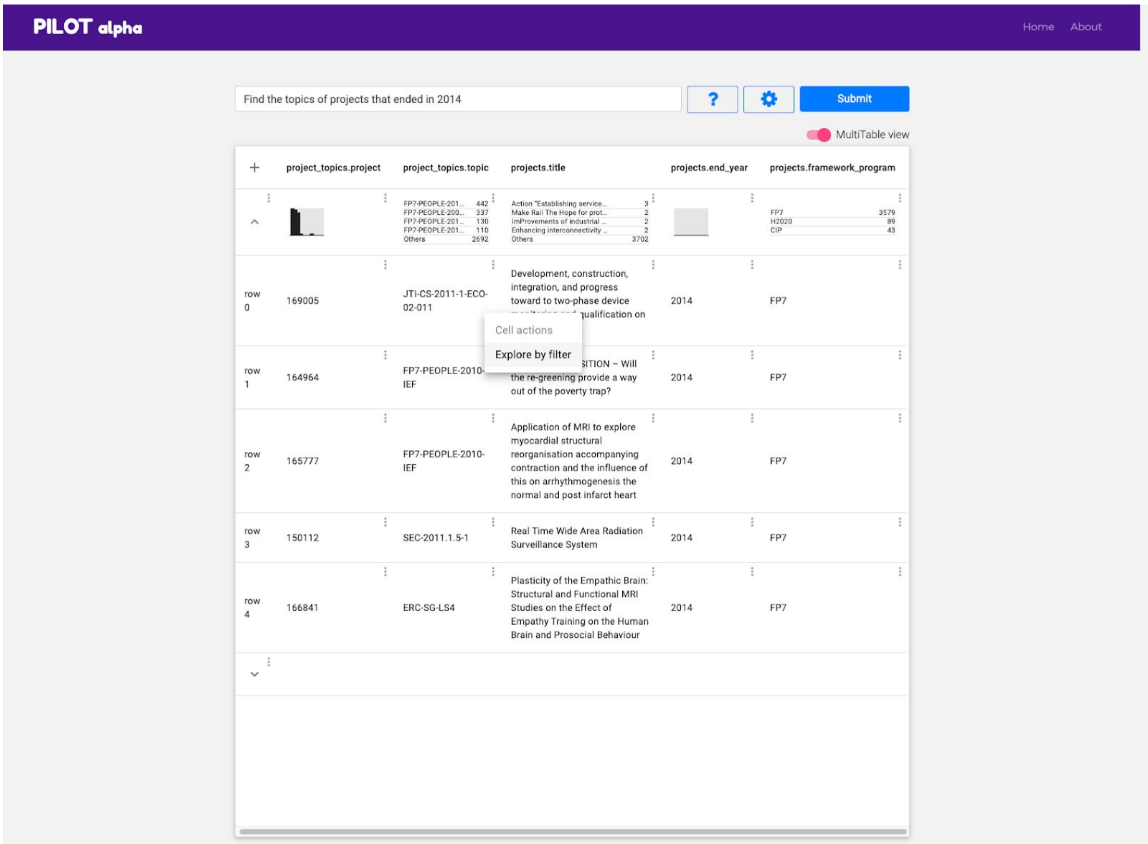


Figure 12: Starting the data pipeline with the "by filter" operator.

In Figure 12, the user has clicked on the drop down symbol for the second row of visualizations in the MultiTable view to open 5 rows of data from the executed query

```
SELECT * FROM project_topics, projects WHERE ((projects.end_year=2014))
AND (project_topics.project=projects.unics_id)
```

The user then clicks on a certain value in the column to filter on, for example from the column topics, the user chooses "FP7-PEOPLE-2010-IEF".

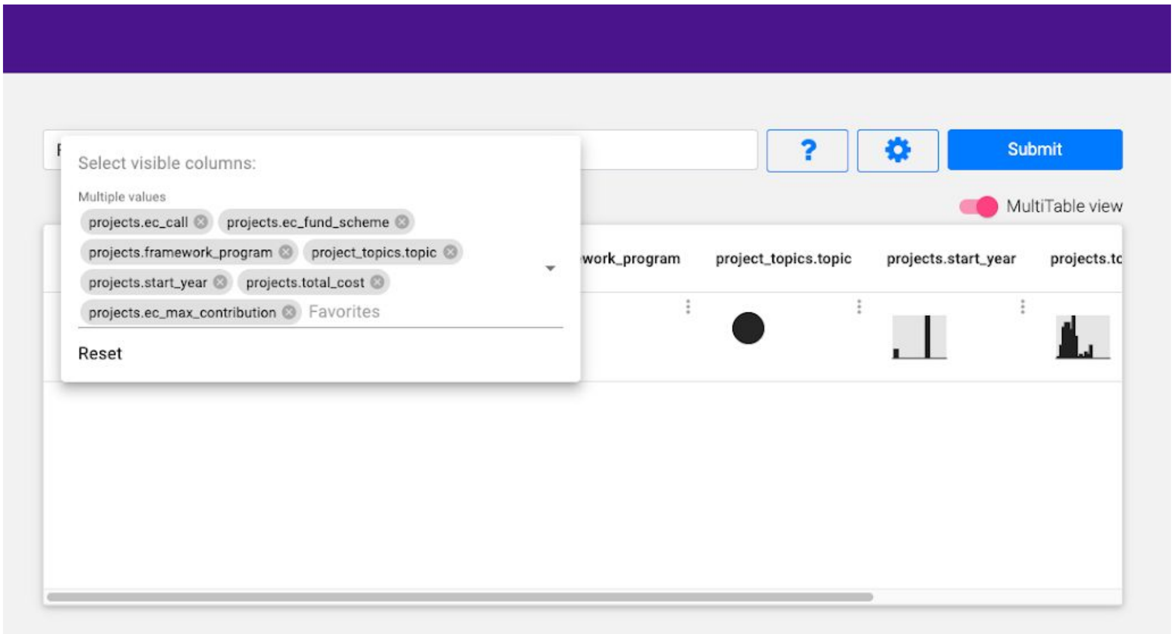


*Figure 13: Results returned from "by filter" operator, filtered on "*FP7-PEOPLE-2010-IEF*".*

Figure 13 shows the results returned from the "by filter" operator. The user is then again able to choose which columns they want to see by clicking on the + sign.

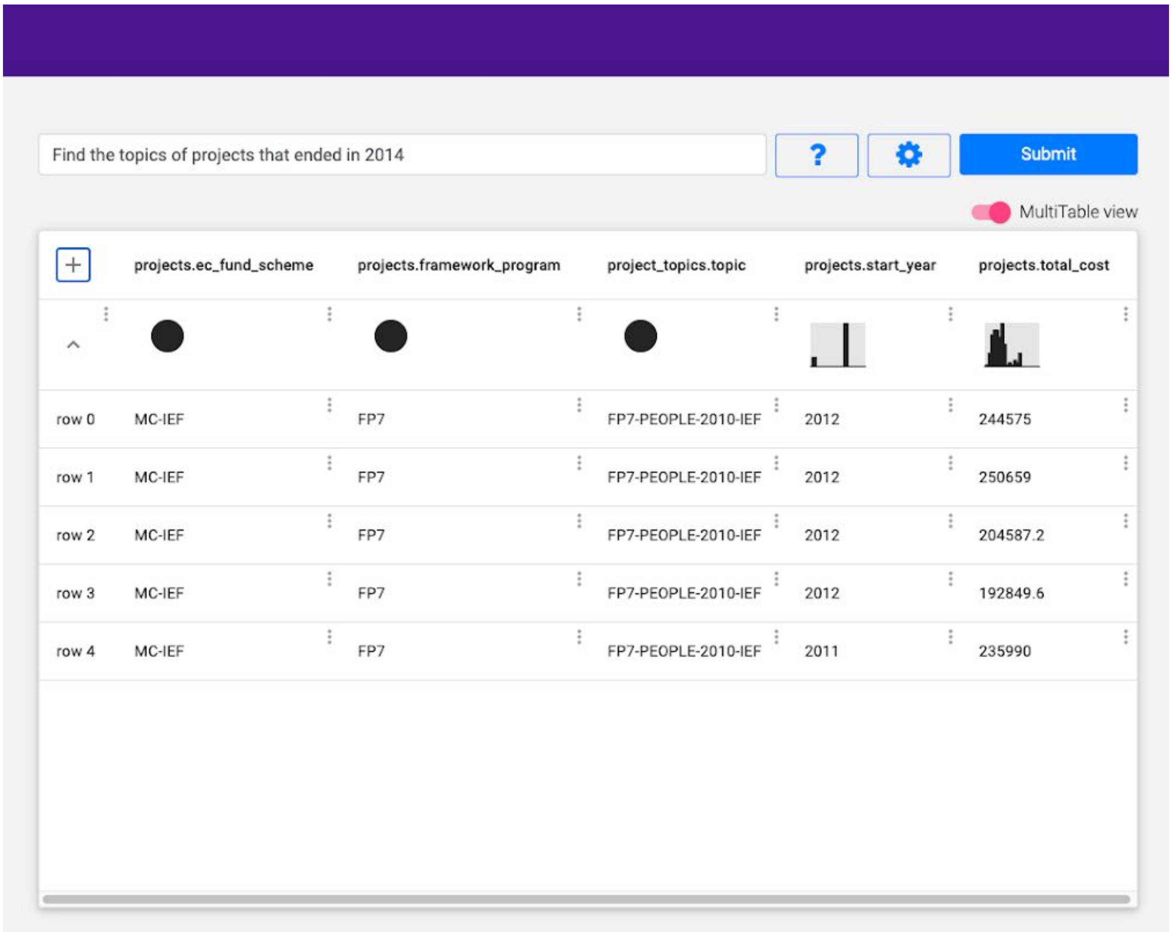


Figure 14: 5 rows of filtered data with visualizations. After filtering on "FP7-PEOPLE-2010-IEF", the data table becomes quite homogenous for ec fund scheme, and framework program, which is not a surprise, but it can be also seen at a glance, that one start year is outnumbering the others and how the projects total costs are distributed.

After clicking the drop down symbol, the user is able to see 5 rows of filtered data as shown in Figure 14.

by superset

The user can begin a "by superset" operation by clicking on the three small dots on the first column of a given table, and by clicking on the "Explore by superset" button. The "by superset" operator enables the user to increase the size of a given set by releasing one of the filters restricting it.

The idea is to find which filter to remove to get the smallest set containing the explored set.

In the following example, the explored set is a list of all the projects started in 2017 and ending in 2019.

The SQL query describing this set would be :

```
SELECT * FROM projects WHERE start_year = 2017 AND end_year = 2019
```

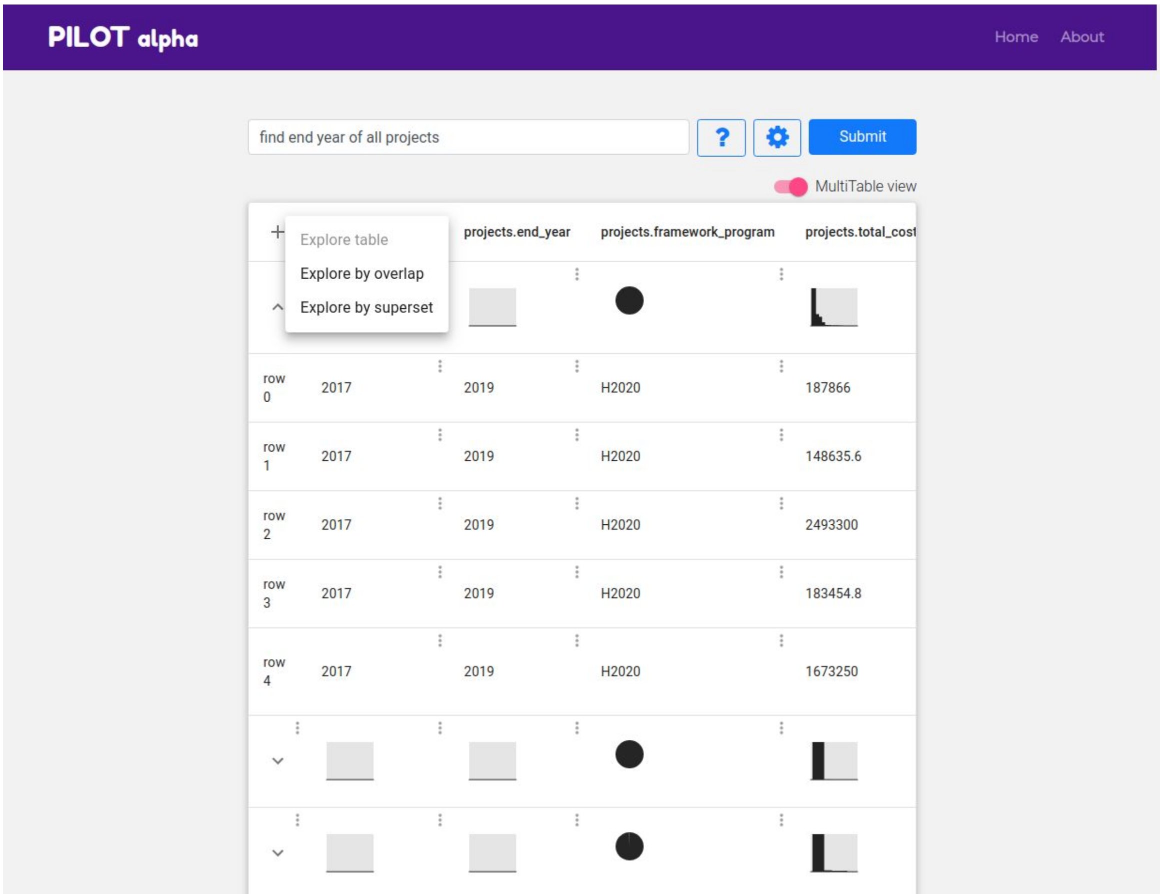


Figure 15: The user can click on the 3 dots at the beginning of the column to display the "by superset" operator.

The user runs the "by superset" operation, as seen in Figure 15, which returns the set of all projects that ended in 2019 :

```
SELECT * FROM projects WHERE end_year = 2019
```

The filter start_year = 2017 was removed.

Future versions of the INODE pilot will have additional data available such as information regarding how many records were contained in each set. In this example, the explored set had 1968 records, the new set has 5748 records. If the user had kept the “start_year” filter but removed the “end_year”, we would have had a set with 6845 records.

by overlap

The user can begin a “by overlap” operation by clicking on the three small dots at the top left of the row, which display the window “Explore by overlap” as seen in Figure 16. The “by overlap” operator enables the user to explore the data by returning neighbouring sets that have the smallest overlap with the input set and overlap the least amongst themselves.

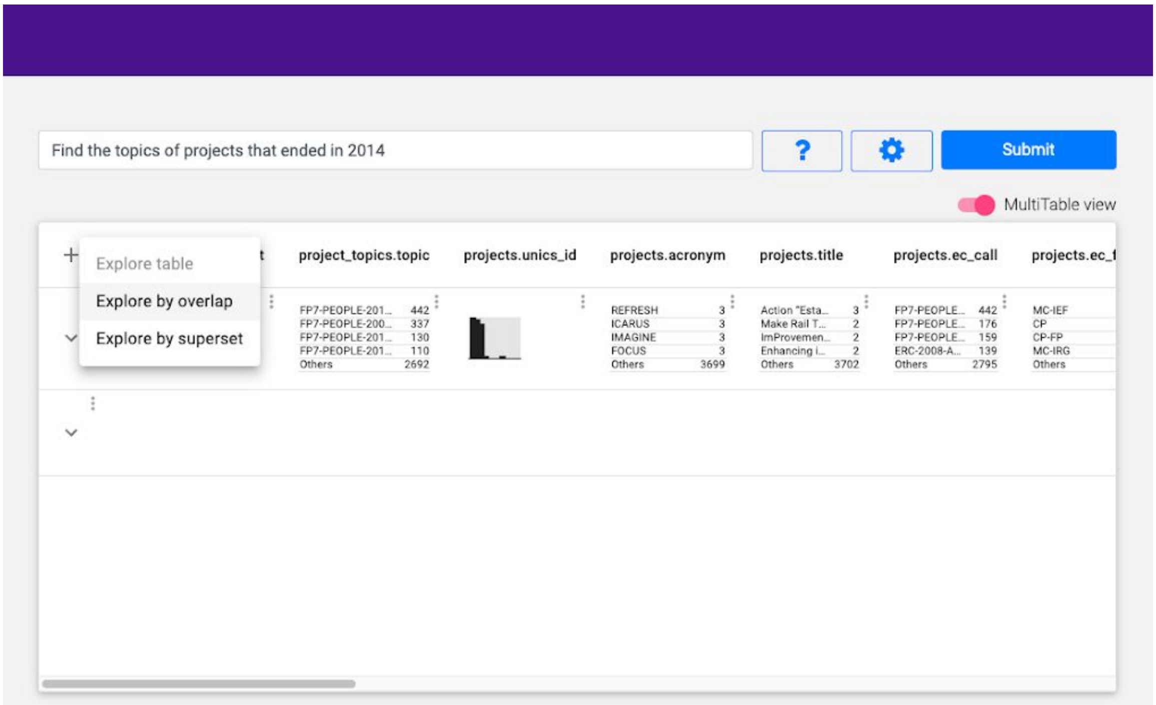


Figure 16: The user selects the “by overlap” operator by clicking the 3 dots at the beginning of the columns.

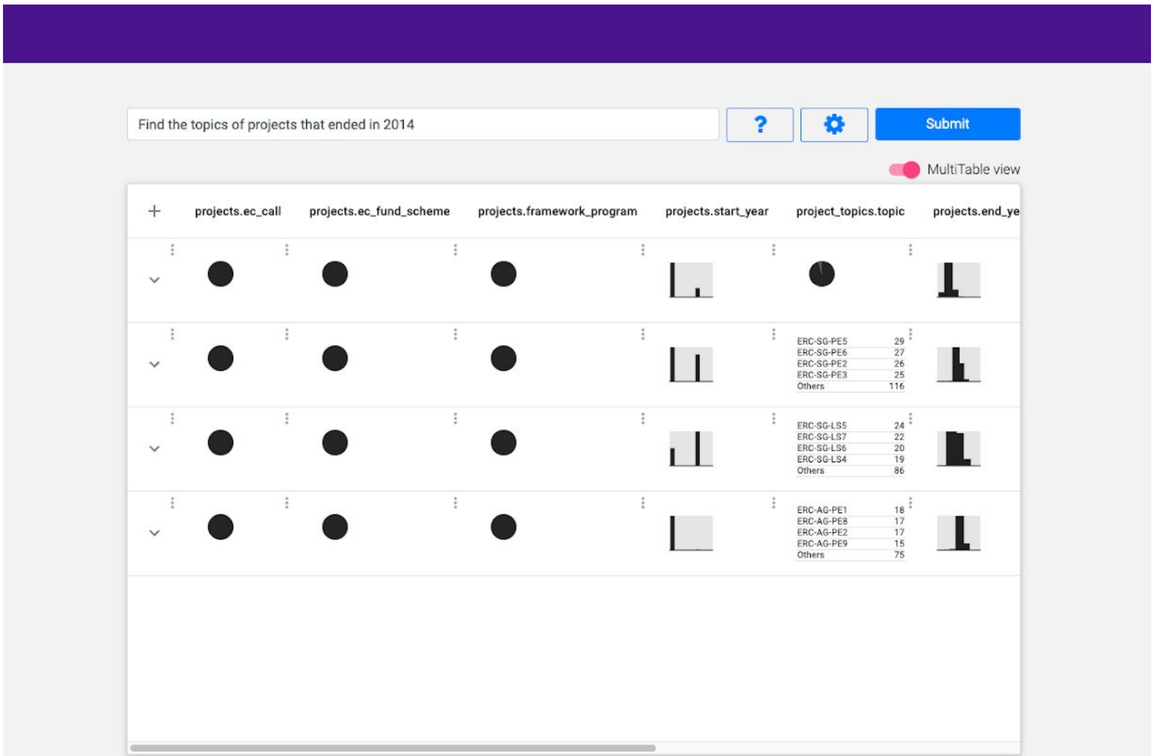


Figure 17: 4 new sets of results and visualizations are returned from applying the "by overlap" operator.

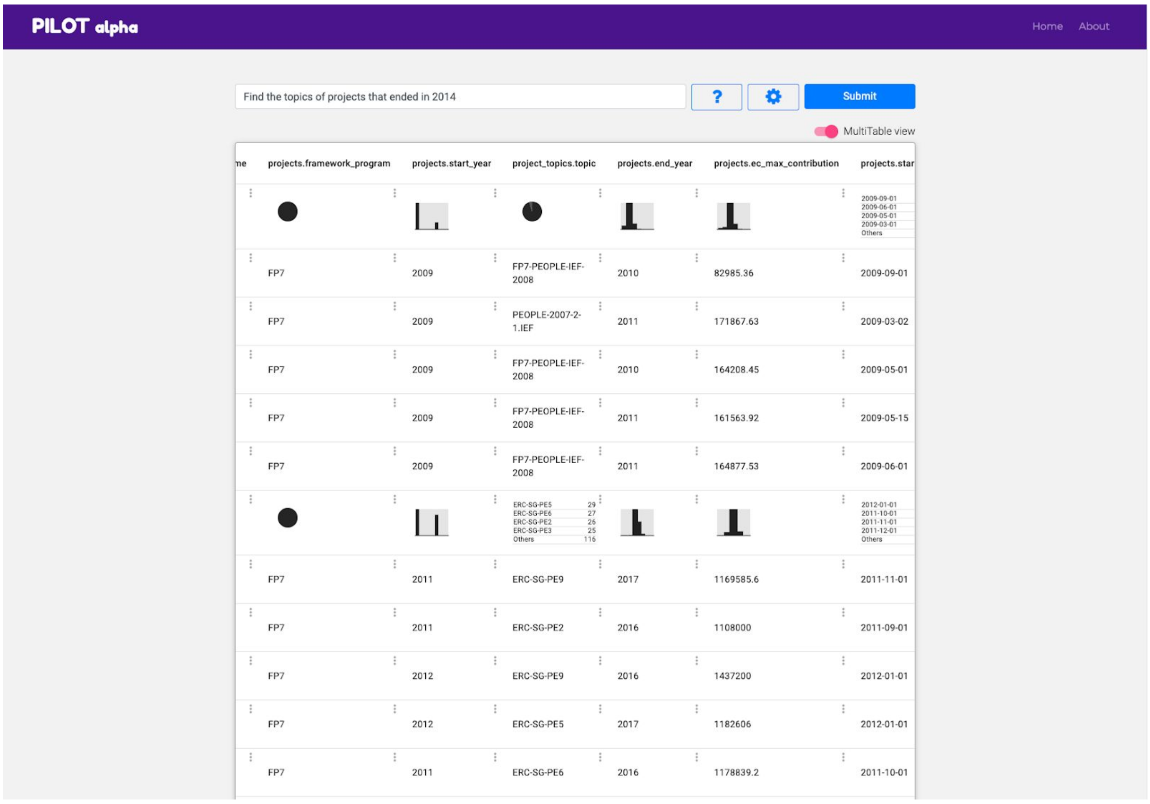


Figure 18: Expanding the returned results shows the query results have been expanded to include projects that ended in other years as well.

The initial query, “Find the topics of projects that ended in 2014”, only returns results from projects that ended in 2014. The results returned by applying the “by overlap” operator are subsets of data that share features in common with the original dataset, but are as distinct as possible from one another.

The returned subsets correspond to the following queries:

- `SELECT * FROM projects join project_topics on (project_topics.project=projects.unics_id) WHERE (projects.framework_program=FP7) AND (projects.ec_fund_scheme=ERC-SG), which returns: 2332 records`

- `SELECT * FROM projects join project_topics on (project_topics.project=projects.unics_id) WHERE (projects.framework_program=FP7) AND (projects.ec_fund_scheme=MC-IEF), which returns: 3911 records`

- `SELECT * FROM projects join project_topics on (project_topics.project=projects.unics_id) WHERE (projects.framework_program=FP7) AND (projects.ec_fund_scheme=ERC-AG), which returns: 1709 records`

In these new sets, by scrolling to the right, as shown in Figure 18, the user can see that each new set returned from the “by overlap” operator includes data on projects from other years as well.

by facet

The user can begin a “by facet” operation by clicking on the three small dots at the top right of a column, which display the window “Explore by facet” as seen in Figure 19. The “by facet” operator enables the user to search for data which is clustered together based on the attributes in the column they have selected to perform the operation on.

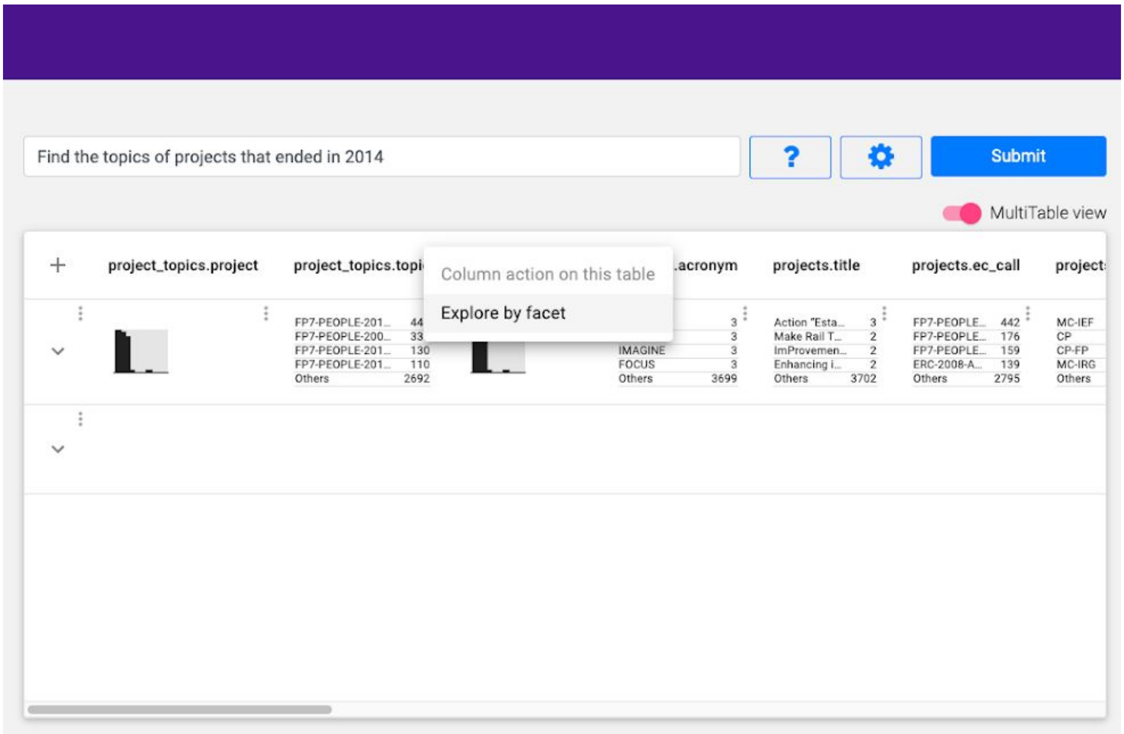


Figure 19: In this example, the user chose to use the “by facet” operator on the column “project_topics.topic”.

During the “by facet” operator execution, the data of the explored set is grouped by the values present in the selected column. The various groups are counted, and the 5 largest groups are returned as distinct SQL queries, displayed as separate tables in the pilot (with no aggregation operation applied to the data).

In the example above, the user selected the column “project_topics.topic” to perform the “by facet” operator on the set of all the projects that ended in 2014.

In Figure 20 we can see that the first 5 results are displayed together because they all have the attribute “FP7-PEOPLE-2011-IEF”, as with the next 5 results who share the attribute “FP7-PEOPLE-2009-RG”.

The “by facet” operator execution returned 5 queries, describing the sets of all the projects that ended in 2014, filtered by the following topics :

- “FP7-PEOPLE-2011-IEF” which results in a set of 442 records
- “FP7-PEOPLE-2009-RG” which results in a set of 337 records
- “FP7-PEOPLE-2010-IEF” which results in a set of 330 records
- “FP7-PEOPLE-2010-IOF” which results in a set of 222 records
- “FP7-PEOPLE-2009-IRSES” which results in a set of 186 records

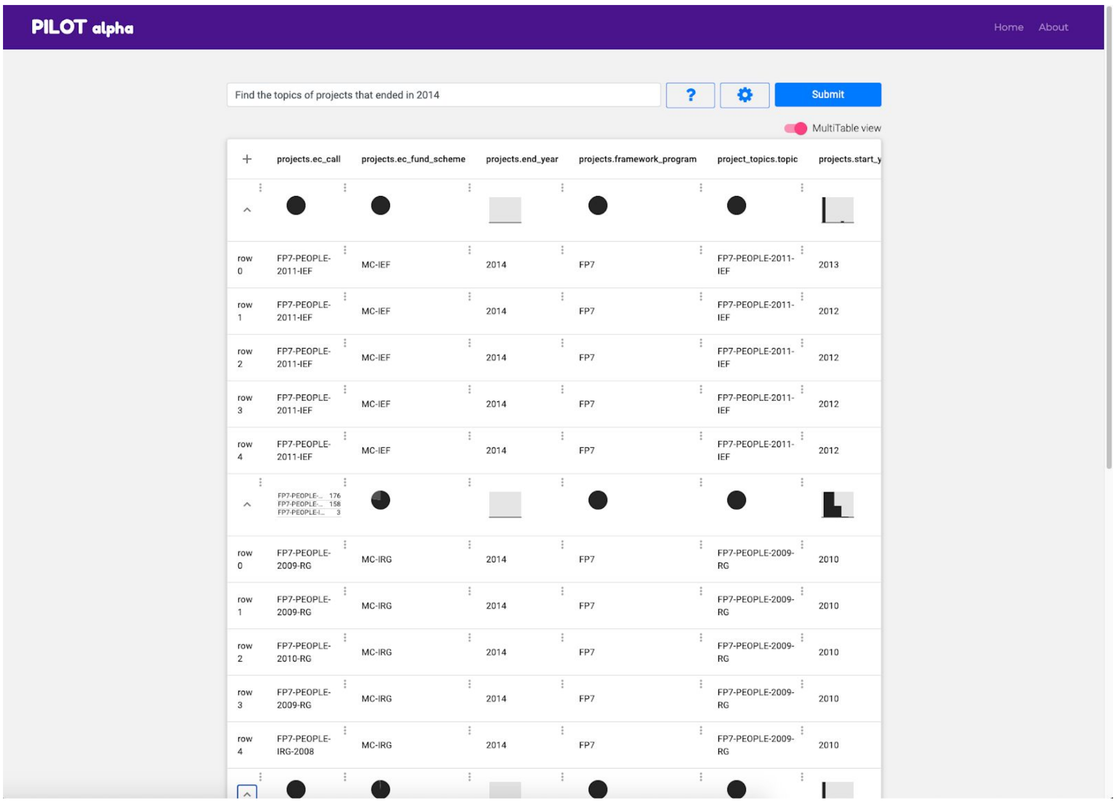


Figure 20: Displays results of the 5 largest sets from the projects that ended in 2014, grouped by topic.

2.2 OpenDataLinking

We showcase the functionalities of the preliminary OpenDataLinking version through two distinct use cases:

1. Cancer Biomarkers Use Case: Information Extraction from PubMed abstracts and Linking with Uberon and OncoMX concepts
2. OpenDataLinking R&I Use Case: Enriching the SIRIS database by linking CORDIS projects based on their NL Objectives

2.2.1 Information Extraction from PubMed abstracts and Linking with Uberon and OncoMX concepts

For this use case, two information extraction systems -from ZHAW (syntax-based) and INF (combining semantic role labelling and deep learning approaches)- were leveraged to extract triples from PubMed articles and map these to existing concepts (anatomical entities) of the Uberon ontology and to genes of the OncoMX database. The linked triples were then added to the latest version of the OncoMX database. A brief description of each engine is given below:

A. The ZHAW Information Extraction Engine

The ZHAW triple extraction system is used to transform unstructured text, in the form of medical research abstracts taken from the PubMed database, into structured data to be used to augment an existing relational database.

The output of the first part of the system comprises a set of subject-predicate-object triples, in annotated natural language text format. For example, for the following PubMed paper title:

Long Non-Coding RNA CCAT2 Promotes Breast Cancer Growth and Metastasis

The first stage involves extracting the following triples:

long non-coding RNA CCAT2 ; promotes ; breast cancer growth
long non-coding RNA CCAT2 ; promotes ; breast cancer metastasis

Since the system is syntax-based, we can leverage syntactic dependencies to give non-linear entity extraction, as shown above. The entities *breast cancer growth* and *breast cancer* *metastasis* are constructed from the noun-phrase *breast cancer growth and metastasis* to give a more accurate representation of the information contained in the text. This procedure is performed over all coordinating conjunctions, and the power set of all permutations of entities is returned. In addition, we also use syntactic rules to annotate the entities and relations, as shown below:

long non-coding RNA CCAT2 ; promotes ; breast cancer growth
long non-coding RNA CCAT2 ; promotes ; breast cancer metastasis

In this example, long non-coding is marked as an adjectival modifier of the entity, and RNA is marked as a compound element of the entity, and the base token is CCAT2. These are based on the syntactic dependencies of the sentence, as shown below:

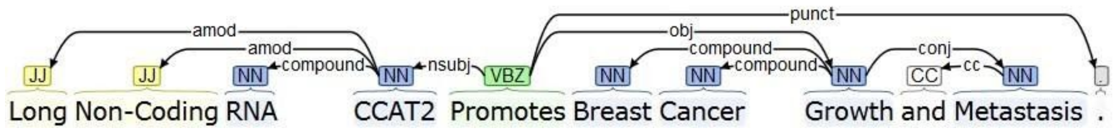


Figure 21: Syntax parsing example from the ZHAW engine.

Other rules exist, and some function on predicates, such as auxiliary verbs and case modifiers. For example, “may” in the relation “may reduce”, can be marked by the auxiliary dependency. With these annotations, we can selectively reduce or expand the information contained in the entities and relations, depending on the current information needed. If more descriptive information is not required, we can reduce the above triples to the following by including only compound entities:

RNA CCAT2 ; promotes ; breast cancer growth
RNA CCAT2 ; promotes ; breast cancer metastasis

This makes the system more extensible to open-domain text, since the level of information required can be easily and efficiently modified based on both user and developer need. We also use negation dependencies to give a polarity for each predicate, indicating whether the triple represents a true or a false relation. For example, *was not contained in* would be modified to *was contained in [False]*.

The first stage of our system outputs these annotated triples, which can be fed into the next stage of the pipeline. For this prototype, we take the use-case of the OncoMX database to demonstrate how this information can be used to augment a structured database.

After the triples are extracted from the PubMed article's abstract and title, we link them to the Uberon anatomical ontology and the OncoMX biomarkers database, in order to insert them into the relational database. To do this, we take the annotated triple, and generate a set of additional entities, varying by level of information. For example, with the entity *long non-coding RNA CCAT2*, we have:

long non-coding RNA CCAT2
RNA CCAT2
CCAT2
long non-coding CCAT2

Which represents the power set of all components contained in the entity (since the number of rules is constant, this remains computationally efficient, while a simple token-based procedure would have exponential complexity). With this set, we search through the Uberon ontology and OncoMX biomarkers database for possible matches. Because we break down the entity itself, we can use a hash-table form of the ontology and database to facilitate constant-time searching, as opposed to linear-time searching if the entity was not annotated. In this example, we find that the gene CCAT2 is contained in the biomarkers set, and we can link this entity with the associated ID for this gene. If a more specific match is found using the additional components of the entity, we would instead select the more descriptive option.

For the final stage, we take all extracted triples in which both the subject and object are linked to both the Uberon ontology and the biomarkers database. For instance, if we have the title from the PubMed paper 28105220:

Overexpression of THY1 Is Associated With Metastasis in Human Gallbladder Carcinoma

Our system would produce as a final output:

pubmed_id: 28105220
gene: THY1
anatomical_entity:UBERON:0002110
subject: overexpression of THY1
predicate: is associated with
object: metastasis in human gallbladder carcinoma
polarity: true

In order to augment OncoMX with additional information extracted from the PubMed medical research database, we add our triples as a supplementary table to the OncoMX relational database. In order to do this, we link the subject and object of our triples with the **biomarker** and **anatomical_entity** tables, shown below:

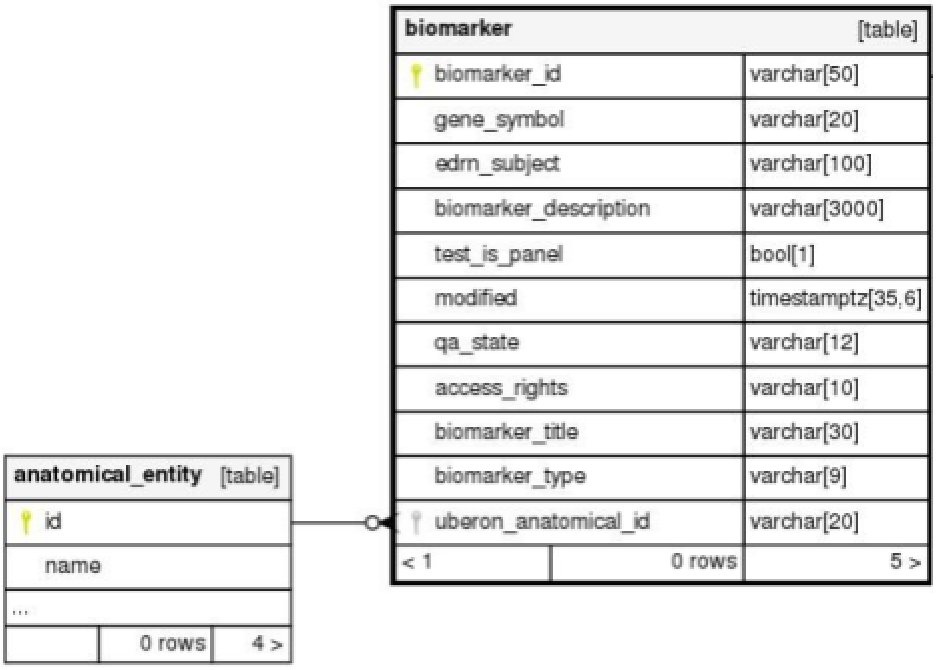


Figure 22: The OncoMX anatomical entity and biomarker tables.

By attaching the gene-symbol from the natural language text to an entity, we can link each triple to the existing database using the *biomarker_id*, meaning that each existing biomarker is now augmented with supplementary structured information. For the Uberon ontology, we use the **anatomical_entity** table, and link each triple using its associated Uberon ID.

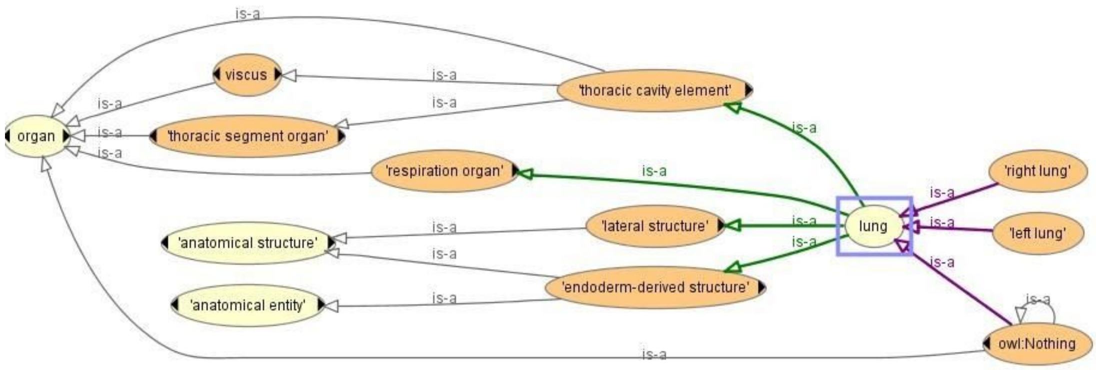


Figure 23: Example of the UBERON ontology.

Uberon is a hierarchical anatomical ontology containing ID-numbered anatomical entities. As described above, we associate the subject or object of each triple with a node in the Uberon ontology. Currently, we consider all nodes which are sub-classes of the **anatomical_entity** node. The structure of the ontology is shown above with the example node **lung**. When matching to the ontology, we attempt to select the most specific node, in order to minimize information loss. For instance, we link to the lower node **pancreatic duct**, rather than the higher node **pancreas**. Each node also contains a list of synonyms and relational adjectives - for example, **lung** can also be matched via **pulmonary**. The following table shows the number of nodes contained in the ontology, and the number of additional nodes created using the provided synonyms and relational adjectives. We also show the number of triples in which both the subject and object are linked to a biomarker and a Uberon node (fully-linked), and the number of triples linked to either the subject or object only (partially-linked).

Total Nodes	11741
Total Synonym Nodes	35622
Fully-Matched Triples	3638
Partially-Matched Triples	19765

Table 1: Triple information from the ZHAW engine.

When the best-match node for an entity has been found, we look up its associated ID, and use this to annotate either the subject or object of the extracted triple. In this manner, we can leverage information extracted from unstructured text to augment an existing database and ontology with additional relations.

B. The INF Information Extraction Engine

The INF information extraction pipeline is used to extract Open Information Extraction (OIE) triples (S-P-O) from PubMed articles and map them to Uberon and OncoMX concepts. The INF information extraction pipeline comprises the following steps:

- an **in-place neural coreference resolution** process: Given that our information retrieval task requires the extraction of dependency relations from sentences, i.e. sets of the form {subject, predicate, object}, and that in many cases the entity is replaced with its coreferential pronoun we consider in-place coreference resolution as a crucial pre-processing step on the each article's body text, to improve the quality of the extracted triples.
- a **parallel triple extraction** process as our core information extraction method: We integrated triple extraction based on multiple OIE engines, relying on the complementarity of different information retrieval approaches (clause-based, learning-based, embeddings-based, etc.) to counter the loss of structural and semantic information.
- an **entity enrichment and cleaning** process: Our pipeline concludes with a series of post-processing activities, including linking the extracted entities to existing ontologies, performing polarity detection on the phrases related to each triple as well as cleaning the duplicate triples that were extracted via the parallel execution of the aforementioned OIE engines.

The graphical summary of our pipeline is as follows:

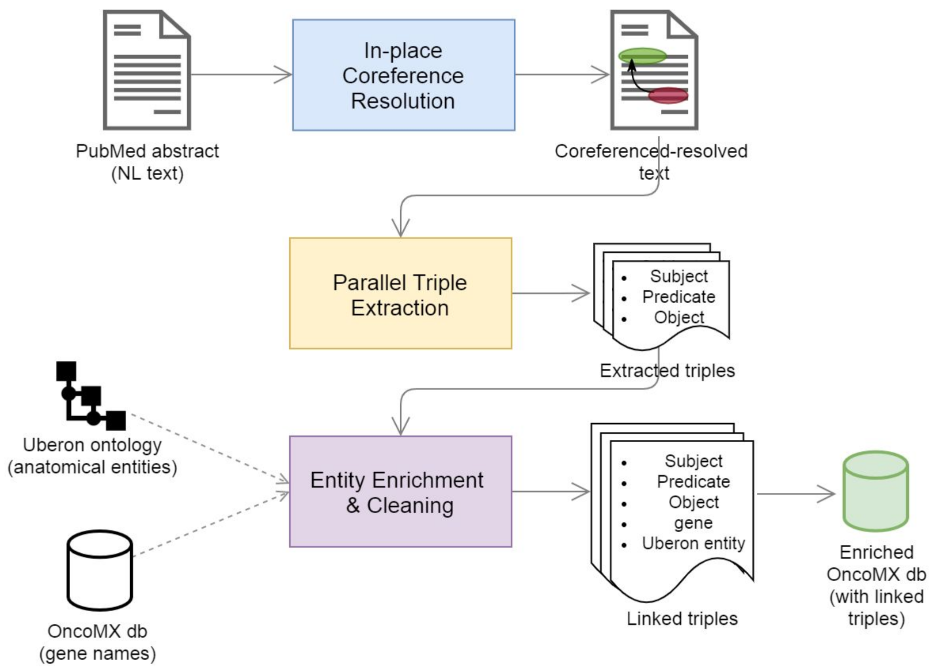


Figure 24: INF Information extraction pipeline overview.

A sample of extracted triples linked to the Uberon and OncoMX concepts is shown below:

title	id	subject	predicate	object	sentiment	sent_num	triple_num	engin	uberon	biomarker
Methylenetetrahydrofolate Reductase and Thymid...	15510613	methylene tetrahydrofolate reductase (MTHFR)...	may be	risk factors for breast cancer because of Func...	positive	286494	7	2	UBERON:0000310	MTHFR
Evaluation of Seven Different Tumour Markers f...	2627971	squamous cell carcinoma antigen SCC cancer ant...	were analyzed	in sera in order to create tumor marker panels...	positive	277343	1	1	UBERON:0000479	CRP
Impact of Helicobacter Pylori Infection on the...	17691020	a significantly higher level of MUC1 IgG than ...	was observed	p irrespective of H. pylori status or stage of...	positive	220219	1	1	UBERON:0000178	MUC1
MicroRNA-224 Promotes Tumor Progression in Non...	26187928	TNFÎ ±	be induced	protein 1 TNFAIP1 and SMAD4	positive	126715	1	1	UBERON:0008933	TNFAIP1
Molecular Regulation of S100P in Human Lung Ad...	18575778	an upregulation of S100P in lung adenocarcinomas	was determined	for early\/T1 stage but not more advanced\/T2 ...	negative	212717	2	1	UBERON:0002048	S100P

Figure 25: Sample of extracted triples linked with Uberon entities and OncoMX genes/biomarkers from the INF engine.

We used the extracted information to populate two new tables which were added to the Postgres OncoMX database. The format of the extracted triples is aligned to that of the ZHAW information extraction engine to ensure compatibility. The new tables have the following properties:

- ***triples_fully_linked***: contains extracted triples that are linked to both a Uberon entity and an OncoMX gene/biomarker. We extracted 2,843 such triples.
- ***triples_partially_linked***: contains extracted triples that are linked only to gene/biomarker. We extracted 18,538 such triples.

Sample rows of *the* ***triples_fully_linked*** table containing triples from both the INF and ZHAW engines, are shown below:

tri_ful_key [PK] integer	pmid integer	gene text	uberon text	uberonname text	subject text	predicate text	object text	polarity text	source text
3142	25527410	XRCC5	UBERON:0008933	primary somatosensory cortex	L 0 and 1 repeats anc	alleles	The alleles of VNTR XRCC5 pol	True	infili
2237	18089803	RET	UBERON:0008933	primary somatosensory cortex	Germ line-activating	allow	this receptor to signal indep	True	infili
218	28249601	DKK3	UBERON:0001235	adrenal cortex	silencing of negative	allows	dedifferentiation of adrenal	True	zhaw
219	28249601	DKK3	UBERON:0001851	cortex	silencing of negative	allows	dedifferentiation of adrenal	True	zhaw
223	28249601	DKK3	UBERON:0002369	adrenal gland	silencing of negative	allows	dedifferentiation of adrenal	True	zhaw
1792	25103640	IFI27	UBERON:0000310	breast	Interferon	alpha-inducible	protein 27 IFI27 is an interf	True	infili
663	17063264	MTHFR	UBERON:0000310	breast	functional polymorphi	alter	risk of breast	True	zhaw
983	25563194	XRCC1	UBERON:0002048	lung	risk for lung cancer	among are	variant Arg399Gln of XRCC1	True	both
982	25563194	XRCC1	UBERON:0002048	lung	risk for lung cancer	among are	variant Arg194Trp of XRCC1	True	both
61	15941951	AMACR	UBERON:0035944	life-death temporal boundary	risk of prostate canc	Among was highe	those with low AMACR score	True	zhaw
508	17210081	IGFBP3	UBERON:0000310	breast	IGFBP3 expression in	Among was highe	patients with benign breast d	True	zhaw
60	15941951	AMACR	UBERON:0035944	life-death temporal boundary	risk of prostate canc	Among was highe	those with low AMACR expressi	True	zhaw
737	23838800	PDCD4	UBERON:0000479	tissue	PDCD4 expression	Among was lower	different differentiated canc	True	zhaw
2452	12820330	TP53	UBERON:0000479	tissue	Screening for TP53 mu	amplification (	to analyze TP53 mutations	True	infili
2453	12820330	TP53	UBERON:0001088	urine	Screening for TP53 mu	amplification (	to analyze TP53 mutations	True	infili
1996	23135313	CRP	UBERON:0000479	tissue	C-reactive protein CF	amyloid	A SAA are acute inflammatory	True	infili
2517	10940270	MUC5AC	UBERON:0008933	primary somatosensory cortex	Apomucin (MUC2 , MU(	analysed	using single and double immu	True	infili
2909	20037207	CYP1B1	UBERON:0000310	breast	COMT and CYP1B1 polym	analyzed	employing polymerase chain r	True	infili
1848	24481866	TNF	UBERON:0002876	nucleus intercalatus	plasma levels of IL-2	analyzed	In this study	True	infili
2350	16424981	MTA1	UBERON:0000310	breast	Immunohistochemical e	analyzed	quantitatively by a novel a	True	infili
1827	24737289	SELENBP1	UBERON:0000479	tissue	Samples of cancer tis	analyzed	for SELENBP1 expression by 2D	True	infili
2349	16424981	MTA1	UBERON:0000479	tissue	Immunohistochemical e	analyzed	quantitatively by a novel a	True	infili
1826	24737289	SELENBP1	UBERON:0000344	mucosa	Samples of cancer tis	analyzed	for SELENBP1 expression by 2D	True	infili

Figure 26: Sample rows of the triples_fully_linked table. The highlighted uberonname column corresponds to each coded UBERON entity.

2.2.2 Enriching the SIRIS database by linking CORDIS projects based on their natural-language Objectives

We leverage the natural language text of each project stored in the SIRIS database to find semantic neighbours of the existing CORDIS projects, based on their vector representation similarities. We then enrich the SIRIS database with the discovered neighbour pairs.

We based our work on the CORDIS database provided by SIRIS in SQL (Postgres) format:

- We focused only on the **unics_cordis.projects** table, which contains –among other fields- the projects acronyms, titles, objectives, unics_id, call etc.
- We aggregated information (NL text) from 3 sources: the **project title**, its **objective** and **call** to create a corpus for each project. That corpus was used as the basis of our entity matching method.
- Stopword cleaning was performed on the corpus (using NLTK).
- Each project's corpus was encoded to its vector representation using fastText embeddings
- We used the acquired semantic representations of each project to find the n=**3** closest neighbours (based on angular distance).
- We created a table consisting of project pairs and their in-between distance based on their NL information. Each project is paired with 3 other projects (its 3 closest neighbours).

A graphical summary of our methodology is as follows:

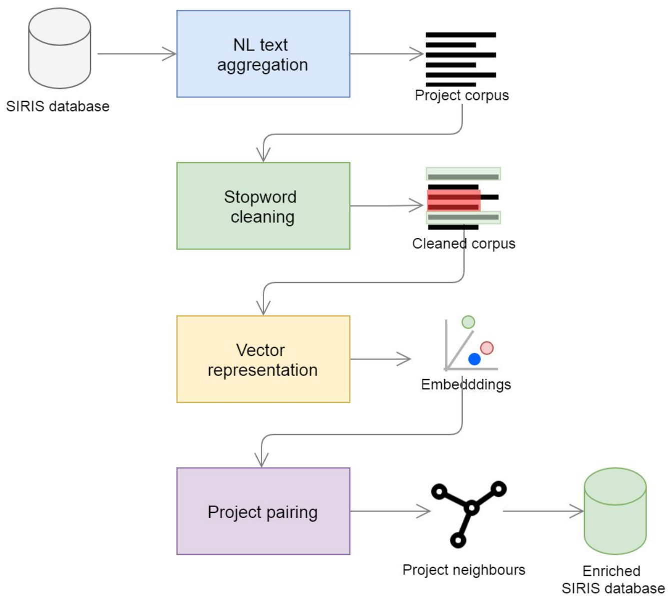


Figure 27: Methodology overview.

For the 50.823 CORDIS projects, we created 152.469 pairs (3 * 50.823) representing the 3 closest semantic neighbours of each project. The created project pairs were added to the existing SIRIS database as a new ***project_neighbours*** table.

	project [PK] integer	neighbour [PK] integer	distance numeric
541	217716	996820	0.07558257
542	996820	217716	0.07558336
543	878580	220675	0.07574264
544	220675	878580	0.07574264
545	163602	153085	0.07601548
546	153085	163602	0.07601548
547	220670	175897	0.0766945
548	175897	220670	0.0766945
549	174304	171709	0.077428475
550	171709	174304	0.077428475
551	982280	220530	0.07879272
552	220530	982280	0.07879348
553	156644	155708	0.07881089

Figure 28: Sample rows from the created "project_neighbours" table.

A list of examples follows:

- Closest neighbours of TRESSPASS (a smart border control project) are also related to border control and screening processes:

```
catted.loc[catted['v1_acronym'] == 'TRESSPASS']
```

	v1	v2	dist	v1_unics_id	v2_unics_id	v1_title	v2_title	v1_acronym	v2_acronym
32532	10844	21733	0.471906	887445	149117	robusT Risk basEd Screening and alert System f...	Protection of European seas and borders throug...	TRESSPASS	PERSEUS
32533	10844	23058	0.473214	887445	147544	robusT Risk basEd Screening and alert System f...	Research on EGNOS/Galileo in Aviation and Terr...	TRESSPASS	EGALITE
32534	10844	42255	0.475244	887445	887579	robusT Risk basEd Screening and alert System f...	CBRNE Detection in Containers	TRESSPASS	COSMIC

Figure 29: TRESSPASS project neighbours.

- Closest neighbours of SOLARGAIN (related to solar heat gain control films for energy-efficiency) are also solutions w.r.t energy efficient constructions:

```
catted.loc[catted['v1_acronym'] == 'SOLARGAIN']
```

	v1	v2	dist	v1_unics_id	v2_unics_id	v1_title	v2_title	v1_acronym	v2_acronym
59958	19986	16673	0.368324	147245	150288	Low-cost switchable reflective polymeric solar...	High thermal insulating window frames for ener...	SOLARGAIN	THINFRAME
59959	19986	12681	0.376281	147245	185207	Low-cost switchable reflective polymeric solar...	Energy efficient greenhouse dehumidifier for w...	SOLARGAIN	Drygair20
59960	19986	42013	0.378171	147245	221982	Low-cost switchable reflective polymeric solar...	Development and Validation of an Innovative So...	SOLARGAIN	SWS-HEATING

Figure 30: SOLARGAIN project neighbours.

- Closest neighbours of SOLUS (optical and ultrasound diagnostics of breast cancer) are also diagnostic and biopsy solutions for chest diseases:

```
catted.loc[catted['v1_acronym'] == 'SOLUS']
```

	v1	v2	dist	v1_unics_id	v2_unics_id	v1_title	v2_title	v1_acronym	v2_acronym
1866	622	27671	0.361932	216445	172588	Smart optical and ultrasound diagnostics of br...	Laser and Ultrasound Co-Analyzer for thyroid n...	SOLUS	LUCA
1867	622	1	0.371948	216445	148282	Smart optical and ultrasound diagnostics of br...	Breast biopsy system guided by Positron Emissi...	SOLUS	MAMMOCARE
1868	622	1259	0.383238	216445	224633	Smart optical and ultrasound diagnostics of br...	Self-Healthcare for breast cancer detection us...	SOLUS	SHINE

Figure 31: SOLUS project neighbours.

2.2.3 Mapping-Patterns Bootstrapper (MPBoot)

Manually writing ontologies and mappings starting from the relational schema of one or more available data sources is a tedious and error-prone process. For this reason, in INODE our objective is to automate as much as possible the generation of an ontology and mappings that are well suited for extracting data from the available data sources.

MPBoot takes as input a configuration file, containing the connection parameters to a relational data source, and produces an ontology and mappings that reflect how the data is organized within the data source.

In its current form, MPBoot adheres to the W3C Direct Mapping Recommendation. The bootstrapped ontology is a direct translation of the relational schema into the OWL language, and the bootstrapped mappings link elements in the DB schema to their corresponding OWL translations in the bootstrapped ontology.

As a minimal example, assume the following table in the database:

Person

SSN	Name	Address
001	Alice	here
002	Bob	there

The bootstrapper will create in the ontology a class Person, whose individuals are built out of the primary key of Person (SSN), and with three data properties corresponding to the three attributes of the Person table. More precisely, the following ontology and target part of a VKG mapping are generated:

```
Prefix Declaration:

@PREFIX : http://www.ongology.org/

Ontology:

<!-- Class declaration corresponding to the DB entity “Person” -->
<owl:Class rdf:about=":Person"/>

Mappings:

target:  :person/{SSN}  rdf:type Person .
         :person/{SSN} :SSN      {SSN} .
         :person/{SSN} :Name     {Name} .
         :person/{SSN} :Address {Address} .
```

The “VKG-setting” above, consisting of a class definition in the ontology and three data properties, when combined with a source part of the mapping that simply retrieves the tuples in the Person table, would generate the following (virtual) RDF triples:

```
:person/001 rdf:type Person .
:person/001 :SSN “001” .
```

```
:person/001 :Name “Alice” .
:person/001 :Address “here” .

:person/002 rdf:type Person .
:person/002 :SSN “002” .
:person/002 :Name “Bob” .
:person/002 :Address “there” .
```

MPBoot is invoked through the command-line-interface of Ontop. For instance, the command we used to bootstrap the Skyserver ontology was the following:

```
./ontop bootstrap -p boot-skyserver-dr16.properties \
-b 'http://www.semanticweb.org/skyserver'
-m boot-skyserver-dr16.obda \
-t boot-skyserver-dr16.owl
```

where the “-p” option indicates the file containing the connection parameters to the Skyserver database, the “-b” option corresponds to our prefix declaration above, the “-m” option indicates the output mapping file, and the “-t” option indicates the output ontology file.

In Section 4 below, when talking about data models, we will provide and discuss visualizations for the ontologies (Cordis, Skyserver, and Oncomx) bootstrapped through MPBoot.

3 API Specification

In this section we provide the API specification of INODE 1.0.

3.1 OpenDataDialog

In this subsection we describe the API documentation of the current status of the integrated OpenDataDialog that we demonstrated in Section 2.

3.1.1 NL-to-SQL and SQL-to-NL

Here we discuss the currently implemented and integrated functionality for translating a natural language question to SQL and for explaining generated SQL-queries in natural language.

NL-to-SQL

This request starts the systems {nalir+, soda} translation of a natural language query to a specified number of SQL queries. There is no 1 to 1 NL-SQL mapping due to the inherent ambiguity of natural language queries.

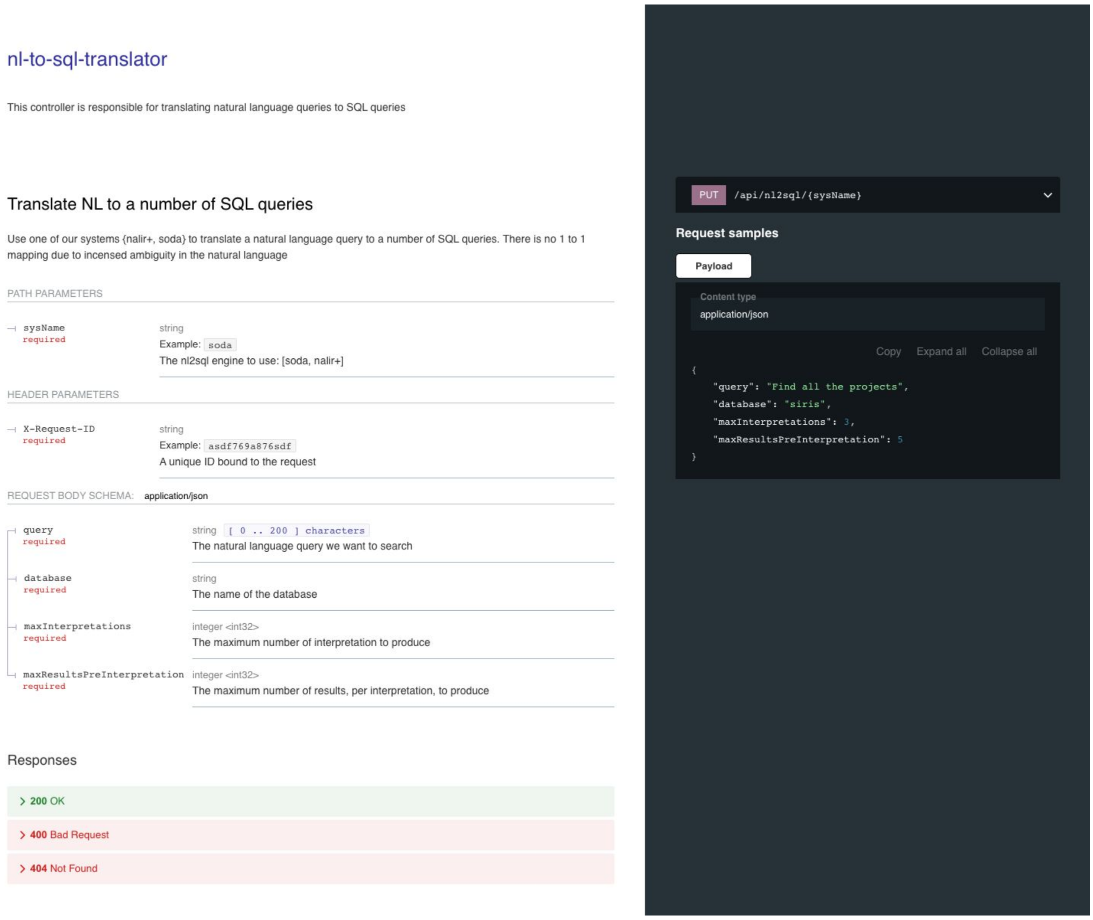


Figure 32: NL to SQL translation.

This request stops the translation process bound to the 'X-Request-ID' unique id.

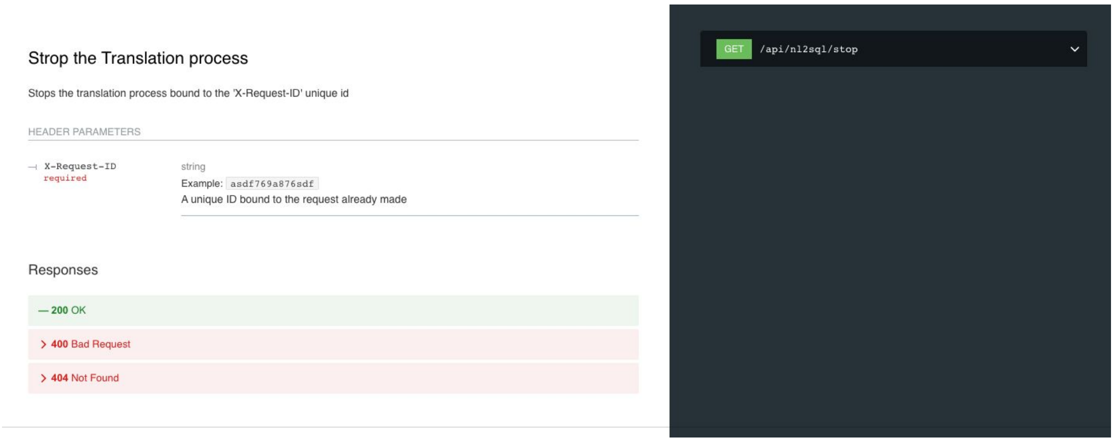


Figure 33: Stopping the translation process.

SQL-to-NL

The sql-to-nl-translator translates natural language queries to SQL queries.

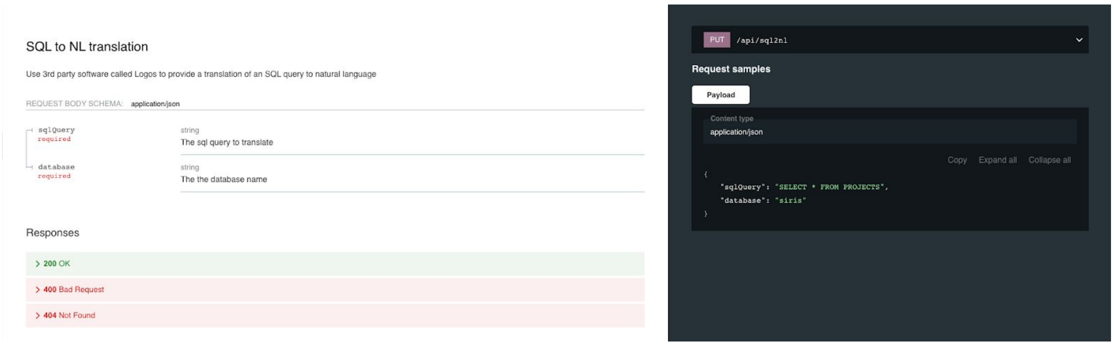


Figure 34: SQL to NL translation.

SQL Parser

This request parses the sql query into a json format (currently unable to handle nesting).

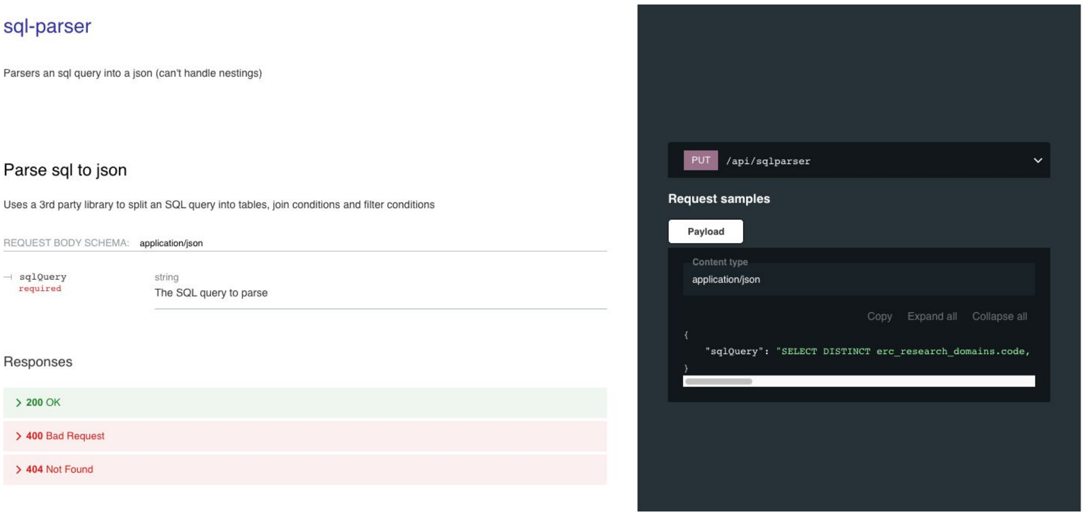


Figure 35: SQL to JSON.

SQL Executor

The SQL to tabular data request uses an SQL query to retrieve data from an underlying RDBMS system (PostgreSQL) and returns the data in a tabular format.

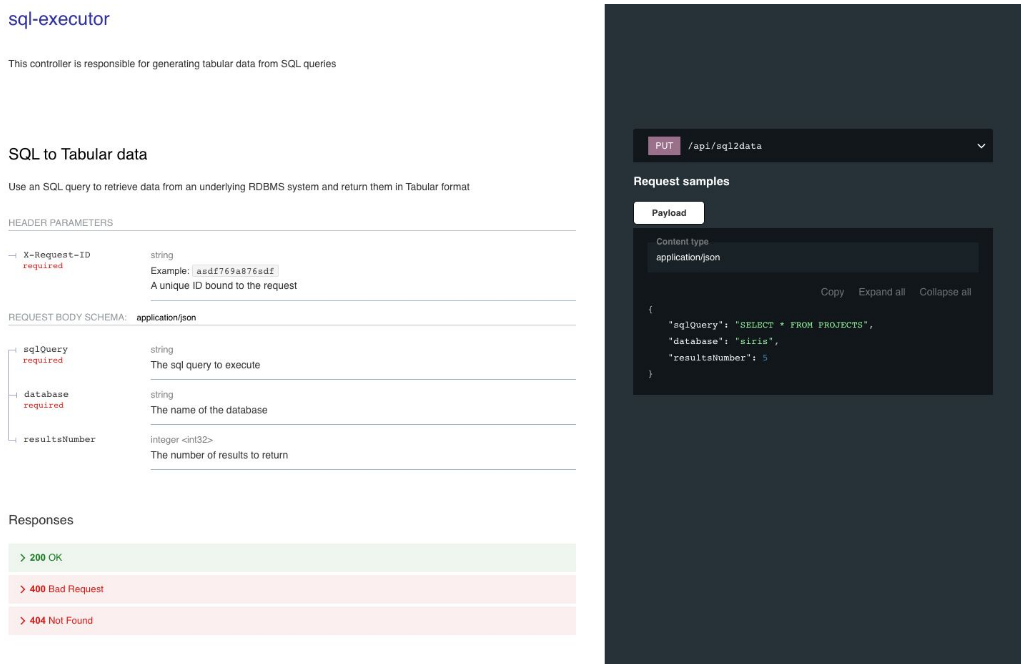


Figure 36: SQL to tabular data.

The stop execution request stops the execution of the query using the unique 'X-Request-ID' bound with it.

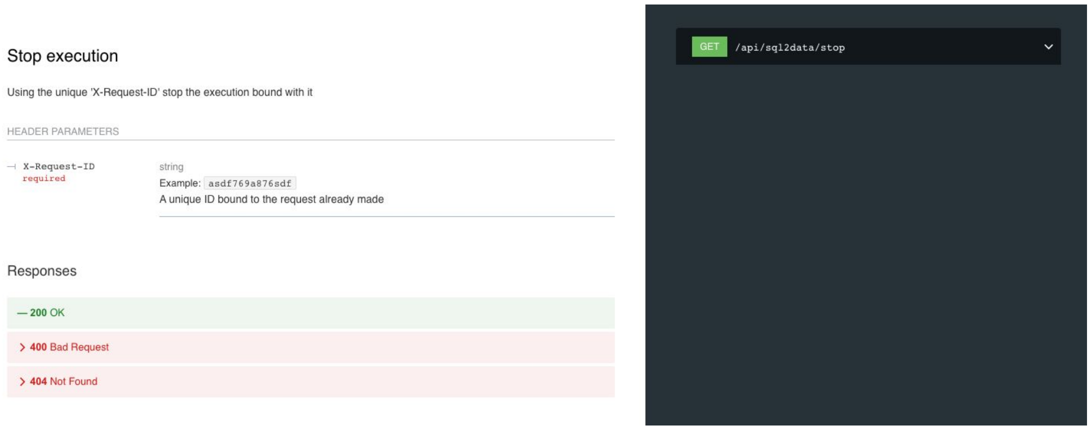


Figure 37: Stopping the execution process.

3.1.2 MultiTable Visualization

To make the INODE vision a reality, Fraunhofer had two major goals for the initiation period of the project: (1) Visualize the data in a way, that provides a better overview over the result than plain tables, and (2) add interaction capabilities to enable the user to explore the results using pipeline operators.

For (1) we integrated methods into the pilot that run the relevant algorithms on the basis of queries, which are generated as outputs of nl2sql translations and pipeline operators. Although these algorithms are mainly used internally by the pilot to post-process queries, they are also exposed as a separate endpoint (see Figure 32 below).

For (2), we added functionality to the MultiTable Visualization to trigger operations on tables, columns, rows and individual cells of the result tables.

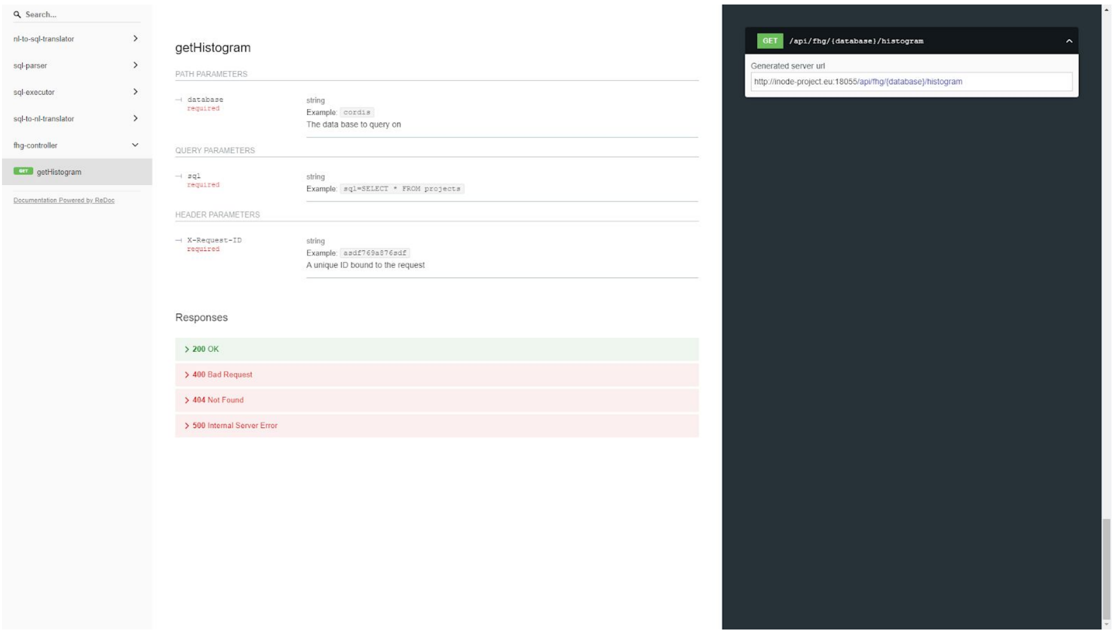


Figure 38: The getHistogram API endpoint produces a data table which is enriched by metadata which allows the depiction of histograms and other data distributions in the frontend without exceeding the browser capabilities.

3.1.3 Pipeline Operators

Here we discuss the currently implemented and integrated pipeline operators.

The pipeline operators are designed to manipulate and explore a database iteratively, while seeing the data as multiple layers of overlapping sets. Each operator takes a single set defined by a conjunction of attribute/value equalities as input, and returns one or multiple sets defined by a conjunction of attribute/value equalities, depending on the operation.

A REST API was designed to make them available to the pilot. As the pilot is only able to handle SQL queries to define the data to display, each input set is described as a parsed SQL query, telling in which tables the data is located, and which attribute/value filters are defining it, and each output set is described as an SQL query, resulting in the set data if executed.

Here is the API documentation for each of these operators:

by filter

By filter allows the user to filter a given set on an additional attribute-value.

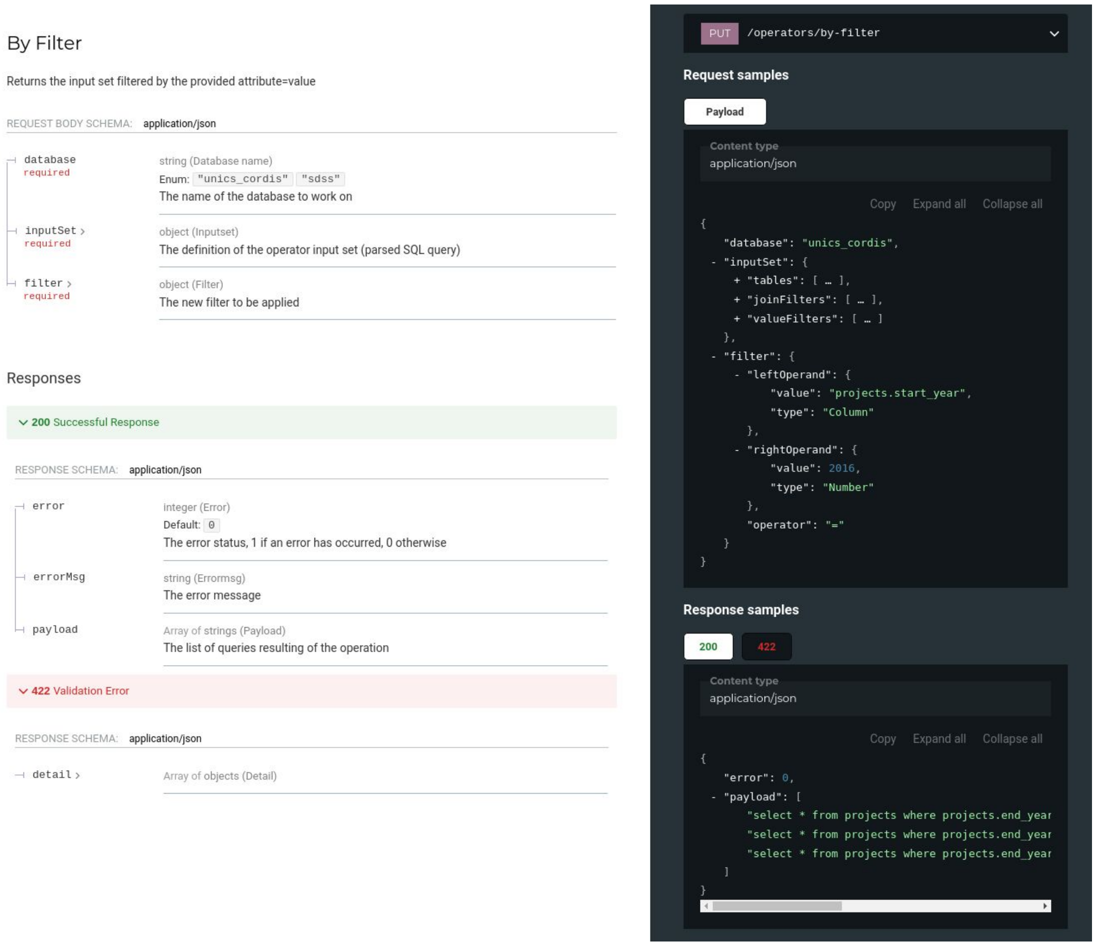


Figure 39: By filter operator.

by superset

By superset allows the user to get a wider scope on the data by returning the smallest set completely overlapping with a given set.

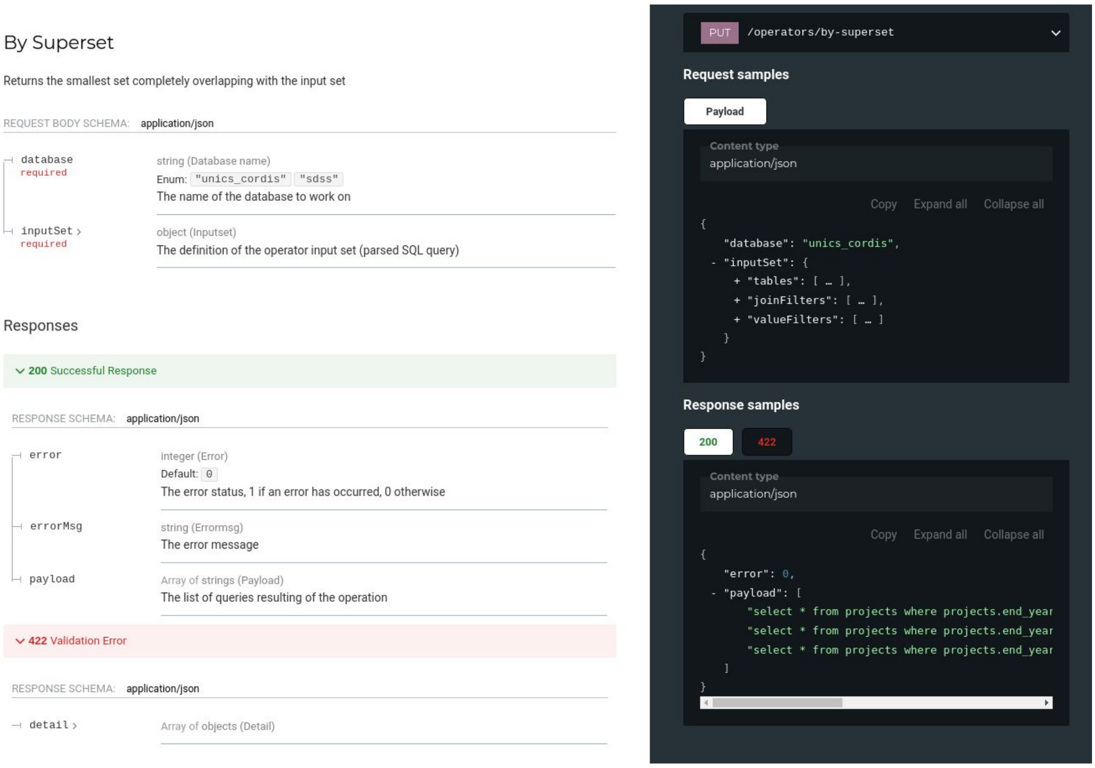


Figure 40: By superset operator.

by overlap

By overlap allows the user to find multiple neighbouring sets to a given set. It looks for the sets with the minimum overlap to the input set and to each other.

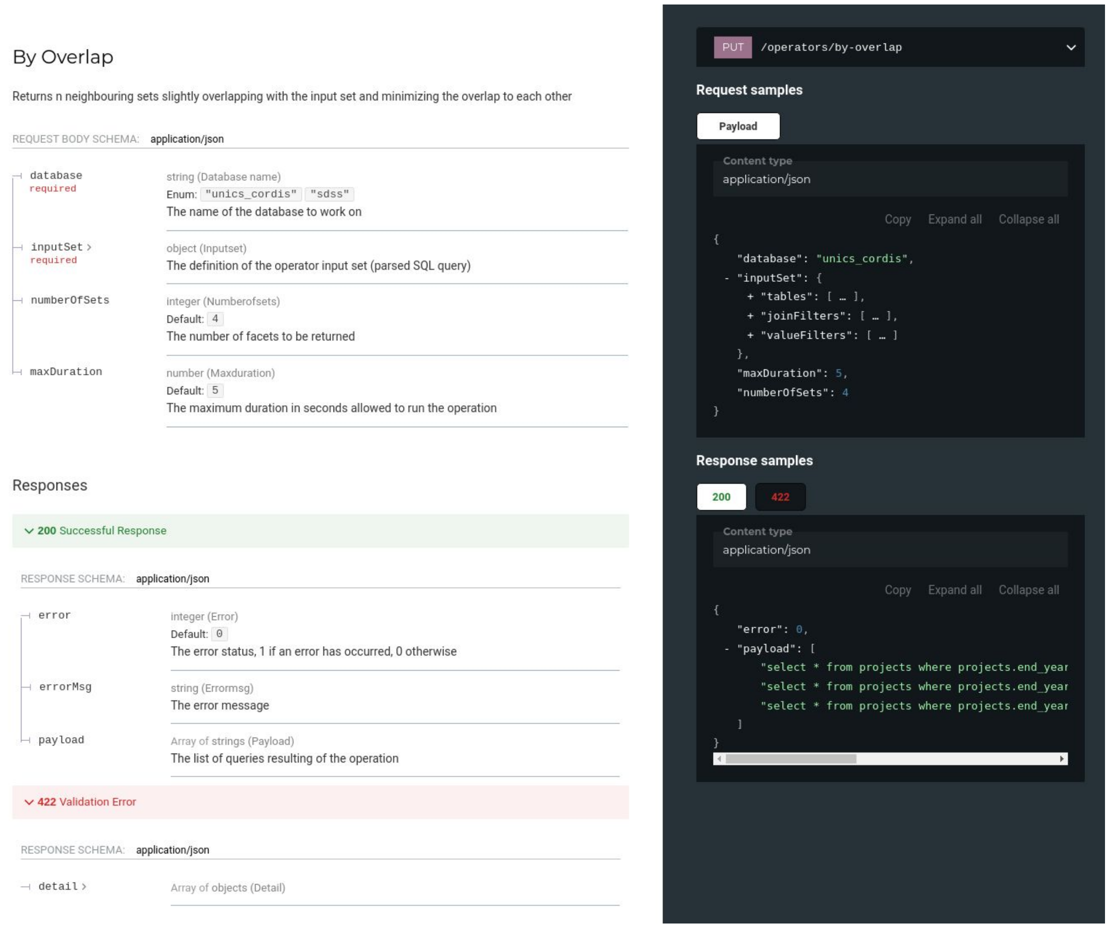


Figure 41: By facet operator.

by facet

By facet allows the user to group an input set data by a list of attributes, and returns the biggest groups.

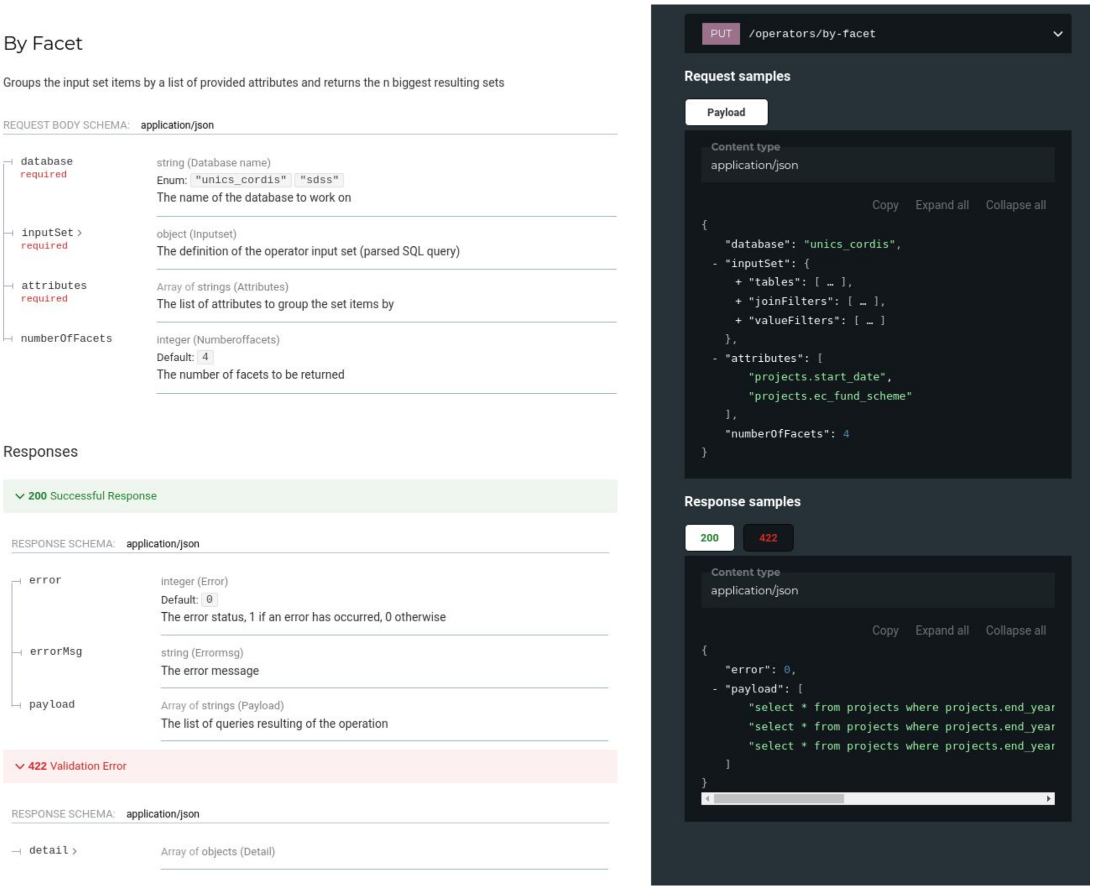


Figure 42: By facet operator.

3.2 OpenDataLinking

Two engines (ZHAW, INF) perform triple extraction on NL-text using different approaches. The endpoints (path, method, request, response) are the same for both engines.

OpenDataLinking Triple Extraction from NL-text Endpoint (REST Endpoint)

- **Path & Method:** URL/extract/ **&** POST request

- **Description:** The triple extraction engine receives a JSON file containing PubMed abstracts as input, performs the aforementioned NLP operations depending on the engine (e.g. syntactic parsing, coreference resolution, parallel triple extraction and triple cleaning), and returns two csv files including extracted triples based on the given input mapped to Uberon entities and Bionarkers.

A sample of a valid JSON file containing PubMed abstracts is given below:

- Sample JSON input file:

{"25584213": {"abstract": {"Background": "Smoking has been considered to be the major cause of lung cancer. However, only a fraction of cigarette smokers develop this disease. This suggests the importance of genetic constitution in predicting the individual's susceptibility towards lung cancer. This genetic susceptibility may result from inherited polymorphisms in genes controlling carcinogen metabolism and repair of damaged deoxyribonucleic acid (DNA). These repair systems are fundamental to the maintenance of genomic integrity. X-ray repair cross complimenting group I (XRCC1), a major DNA repair gene in the base excision repair (BER) pathway. It is involved in repair by interacting with components of DNA at the site of damage. Inconsistent results have been reported regarding the associations between the Arg399Gln polymorphism of XRCC1. This study demonstrates the importance of recognition of this relationship of lung carcinoma and genetic constitution of the person which will help guide clinicians on the optimal screening of this disease.", "Aim": "To assess the role of XRCC1 gene polymorphism (Arg399Gln) directly on the variation in susceptibility to development of lung cancer in North Indian subjects.", "Materials and methods": "One hundred males with diagnosed cases of lung cancer were recruited from Delhi State Cancer Institute (DSCI). Hundred healthy volunteers were taken as controls. DNA isolation was done and Polymerase chain reaction-Restriction Fragment Length Polymorphism (PCR-RFLP) procedure undertaken to amplify the region containing Arg/Gln substitution at codon 399 (in exon 10).", "Results": "XRCC1 gene polymorphism is associated with increased risk of lung cancer when the Arg/Arg genotype was used as the reference group. The Arg/Gln and Gln/Gln was associated with statistically increased risk for cancer.", "Conclusion": "Arg399Gln polymorphism in XRCC1 gene polymorphism is associated with lung cancer in North Indian subjects and screening for this polymorphism will help in targeting predisposed individuals and its prevention."}, "title": "XRCC-1 Gene Polymorphism (Arg399Gln) and Susceptibility to Development of Lung Cancer in Cohort of North Indian Population: A Pilot Study"}}

- **Request Body:**

The format of the input:

```
curl -X POST "http://localhost:8000/extract/" -H  "accept: application/json" -H  "Content-Type: multipart/form-data" -F "filename=@/filepath/file.json"
```

-**Response:**

Two csv files:

- ***fully_linked_triples_demo_YYYYMMDD_hhddss.csv*** : which contains extracted triples that are linked to both a Uberon entity and a gene/biomarker.
- ***partially_linked_triples_demo_YYYYMMDD_hhddss.csv***: which contains extracted triples that are linked only to a gene/biomarker.

-**Example:**

A valid request followed by a response:

Request (The client uses a POST request to initiate triple extraction on a JSON file):

```
infili@lab:~$ curl -X POST "http://localhost:8000/extract/" -H  "accept: applicati
on/json" -H  "Content-Type: multipart/form-data" -F "filename=@/home/infili/transl
ation/DimPapSandbox/triple-extraction/pubmed_sample2.json"
```

Response (The server performs triple extraction and the extracted triples mapped to genes and Uberon entities):

Server side:

```
    .................................................................
    .................................................................
    ...............INODE - Parallel Triple Extraction - .............
    .................................................................
    ..............................................Noima-EN_v0.2.

Loading dependencies...
** Server ready! **


Information extraction job started at 2020-07-28 08:58:00

Performing coreference resolution...
100%|████████████████████████████████████████| 1/1 [00:00<00:00,  4.13it/s]
Coreference resolution completed.

Starting triple extraction...
1it [00:00,  1.14it/s]
Triple extraction completed.

Cleaning duplicate triples...
Cleaning completed.

Starting biomarker mapping...
100%|████████████████████████████████████| 809/809 [00:00<00:00, 1375.94it/s]
Biomarker mapping completed.

Starting UBERON mapping...
100%|██████████████████████████████████| 15183/15183 [00:05<00:00, 2710.99it/s]
15183it [00:11, 1305.57it/s]
UBERON mapping completed.

Partially linked triples saved in: partially_linked_triples_demo_20200728-115824.csv and fully linked
triples saved in: fully_linked_triples_demo_20200728-115824.csv

Job finished in 24.76 seconds
```

Client side (received triples):

```
infili@lab:~$ curl -X POST "http://localhost:8000/extract/" -H  "accept: applicati
on/json" -H  "Content-Type: multipart/form-data" -F "filename=@/home/infili/transl
ation/DimPapSandbox/triple-extraction/pubmed_sample2.json"
["[{\"pmid\":25584213,\"gene\":\"XRCC1\",\"subject\":\"Arg399Gln polymorphism in X
RCC1 gene\",\"predicate\":\"is\",\"object\":\" associated\",\"polarity\":\"positiv
e\",\"source\":\"infili\"}]","[{\"pmid\":25584213,\"gene\":\"XRCC1\",\"uberon\":\"
UBERON:0002048\",\"subject\":\"Arg399Gln polymorphism in XRCC1 gene polymorphism\"
,\"predicate\":\"associated\",\"object\":\" with lung cancer in North Indian subje
cts and screening for Arg399Gln polymorphism\",\"polarity\":\"positive\",\"source\
":\"infili\"}]"]infili@lab:~$
```

A bad request followed by a response:

Request (The client uses a POST request to initiate triple extraction on a non-existent JSON file):

```
infili@lab:~$ curl -X POST "http://localhost:8000/extract/" -H  "accept: application/json" -H  "Content-Type: mu
ltipart/form-data" -F "filename=@/home/infili/translation/DimPapSandbox/triple-extraction/nonexistent_file.json"
```

Response (The server responds with an error):

```
^Cinfili@lab:~/translation/DimPapSandbox/triple-extraction$ uvicorn tripleAPI-POST:app --reload --log-level error

    .................................................................
    .................................................................
    ...............INODE - Parallel Triple Extraction - .............
    .................................................................
    ..................................................Noima-EN_v0.2.

Loading dependencies...
** Server ready! **

ERROR:    Error getting request body:
```

Client side:

```
Warning: setting file
Warning: /home/infili/translation/DimPapSandbox/triple-extraction/nonexistent_f
Warning: ile.json  failed!
curl: (26) read function returned funny value
```

Automated documentation for the above POST request (using ReDoc):

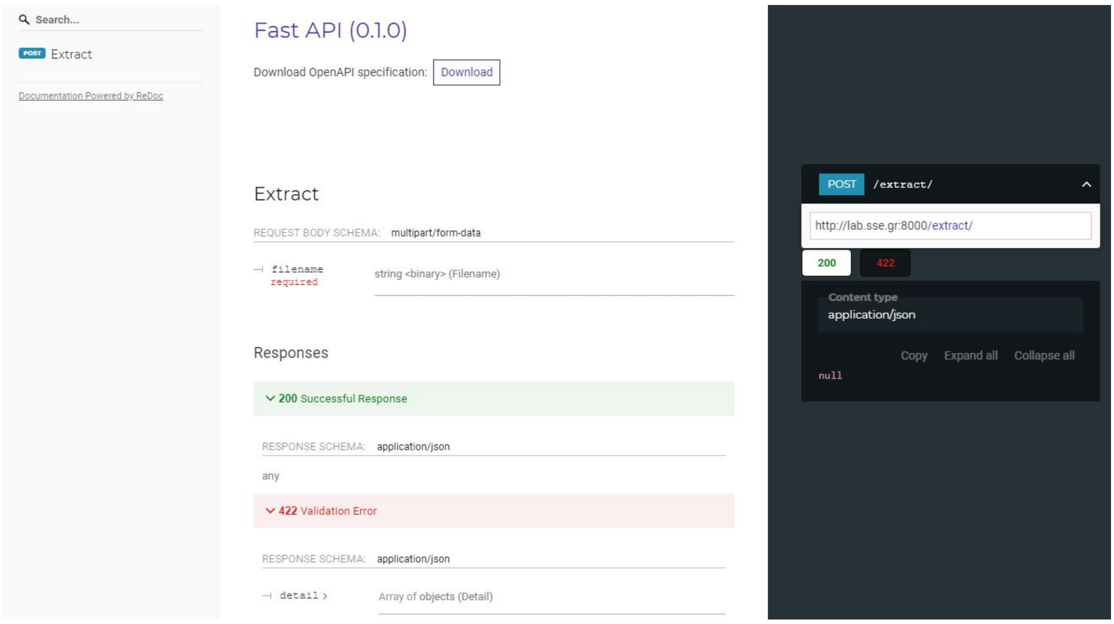


Figure 43: Download API.

4 DATA MODELS

In this section we provide the entity-relationship diagrams and corresponding ontology views for the three use cases *research & innovation policy making*, *astrophysics* and *cancer research*.

4.1 Research & Innovation Policy Making

An initial version of the Cordis database has been developed by Siris. Such a database has been augmented with a table "project_neighbours", containing pairs of projects with similar objectives which have been derived through the automated analysis of texts in natural language. The final database schema is as follows

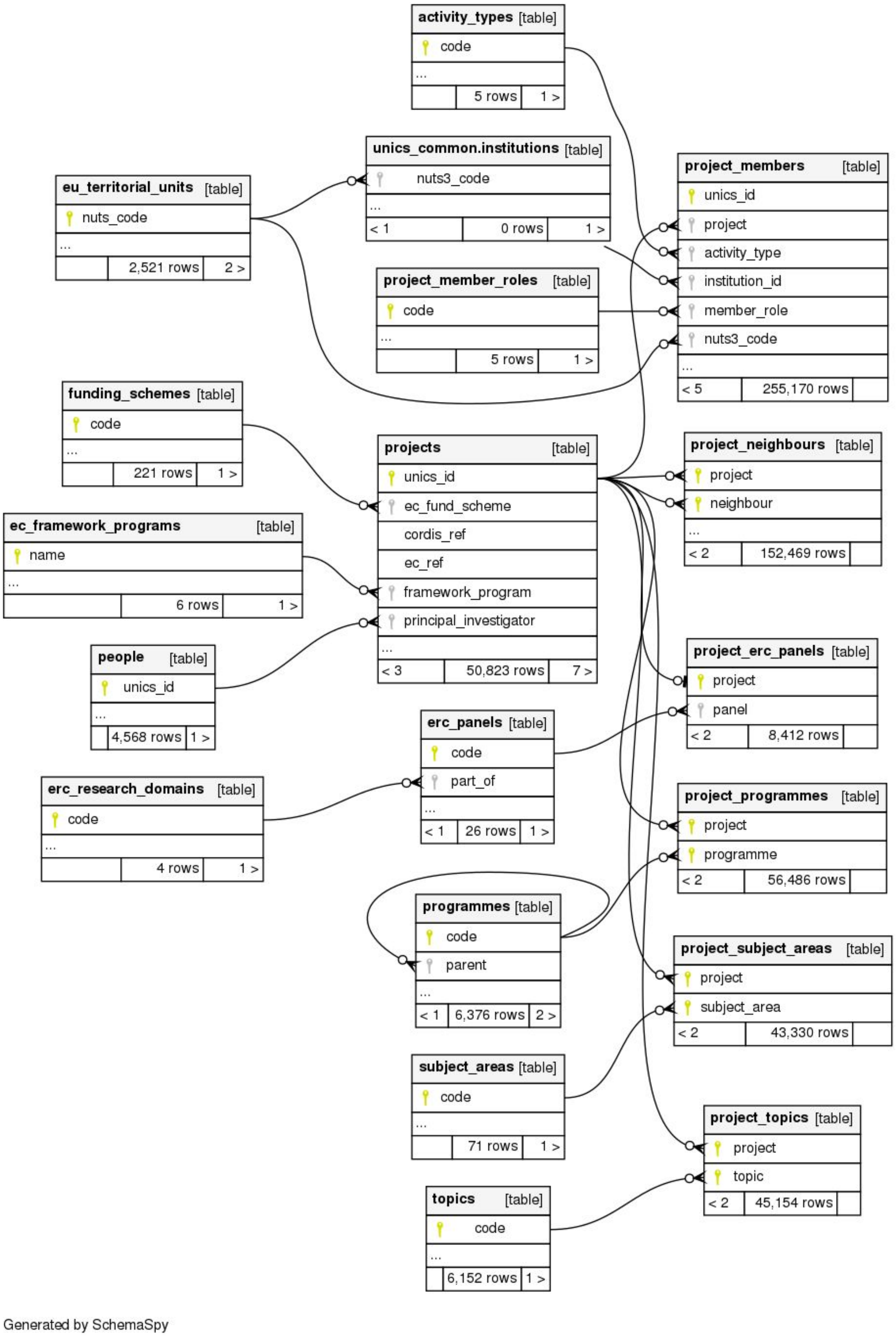


Figure 44: Cordis database schema.

An ontology and relative mappings have been manually-crafted by Siris. The ontology has the following structure (graph view):

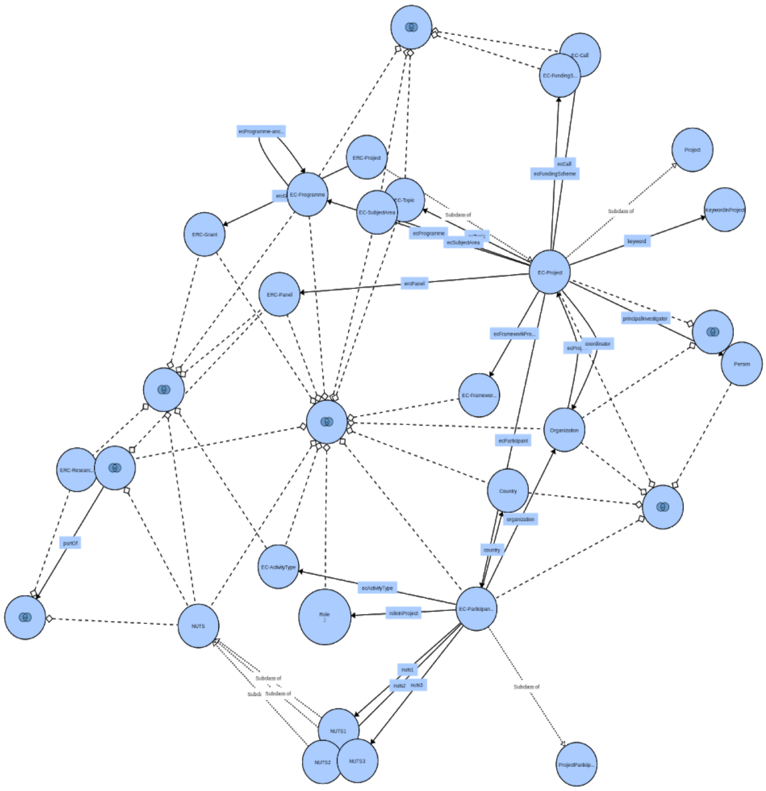


Figure 45: Cordis ontology.

Nodes in such a graph represent classes in the ontology, and links between them represent object properties between two classes (i.e., object properties whose domain and range have been declared). In order not to overload the figure with too much information, we have left the data properties out of the visualizations (and of all visualizations in this section).

We used MPBoot to automatically bootstrap an ontology for the Cordis database:

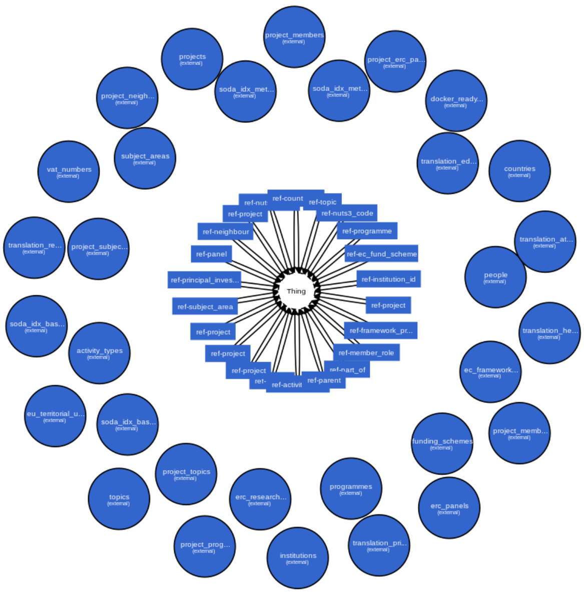


Figure 46: Cordis bootstrapped ontology.

By using MPBoot, we can significantly speed up the time needed to generate ontologies. However, we can observe that the structure of the bootstrapped ontology looks quite poor when compared to the manually crafted ontology. One reason for this is the fact that the W3C direct mapping recommendation, and therefore MPBoot, does not specify domains and ranges of object properties. Therefore, all such object properties appear in our visualization as connecting objects of the class "Thing", which is a superclass of all other classes in the ontology. This is one of the limitations of the Direct Mapping approach that we plan to overcome in the remaining part of the project.

To overcome this specific limitation, we will consider the foreignkey-constraints linking different tables in the database, and we will encode them through suitable OWL domain and range axioms. We observe that currently MPBoot already uses these foreign keys to generate object properties. More in general, we plan to consider different forms of patterns that are present in the combination of keys and foreign keys of the database, and generate corresponding combinations of OWL axioms that capture at best the semantics of the data encoded through these patterns of database constraints. Importantly, we will consider not only constraints that are explicitly declared in the database, but also constraints that can be derived by analyzing the actual data.

4.2 Astrophysics

For this scenario, we have limited the Skyserver-data (DR16) to a portion of the sky and selected 5 tables of particular interest. Some of these tables were actually views, which were missing the specification of foreign key-constraints in the schema. Since foreign keys are crucial in order to understand the structure of the original data, we have added manually those that we could infer from the available information. More specifically, some of the foreign keys could be inferred by considering the view definitions, together with the constraints specified on the database tables appearing in those views. Others were discovered by actually looking at the data. As an output of all these activities, we devised the following database schema:

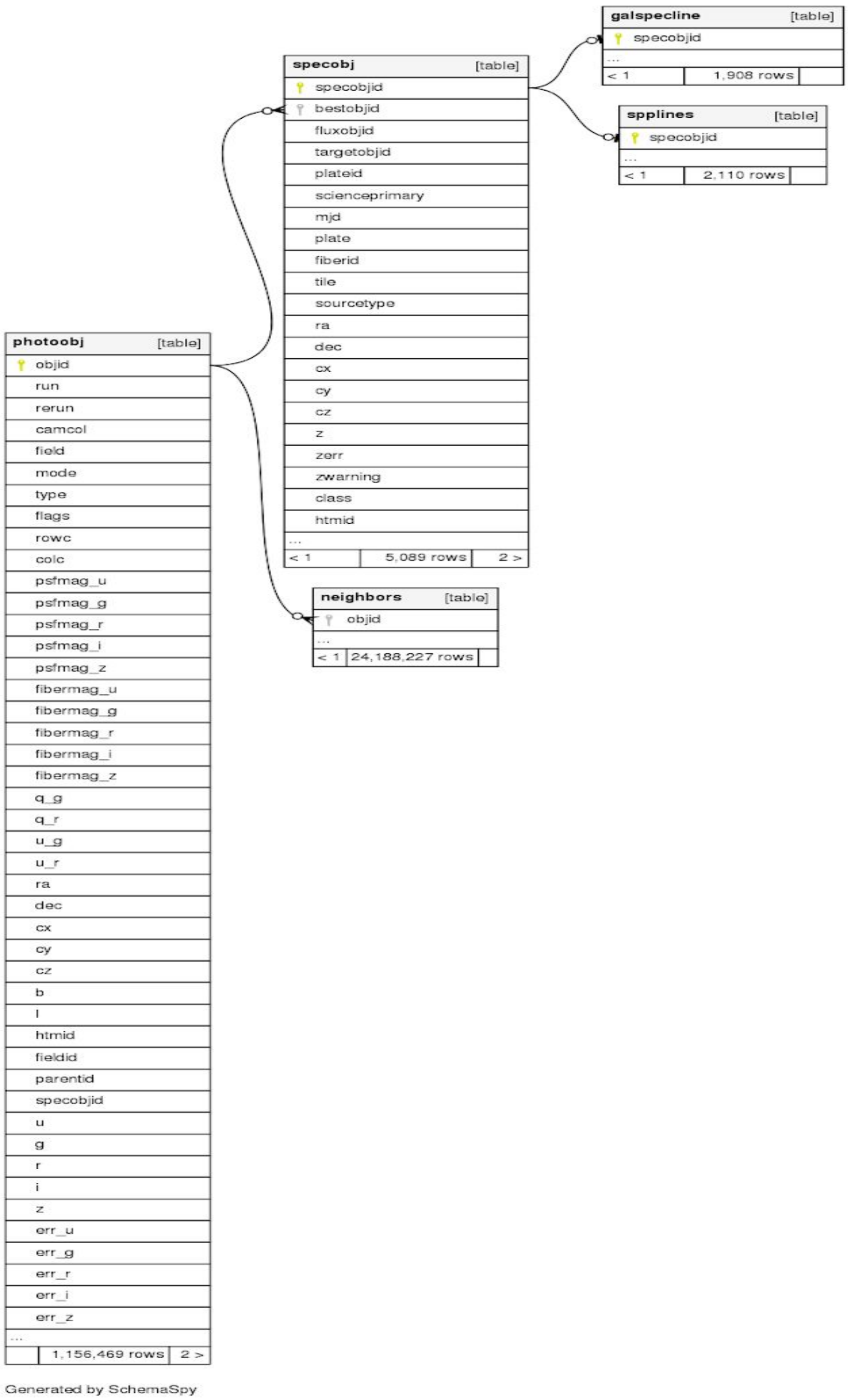


Figure 47: SDSS database schema.

We began to manually craft an ontology for the Skyserver scenario, along with mappings connecting the ontology elements to queries over the Skyserver database schema. At the moment, the ontology consists of 29 classes, 2 object properties, and 43 data properties:

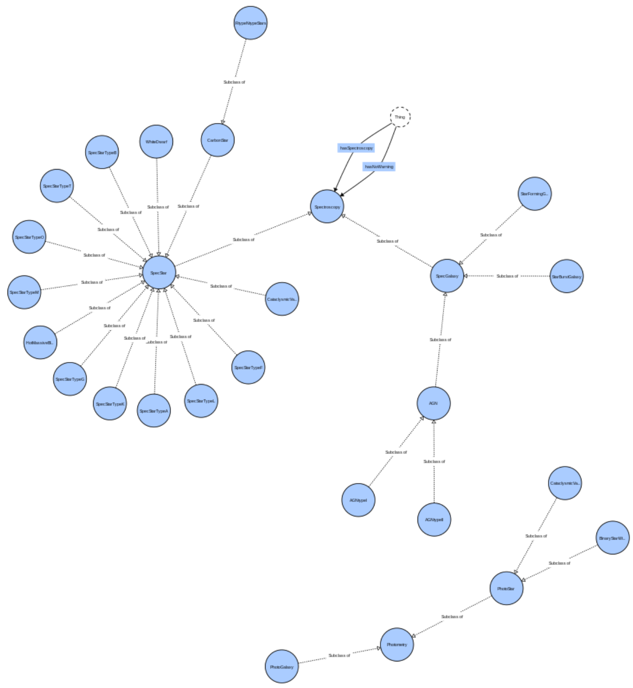


Figure 48: SDSS ontology.

The bootstrapped ontology, produced by MPBoot, again, displays a quite poor structure:

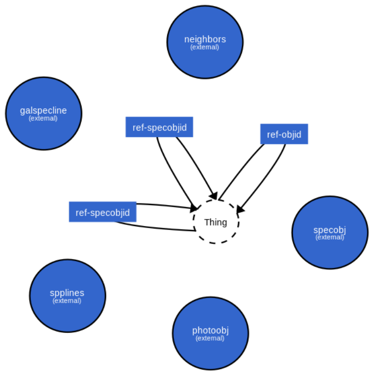


Figure 49: SDSS bootstrapped ontology.

For the same considerations as in the Cordis ontology, we decided to leave data properties out of the visualization. The plan of improving MPBoot sketched for the Research & Innovation setting is general, in the sense that it is not tailored towards that specific scenario only, and applies also to this scenario.

4.3 Cancer Research

Entity-Relationship Diagram

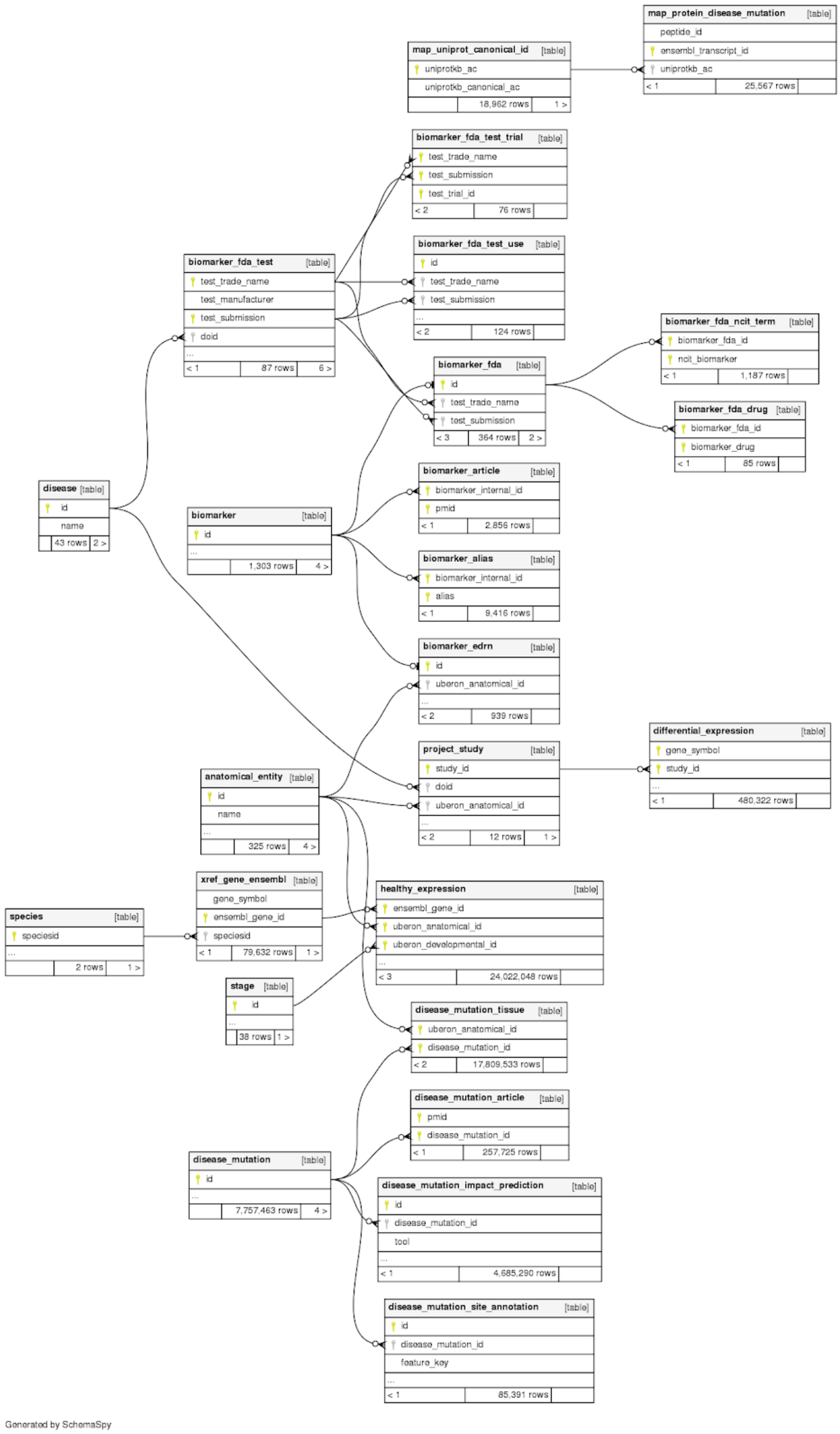


Figure 50: Cancer Biomarker database schema.

Automatically-Generated Ontology with MPBoot (class view)

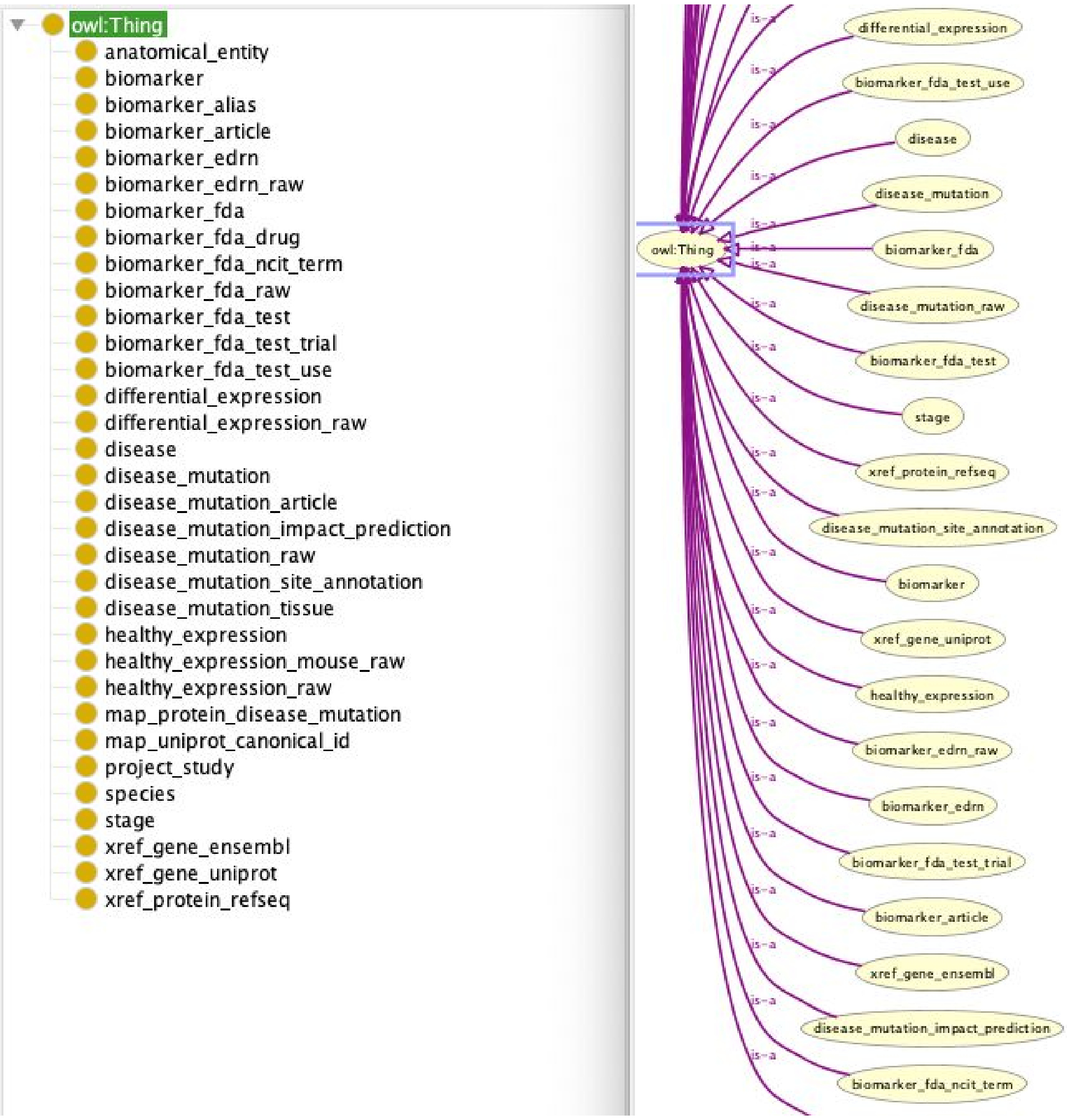


Figure 51: Cancer Biomarker bootstrapped ontology.

A Portion of the In-progress Hand-Crafted Ontology

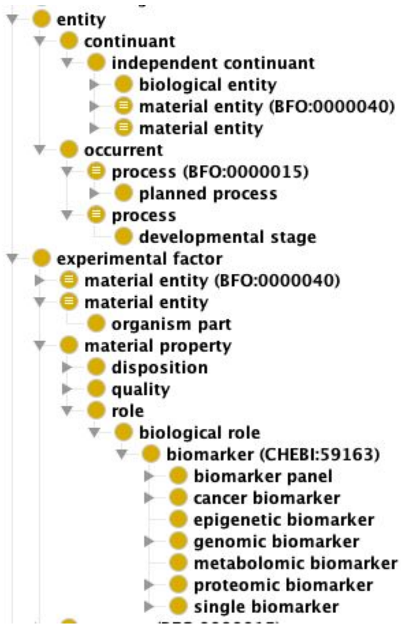


Figure 52: Cancer Biomarker hand-crafted ontology.

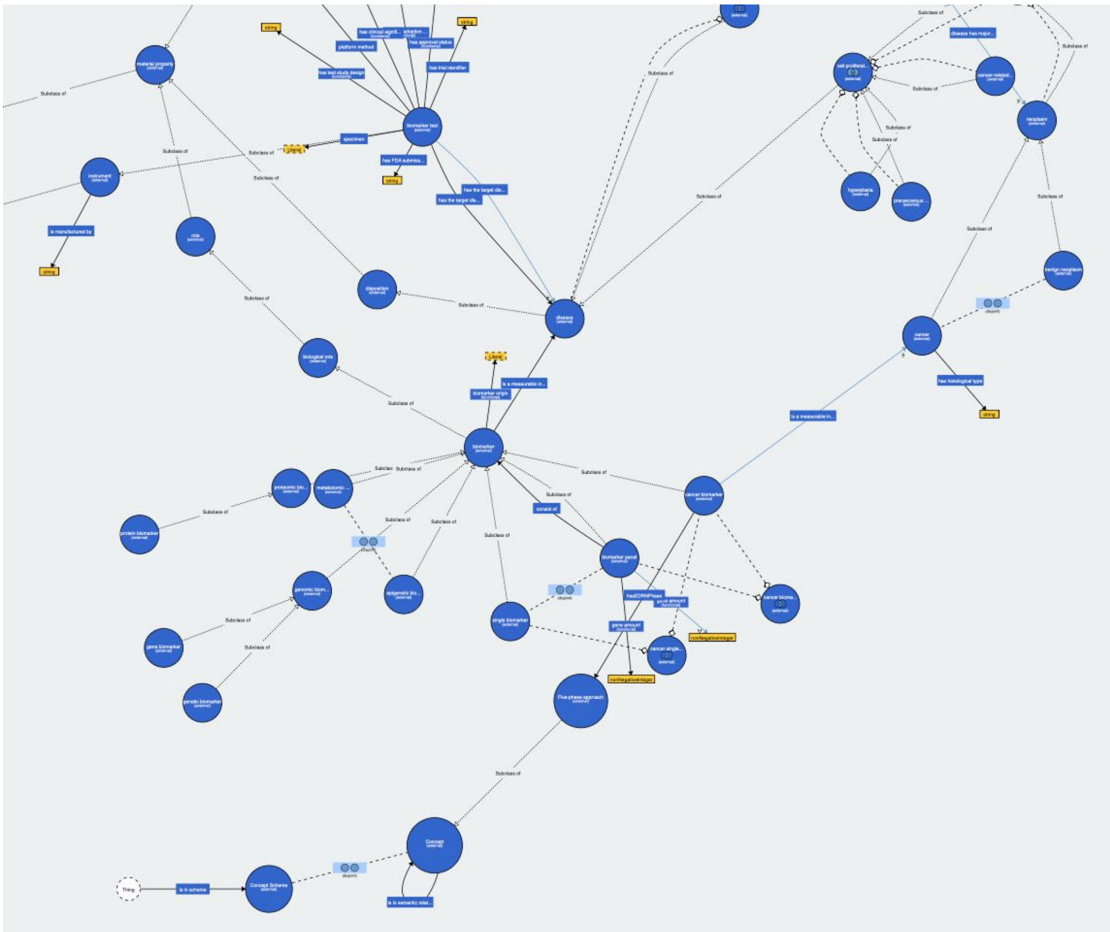


Figure 53: Visualizing the hand-crafted Cancer Biomarker ontology.