User Manual - CITS3005 Project

Henri Scaffidi (23245207)

20 October 2023

This document contains:

- Quick Start: How to quickly run everything.
- Overview: Outlines the schema, constraints and ontology rules.
- Querying the Knowledge Graph: How to construct and interpret queries.
- Altering Data and Rules in Knowledge Graph: How to edit knowledge graph data/rules.

1 Quick Start

1. Place the following files in the same directory:
   - handbook_knowledge_graph.py
   - handbook_shapes.ttl
   - handbook_validation.py
   - handbook_ontology.py
   - majors.json
   - units.json
2. 'cd' into said directory.
3. Run **handbook_knowledge_graph.py**
   - This creates the knowledge graph, exporting it as handbook_knowledge_graph.rdf and handbook_knowledge_graph.ttl
   - Additionally, this runs SPARQL queries on the graph, where the output is printed to stdout
4. Run **handbook_validation.py**
   - This validates the previously created graph against the shapes graph (can take up to 2 minutes)
   - The results of the validation are printed to stdout
5. Run **handbook_ontology.py**
   - This creates the general ontology, outputting it to handbook_ontology.owl
   - It then loads in the knowledge graph from handbook_knowledge_graph.rdf
   - It then runs the reasoner, does inferences, and outputs the final ontology to handbook_ontology.rdf
   - Finally, SPARQL query results are printed to stdout, demonstrating newly inferred knowledge

2 Overview

2.1 Entities

The graph has two main entity types, each with several attributes/properties:

- **Unit** - An academic Unit at UWA.
  - **Code** - Code for the unit.
  - **Title** - Title of the unit.
  - **School** - School offering the unit.
  - **Board of Examiners** - Board responsible for the unit's examination.
  - **Delivery Mode** - Mode of unit delivery (e.g., Online).
  - **Level** - Level of the unit (e.g., 1, 2, 3).
  - **Description** - Description of the unit's content and purpose.
  - **Credit** - Credit value of the unit.
  - **Outcomes** - Expected outcomes after completing the unit.
  - **Assessment** - Methods and criteria for assessing students.
  - **Prerequisites Text** - Textual information on prerequisites for the unit.
  - **Contact** - Weekly contact hours for the unit.
  - **Majors** - Majors associated with or requiring the unit.
  - **Offering** - Offering details of the unit.
  - **Note** - Any additional notes or remarks about the unit.
  - **Text** - Additional required texts for the unit.
- **Major** - An academic Major at UWA.
  - **Code** - Code for the major.
  - **Title** - Title of the major.
  - **School** - School offering the major.
  - **Board of Examiners** - Board responsible for the major's examination.
  - **Delivery Mode** - Mode of major delivery.
  - **Description** - Description of the major's content and purpose.
  - **Outcomes** - Expected outcomes after completing the major.
  - **Prerequisites** - Prerequisites required for the major.
  - **Courses** - Courses associated with the major.

2.2 Relations

The graph has numerous relations which connect the entities:

- **hasPrerequisite** - Unit A has prerequisite Unit B.
- **hasAdvisedPriorStudy** - Unit A has advised prior study in Unit B.
- **hasUnit** - Major A includes studying Unit A.
- **hasBridgingUnit** - Major A requires Unit A to be studied as a Bridging Unit (beforehand).

2.3 Constraints

Unit Property Constraints:

- A unit may have 0 or more prerequisite units.
- Each unit must have exactly one code.
- Each unit must have exactly one level between 0 and 9.
- A unit's contact hours should be non-negative.
- Each unit's code should be unique.
- Every prerequisite for a level X unit should have a level less than X.
- No unit should be its own prerequisite.

Major Property Constraints:

- The average contact hours for units in a major should not exceed 10 hours per week.
- Each major must have exactly one code.
- Each major should have at least one required unit to be completed.

Relation Constraints:

- hasUnit - Subject of hasUnit must be a Major, object must be a Unit.
- hasPrerequisite - Subject of hasPrerequisite must be a Unit, object must be a Unit.
- hasAdvisedPriorStudy - Subject of hasAdvisedPriorStudy must be a Unit, object must be a Unit.
- hasBridgingUnit - Subject of hasBridgingUnit must be a Major, object must be a Unit.

2.4 Ontology

The ontology defines various classes, object properties, data properties, and rules.

2.4.1 Classes:

- **Unit** - Represents an academic unit.
- **Major** - Represents an academic major.
- **RequiredBridgingMajor** - Represents a major with required bridging units. Equivalent to majors that have at least one bridging unit.

2.4.2 Object Properties:

- **hasBridgingUnit** - Links a major to its bridging units.
- **hasUnit** - Links a major to its units.
- **hasPrerequisite** - A transitive property that links a unit to its prerequisites.
- **isPrerequisiteFor** - Inverse of the hasPrerequisite property.
- **hasAdvisedPriorStudy** - A transitive property that links a unit to its advised prior studies.
- **isAdvisedPriorStudyFor** - Inverse of the hasAdvisedPriorStudy property.

2.4.3 Data Properties:

- **code** - Represents the code for both units and majors.
- **title** - Represents the title for both units and majors.
- **school** - Represents the school for both units and majors.
- **board_of_examiners**, **delivery_mode**, **description**, **outcomes**, **text** - Other descriptive properties applicable to both units and majors.
- **level**, **credit**, **assessment**, **prerequisites_text**, **contact**, **majors**, **offering**, **note** - Descriptive properties specific to units.
- **courses**, **prerequisites** - Descriptive properties specific to majors.

2.4.4 Rules:

- If a unit has an outcome and is part of a major, then the major also has that outcome.
- If a unit has a text and is part of a major, then the major also has that text.

3 Querying the Knowledge Graph

3.1 Example Queries:

- Find all units with more than 6 outcomes.
- Find all level 3 units that do not have an exam, and where none of their prerequisites have an exam.
- Find all units that appear in more than 3 majors.
- Basic search functionality: Given a query string (e.g., "environmental policy"), find the units that contain this string in the description or outcomes.
- Find the top 10 highest-level units at UWA.

3.2 Writing Queries:

These example queries can be written using SPARQL. They can then be placed in the file **handbook_knowledge_graph.py** (as I have already done). If you instead would like to query the graph after inferences have been made, place queries into **handbook_ontology.py** (you will see I have written some already).

```
print("\n-----------------QUERY 5-----------------\n")
print("Find the top 10 highest level units at UWA.\n")
query5 = """
    PREFIX uwa: <http://example.org/uwa/>

    SELECT ?unit ?level
    WHERE {
        ?unit a uwa:Unit;
              uwa:level ?level .
    }
    ORDER BY DESC(?level)
    LIMIT 10
"""
for row in g.query(query5):
    print("\tUnit:", row["unit"])
    print("\tLevel:", row["level"], "\n")
```

Inside the large Python string, simply replace the text with any valid SPARQL query, ensuring to include the uwa namespace. Whichever variables you 'SELECT', make sure to place their names into the for loop so they can be printed to stdout.

3.3 Interpreting the Output of Queries:

- Ensure **units.json** and **majors.json** are contained within the same directory as handbook_knowledge_graph.py.
- In a shell, type 'cd directory_path' command, where 'directory_path' is the path of the folder in which handbook_knowledge_graph.py resides.
- In the same shell, run command 'python handbook_knowledge_graph.py', which will write the knowledge graph to a .rdf and .ttl file, and print out the results of the queries in the shell.

4 Altering Data and Rules in Knowledge Graph

4.1 Data

The following instructions are a guide through the process of adding, updating, or removing data from the knowledge graph using the script **handbook_knowledge_graph.py**:

1. **Prepare JSON Files:** Modify the respective JSON files (**units.json** and **majors.json**). These files contain structured data representing the units and majors, respectively.

2. **Load JSON Data:** Load the data:

```
with open('units.json', 'r') as file:
    units_data = json.load(file)
```

3. **Initialize RDF Graph:** Create an RDF graph using `rdflib`. This is where the data will be stored in RDF format.

```
g = Graph()
```

4. **Add Entities to Graph:** Traverse the loaded JSON data to create entities and properties in the RDF graph. For instance, each unit and major is added as an individual entity with associated properties:

```
for unit in units_data.keys():
    unit_uri = UWA[unit]
    g.add((unit_uri, RDF.type, UWA.Unit))
```

5. **Process Properties:** Loop through the properties of each unit or major and convert them to RDF triples that can be added to the graph. This includes relations like prerequisites, advisability of prior study, bridging units etc.

6. **Export the Graph:** Once all modifications are made and the graph is constructed, it can be serialized and exported in different RDF formats:

```
ttl_data = g.serialize(format='turtle')
with open('handbook_knowledge_graph.ttl', 'wb') as f:
    f.write(ttl_data)
```

7. **Verify Data:** After exporting the graph (after running the script), open and verify the generated .rdf or .ttl files.

4.2 Rules

The "rules" governing the structure and validity of the knowledge graph are defined within two main components: the shapes graph **handbook_shapes.ttl** and the ontology **handbook_ontology.py**. Here's a breakdown on how to alter rules within each component:

4.2.1 Shapes Graph

The shapes graph in **handbook_shapes.ttl** employs SHACL (Shapes Constraint Language) to validate the data in the RDF graph. SHACL shapes describe the constraints and patterns that the RDF data should adhere to.

1. **Understand Existing Shapes:** Before modifying the shapes graph, understand the existing shapes and their constraints. This helps in ensuring consistency and compatibility.
2. **Add/Update a Shape:** To add or modify a shape, define a new shape or modify an existing one within the shapes graph. This could involve specifying the properties a node must have, the kind of values those properties can have, or more advanced structural rules using SPARQL.
3. **Remove a Shape:** Remove the RDF triples associated with the shape you want to delete. Ensure that this does not leave other shapes or data in an inconsistent state.
4. **Validate Data Against Shapes:** After modifying the shapes graph, run **handbook_validation.py** (in the same directory) to validate the knowledge graph data against the updated shapes. The output in stdout shows which shapes have been violated. Note: You must run **handbook_knowledge_graph.py** in the same directory before doing this (with majors.json and units.json in the directory, too).

4.2.2 Ontology

The ontology, defined in **handbook_ontology.py**, defines the terms used in the RDF data, their relationships, and semantics.

1. **Review the Ontology:** Familiarize yourself with the existing classes, properties, and relationships to ensure that any modifications are consistent with the established structure.
2. **Add/Update Classes or Properties:** To introduce a new term or modify an existing one, add or update the respective class or property definition in the ontology. Ensure that the semantics and relationships are clearly defined. Additionally, use SWRL rules to have fine-grained control of inferences that the reasoner can make.
3. **Remove Classes or Properties:** Delete the class or property you want to remove. However, ensure that this removal does not create inconsistencies or ambiguities in the knowledge graph.
4. **Ensure Compatibility:** If you've made modifications to the ontology, ensure that it remains compatible with the shapes graph. It's vital that the rules in the shapes graph align with the terms defined in the ontology.
5. **Run Ontology Script:** Run **handbook_ontology.py** in the same directory where the knowledge graph was created and exported previously. This will create a .owl file which defines the general ontology, and a .rdf file which incorporates the knowledge graph data AND has newly inferred information. Additionally, some query output is printed to stdout, showing newly inferred knowledge.