Finding Extracurricular Courses using a RDF Dataset

Tim Brandt Corstius, Azra Çinar, Satiga Godrie, Benn Samuel Stroschein

Vrije Universiteit Amsterdam

1 Goal

The StudyGuide [1] from the Vrije Universiteit Amsterdam offers information on all offered courses. However, a specific course can only be found by knowing the name, identifier or the study program where the course is a part of the curriculum. Not only are some courses not part of an official curriculum but finding courses by specific properties such as the number of credits, level, faculty or which professor teaches the course is currently not possible. The goal of this project is develop an enhanced studyguide for the Vrije Universiteit Amsterdam that makes use of RDF (Resource Description Framework) in order to deliver users/students an enhanced method of finding courses by certain properties.

2 Users

The target audience of the project are students who are searching for courses that can be taken at the Vrije Universiteit Amsterdam which are not part of their official curriculum. Finding extracurricular courses is hampered by the non-existent possibility to search by properties on the universities in-house platform to find courses named "StudyGuide". A simple search by properties could be a student that is interested in which courses the faculty of science generally offers or, a more specific example, which courses the faculty of science offers in period 4 that have the level 200 (2^{nd} year bachelor). Neither simple or more complex searches for courses are currently possible. A workaround students use is to search for specific keywords like "programming". However, neither is guaranteed that all relevant courses about programming are found, nor that courses have the keyword "programming" in their name. An example of a course that is about programming but does not contain the keyword "programming" in it's title is the course "Project Application Development". A student could have found the course by looking for courses offered by the faculty of science without being overwhelmed by all 2064 courses the Vrije Universiteit has to offer.

3 Design

The website is designed as one page website. Figure 1 illustrates an early prototype of the design when a user first opens the website. The design revolves around the possibility to search for courses by filtering on properties. These properties include: period in which the course is given, difficulty level of the course, number of credits for the course, professor who is giving the course and language of the course. When a certain property has less or equal to 6 options, the requirements are displayed using check boxes. If a property has more than 6 options then it will be displayed using a search box which a user can search words in, the search box will give the options in a drop-down list dynamically while searching. Displaying more than 6 options using check boxes creates too many options which are difficult to search trough as a user. After having selecting all criterion the user a button named "GO" can be pressed. This button executes a

SPARQL query. While the SPARQL query is executing, a loading animation will indicate to the user that the query is executing. This step is beneficial as it makes sure the user does not think the application is broken when it is still fetching. The results will be displayed as can be seen in Figure 2. All course names that match the criteria of the query are displayed in an ordered list. Every course name is expandable and once expanded the user can read more information about that particular course. All courses are collapsed in the first instance in order to give a more clear overview of all courses that have matched the user's input.

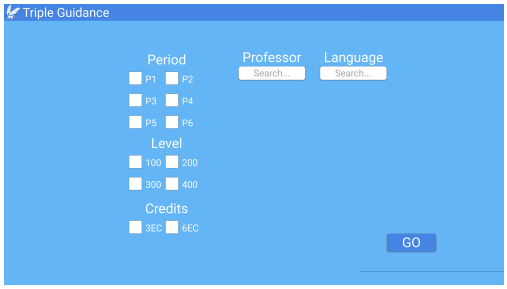

Fig. 1. Early Prototype of the Home Screen

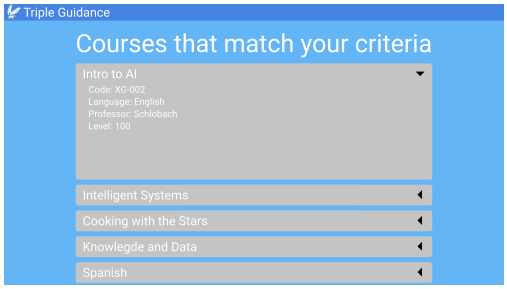

Fig. 2. Early Prototype of the Results

4 Walkthrough

To illustrate a walkthrough of the product, a fictional student who is looking for a course with the following properties is used as illustration: Courses offered in period 4 that have the level 200 (2^{nd} year bachelor).

The student starts on the left side (see Figure 3) in the top by selecting the desired period the courses are taught. When selecting the 4^{th} period, all courses that start in the 4^{th} period will be displayed when executing the search, thus courses in this period that have a duration of more periods and have started before this period will not be displayed.
The level of the course can be selected, see Figure 3, indicating the minimum required academic level of the student. The student chooses level 200. This search will only display courses with this exact level of 200 and not the courses below this level, for which the student is also qualified.
The student is able to indicate the number of study credits the course should have, similarly to the course level choice, see Figure 3. Only the courses with the exact number of chosen study credits will be displayed. Similar to the course level selection, multiple boxes can be chosen for EC credits as well. In the filter boxes on the left, the student is able to narrow down the courses by choosing a specific professor and a specific language the course will be taught in.
Once the student has selected his properties of interest, clicking on the "GO" button executes the search. Afterwards the results are shown in which the student can find the courses of interest. Only chosen filters will be executed on the results, the filters which are not chosen will be disregarded.

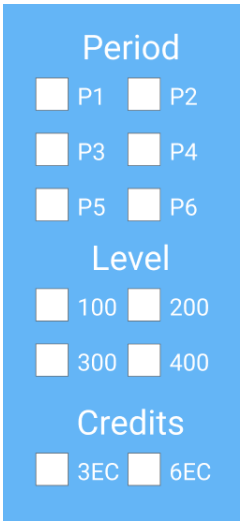

Fig. 3. Left-hand side property selection of the Home Screen

5 Identify 2 ontologies

The first ontology language we will be using is the TEACH [2] which stands for Teaching Core vocabulary. TEACH [2] contains classes such as Course, Student, and Module which will allow us to gather different types of instances within proper classes. Additionally, the properties of TEACH [2] will enable us to form the relevant inter-class and intra-class relations. For instance, the TEACH [2] properties hasTitle and hasCourse-Material could be use to define characteristics of a course whereas the properties teacherOf and teacher could allow us to relate the courses with the the relevant professors. The second ontology which will be considered is AIISO **AISSO**. AIISO **AISSO**, in other words The Academic Institution Internal Structure Ontology provides classes and properties to describe the internal organizational structure of an academic institution. Using AISSO **AISSO** could be helpful when identifying further characteristics of academic properties. For instance, the terms Faculty, Department and Programme could be used to gather the different information types related to the courses. Additionally, the course codes could be displayed under the term Code and the courses could be gathered according to their subjects through the term Subject. Moreover, the property partOf could define the belonging of a programme to a specific faculty/department.

6 Sources

One of the external sources of data which will be used is the Study Guide website of the Vrije Universiteit Amsterdam. Since the project aims to improve searching for courses for students, data about the courses including their properties offered at the Vrije Universiteit Amsterdam is necessary. In this regard, Study Guide displays a broad amount of course data which is necessary for the application. Since this source is not in an RDF-schema, nor has a SPAQRL endpoint, the data needs to be gathered and converted to RDF. The second source will be DBPedia. Through the use of DBPedia, we aim to enlarge the information range

about instances. For example, if a specific course and/or professor has information on DBPedia, then this information could be added to give a more detailed insight on these instances.

7 Domain and Scope

The application aims to help students finding specific courses through selection of given filters. Therefore, the ontology consists of courses with their titles, content, objectives, levels, target audience and their teaching methods as well as professors by whom the courses are taught, academic term in which the courses are offered and faculty where the courses belong. In this regard, the existing material on StudyGuide of Vrije Universiteit Amsterdam as well as the information within the vocabularies TEACH and AISSO are used but a conceptualization had to be made according to the type of each instance and their relationship with each other. Thus, the classes such as course level, course period, course credits (ECTS), course language, and professor are created and the related instances are connected to them through the appropriate properties. In this manner, the final version of the ontology is obtained.

8 Methodology

The ontology was created by scraping all courses and their corresponding information that the Vrije Universiteit listed in their StudyGuide [1]. Since the platform StudyGuide does not display all offered courses, a list with all courses, in particular their unique identifier, had to be gathered. The unique identifiers of the courses were used to scrape the corresponding StudyGuide pages of the course. The information on a course page was then transformed into triples. An example would be vuc:courseId teach:courseTitle Literal(courseTitleOfScrapedCourse). While converting the course data into triples, reusing existing ontologies/vocabularies was emphasized. Three existing vocabularies were used, namely TEACH [2], AIISO [3] and DBO [4]. Properties for which no existing vocabulary was found, a namespace VU, which links to the official website of the Vrije Universiteit Amsterdam was created [5]. All courses instances have the namespace VUC which stands for VUCourses and links to the official course page on StudyGuide. Furthermore, suitable course properties link to external instances. For example, a course has a language property in which the course is taught, that language property does not point towards a Literal but instead to DBPedia's page about the language.

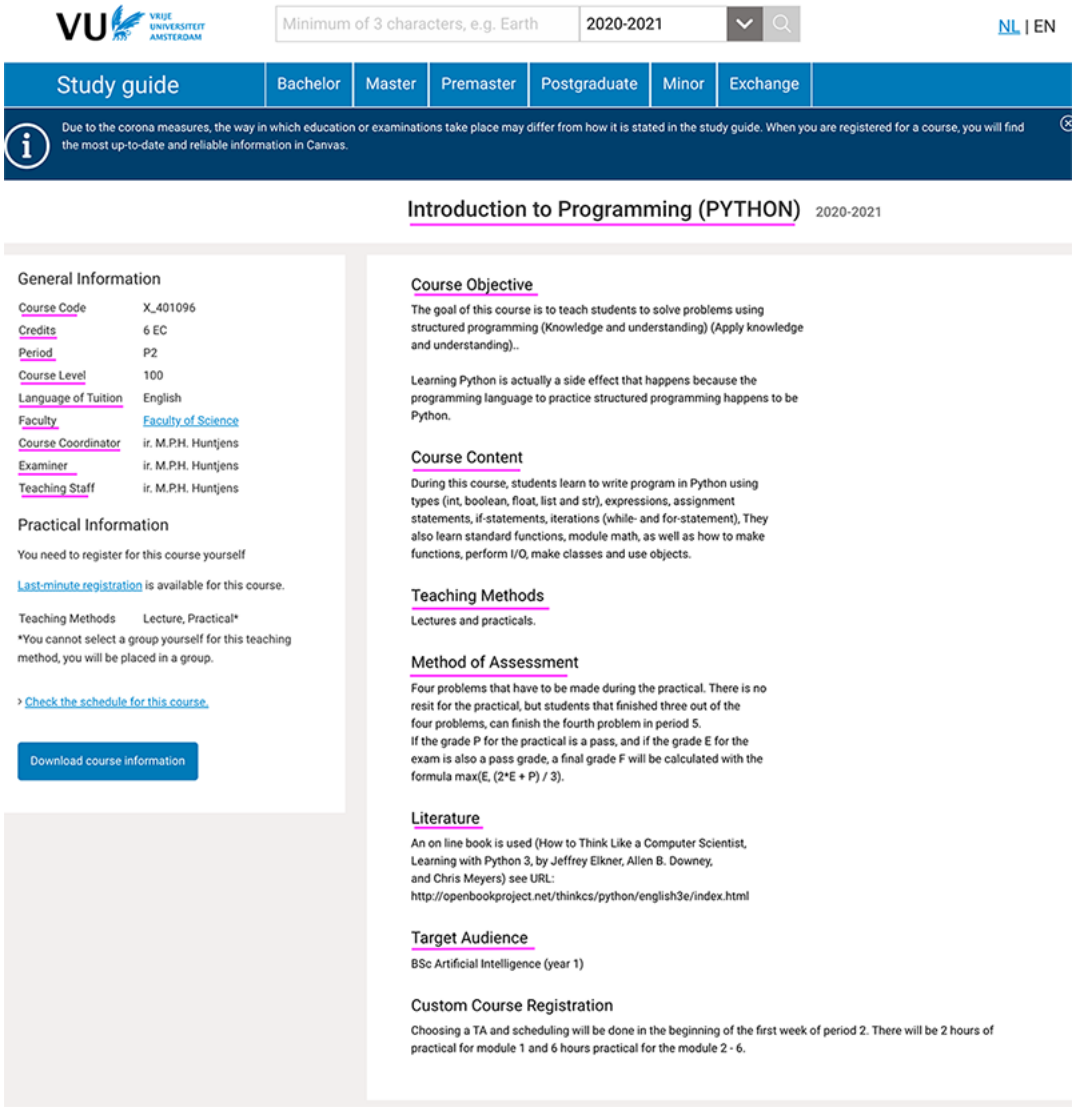

Fig. 4. An exemplary course page on StudyGuide. The pink underlined attributes are transformed into triples if available.

9 Conceptualization

The main classes in the ontology are Course, Faculty and Person. The TEACH vocabulary is used to create the class Course and the Person subclass Teacher. Properties are also created using the TEACH vocabulary such as academicTerm, courseTitle, ects and grading. Besides these, through StudyGuide, the following

properties were created: courseContent, courseLevel, courseObjective, literature, offeredByFaculty, recommendedBackground, targetAudience, taughtBy and teachingMethods. With these properties, we obtained a more detailed insight into the content of the courses.

The classes Ects3 and Ects6 are created to assign the courses based on the number of study credits they have. Similarly, the classes CourseLevel100, CourseLevel200, CourseLevel300, CourseLevel400, CourseLevel500 and CourseLevel600 are created to assign the courses to the corresponding level. DBPedia ontology is used for the class Person, what is used to create the two subclasses Teacher and Professor for.

As for the vocabulary AISSO, it is used to create the class Faculty. Additionally, a number of classes were created within the created namespace VU.These classes were used to assign the courses through their types regarding the faculty they are offered in, such as ScienceCourse, SocialScienceCourse, HumanitiesCourse, LawCourse and TheologyCourse. Figure shows a drawing of the concept of the ontology.

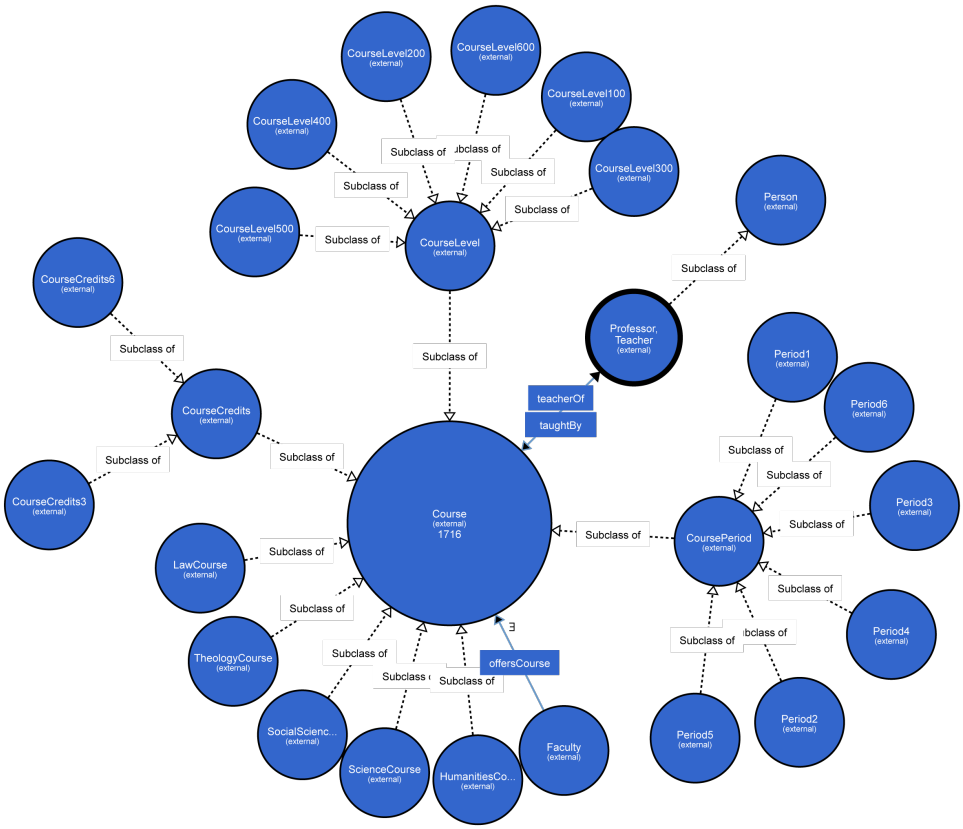

Fig. 5. The concept overview

10 Ontology

The ontology displays the conceptualization described in the chapter Conceptualization. As mentioned earlier, different vocabularies are used to create the final version of our ontology such TEACH and AIISO vocabularies as well as the StudyGuide and DBPedia sources. Some of the main classes of our ontology, for instance class Course and class Faculty, are respectively created through TEACH and AISSO vocabularies. The subclasses of Course are defined with the help of the corresponding information available in the StudyGuide namely for course credits, the level of courses, course periods, course types regarding the faculty they belong to as well as for professors. The owl class restriction "Equivalent To" is used to extract the required instances to related subclasses. For instance, to concatenate the courses through the course credits they have, the course levels and the academic term they belong, and the course type they are assigned to, the owl restriction "value" is used. Besides, the owl restriction type "some" is also used to relate the course instances and the faculty instances they are offered in and thus the Faculty class is inferred. Furthermore, the teachers and professors are respectively gathered separately within the class Teacher and within the class Professor through the same owl restriction type. Additionally, class equivalences are made through the owl class restriction "owl:equivalentClass" for instance, to make two subclasses equal to each other (the subclass Teacher and the subclass Professor). As for the object properties, the owl property restriction "owl:equivalentProperty" is used to assign two properties as same and "owl:inverseProperty" is used to define the inverse property relationship.

11 Integration of External Data

The integration of external data happened at the beginning of project when we transformed all course data from StudyGuide to a triple format. While creating the rules for the triples, the additional source DBPedia was directly integrated. In other words, while creating the ontology both external sources where interwoven. Creating the ontology from scratch enabled to us to use different vocabularies such as TEACH and AIISO and partially DBO right away. A concrete example is the language property of courses which directly links to the corresponding DBPedia resource: vuc:someCourse dbo:language dbr:Dutch_language. Linking directly do DBPedia allows to theoretically display any additional information about properties in the web app. However, we only implemented a proof of concept in the form of linking to DBPedia but not embedding information from DBPedia directly into the website. Although this approach simplified the ontology itself because no mapping of classes and properties from different vocabularies had to be made, it complicated fulfilling the assignments requirements of having at least 15 different classes. If we would have merged two ontology's from the Web, the resulting ontology would likely contain at least 15 different classes. Mapping the different classes with owl restrictions is then a must. However, no suitable dataset was found. The possibility to gather course data from another Dutch university, merge them and creating a regional StudyGuide was considered but discarded because the classes and properties would have been identical.

12 Inferencing

The ontology created contains 27 different classes. The main class is called Course. This class contains all the instances of courses. All other classes (as can be seen in Figure 6) only contain inferenced instances. The CourseCredits, CourseLevel and CoursePeriod classes all make use of the same type of inferencing rules, only the subclasses of these classes contain rules. For example: CourseCredits3 contains the rule Equivalent To 'ects value 3' see Figure 7. Hereby ects is a data property and every course with 3 EC's has to be in

that class, therefore the value is 3. The same is done for CourseCredits6, only then with 'value 6' instead of 'value 3'. The CourseLevel classes have all instance courses with their corresponding level inferenced to it. For example: CourseLevel100 contains all level 100 courses. It contains the rule Equivalent To 'courseLevel value 100', whereby courseLevel is a Data property which contains the level of each course. This is done for every CourseLevel class, only with their corresponding levels. The CoursePeriod class contains Period classes with each their own period. For example: Period1 contains all courses that are given in period 1 of the year. This is done by the inferencing rule Equivalent To 'academicTerm value "P1"@en', whereby academicTerm is a data property which has the label English. This is repeated for all other Periods. The classes HumanitiesCourse, LawCourse, ScienceCourse, SocialScienceCourse and TheologyCourse contain courses that are given at that particular faculty. For example the class HumanitiesCourse contains the inferencing rule Equivalent To 'offeredByFaculty value 'Faculty of Humanities' whereby offeredByFaculty is a object property that connects courses to their faculty. The same is done for all other faculties only then with their required value. The class Faculty contains a list of all faculties, see Figure 8, created by the inferencing rule Equivalent To 'offersCourse some Course', whereby offersCourse is a object property. offersCourse is the inverse of offeredByFaculty. The class Person contains the subclass Professor and Teacher. Teacher has the rule Equivalent To 'Professor', whereby 'Professor' is referring to the class Professor. The class Professor contains the rule Equivalent To 'teacherOf some Course', whereby teacherOf is an object property. teacherOf is an inverse of taughtBy and therefore if an course is taught by someone then a professor is the teacher of that course. If someone officially teaches some course he/she is a professor see Figure 9. The Teacher class is Equivalent to the Professor class and therefore also contains the same professors.

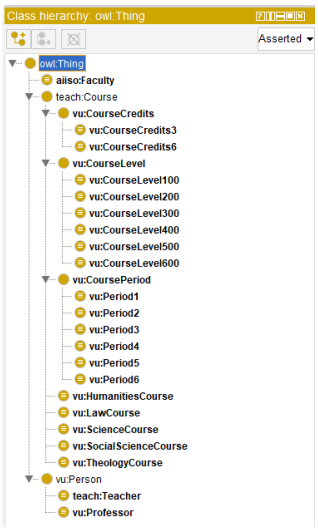

Fig. 6. An overview of all classes

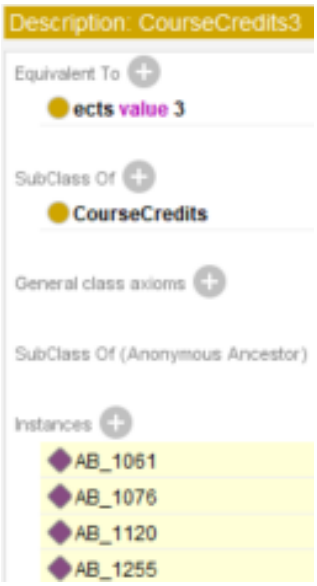

Fig. 7. Part of the inferenced results in the CourseCredits3 class

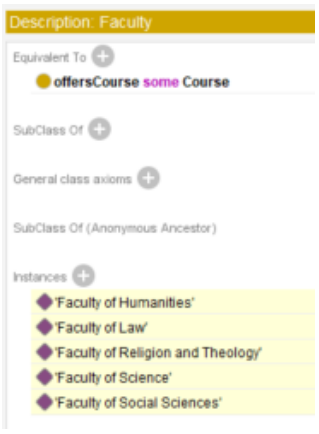

Fig. 8. Part of the inferenced results in the Faculty class

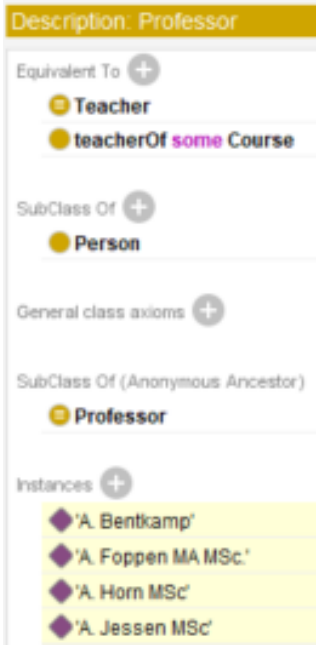

Fig. 9. Part of the inferenced results in the Professor class

13 Example SPARQL Queries

The SPARQL Query in Figure 10 used in the application can be seen below. This query selects almost everything except the variables (denoted by ?*varbiablename*) that return a URI as this is not something we want to show to the end user. Each variable that is selected gets used as a label in the resulting treeview. For example: ?Level will result in the treeview as Level: result from ?level. This way of creating labels does cause the problem that labels can't have spaces and that some variables are a bit unclear. One example of a variable being unclear is that instead of having *?Faculty rdfs:label ?FacultyLabel* we have *?FacultyName rdfs:label ?Faculty*. This is because otherwise the label would become TeacherLabel: and that was unwanted for the application. The filters at the bottom are used to make the filter functionality is possible on the website. The filters get parsed to the full query when pressing one of the filter buttons. By default the filter has all possible options separated by || (or) statements. For example: the ecFilter has *"?Credits = '6'^^xsd:integer || ?Credits = '3'^^xsd:integer"* as the default value and when selecting 3 EC it becomes *"?Credits = '3'^^xsd:integer"* and when selecting 6 EC it becomes *"?Credits = '6'^^xsd:integer"*. This method was done for every filter that used radio buttons. For filtering professors it was unfeasible to create a default query for every value separated by || as there are more than a 1000 professors at the VU. Therefore we first wanted to only insert a FILTER query when a professor was selected but our SPARQL endpoint only allows a maximum of 10 seconds running time per query (limitation of Vercel, which is used to build the query, see Application Development section) and this filter function took too long. An if statement was created in order to check if a professor was selected in the filter. If it was empty it used below query, if it was not empty a new query was created but 2 lines were added, below the *?StudieGids_URL rdf:type teach:Course;* the line

vu:taughtBy ?TeacherName. was added. Below *teach:courseTitle ?Title;* the line *?TeacherName rdfs:label 'teacherFilter'@en.* was added. Hereby the teacherFilter is a teacher that has been selected in the filter. This query speeds up the query processing code at the endpoint enough so it is able to run within 10 seconds. This SPARQL query in Figure 10 is used by the application and is fully functional. The query is limited to 800 results. Testing has confirmed that when a query has to return more than 800 results it takes more than 10 seconds. Therefore each query is limited to 800 results.

```
    select DISTINCT ?StudieGids_URL ?Credits ?Level ?Period ?Title ?Grading ?Content
    ?Objective ?Teaching_Method ?Literature ?Language ?Faculty
    where {
          ?StudieGids_URL rdf:type vu:CourseCredits;
                      teach:ects ?Credits.
          ?StudieGids_URL rdf:type vu:CourseLevel;
                      vu:courseLevel ?Level.
          ?StudieGids_URL rdf:type vu:CoursePeriod;
                      teach:academicTerm ?Period.
          ?StudieGids_URL rdf:type teach:Course;
                      dbo:language ?LanguageName;
                      teach:courseTitle ?Title.
           OPTIONAL {
         ?StudieGids_URL teach:grading ?Grading;
            vu:courseContent ?Content;w
            vu:courseObjective ?Objective;
            vu:offeredByFaculty ?FacultyName;
            vu:literature ?Literature;
            vu:teachingMethods ?Teaching_Method.
          ?FacultyName rdfs:label ?Faculty.
          ?LanguageName rdfs:label ?Language.
     }
FILTER (${$ecFilter})
FILTER (${$levelFilter})
FILTER (${$periodFilter})
FILTER (${$languageFilter})
    }
    LIMIT 800
```

Fig. 10. Sparql Query

The original plan was to gather additional information about the instruction language of courses using DBPedia. Unfortunately RDFLib, which is used for our SPARQL Endpoint [6] does not support a SERVICE query 11. The addition to the query made according to the original plan did work in GraphDB but not when using our own SPARQL endpoint. Therefore we have chosen to automate the adding of a language and its label to each course using python, this is done while scraping. The SERVICE query can be seen below.

```
SERVICE <http://dbpedia.org/sparql> {
      ?language rdfs:label ?languagelabel .
      FILTER ( LANG ( ?languagelabel ) = "en")
}
```

Fig. 11. Service Query Part

Another problem that was encountered was that a query using no filter for the professor returned multiple instances of the same courses, each with a different professor as a property. This is fixable with the below Concat Query 12. Unfortunately RDFLib also does not support concat in a SPARQL query. Therefore it was decided to remove the teacher from the treeview on the website. It is still possible to filter on a teacher only the filtered teacher will not be displayed in the course overview.

```
select DISTINCT ?StudieGids_URL (group_concat ( distinct ?Teacher ; separator = ", ") AS ?Teacher)
 {
?StudieGids_URL rdf:type teach:Course;
          vu:taughtBy ?TeacherName;
           OPTIONAL {
?TeacherName rdfs:label ?Teacher.
  }
}
group by ?StudieGids_URL
```

Fig. 12. Concat Query Part

13.1 Description of SPARQL Queries

The part of the whole SparQL query 10 that makes use of inferences triples can be seen below in Figure 13.

```
   select DISTINCT ?StudieGids_URL ?Credits ?Level ?Period where {
?StudieGids_URL rdf:type vu:CourseCredits;
     teach:ects ?Credits.
    ?StudieGids_URL rdf:type vu:CourseLevel;
     vu:courseLevel ?Level.
    ?StudieGids_URL rdf:type vu:CoursePeriod;
              teach:academicTerm ?Period.
}
```

Fig. 13. Concat Query Part

The query above in Figure 13 fetches the instances from the class CourseCredits and the ects value from each instance. As explained in the Inferencing section 12 the class CourseCredits contains subclasses which make use of inferencing rules. The only instances that are in the CourseCredits class are inferenced instances. The inferencing part of the query in Figure 13 also fetches every CourseLevel and CoursePeriod class and their corresponding courseLevel and academicTerm instances. These classes were also described in **??** as classes that contain multiple subclasses. Each subclass contains only inferred instances, therefore everything that is fetched only works if inferencing is turned on. Provided below are screenshots which show that the query only works when inferencing is turned on.

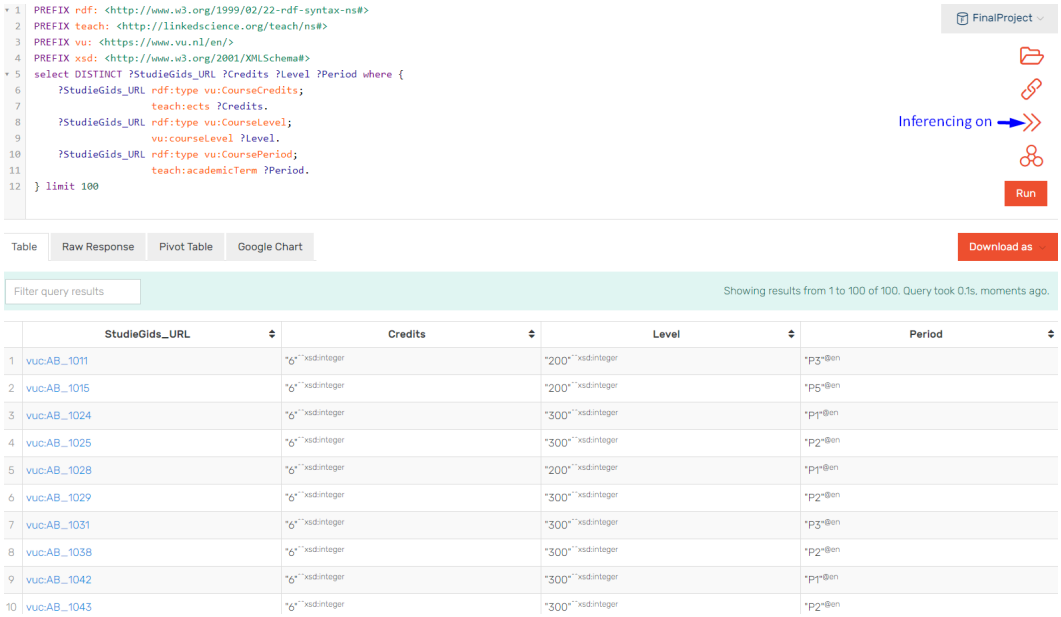

Fig. 14. Sparql query results in GraphDB with inferencing turned on.

Fig. 15. Sparql query results in GraphDB with inferencing turned off.

14 Application Development

Because we think that this project might be used by students, or the source code is used to improve this project, we emphasized developing a self-running application without the need to run a server hosting our constructed ontology. Therefore, not only did we develop a working prototype but we implemented a functional SPARQL endpoint under the hosted domain followed by *.../api/sparql* [6]. The SPARQL endpoint solved two problems: First it allowed us to run the application serverless e.g. not having to start GraphDB on localhost, thus allowing everyone to use the site. Secondly, forking this project should become easier because the use of GraphDB is not necessary. The website itself uses Svelte [7] as framework which is an easy to learn, likely the easiest to learn, JavaScript frontend framework. Vercel [8] is used to host the website because of a generous free tier, and more importantly because Vercel allows us to run the SPARQL endpoint in a serverless function. A serverless function can act as an HTTP endpoint but in contrast to a constantly running server, computing time is only used when the function, a client making a request, is executed. The on-demand execution of the SPARQL endpoint in a serverless function costs 0 Euros and reduces complexity by eliminating the need for a server.

15 Honorary Title

As the data that has been scraped from the Vrije Universiteit Amsterdam's StudyGuide [1] was not available in a triple format, we have chosen to go for the Linked Data Producer Honorary Title and publish the data set on Zenodo.org. Permission from Viktor de Boer to upload the data set on Zenodo.org has been granted on 25-10-2020. This data set can be used in order to find specific information about courses without knowing the exact course name or code. For example, you can fetch all courses that have 3 credits and are offered in English, given at the faculty of Science. Therefore it becomes much easier for students to find courses which are outside of their major/master and which also fit their requirements. In the current situation without adding this data set, a student would need to search through all 2064 courses given at the VU to find a course that they like. However with this data set, the process requires remarkably less effort and thus becomes quicker and more efficient. The published data set can be accessed from here [9].

References

[1] V. Universiteit, *Studyguide*, https://studiegids.vu.nl/en.
[2] L. Science, *Teach ontology*, http://linkedscience.org/teach/ns/.
[3] Vocab.org, *Aiiso ontology*, https://vocab.org/aiiso/.
[4] DBPedia.org, *Dbpedia ontology*, http://dbpedia.org/ontology/.
[5] V. U. Amsterdam, *Vrije universiteit amsterdam website*, https://www.vu.nl/en/.
[6] *Sparql endpoint for the ontology developed in this project*, https://enhanced-vu-studyguide.vercel.app/api/sparql.
[7] Svelte, *Svelte introduction and basics*, https://svelte.dev/tutorial/basics.
[8] V. H. Platform, https://vercel.com/.
[9] B. S. Stroschein, T. B. Corstius, Satiga Godrie, and A. Çinar, *[rdf triples] courses given at the vrije universiteit amsterdam 2020/2021*, en, 2020. DOI: 10.5281/ZENODO.4129774. [Online]. Available: https://zenodo.org/record/4129774.