A black and red logo

Description automatically generated with medium confidence

**COMP 6741: INTELLIGENT SYSTEMS**

**WINTER 2024**

**PROJECT ASSIGNMENT #1**

**Instructor**

Dr. René Witte

**Submitted By**

Saurabh Sharma

Anubhav Mahajan

**GitHub URL:**

<https://github.com/saurabhs679/COMP-6741-Project>

**Vocabulary**

The knowledge base schema is represented using RDF (Resource Description Framework), with specific classes and properties developed to describe the domain's entities and relationships. Here is a summary of the modeling decisions:

**Standard Vocabularies:**

Standard vocabularies such as rdf, rdfs, and their associated properties (rdf:type, rdfs:label, rdfs:domain, rdfs:range) are utilized for defining classes, properties, and their characteristics

**Classes and Properties:**  
**Classes** - ex:Course, ex:Lecture, ex:Student, ex:Topic, and ex:University. These classes provide a structured framework for categorizing and organizing information.

**Properties** - Properties explain an entity's attributes and relationships. Each property is connected with a domain and a range, which define its applicability and possible values. ex:CourseName, ex:CourseNumber, and ex:Credits are characteristics that record precise facts about courses, whereas ex:IDNumber, ex:firstName, and ex:lastName describe student-related information.

**Vocabulary Extensions:**

We also developed few vocabularies extension such as ex:topic\_in\_course, ex:topic\_in\_lecture to defines the relationship between different classes like first one shows relationship between course and topic . Similarly , ex:topic\_in\_lecture links individual lectures to the topics they address. These extensions enhance the expressiveness of the schema, enabling more accurate querying and inference and facilitate a more comprehensive representation of the knowledge base.

This design structure lays the groundwork for efficient organization, retrieval, and integration of information.

**Knowledge Base Construction**

Our dataset consists of five distinct datasets, each based on specific classes and their corresponding properties within our knowledge base schema. The datasets contain structured information relevant to courses, lectures, students, topics, and universities.

**Courses Dataset:** Contains information about various courses offered, including course name, number, credits, description, subject, outline, and webpage link.

**Lectures Dataset:** Provides details about individual lectures, such as lecture name, number, content, and webpage link.

**Students Dataset:** Includes information about enrolled students, such as student ID, first name, last name, email, and completed courses.

**Topics Dataset:** Contains data regarding topics covered within lectures or courses, along with topic names and DBpedia links for additional context.

Universities Dataset: Provides information about universities, including university name and DBpedia link.

We are using python script to parse and read all the datasets and converting it into a structured format compatible with the Turtle serialization.We are doing this for all 5 datasets merging them together with main vocubalary to populate the knowledge graph.

To run the tools and create the knowledge base using the provided Python notebook, follow these steps:

**Prepare CSV Files:**

Ensure that you have the necessary CSV files containing the data for students, courses, universities, lectures, and topics. Each CSV file should follow a specific format with columns representing different attributes such as student ID, course name, lecture content, etc.

**Open the Python Notebook:**

Open the Python notebook file containing the code for populating each aspect of the knowledge base using the rdflib library.

**Execute Cells:**

Execute each cell in the notebook using a Python interpreter or a Jupyter notebook environment. Each cell corresponds to a section of code responsible for populating a specific aspect of the knowledge base (e.g., student data, course data, etc.).

**Verify Output:**

After executing each cell, verify that the output is generated successfully without any errors. The output will typically be in the form of RDF triples serialized in Turtle format, printed to the notebook's output cell.

**Load RDF Triples:**

Once you have generated RDF triples for each aspect of the knowledge base, load them into a triple store or graph database for storage and querying. You can use a triple store such as Apache Jena.

**Execute SPARQL Queries:**

Within the notebook, execute SPARQL queries directly against the populated RDF graph. Use the rdflib.plugins.sparql.prepareQuery function to prepare the query, and then execute it using the RDF graph object. For example, you can execute a SPARQL query to retrieve competencies related to a specific course as follows:

*from rdflib.plugins.sparql import prepareQuery*

*query = prepareQuery(f"""*

*PREFIX ex: <http://example.org/>*

*SELECT ?competency*

*WHERE {{*

*?course ex:completedCourse "COMP 6481" ;*

*ex:competentIn ?competency .*

*}}*

*""", initNs={"ex": ex})*

*# Execute the query and print the competencies*

*results = g1.query(query)*

*for row in results:*

*print(row.competency)*

**Load RDF Triples:**

Once you have generated RDF triples for each aspect of the knowledge base, load them into a triple store or graph database for storage and querying. You can use a triple store such as Apache Jena.

By following these steps and executing the provided Python notebook, you can effectively populate the knowledge base from the CSV files and perform querying and analysis tasks.

**Graph Queries**

1. List all courses offered by [university]:

* This query retrieves the names and numbers of all courses offered by a specific university.
* It uses the ex:CourseName and ex:CourseNumber properties to fetch the course details.
* The ?course variable represents each course in the graph that matches the criteria.
* It selects the course name and course number using the SELECT clause.
* The ?course variable is bound to instances of the ex:Course class.

1. In which courses is [topic] discussed?

* This query finds all courses where a specific topic is discussed.
* It looks for instances of the topic with the given name using ex:TopicName.
* It then retrieves the names of courses where the topic is discussed using the ex:topic\_in\_course property.
* The ?courseName variable represents the names of courses where the topic is discussed.

1. Which [topics] are covered in [course] during [lecture number]?

* This query retrieves the names of topics covered in a specific course during a particular lecture.
* It selects topics associated with the given lecture number using the ex:lectureNumber property.
* The ?topicName variable represents the names of topics covered in the specified lecture of the course.

1. List all [courses] offered by [university] within the [subject] (e.g., "COMP", "SOEN"):

* This query fetches the names and numbers of courses offered by a university within a specific subject area.
* It filters courses based on the subject using the ex:Subject property.
* The ?university variable represents the university, and the ?courseName and ?courseNumber variables represent course details.

1. #6 How many credits is [course] [number] worth?

* This query retrieves the number of credits associated with a specific course number.
* It filters courses based on the course number using the ex:CourseNumber property.
* The ?credits variable represents the number of credits associated with the course.

**Triplestore and SPARQL Endpoint Setup**

We are using Apache Jena Fuseki to set up a Triplestore and SPARQL Endpoint.

We are downloading the Apache Jena Fuseki distribution package and extracting it. We also need to make sure java is installed in our system and its environment path is configured. We can open it from command prompt using fuseki-server command and it runs on localhost 3030. We create a new dataset and Upload RDF data into the newly created dataset using the web interface. To access the SPARQL endpoint provided by Fuseki to query the loaded data, construct and execute SPARQL queries using the web interface.