

Master degree in
Intelligent and Interactive Systems
Modul 39-M-Inf-P Projekt
Modul 39-M-Inf-RDM

DDRUM: Drugs, Diseases, RDF und MeSH



Andrea Loretto

Andrea Schinoppi

Contents

1	Introduction	3
1.1	Task description	3
1.2	Use cases	3
2	Data Management Plan	5
2.1	Project description	5
2.1.1	Topic: what is the main research question?	5
2.1.2	Research field	5
2.1.3	Responsible for the project coordination	5
2.1.4	Project partners and funding	5
2.1.5	Duration	5
2.2	Content classification	5
2.2.1	Datasets: What kind of data are you working with?	5
2.3	Technical classification	6
2.3.1	Data size: What is the size of the data?	6
2.3.2	Formats: which file formats are used?	6
2.4	Data usage	6
2.4.1	Data organisation	6
2.5	Metadata and referencing	7
2.5.1	Which standards, ontologies, classifications etc. are used to describe the data and context information?	7
2.6	Legal and ethics	7
2.6.1	Personal data: does the data contain personal data?	7
2.6.2	Licenses	7
2.7	Data exchange	8
2.8	Responsibilities and duties	8
2.9	Costs	9
2.10	Resources	9
3	Proposed solution	10
3.1	Early phases	10
3.2	Implemented solution	11
3.2.1	RDF graph creation	11
3.2.2	SPARQL database and frontend	11
3.3	Experiments and reproducibility	15
4	Conclusions	17
4.1	Future work	17

A	Technical documentation	18
A.1	Adding queries	18
A.2	How the RDF graph is generated	19
A.2.1	Drug Entity and Properties	19
A.2.2	Connecting Drugs with Clinical Trial Files	19
A.2.3	MeSH Information	20

Introduction

1.1 Task description

The aim of this project is to develop a system that integrates multiple medical databases and provides a user-friendly web interface for querying a unified SPARQL database derived from them. This integration will simplify the process of accessing and retrieving medical data across various sources, enhancing research and analysis capabilities for medical professionals and researchers.

1.2 Use cases

1. **Most used drugs to treat a condition:**

This use case involves querying the database to determine which drugs are most commonly used to treat a specific medical condition. It could be helpful for healthcare providers and researchers to understand the prevailing treatment options for a particular condition.

2. **Drugs used to treat a given condition:**

In this use case, the user specifies a medical condition, and the system retrieves a list of drugs commonly used to treat that condition. It provides valuable information for healthcare professionals when selecting appropriate medications for their patients.

3. **Studies (divided by phase) in which a drug is cited:**

This use case focuses on retrieving information about clinical studies where a specific drug is mentioned. The studies are categorized by their respective phases, providing insights into the research and development stages of the drug.

4. **Studies (divided by phase) in which a condition is cited:**

Similar to the previous use case, this one involves querying the database for clinical studies that reference a particular medical condition. The studies are categorized by phases, allowing users to explore research related to the condition across different stages.

5. **All studies citing a drug:**

This use case involves retrieving a comprehensive list of all clinical studies that mention a specific drug. It enables users to access a wealth of information about the drug's efficacy, safety, and usage across various research studies.

6. All studies citing a condition:

Similarly, this use case involves retrieving a comprehensive list of all clinical studies that reference a particular medical condition. It allows users to explore research findings related to the condition and its management.

Data Management Plan

2.1 Project description

2.1.1 Topic: what is the main research question?

This project aims to discover relations among databases containing drugs and diseases starting from scientific studies.

2.1.2 Research field

Life Sciences / Basic Biological and Medical Research

2.1.3 Responsible for the project coordination

- Prof. Dr. Philipp Cimiano
- Andrea Loretto
- Andrea Schinoppi

2.1.4 Project partners and funding

None

2.1.5 Duration

The duration of the project goes from 30th October 2023 to 29th February 2024.

2.2 Content classification

2.2.1 Datasets: What kind of data are you working with?

Existing data

Dataset “MeSH”: MeSH (Medical Subject Headings) is the NLM controlled vocabulary thesaurus used for indexing articles for PubMed. Database dumped on 29th December 2023 as .xml file containing all the available information about MeSH elements.

Dataset “ClinicalTrials”: ClinicalTrials.gov is a database of privately and publicly funded clinical studies conducted around the world.

Database dumped on 08th December 2023 consisting of .xml files representing studies information.

Dataset “DrugCentral”: DrugCentral is online drug information resource created and maintained by Division of Translational Informatics at University of New Mexico.

Database dumped in .tsv format on 10th December 2023.

Generated data

Dataset “DDRUM”: Serialized Turtle RDF graph containing relations among drugs and conditions found in studies.

Dataset created in the scope of this project.

2.3 Technical classification

2.3.1 Data size: What is the size of the data?

Dataset “MeSH”: exact size: 3.0 GiB

Dataset “ClinicalTrials”: exact size: 10.6 GiB

Dataset “DrugCentral”: exact size: 4.3 MiB

Dataset “DDRUM”: exact size: 41.2 MiB

2.3.2 Formats: which file formats are used?

Datasets “MeSH”, “ClinicalTrials” and “DrugCentral”: .xml

Dataset “DDRUM”: .ttl

2.4 Data usage

2.4.1 Data organisation

Where is the data stored?

Datasets “MeSH”, “ClinicalTrials” and “DrugCentral”: On a local machine, and a backup on a cloud service (OneDrive).

Dataset “DDRUM”: On a local machine and on Github¹.

¹<https://github.com/Stintipacchio/DDRUM>

Versioning

We will use Git for the versioning of project files.

2.5 Metadata and referencing

2.5.1 Which standards, ontologies, classifications etc. are used to describe the data and context information?

The standards used are:

- Python v.3.12.0;
- .ttl, .tsv and .xml file formats;
- schema.org properties.

Using Python we wrote a script to elaborate the data:

- The script traverses through all XML studies files and it extracts useful information.
- This information is used to create instances of the **Study** and **Drug** classes.
- The script reads drug information from the drugs .tsv file.
- It filters the data to include only drugs mentioned in the XML files and extracts relevant information.
- This information is then associated with the respective **Drug** objects created from the XML files.
- Finally, all the information gets serialized into a turtle format RDF graph.

2.6 Legal and ethics

2.6.1 Personal data: does the data contain personal data?

No dataset contains personal information.

2.6.2 Licenses

Dataset “MeSH”:

- https://www.nlm.nih.gov/databases/download/terms_and_conditions_mesh.html
 - Users of the data agree to:
 - * Acknowledge NLM as the source of the data in a clear and conspicuous manner,
 - * Not indicate or imply that NLM has endorsed its products/services/applications.
 - Users who republish or redistribute the data (services, products or raw data) agree to:
 - * Maintain the most current version of all distributed data, or
 - * Make known in a clear and conspicuous manner that the products/services/applications do not reflect the most current/accurate data available from NLM and/or identify the version of MeSH being used.

Dataset “DrugCentral”:

- Creative Commons Attribution Share Alike (CC BY SA)

Dataset “ClinicalTrials”:

- <https://www.clinicaltrials.gov/about-site/terms-conditions>
 - Attribute the source of the data as ClinicalTrials.gov
 - Update the data such that they are current at all times
 - Clearly display the date the data were processed by ClinicalTrials.gov
 - State any modifications made to the content of the data, along with a complete description of the modifications

Dataset “DDRUM”:

- Creative Commons Attribution Share Alike (CC BY SA)

2.7 Data exchange

It will be possible to download the DDRUM database from the Github repository. Using a standard data format (.ttl), interoperability will be granted.

2.8 Responsibilities and duties

The people responsible for data management are:

- Andrea Loretto;

- Andrea Schinoppi.

2.9 Costs

The whole project will not have costs.

2.10 Resources

Dataset “MeSH”: <https://www.nlm.nih.gov/databases/download/mesh.html>

Dataset “DrugCentral”: <https://drugcentral.org/download>

Dataset “ClinicalTrials”:

<https://classic.clinicaltrials.gov/AllPublicXML.zip>

Dataset “DDRUM”:

<https://github.com/Stintipacchio/DDRUM/blob/main/out/output.ttl>

Proposed solution

3.1 Early phases

In the early phases of the project, we studied some documentation about RDF and SPARQL in order to have a better understanding of the topics.

Then we started looking for the data we needed to complete the project: we dumped MeSH, ClinicalTrials and DrugCentral databases (see chapter 2) and started looking into them to understand how to find relations among data.

Once we found some plausible associations among these databases, we wrote them down and decided to use Python to implement the project.

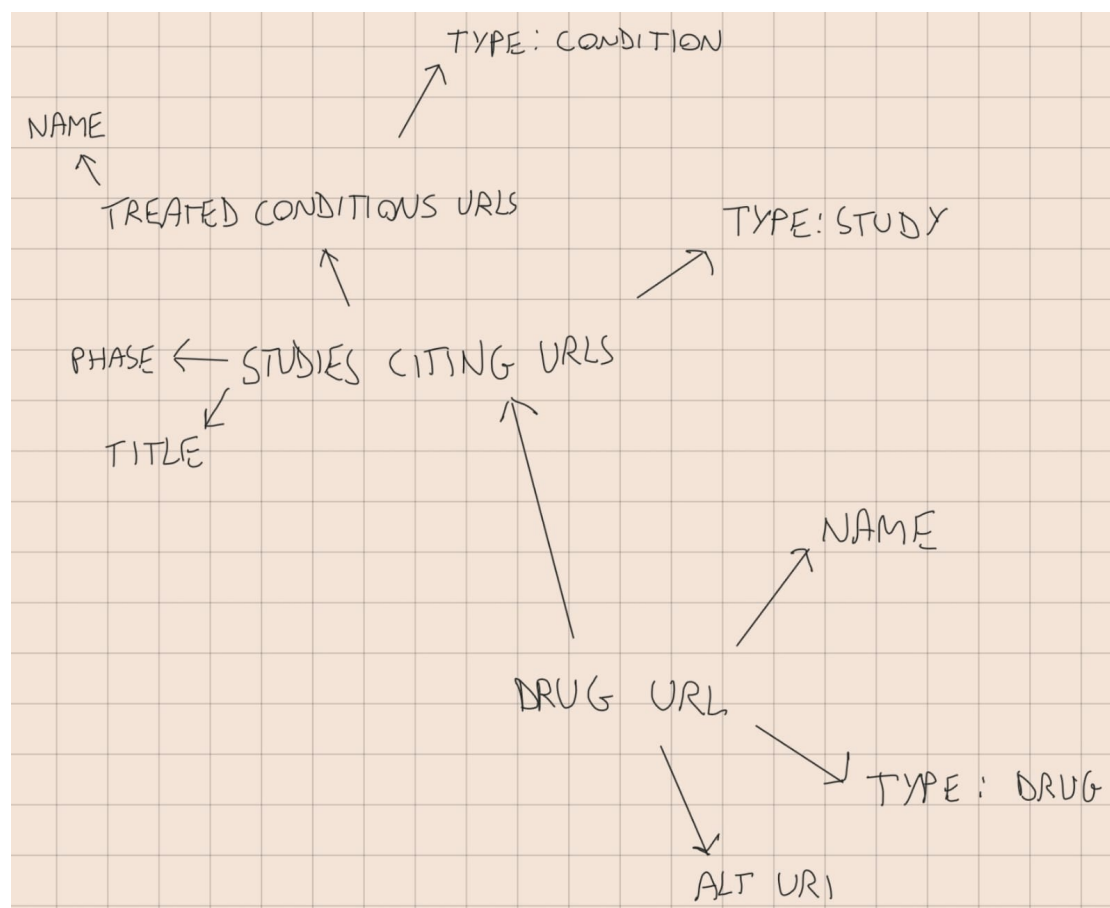


Figure 3.1: Early phases idea of the RDF graph

We then documented on schema.org RDF properties to be sure to use standard

properties to ensure compatibility.

3.2 Implemented solution

3.2.1 RDF graph creation

The implemented solution is based on a Python script designed to process XML files containing information about clinical trials and represent this data in RDF (Resource Description Framework) format.

The script utilizes libraries for file manipulation, XML parsing, data processing, RDF manipulation, and visualization.

It begins by defining three classes: `Study`, `Drug`, and `MeshElement` containing the necessary fields of the respective elements.

Then the script initializes an RDF graph and starts to process XML files.

Processing XML files involves extracting information about drug interventions, study details, and MeSH terms.

Then, information from the TSV file is filtered based on drugs mentioned in the XML files, and MOA URLs, activity URLs, and structural IDs are associated with corresponding drug objects.

MeSH information for drugs is subsequently extracted from a MeSH XML file and also linked to drug objects.

Finally, RDF triples are generated and the RDF graph is serialized into JSON-LD and Turtle formats.

Optionally, the script can generate an SVG visualization of the RDF graph to aid in visual understanding of entity relationships, but this results in a chaotic visualization for bigger graphs, so it has been disabled.

3.2.2 SPARQL database and frontend

The backend runs on a Flask application written in Python for querying a SPARQL endpoint hosted by a Fuseki server initialized with the serialized turtle database.

The Flask application defines routes for handling HTTP requests.

The “/” route renders an HTML document for the index page, which is the interface the user will use to query the server.

The “/drugs” route queries the SPARQL endpoint to retrieve all the distinct drug names.

The “/conditions” route retrieves all the distinct medical condition names. Finally, the “/submit” route handles POST requests containing query parameters and dynamically constructs SPARQL queries based on the request type, then forwards the correct query to the Fuseki server and sends the response to the frontend, which renders it.

User interface

We decided to realize a simple user interface using HTML and CSS with JQuery and Bootstrap:

DDRUM
Drug Diseases RDF Und MeSH

Author: [Andrea Loretta](#)
Author: [Andrea Schinoppi](#)
License: [CC BY-SA 4.0](#)

Select a Form:

Most used drugs to treat a condition ▼

Search for a condition

Maximum number of results to display:

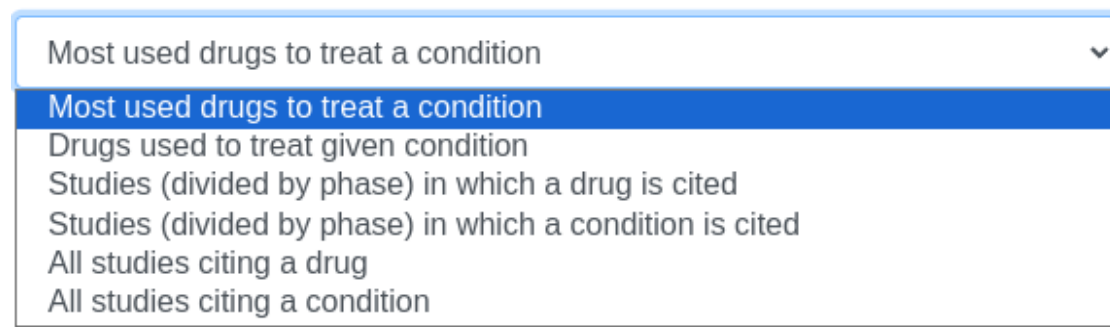
10

Submit

Figure 3.2: User interface

The user is allowed to select the desired query from a drop-down menu:

Select a Form:



A dropdown menu titled "Select a Form:" with a light blue border. The menu is open, showing a list of options. The first option, "Most used drugs to treat a condition", is highlighted in blue. The other options are: "Drugs used to treat given condition", "Studies (divided by phase) in which a drug is cited", "Studies (divided by phase) in which a condition is cited", "All studies citing a drug", and "All studies citing a condition".

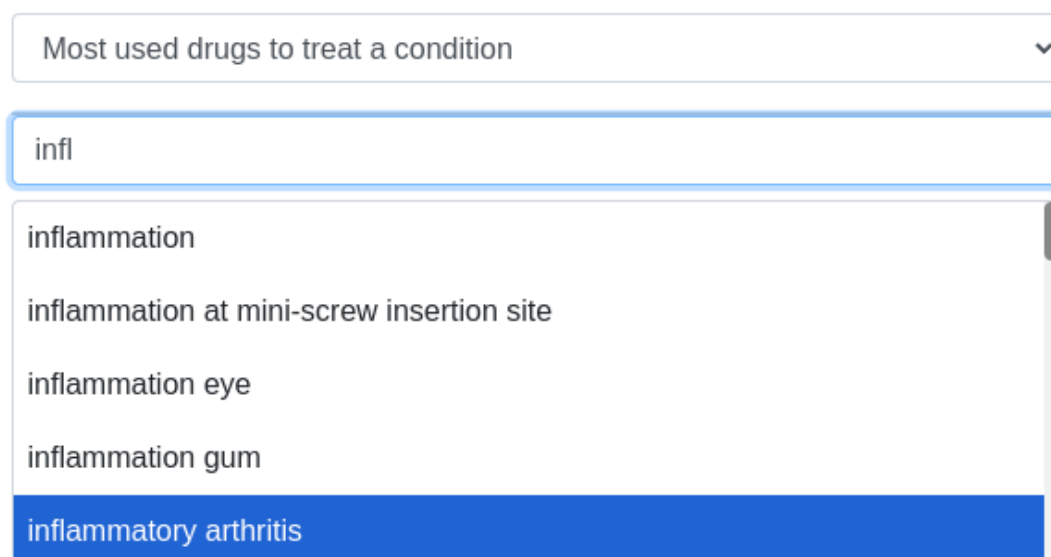
- Most used drugs to treat a condition
- Most used drugs to treat a condition
- Drugs used to treat given condition
- Studies (divided by phase) in which a drug is cited
- Studies (divided by phase) in which a condition is cited
- All studies citing a drug
- All studies citing a condition

Figure 3.3: Query selector

Then they are able to write in the input box the drug or medical condition (according to the query type) and receive suggestions based on which drugs/conditions are in the database.

In order to insert suggestions, the frontend queries the “/drugs” and “/conditions” endpoints to receive a list of all elements in the database. Then data is inserted in two trie data structures which store respectively drugs and conditions and are used to retrieve elements based on what the user is typing. When the user clicks on a suggestion, it gets inserted into the search bar.

Select a Form:

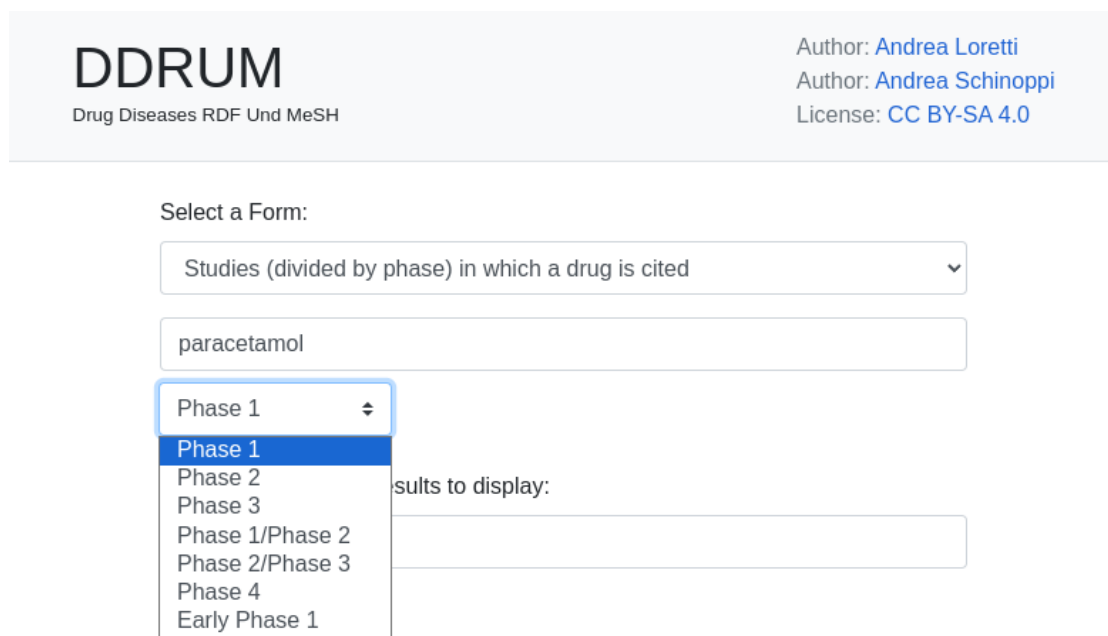


A dropdown menu titled "Select a Form:" with a light blue border. The menu is open, showing a list of suggestions. The first suggestion, "infl", is highlighted in blue. The other suggestions are: "inflammation", "inflammation at mini-screw insertion site", "inflammation eye", "inflammation gum", and "inflammatory arthritis".

- Most used drugs to treat a condition
- infl
- inflammation
- inflammation at mini-screw insertion site
- inflammation eye
- inflammation gum
- inflammatory arthritis

Figure 3.4: Suggestions for the user

For dedicated queries, the user can choose the phase of the studies they are interested in:



The screenshot shows the DDRUM web interface. At the top left, the logo "DDRUM" is displayed with the subtitle "Drug Diseases RDF Und MeSH". To the right, the author information is listed: "Author: [Andrea Loretto](#)", "Author: [Andrea Schinoppi](#)", and "License: [CC BY-SA 4.0](#)". Below the header, there is a section titled "Select a Form:" with a dropdown menu currently showing "Studies (divided by phase) in which a drug is cited". Below this, a text input field contains the word "paracetamol". To the right of the input field, there is a label "Results to display:" followed by another empty text input field. A dropdown menu is open below the "paracetamol" input, showing a list of phases: "Phase 1", "Phase 2", "Phase 3", "Phase 1/Phase 2", "Phase 2/Phase 3", "Phase 4", and "Early Phase 1". The "Phase 1" option is currently selected and highlighted in blue.

Figure 3.5: Phases drop-down menu

Finally, the user can choose how many results to include and submit the query: the results will be shown as a table.

DDRUM

Drug Diseases RDF Und MeSH

Author: [Andrea Loretto](#)

Author: [Andrea Schinoppi](#)

License: [CC BY-SA 4.0](#)

Select a Form:

Most used drugs to treat a condition

inflammation

Maximum number of results to display:

10

Submit

DrugName	Drug	UsageCount
Propofol	https://drugcentral.org/drugcard/2302	4
Difluprednate	https://drugcentral.org/drugcard/3142	4
Atorvastatin	https://drugcentral.org/drugcard/257	3
Meloxicam	https://drugcentral.org/drugcard/1676	2
Sevoflurane	https://drugcentral.org/drugcard/2439	2
Bromfenac	https://drugcentral.org/drugcard/401	2
Dexmedetomidine	https://drugcentral.org/drugcard/835	2

Figure 3.6: Table of results

3.3 Experiments and reproducibility

Our first experiments focused on understanding if the Python script outputs were correct. To do so, we used a RDF graph visualizer (issemantic.net/rdf-visualizer) in order to manually check the serialized graph. All the tests were done with a few (<5) studies in order to have a better understanding of the graph.

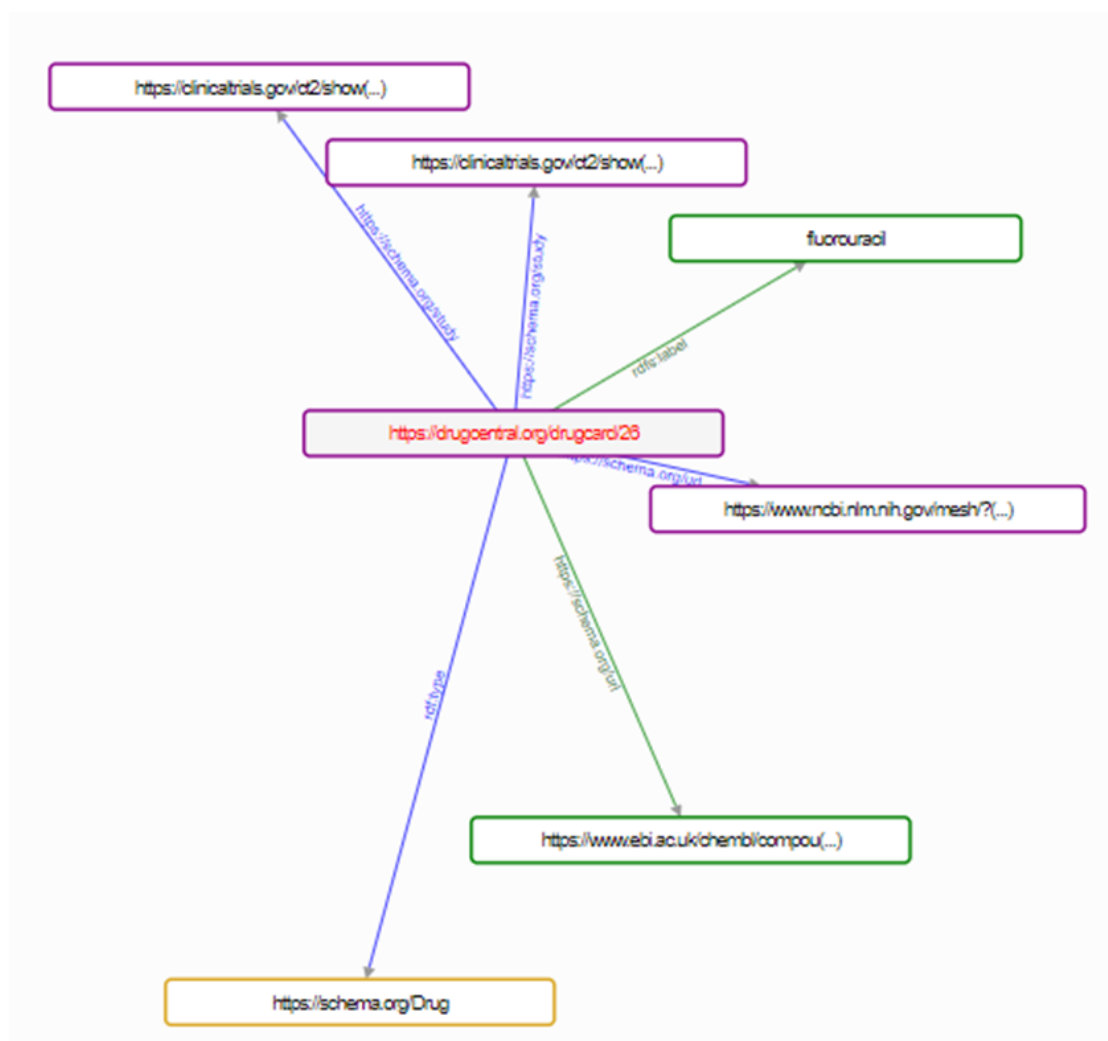


Figure 3.7: Test graph

Then, after we were sure the graphs were generated correctly, we ran the code on all the studies and created the complete database.

The reproducibility is maintained as long as the same Python code is used with the same version and the dataset used are the ones archived with their relative dump date specified in chapter 2.

It is possible to find all the databases we used for this project as .tar.gz file on Google Drive at *this link* and all the code we developed on Github at *this link*.

Conclusions

In this project, we developed a system aimed at integrating multiple medical databases and providing a user-friendly web interface for querying a unified SPARQL database derived from them. The system's primary goal was to simplify the process of accessing and retrieving medical data across various sources, thereby enhancing research and analysis capabilities for medical professionals and researchers.

By addressing some use cases, with the possibility to add new ones, our system offers valuable insights and information in healthcare and medical research.

The implementation phase involved multiple stages, starting from early conceptualization and data exploration to the actual development of the solution. We used Python for data processing, RDF manipulation, and backend development.

Additionally, we adhered to standard RDF properties from schema.org to maintain compatibility and interoperability with existing systems and standards.

The experiments conducted during the development phase provided valuable insights into the accuracy and reliability of our system. By testing with a subset of data and verifying the generated RDF graphs, we ensured the correctness of our approach before scaling it to handle the entire dataset.

4.1 Future work

The first possible future work would be deploying this website as a service for scholars and medics.

It would be also interesting to develop this topic into finding new relations among studies, drugs and medical conditions to enrich the dataset, allowing even more queries.

Technical documentation

A.1 Adding queries

To add new queries it is sufficient to:

- Add the new query name to the `#dropdownMenu` in `index.html` file and select a new value of your choice: this value will be the name which will identify the new query type for the server.

```
1 <div class="form-group">
2   <label for="dropdownMenu">Select a Form:</label>
3   <select class="form-control" id="dropdownMenu" ...>
4     <option value="Mdrug_cond">Most used ...</option>
5     <option value="drug_cond">Drugs used ...</option>
6     <option value="Pstud_drug">Studies ...</option>
7     <option value="Pstud_cond">Studies ...</option>
8     <option value="stud_drug">All studies ...</option>
9     <option value="stud_cond">All studies...</option>
10    <!-- HERE -->
11  </select>
12 </div>
13
```

- Update the code in the script part to show or hide conditions and drugs search bars based on the new value of the query:

```
1 var selectedValue = $("#dropdownMenu").val();
2
3 // Toggle visibility of drugSearch and conditionSearch
4 if (selectedValue === "Pstud_drug" || ...) { //HERE
5   $("#drugSearch").show();
6   $("#conditionSearch").hide();
7 } else {
8   $("#drugSearch").hide();
9   $("#conditionSearch").show();
10 }
11 // Toggle visibility of phases
12 if (selectedValue === "Pstud_drug" || ...) { //HERE
13   $("#phaseDropdownContainer").show();
14 } else {
15   $("#phaseDropdownContainer").hide();
16 }
17
```

A.2 How the RDF graph is generated

This part of the documentation was written with the help of Large Language Models. The output was manually checked and, if necessary, corrected.

A.2.1 Drug Entity and Properties

- For each drug extracted from the data sources, the code creates RDF triples to represent the drug entity and its properties.
- It starts by creating a URI reference for the drug using the DrugCentral namespace (`DRUG_CARD`) and the structural identifier (`struct_id`) associated with the drug.
- RDF triples are then created to denote the type of entity (`RDF.type`) as a Drug according to the schema.org vocabulary.
- The drug's name is represented using the `SCHEMA.name` property, linking the drug URI to a literal value representing the drug's name.
- If available, MOA URLs and ACT URLs associated with the drug are also represented using RDF triples. Each URL is linked to the drug URI using the `SCHEMA.url` property.

A.2.2 Connecting Drugs with Clinical Trial Files

- The code iterates over each drug and its associated clinical trial files.
- For each file, RDF triples are created to represent the file entity and its properties.
- A URI reference is created for each file using the ClinicalTrials namespace (`STUDY`) and the file identifier.
- RDF triples are then created to denote the type of entity (`RDF.type`) as a MedicalStudy according to the schema.org vocabulary.
- Properties of the clinical trial file, such as title, alternate title (if available), and medical condition, are represented using appropriate schema.org properties (`SCHEMA.title`, `SCHEMA.alternateName`, `SCHEMA.MedicalCondition`).
- If the clinical trial phase information is available and not "N/A", it is also represented using the `SCHEMA.phase` property.

- Finally, RDF triples are created to connect the drug entity with the clinical trial file entity. This relationship is represented using the **SCHEMA.study** property, linking the drug URI to the file URI. Additionally, the code establishes a relation between the clinical trial file entity and the MeSH terms associated with it using the **SCHEMA.about** property.

A.2.3 MeSH Information

- If MeSH information is available for a drug (i.e., if the drug is found in the MeSH XML data), RDF triples are created to represent this information.
- MeSH terms associated with the drug are represented using appropriate schema.org properties (**SCHEMA.url**).
- Each MeSH term is represented as a URI reference using the MeSH namespace (**MESH**), with the term's descriptor UI as the identifier.
- RDF triples are created to connect the drug entity with the MeSH term entities, indicating that the drug is associated with these MeSH terms. Additionally, the code establishes a symmetric relation between the MeSH term entity and the clinical trial file entity using the **SCHEMA.citedby** property.