Knowledge-Based Systems Laboratory Project

Iulia-Olivia Cipleu

May 2025

1 Introduction

The project aims to create a Faculty Ontology in Protégé. The objective is to define main concepts such as: departments, study programs, students, teachers, rooms, courses, timetables, and their relationships. The Graph Ontology may be seen in Fig. 1.

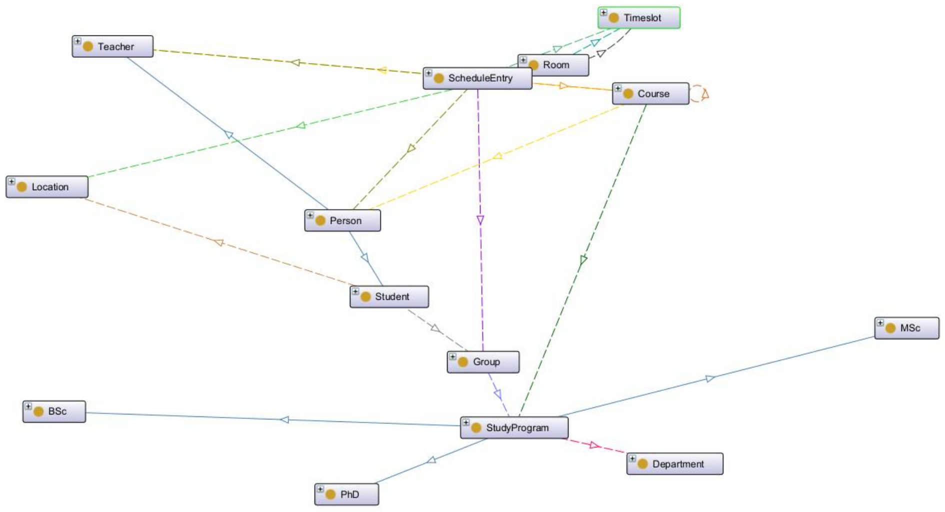


Figure 1: Ontology Graph

2 Tools and Technologies

- **Protégé 5.5.0** – used for modeling and editing the ontology.
- **OWL (Web Ontology Language)** – formal language for defining classes, properties, and individuals.
- **Pellet Reasoner** – for logical consistency checks and inference.

3 Ontology Design

3.1 Classes

The ontology includes key classes such as:

- `Department, StudyProgram, BSc, MSc, PhD, Course, Student, Teacher`
- `Room, Timeslot, ScheduleEntry, Group`

3.2 Enum

The ontology defines a classification for locations based on their proximity to a reference point, specifically `26-28 Gh. Baritiu Street`. This classification uses classes, individuals, object and data properties, and OWL axioms to model semantic proximity. Below, it is explained each modeling step in detail:

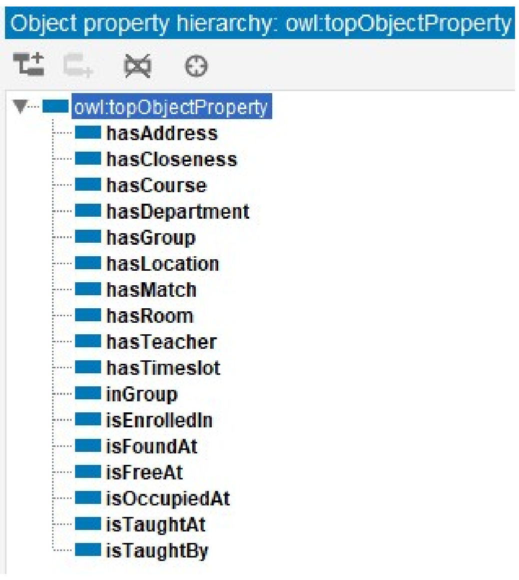
1. **Class Declarations:**
   - `CloseLocation`, `MiddleLocation`, and `FarLocation` represent categories of location proximity.
   - `Distance` is a general concept encompassing the three proximity types.
2. **Individual Enumeration of Distance Types:**
   The class `Distance` is defined as equivalent to a specific set of individuals using the `ObjectOneOf` constructor:

   `Distance` ≡ { `Close`, `Middle`, `Far` }
3. **CloseLocation Definition:**
   - Has a `hasCloseness` object property pointing to the individual `Close`.
   - Has a `distanceTo26-28_Gh._Baritiu_Street` data property with a decimal value strictly less than 2.0.
4. **MiddleLocation Definition:**
   - Subclass of `Location`.
   - Has a `hasCloseness` property pointing to `Middle`.
   - The distance to the reference location is defined as a decimal value between 2.0 and 5.0, inclusive.
5. **FarLocation Definition:**
   - Subclass of `Location`.
   - Has a `hasCloseness` property pointing to `Far`.
   - The distance to the reference location is defined as a decimal strictly greater than 5.0.
6. **Disjoint Classes:**
   The ontology ensures that the classes `Student` and `Teacher` are disjoint, meaning that no individual can belong to both.

3.3 Object Properties

Main object properties that define relationships include (see Fig. 2):

- `hasCourse, hasTeacher, isEnrolledIn, isTaughtAt, hasRoom, hasTimeslot`
- `hasGroup, inGroup, isFreeAt, isOccupiedAt`


Object property hierarchy: owl:topObjectProperty

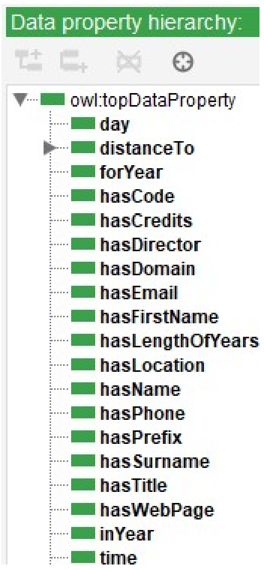
```
owl:topObjectProperty
    hasAddress
    hasCloseness
    hasCourse
    hasDepartment
    hasGroup
    hasLocation
    hasMatch
    hasRoom
    hasTeacher
    hasTimeslot
    inGroup
    isEnrolledIn
    isFoundAt
    isFreeAt
    isOccupiedAt
    isTaughtAt
    isTaughtBy
```

Figure 2: Object properties

3.4 Data Properties

Attributes of individuals are represented using data properties such as (see Fig. 3):

- `hasName, hasEmail, hasCredits, hasCode, hasDomain, hasTitle`
- `forYear, day, time, inYear`



Data property hierarchy:

- owl:topDataProperty
  - day
  - distanceTo
  - forYear
  - hasCode
  - hasCredits
  - hasDirector
  - hasDomain
  - hasEmail
  - hasFirstName
  - hasLengthOfYears
  - hasLocation
  - hasName
  - hasPhone
  - hasPrefix
  - hasSurname
  - hasTitle
  - hasWebPage
  - inYear
  - time

Figure 3: Data properties

3.5 Individuals

Example individuals instantiated in the ontology include:

- `Teachers: A_Groza, C_Feier, A_Colesa`
- `Courses: Semantic_Web_Agents, Information_Theory_and_Statistics, Windows_Internals_and_Kernel_Driver_Development`
- `Students: Student_Achim_Andrei, Student_Bologa_Horatiu`
- `Rooms: BT5_01, AulaInstalatii`
- `Timeslots: Monday_10_12, Friday_14_16`

4 Creating Individuals

In this section, it is described how individuals were generated and added to the ontology. The process was done in two ways: programmatically using Python and interactively using `Cellfie`, a Protégé plugin that supports spreadsheet-based ontology population.

In this section, it is described how individuals were generated and added to the ontology. The process was carried out in two main ways: programmatically using Python and interactively using `Cellfie`, a Protégé plugin that supports spreadsheet-based ontology population.

4.1 With Python

To create individuals programmatically, a Python-based pipeline involving Optical Character Recognition (OCR), HTML parsing, and semantic mapping was used. The pipeline processed various input formats:

- **OCR from Screenshots:** For several data sources, including department pages and course listings on the faculty website, screenshots were taken and processed using OCR tools such as `pytesseract`. The extracted text was cleaned and mapped to individuals such as students, teachers, and courses.

- **Parsing PDFs:** Official documents like course catalogs and schedules were downloaded as PDF files. These were parsed using PDF-to-text libraries (e.g., `pdfplumber` or `PyMuPDF`), and the structured data was used to instantiate corresponding OWL individuals.
- **Parsing HTML Files:** For web pages available in HTML format, such as dynamically generated schedules or department directories, Python's `BeautifulSoup` library was used to parse and extract relevant information. This data was then normalized and converted into OWL individuals.

4.1.1 Matching Course Titles Across Sources

An important challenge in constructing the ontology was handling inconsistent course title formats across multiple data sources. Course information was obtained from several OWL modules, including `courses.owl` and `timetables.owl`, and it was observed that:

- Some courses were listed in **English**.
- Others were written in **Romanian**.
- Several entries used only **abbreviations** (e.g., `SO`, `PL`, `IS`).

To identify and merge equivalent courses, a Python-based normalization and matching pipeline was implemented. The key steps included:

1. **Translation and Normalization:** Titles were first translated from Romanian to English using the Google Translate API, then processed to remove accents and punctuation using `unidecode` and `regex`. This ensured a uniform lowercase alphanumeric representation.
2. **Abbreviation Mapping:** A manually curated mapping dictionary was defined to associate full course names with common abbreviations. For example:

   ```
   "electrotechnics" => ["et", "electrotehnica"]
   "logicprogramming" => ["pl", "lp", "programarelogica"]
   ```

   This allowed robust matching even when the same course appeared with different abbreviations.
3. **Similarity Matching:** When abbreviation logic did not apply, `Levenshtein` string similarity was used to compute the normalized title similarity. If the similarity exceeded a defined threshold (e.g., 85%), the two course individuals were considered a match.
4. **Storing Matches:** Matches were saved as a new OWL graph in `matched_courses.owl`, using the custom object property `hasMatch` and datatype property `hasSimilarity`. This structure supports semantic alignment without data loss.

This approach provided a scalable and semi-automated solution for reconciling heterogeneous course data across ontology modules, improving consistency and query accuracy.

4.1.2 Merging the OWL Files

In all cases, serializing the data as RDF/XML using the `rdflib` library, ensuring compatibility with Protégé and OWL standards.

After generating and organizing individuals across several domain-specific OWL files (e.g., for students, courses, departments, timetables), the final step was to integrate them into a unified ontology. This was achieved using a Python script based on the `rdflib` library.

The script performs the following steps:

1. **Backup:** Creates a backup of the existing merged ontology (`merged_ontology.owl`) if it already exists.
2. **Graph Initialization:** Initializes an empty RDF graph to serve as the container for all merged triples.
3. **File Loading:** Iteratively loads a list of modular OWL files such as:
   - `study_programs.owl`, `departments.owl`, `cs_department.owl`
   - `courses.owl`, `students.owl`, `timetables.owl`
   - `matched_courses.owl`, `inferred.owl`

Optionally, if a backup version of the merged ontology exists, it is also included in the merging process to preserve continuity.

4. **Merging:** Each file is parsed and its triples are added to the unified graph.
5. **Serialization:** The final merged graph is written back to disk as an RDF/XML OWL file, creating an updated and complete ontology.

4.2 With Cellfie

In addition to Python, the Cellfie plugin is used to batch-create individuals from tabular data. For this, two Excel sheets were prepared: one containing room data and another containing student entries.

Custom Cellfie rules were defined to map spreadsheet columns to OWL class assertions and property assertions. For example:

- Each row in the rooms sheet was converted into a `Room` individual with properties `hasLocation`.
- Each student entry was mapped to a `Student` individual with associated data properties (e.g., `hasFirstName`, `hasAddress`).

This method was efficient for bulk population where the source data was already available in spreadsheet format.

4.2.1 Cellfie Rules for Students

The following rules were defined in the Cellfie JSON configuration to extract and transform the data into OWL individuals:

- **Location Individuals:** Each entry in column `D` (the address of the student) is used to create an individual of type `Location`.

  ```
  Individual: @D*
    Types: Location
  ```

- **Distance Properties for Locations:** The address in column `D` is extended with facts expressing distances to specific landmarks. These distances are taken from various columns:
  - Column `E`: distance to `26-28 Gh.  Baritiu Street`
  - Column `F`: distance to `2 Observatorului Street`
  - Column `G`: distance to `71-73 Dorobantilor Street`
  - Column `H`: distance to `TUCN Swimming Complex`

  These are mapped using facts such as:

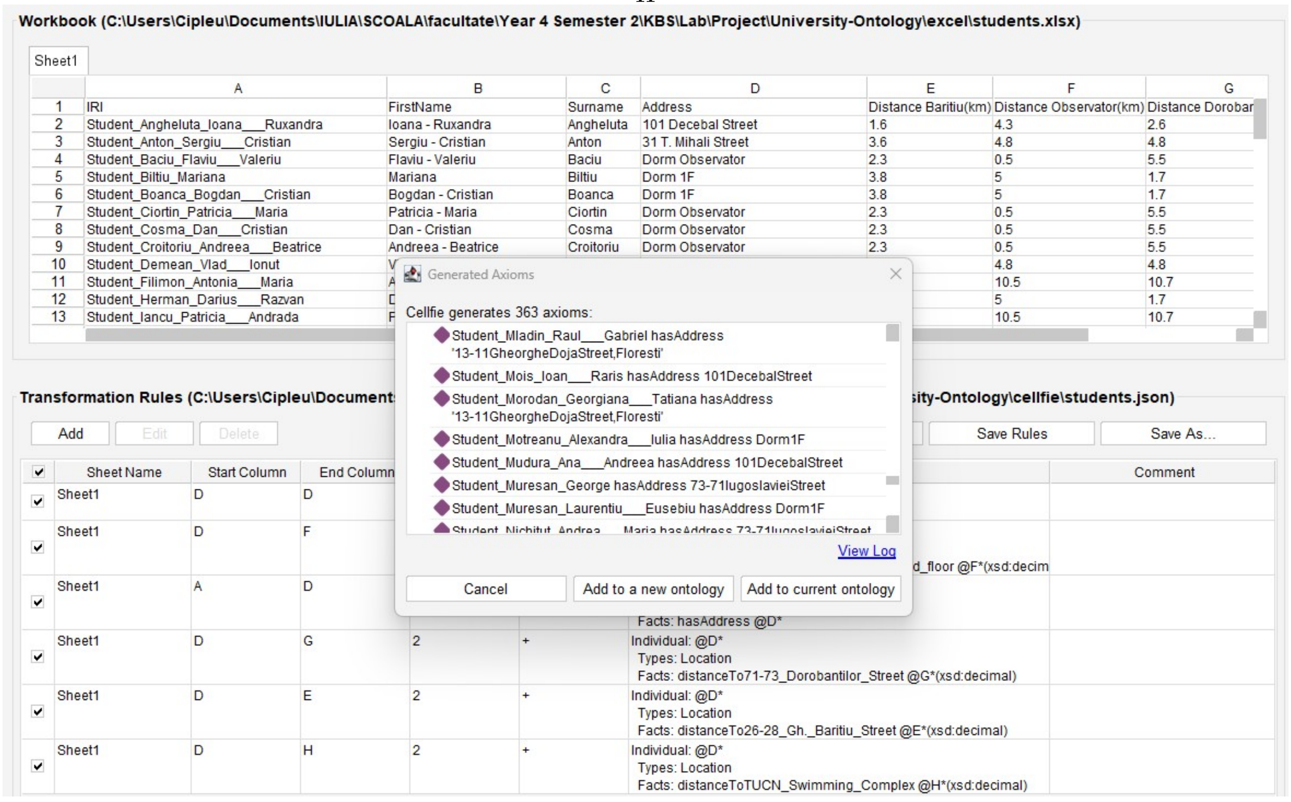
  ```
  Facts: distanceTo26-28_Gh._Baritiu_Street @E*(xsd:decimal)
  ```

- **Student Individuals:** Each entry in column `A` (student name) is used to create a `Student` individual. That individual is then linked to a location (from column `D`) via the `hasAddress` object property:

  ```
  Individual: @A*
    Types: Student
    Facts: hasAddress @D*
  ```

This rule set allowed us to generate multiple individuals and enrich them with structured geographic information directly from Excel, making it efficient for bulk population tasks. Part of the generated results can be seen in Fig. 4.

H



Workbook (C:\Users\Cipleu\Documents\IULIA\SCOALA\facultate\Year 4 Semester 2\KBS\Lab\Project\University-Ontology\excel\students.xlsx)

Sheet1

	A	B	C	D	E	F	G
1	IRI	FirstName	Surname	Address	Distance Baritiu(km)	Distance Observator(km)	Distance Dorobar
2	Student_Angheluta_Ioana___Ruxandra	Ioana - Ruxandra	Angheluta	101 Decebal Street	1.6	4.3	2.6
3	Student_Anton_Sergiu___Cristian	Sergiu - Cristian	Anton	31 T. Mihali Street	3.6	4.8	4.8
4	Student_Baciu_Flaviu___Valeriu	Flaviu - Valeriu	Baciu	Dorm Observator	2.3	0.5	5.5
5	Student_Biltiu_Mariana	Mariana	Biltiu	Dorm 1F	3.8	5	1.7
6	Student_Boanca_Bogdan___Cristian	Bogdan - Cristian	Boanca	Dorm 1F	3.8	5	1.7
7	Student_Ciortin_Patricia___Maria	Patricia - Maria	Ciortin	Dorm Observator	2.3	0.5	5.5
8	Student_Cosma_Dan___Cristian	Dan - Cristian	Cosma	Dorm Observator	2.3	0.5	5.5
9	Student_Croitoriu_Andreea___Beatrice	Andreea - Beatrice	Croitoriu	Dorm Observator	2.3	0.5	5.5
10	Student_Demean_Vlad___Ionut	V				4.8	4.8
11	Student_Filimon_Antonia___Maria	A				10.5	10.7
12	Student_Herman_Darius___Razvan	D				5	1.7
13	Student_Iancu_Patricia___Andrada	F				10.5	10.7

Generated Axioms

Cellfie generates 363 axioms:

```
◆ Student_Mladin_Raul___Gabriel hasAddress
  '13-11GheorgheDojaStreet,Floresti'
◆ Student_Mois_Ioan___Raris hasAddress 101DecebalStreet
◆ Student_Morodan_Georgiana___Tatiana hasAddress
  '13-11GheorgheDojaStreet,Floresti'
◆ Student_Motreanu_Alexandra___Iulia hasAddress Dorm1F
◆ Student_Mudura_Ana___Andreea hasAddress 101DecebalStreet
◆ Student_Muresan_George hasAddress 73-71IugoslavieiStreet
◆ Student_Muresan_Laurentiu___Eusebiu hasAddress Dorm1F
◆ Student_Nichitut_Andrea___Maria hasAddress 73-71IugoslavieiStreet
```

Transformation Rules (C:\Users\Cipleu\Document ... sity-Ontology\cellfie\students.json)

✓	Sheet Name	Start Column	End Column				Comment
✓	Sheet1	D	D				
✓	Sheet1	D	F			d_floor @F*(xsd:decim	
✓	Sheet1	A	D			Facts: hasAddress @D*	
✓	Sheet1	D	G	2	+	Individual: @D* Types: Location Facts: distanceTo71-73_Dorobantilor_Street @G*(xsd:decimal)	
✓	Sheet1	D	E	2	+	Individual: @D* Types: Location Facts: distanceTo26-28_Gh._Baritiu_Street @E*(xsd:decimal)	
✓	Sheet1	D	H	2	+	Individual: @D* Types: Location Facts: distanceToTUCN_Swimming_Complex @H*(xsd:decimal)	

Figure 4: Generating Students with Cellfie

4.2.2 Cellfie Rules for Rooms

To define individuals for rooms and their associated data, a worksheet named `Legenda` and several Cellfie rules are defined to extract room names, locations, and web pages.

- **Room Individuals:** Each entry in column `A` represents a room (e.g., `BT5_01`, `AulaInstalatii`) and is used to declare an individual of type `Room`:

```
Individual: @A*
  Types: Room
```

- **Location Individuals:** Each corresponding value in column `B` refers to a physical location of that room. These entries are used to create `Location` individuals:

```
Individual: @B*
  Types: Location
```

- **Linking Rooms to Locations:** Rooms (from column `A`) are connected to their locations (from column `B`) using the object property `hasLocation`:

```
Individual: @A*
  Facts: hasLocation @B*
```

- **Adding Web Pages:** If column `C` includes URLs or descriptions, these are added as values for the data property `hasWebPage` for each room:

```
Individual: @A*
  Types: Room
  Facts: hasWebPage @C*
```

This approach allows structured, batch creation of room-related data directly from Excel, mapping real-world attributes into OWL individuals with associated object and data properties.

5 Constraints in SHACL

SHACL (Shapes Constraint Language) allows the definition of constraints on RDF data to ensure its validity against a given schema. Below are three example constraints defined using SHACL for validating the properties of different classes in an ontology.

5.1 Constraint on `Group` Class: Year of Study

The `GroupShape` defines a constraint on the `ex:Group` class, specifically on the `ex:inYear` property. The constraint ensures that the value must be an integer and one of the valid academic years: 1, 2, 3, or 4.

```
ex:GroupShape
  a sh:NodeShape ;
  sh:targetClass ex:Group ;
  sh:property [
    sh:path ex:inYear ;
    sh:datatype xsd:integer ;
    sh:in (1 2 3 4) ;
    sh:message "inYear must be one of: 1, 2, 3, or 4." ;
  ] .
```

Explanation:

- `sh:targetClass` applies the shape to all instances of the `Group` class.
- `sh:datatype` ensures the value is of type `xsd:integer`.
- `sh:in` restricts the value to a specific set of integers: 1, 2, 3, or 4.

5.2 Constraint on `Timeslot` Class: Valid Weekdays

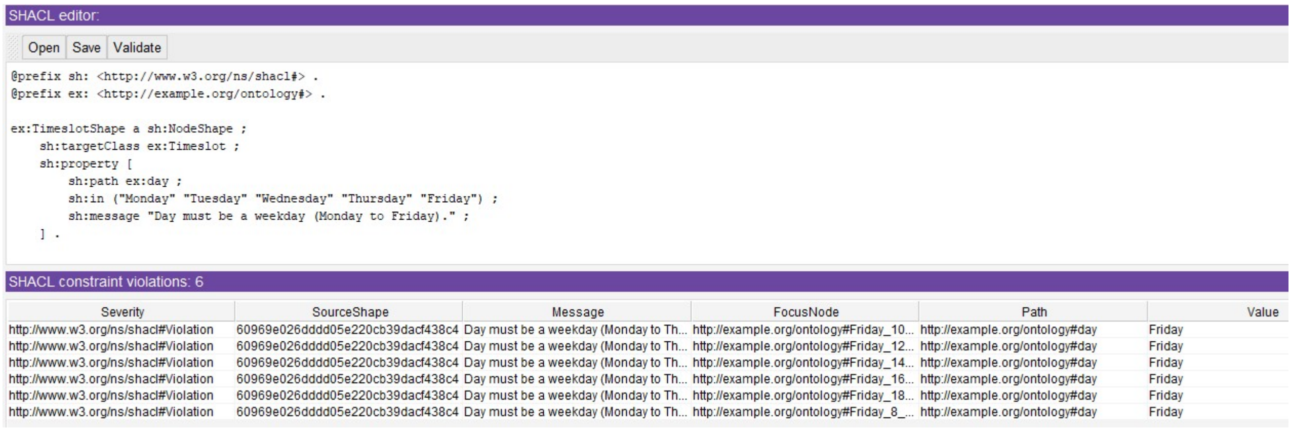
The `TimeslotShape` applies to the `ex:Timeslot` class and constrains the `ex:day` property to allow only valid weekdays.

```
ex:TimeslotShape a sh:NodeShape ;
    sh:targetClass ex:Timeslot ;
    sh:property [
        sh:path ex:day ;
        sh:in ("Monday" "Tuesday" "Wednesday" "Thursday" "Friday") ;
        sh:message "Day must be a weekday (Monday to Thursday)." ;
    ] .
```

Explanation:

- The `sh:in` constraint restricts the allowed values to five specific weekday strings.
- This helps ensure that no invalid or weekend days (e.g., ”Sunday”) are assigned as timeslot values.

In order to validate the constraint, ”Friday” was eliminated from the list. As a result, the Individuals having this day were detected as violations (see Fig. 5).



```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix ex: <http://example.org/ontology#> .

ex:TimeslotShape a sh:NodeShape ;
    sh:targetClass ex:Timeslot ;
    sh:property [
        sh:path ex:day ;
        sh:in ("Monday" "Tuesday" "Wednesday" "Thursday" "Friday") ;
        sh:message "Day must be a weekday (Monday to Friday)." ;
    ] .
```

SHACL constraint violations: 6

Severity	SourceShape	Message	FocusNode	Path	Value
http://www.w3.org/ns/shacl#Violation	60969e026dddd05e220cb39dacf438c4	Day must be a weekday (Monday to Th...	http://example.org/ontology#Friday_10...	http://example.org/ontology#day	Friday
http://www.w3.org/ns/shacl#Violation	60969e026dddd05e220cb39dacf438c4	Day must be a weekday (Monday to Th...	http://example.org/ontology#Friday_12...	http://example.org/ontology#day	Friday
http://www.w3.org/ns/shacl#Violation	60969e026dddd05e220cb39dacf438c4	Day must be a weekday (Monday to Th...	http://example.org/ontology#Friday_14...	http://example.org/ontology#day	Friday
http://www.w3.org/ns/shacl#Violation	60969e026dddd05e220cb39dacf438c4	Day must be a weekday (Monday to Th...	http://example.org/ontology#Friday_16...	http://example.org/ontology#day	Friday
http://www.w3.org/ns/shacl#Violation	60969e026dddd05e220cb39dacf438c4	Day must be a weekday (Monday to Th...	http://example.org/ontology#Friday_18...	http://example.org/ontology#day	Friday
http://www.w3.org/ns/shacl#Violation	60969e026dddd05e220cb39dacf438c4	Day must be a weekday (Monday to Th...	http://example.org/ontology#Friday_8_...	http://example.org/ontology#day	Friday

Figure 5: Intentional Violation

5.3 Constraint on `Department` Class: Email Pattern

The `DepartmentShape` applies a pattern constraint on the `ex:hasEmail` property of the `Department` class to ensure that email addresses follow a valid format.

```
ex:DepartmentShape
  a sh:NodeShape ;
  sh:targetClass ex:Department ;
  sh:property [
    sh:path ex:hasEmail ;
    sh:datatype xsd:string ;
    sh:pattern "^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$" ;
    sh:message "Email must be a valid email address." ;
  ] .
```

Explanation:

- The `sh:pattern` defines a regular expression that matches standard email formats.
- The `sh:datatype` ensures the value is a string.
- If the email does not match the pattern, the custom message informs the user of the invalid format.

6 Reasoning and Consistency Checking

Using the Pellet reasoner, the ontology was checked for consistency. It correctly inferred subclass relationships such as `BSc, MSc, PhD` being subclasses of `StudyProgram`, and confirmed disjointness between `Student` and `Teacher` classes.

7 SWRL Tab Query

In this section, it is presented a set of SWRL (Semantic Web Rule Language) rules used to infer relationships and properties within our ontology. These rules operate on the concepts of schedule entries, rooms, timeslots, courses, and teachers to derive additional knowledge.

- **Occupancy inference:**

$$\begin{aligned} &\wedge \texttt{autogen0:ScheduleEntry}(?se)\wedge \\ &\wedge \texttt{autogen0:hasRoom}(?se, ?r)\wedge \\ &\wedge \texttt{autogen0:hasTimeslot}(?se, ?t) \\ \rightarrow\ &\wedge \texttt{autogen0:isOccupiedAt}(?r, ?t) \end{aligned}$$

- **Course equivalence:**

$$\begin{aligned}\wedge\texttt{autogen0:hasMatch}(?a, ?b)\wedge\\ \wedge\texttt{autogen0:Course}(?a)\wedge\\ \wedge\texttt{autogen0:Course}(?b)\\ \rightarrow\ \wedge\texttt{sameAs}(?a, ?b)\end{aligned}$$

- **Teaching assignment:**

$$\begin{aligned}\wedge\texttt{autogen0:ScheduleEntry}(?se)\wedge\\ \wedge\texttt{autogen0:hasTeacher}(?se, ?t)\wedge\\ \wedge\texttt{autogen0:hasCourse}(?se, ?c)\\ \rightarrow\ \wedge\texttt{autogen1:isTaughtBy}(?c, ?t)\end{aligned}$$

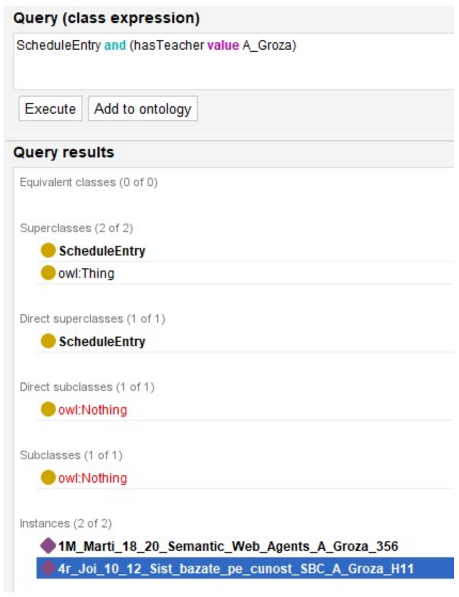
- **Teacher location inference:**

$$\begin{aligned}\wedge\texttt{autogen0:ScheduleEntry}(?se)\wedge\\ \wedge\texttt{autogen0:hasTeacher}(?se, ?t)\wedge\\ \wedge\texttt{autogen0:hasRoom}(?se, ?r)\\ \rightarrow\ \wedge\texttt{autogen1:isFoundAt}(?t, ?r)\end{aligned}$$

These SWRL rules enable automated reasoning to enrich the ontology with inferred facts about room occupancy, course identity, teaching assignments, and teacher locations, facilitating advanced queries and analysis of scheduling data.

8 DL Queries

The following is an example of a Description Logic (DL) query used to retrieve schedule entries taught by a specific teacher (see Fig. 6):


Query (class expression)
ScheduleEntry and (hasTeacher value A_Groza)

Query results
Equivalent classes (0 of 0)
Superclasses (2 of 2)
● ScheduleEntry
● owl:Thing
Direct superclasses (1 of 1)
● ScheduleEntry
Direct subclasses (1 of 1)
● owl:Nothing
Subclasses (1 of 1)
● owl:Nothing
Instances (2 of 2)
◆ 1M_Marti_18_20_Semantic_Web_Agents_A_Groza_356
◆ 4r_Joi_10_12_Sist_bazate_pe_cunost_SBC_A_Groza_H11

Figure 6: DL Query

Query: Retrieve all schedule entries taught by the teacher `A_Groza`.
In DL (Manchester Syntax), this can be expressed as:

$$\texttt{ScheduleEntry} \sqcap (\texttt{hasTeacher value A\_Groza})$$

This query selects all individuals of type `ScheduleEntry` that have the `hasTeacher` relationship with the individual `A_Groza`.

9 SPARQL Queries

This section presents a variety of SPARQL queries used to extract information from the ontology. These queries were tested using tools such as the SPARQL query tab in Protégé or external engines. Each query was designed to illustrate practical retrieval tasks from the university domain.

1. Query 1: List Students and Their Groups: This query retrieves students along with their first name, surname, and assigned group.

   ```
   SELECT ?student ?firstName ?surname ?group WHERE {
     ?student a base:Student ;
              base:hasFirstName ?firstName ;
              base:hasSurname ?surname ;
              base:isInGroup ?group .
   }
   ORDER BY ?group ?surname
   ```

2. Query 2: Students with Study Programs: Find students and the study programs they are enrolled in via their group.

   ```
   ?group base:isEnrolledIn ?studyProgram
   ```

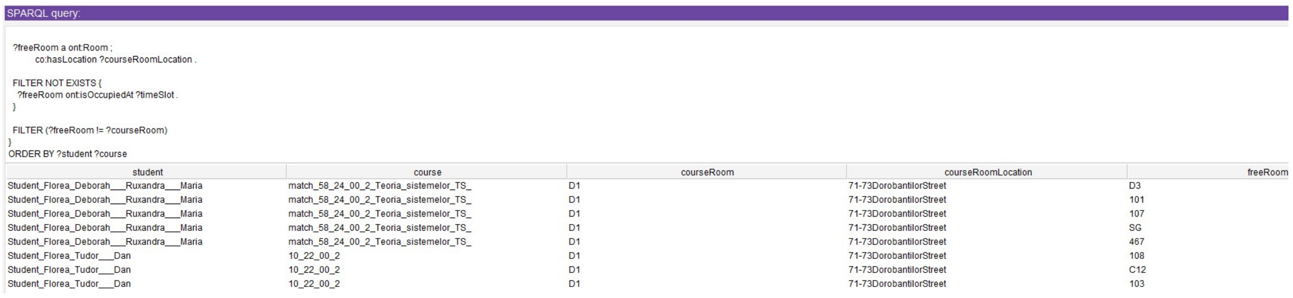
3. Query 3: Students Enrolled in ”Electrotechnics”: Filters students who take a course titled ”Electrotechnics”.

   ```
   FILTER(str(?courseTitle) = "Electrotechnics")
   ```

4. Query 4–5: Courses in Schedule Entries: Extracts courses linked to schedule entries, optionally including course titles.
5. Query 6: Total Credits per Student: Aggregates the total number of credits for each student across their program.
6. Query 7–8: Teachers, Rooms, and Courses: These queries show which teachers are scheduled in which rooms, and which courses they are teaching.
7. Query 9–10: Students and Their Full Schedule: Combines students with the full schedule, including course, room, and teacher details.
8. Query 11: Teachers and Their Office Location: Retrieves the physical location (office) of each teacher using custom properties like `co:isFoundAt`.
9. Query 12: Full View — Students to Teacher Location: A comprehensive query that joins student, course, room, and teacher location.
10. Query 13: Shared Rooms by Teachers: Finds rooms where more than one distinct teacher has scheduled entries.
11. Query 14: Number of Students per Course: Counts how many students are linked to each course through their program enrollment.
12. Query 15: Classroom and Location per Student Course: Retrieves classrooms and physical locations for all scheduled sessions attended by students.
13. Query 16: Course Titles and Their Rooms: Links courses to the rooms where they are held.
14. Query 17: Student Peers in the Same Program: Finds peer students attending the same program via shared course teaching relationships.
15. Query 18: Course Count Per Room: Counts the number of distinct courses scheduled in each room.
16. Query 19: Students in ”Aula Instalatii”: Lists students who are scheduled to attend courses in the ”Aula Instalatii” room.
17. Query 20: Students and Alternative Free Rooms: Finds alternative rooms at the same location that are free during the same time slot a student has a course (see Fig. 7).

18. Query 21: Most Occupied Room: Finds rooms with the most occurrences in the timetable.
19. Query 22: Locations and Their Distances: Retrieves all locations along with their distance to 26–28 Gh. Baritiu Street.
20. Query 23: Medium Distance Locations: Finds locations that are between 2.0 and 5.0 units of distance from 26–28 Gh. Baritiu Street.
21. Query 24: Teachers per Study Programme: Counts how many distinct teachers are associated with each study programme by linking students to their programme and identifying the teachers of courses in that programme.

This diverse set of SPARQL queries demonstrates the richness of the ontology and its potential in real-world scheduling, reporting, and analysis tasks.



```
 ?freeRoom a ont:Room ;
        co:hasLocation ?courseRoomLocation .

 FILTER NOT EXISTS {
   ?freeRoom ont:isOccupiedAt ?timeSlot .
 }

 FILTER (?freeRoom != ?courseRoom)
}
ORDER BY ?student ?course
```

student	course	courseRoom	courseRoomLocation	freeRoom
Student_Florea_Deborah___Ruxandra___Maria	match_58_24_00_2_Teoria_sistemelor_TS_	D1	71-73DorobantilorStreet	D3
Student_Florea_Deborah___Ruxandra___Maria	match_58_24_00_2_Teoria_sistemelor_TS_	D1	71-73DorobantilorStreet	101
Student_Florea_Deborah___Ruxandra___Maria	match_58_24_00_2_Teoria_sistemelor_TS_	D1	71-73DorobantilorStreet	107
Student_Florea_Deborah___Ruxandra___Maria	match_58_24_00_2_Teoria_sistemelor_TS_	D1	71-73DorobantilorStreet	SG
Student_Florea_Deborah___Ruxandra___Maria	match_58_24_00_2_Teoria_sistemelor_TS_	D1	71-73DorobantilorStreet	467
Student_Florea_Tudor___Dan	10_22_00_2	D1	71-73DorobantilorStreet	108
Student_Florea_Tudor___Dan	10_22_00_2	D1	71-73DorobantilorStreet	C12
Student_Florea_Tudor___Dan	10_22_00_2	D1	71-73DorobantilorStreet	103

Figure 7: Output of Query 20

10 Challenges and Solutions

Challenges included:

- Designing appropriate properties to avoid circular dependencies.
- Ensuring consistency across multiple hierarchical classes.
- Mapping complex real-world scheduling scenarios into OWL.

Solutions involved modular ontology design and use of reasoning to validate constraints.

11 Conclusion

The ontology successfully models the key entities of a faculty environment, including academic staff, students, curricula, rooms, and schedules. Future improvements could involve integration with external ontologies (e.g., FOAF for people) and building a front-end query interface using SPARQL.

12 References

- Protégé OWL Editor
- OWL 2 Web Ontology Language Document Overview
- Pellet Reasoner
- Pytesseract