INTERNATIONAL HELLENIC UNIVERSITY	**School of Science and Technology** **MSc in Data Science**
Student ID: 3308210026 - Michailidis Alexios	**Intake: June 2022**
Subject: Knowledge Management - Project in XML	
Title of work: Project in OWL, SPARQL, SWRL	
Course leader: Professor Nick Bassiliades	**Submission date**:
I confirm that the work I have submitted is: My own unaided work Date:6/6/2022	
Marker's Feedback:	
	Final Mark:

TABLE OF CONTENTS

Presentation of the Library Model Classes 3

Object Attributes of the model 6

Data Properties of the model 8

Library Model Instances and Inferencing 9

Inconsistency Examples 12

SPARQL Queries 14

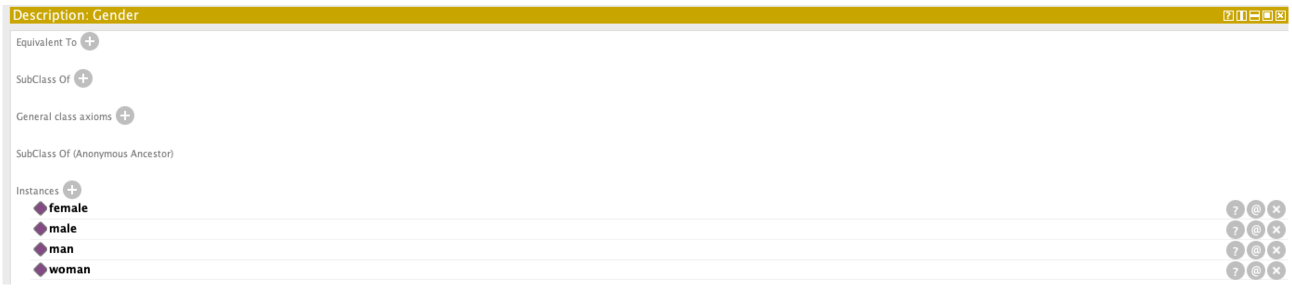
SWRL Rules 16

Conclusion 18

Presentation of the Library Model Classes

In this document, we will represent the university library ontology model, but this time the ontology will be constructed in OWL and the Protege software. For the construction of the ontology, except for the remarks provided on the previous deliverable also some further classes and properties were inserted both for completion and representation reasons.

So, let us recap the main ideas behind our model. The main purpose of the model is to represent the process that a user performs to proceed with a transaction (reservation, loan, return) on a copy of a book. For this reason, we have inserted four main classes. The first one was named "Person", which captures all the instances that have the property of being a person. Each person has a gender, and on the previous deliverable, we had "pGender", which had two sub-properties "Female" and "Male" that take boolean values. We changed the way this process is implemented after a comment, by inserting a new class called "Gender", which is a subclass of "owl:Thing" and has four instances, as presented in Figure 1. The male and man are equivalent individuals and different from females and women who are also equivalent to each other.



female
male
man
woman

Figure 1: Instances of class Gender

In Figure 2, we can have a clear image of how our classes are interconnected, after the inferred relationships. On the subclasses of the "Person", the first change due to comment, was to remove the subclass of "Writer" and transform it into "Author", since there was no conceptual difference between the subclasses of the "Writer". Furthermore, we also inserted a subclass called "PeopleAtUni", which is the union of "AcademicStaff", "Student" and "AdministrativeStaff". Furthermore, it was also suggested to include the department an academic staff works or a student belongs to, so as to be able to locate them in the case of a loan delay. So, a new class "Department" was inserted, having two instances "school_of_humanities" and "school_of_science". This way we were also able to add some necessary and sufficient relationships.

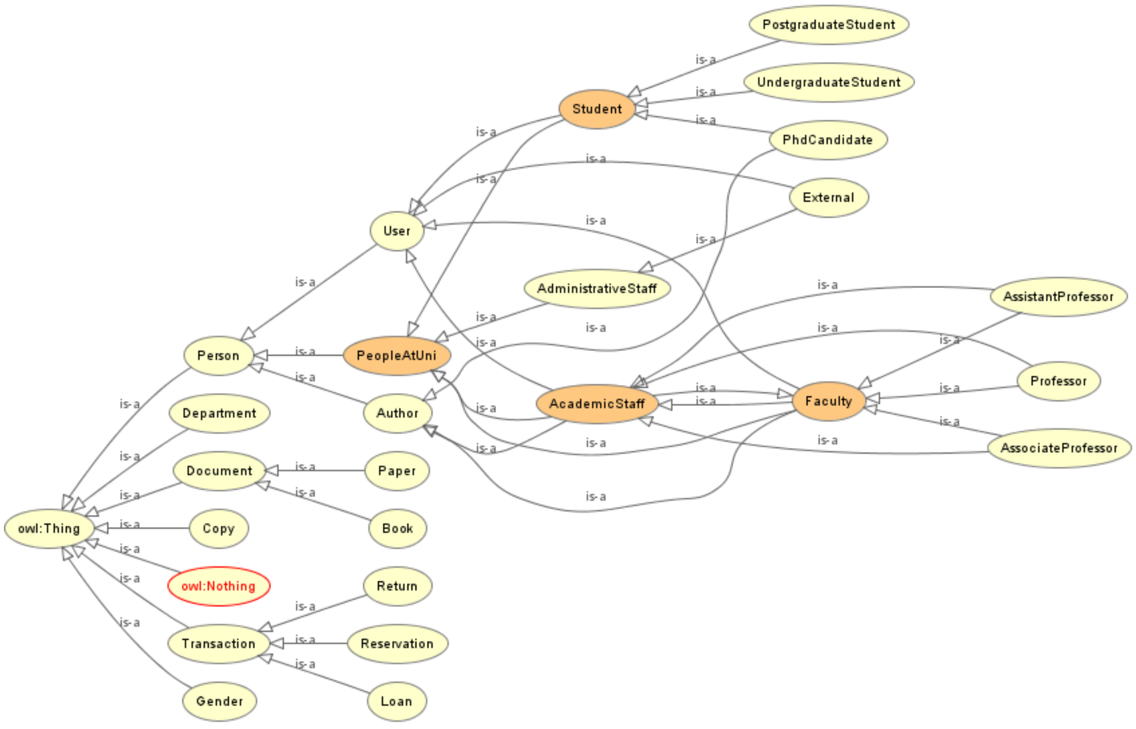
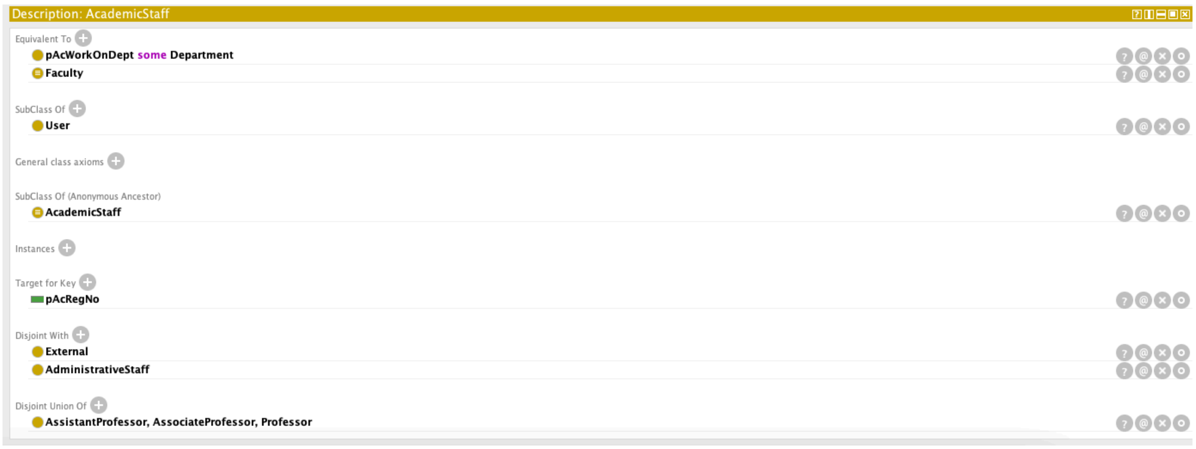


Figure 2: Classes of the Ontology including inferred relationships

In the case of “AcademicStaff”, for someone to be considered as academic staff, he/she should work at least on a department, which is a necessary and sufficient relationship, meaning that if someone works in a “Department” then he/she is also an “AcademicStaff”. Furthermore, as presented in Figure 3, an “AcademicStaff” is the disjoint union of its subclasses, which means that the class “AcademicStaff” is the union of the subclasses and the subclasses are disjoint with each other. In addition, the class is disjoint with “External” and “AdministrativeStaff”, but not with the “Student” class, which means that a “Professor” can also attend an undergraduate program. This is not a frequent case, but it can happen. Furthermore, it also has a unique key, the data property “pAcRegNo”. Lastly, we also created a class called “Faculty”, for consistency reasons with other ontologies, which is the same as our class.



Description: AcademicStaff

Equivalent To
- pAcWorkOnDept some Department
- Faculty

SubClass Of
- User

General class axioms

SubClass Of (Anonymous Ancestor)
- AcademicStaff

Instances

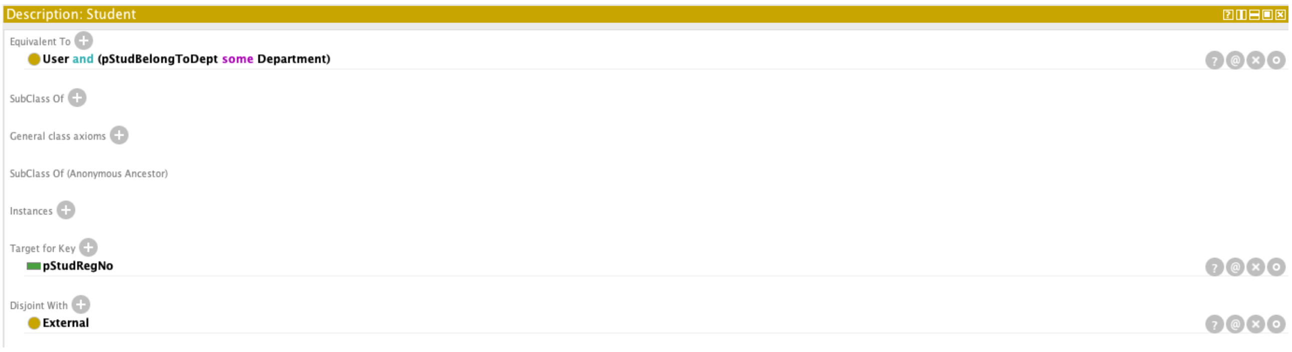
Target for Key
- pAcRegNo

Disjoint With
- External
- AdministrativeStaff

Disjoint Union Of
- AssistantProfessor, AssociateProfessor, Professor

Figure 3: Description of AcademicStaff class

More or less the same applied for the “Student” class, but in this case, we are using the “pStudBeongToDept” property to represent a necessary and sufficient relationship for a student. For representation reasons, instead of setting “Student” as a sub-class of “User”, we inserted this relationship on the equivalence expression. The result is the same as shown in Figure 2. As for the unique key, it is the data property “pStudRegNo” and is only disjoint with the “External” class.


Description: Student

Equivalent To
User and (pStudBelongToDept some Department)

SubClass Of

General class axioms

SubClass Of (Anonymous Ancestor)

Instances

Target for Key
pStudRegNo

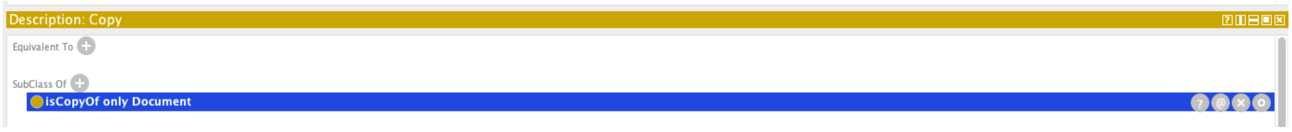
Disjoint With
External

Figure 4: Description of Student class

Concerning all the other subclasses of “Person”, as presented in Figure 2, we have the following structure. The subclasses of “Person” are, the “AdministrativeStaff’, the “User”, the “Author” and the “PeopleAtUni”. The “User” subclass consists of three subclasses (“AcademicStaff”, “Student”, “External”). The “AcademicStaff”, has as subclasses the following: “Professor”, “AssistantProfessor” and “AssociateProfessor”, which are disjoint to each other as already stated. In the same concept, the “Student” subclass consists of three subclasses (“PhdCandidate”, “PostgraduateStudent” and “UndergraduateStudent”), and only the last two are disjoint from each other. Lastly, the third subclass of “User” is the “External”, in which all the other persons belong, including a member of the “AdministrativeStaff”.

As far as the “Author” subclass, all the members of the “AcademicStaff” and the “PhdCandidate” are subclasses of “Author” since there could have written a book or a paper we have in our library.

As already stated above and can be seen in Figure 2, the library has some material that can be loaned by a “User”. This material is described with the second main class “Document” which consequently has two subclasses, “Book” and “Paper”. From a document, we can have multiple copies and this is the reason why a class “Copy” was created. Worth noting is that a copy can only be of a document, so there was created a necessary condition based on which if something “isCopyOf” this means that it is a Document (Figure 5). But this does not work the other way around.


Description: Copy

Equivalent To

SubClass Of

isCopyOf only Document

Figure 5: Description of Copy class

Finally, transactions can be issued between the “Copy” and a “User”. The transactions are included in the class “Transaction” and are more specifically described by the “Reservation”, “Loan” and “Return”.

Object Attributes of the model

The classes created above were formed to perform some operations between them. And its operation is described by the main object properties depicted in Figure 6. A document can be written by an “Author”. This means that the property “isWrittenBy” has as domain the class “Document” and as a range the class “Author”. Also, the inverse object property was created, called “wrote”. A document can also have multiple copies, so the property “hasCopy” has as a range the class “Copy”, the inverse of which is called “isCopyOf”.

From a copy, we can perform multiple transactions. We can perform a “Reservation” or a “Loan”. So the properties “transCopyReserved” and “transCopyLoaned” have both as a range the “Copy” class. In the same concept, a “Reservation” or a “Loan” can be made by a “User”, so we have created two properties to depict it “transIsReservedBy” and “transIsLoanedBy” having as range the “Copy” class, having as inverse the object properties “transReserves” and “transLoans” respectively. Furthermore, a “Loan” can be issued after a “Reserve”, so the “transFromReserve” has as a domain the “Loan” and range “Reservation”. Same case for the “transFromLoan”, in which a “Return” is made based on a “Loan”. Last but not least is that every “Transaction” is registered by a staff. So, the property “transRegisteredFrom” has as a domain a “Transaction” and as a range the “AdministrativeStaff”, with an inverse object property the “transRegisters”.

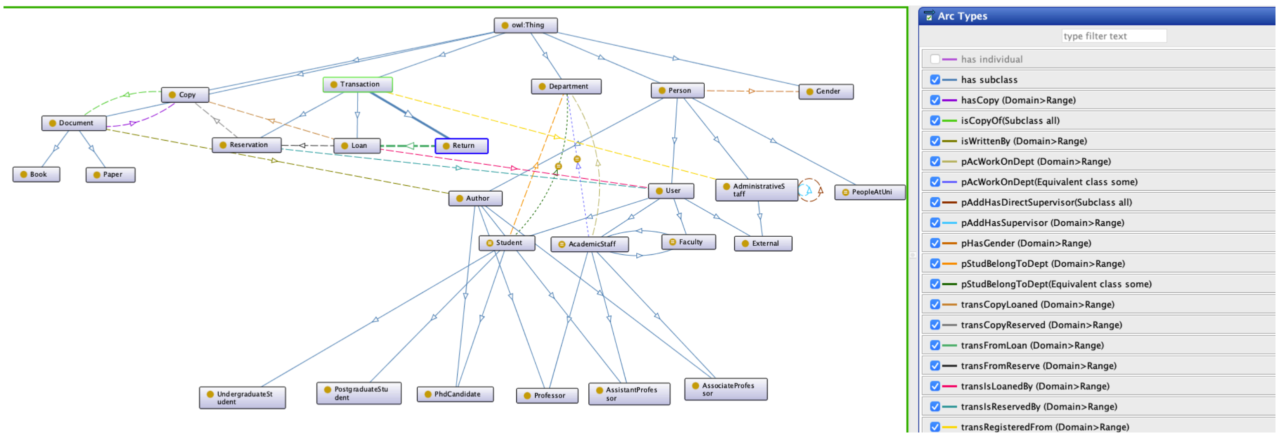
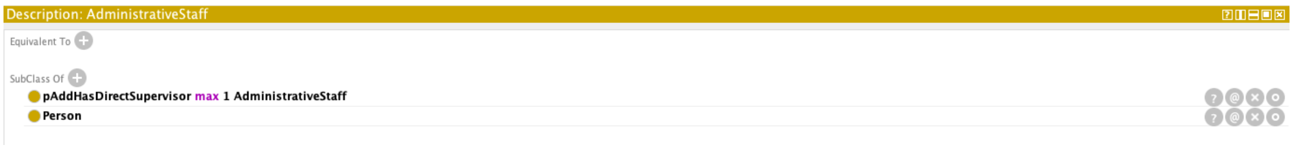

owl:Thing
Copy
Transaction
Department
Person
Gender
Document
Reservation
Loan
Return
Book
Paper
Author
User
AdministrativeStaff
PeopleAtUni
Student
AcademicStaff
Faculty
External
UndergraduateStudent
PostgraduateStudent
PhdCandidate
Professor
AssistantProfessor
AssociateProfessor
Arc Types
type filter text
has individual
has subclass
hasCopy (Domain>Range)
isCopyOf(Subclass all)
isWrittenBy (Domain>Range)
pAcWorkOnDept (Domain>Range)
pAcWorkOnDept(Equivalent class some)
pAddHasDirectSupervisor(Subclass all)
pAddHasSupervisor (Domain>Range)
pHasGender (Domain>Range)
pStudBelongToDept (Domain>Range)
pStudBelongToDept(Equivalent class some)
transCopyLoaned (Domain>Range)
transCopyReserved (Domain>Range)
transFromLoan (Domain>Range)
transFromReserve (Domain>Range)
transIsLoanedBy (Domain>Range)
transIsReservedBy (Domain>Range)
transRegisteredFrom (Domain>Range)

Figure 6: Object Properties of the Ontology

Furthermore, we also included the property “hasAddSupervisor”, which depicts the relation between the “AdministrativeStaff” members. This relationship was highlighted as transitive, since the supervisor of ones supervisor is also his/her supervisor. But, we also included the object property of “pAddHasDirectSupervisor”, which is not transitive, but as presented on the figure below (Figure 7) and based on the restriction inserted on the class it can have max one direct supervisor from the “AdministrativeStaff” class.


Description: AdministrativeStaff

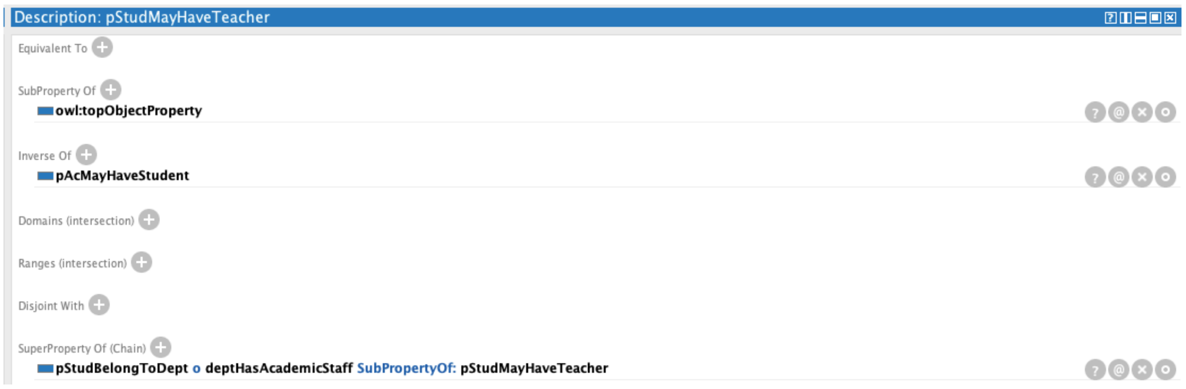
Equivalent To

SubClass Of
- pAddHasDirectSupervisor max 1 AdministrativeStaff
- Person

Figure 7: Object Property for the teachers a student may have

As already stated above, we inserted the object property “pHasGender”, which is also functional, since only one gender can be selected for a user. We also added the functional property of “pAcWorkOnDept” with domain “AcademicStaff” and range the “Department”, having as inverse property the “deptHasAccademicStaff”. This was able to be implemented since the relationship between the “AcademicStaff” and “Department” is necessary and sufficient. The same also applies to students via the “pStudBelongToDept” property, with only the difference being that it is not a functional property, since a student can belong to many departments. The inverse of this object property was named “deptHasStudents”.

Lastly, as illustrated in Figure 8, for representation reasons we created an object property that was about to find the teachers a student may have. This object property was created as a chain between the property “pStudBelongToDept” and “deptHasAcademicStaff”. To make it more clear, if a student belongs to a department and a faculty member works in a department, then this member can be the teacher of this student. The inverse of this relationship was named "pAcMayHaveStudent".


Description: pStudMayHaveTeacher

Equivalent To

SubProperty Of
owl:topObjectProperty

Inverse Of
pAcMayHaveStudent

Domains (intersection)

Ranges (intersection)

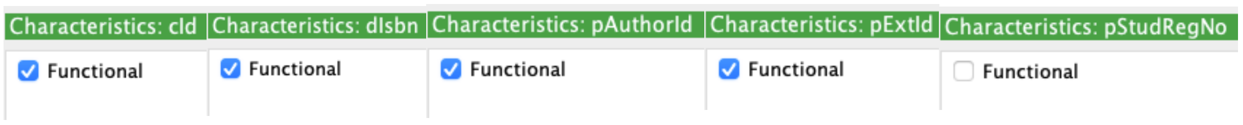
Disjoint With

SuperProperty Of (Chain)
pStudBelongToDept o deptHasAcademicStaff SubPropertyOf: pStudMayHaveTeacher

Figure 8: Object Property for the teachers a student may have

Data Properties of the model

Except for the object properties analysed above additional data properties were created, which had for range one of the following: "date", "integer", "decimal" "string", and "boolean". Almost all the data properties that where used as target keys in classes were also characterised as functional (Figure 9). The only one that was not characterised as functional was the "pStudRegNo", which means that a student can have multiple registration numbers. And more specifically this only applies to a PhD student who can also be an undergraduate or postgraduate. The undergraduate and postgraduate students are disjoint with each other, so this will lead to an inconsistency.



Characteristics: cId	Characteristics: dIsbn	Characteristics: pAuthorId	Characteristics: pExtId	Characteristics: pStudRegNo
✓ Functional	✓ Functional	✓ Functional	✓ Functional	☐ Functional

Figure 9: Object Property for the teachers a student may have

In general, we tried to capture most of the functionalities that will be requested for our model. For instance, the copy could have an ID "cId", a price "cPrice", a date that it purchased "cPurDate". A paper except for the general properties that a document has ie. title and isbn, also a unique property was created the "dPaperJournal" that presents the journal that a paper was published. The same process was followed for the class "Person", which has some common properties for all the subclasses, but also some unique properties for each subclass. Lastly, due to the comment provided we included the data property "pExtHasAddress", so we can locate an external user that has an overdue loan, but also the properties "transLoanIsOverdue" and "TransResIsOverdue", which are boolean properties that are becoming true when a loan or a reservation is overdue.

Library Model Instances and Inferencing

A great example to present is the ontology graph of user "U0002" presented in Figure 10. User "U0002" has the name Anna Karamatsouka and is a female. She is both a PhD Candidate and an undergraduate student in two different departments of the university. So, she may have as teachers all the four instances we have for academic staff. Also, we have an extra instance of a student at each department which is also presented on the graph. Furthermore, Anna has made a reservation "R0001" of copy "C001", and the transaction registered from staff "S001". That reservation was transformed to loan "Loan001" and in addition, the same book and staff were kept in the process of the loan. After a few days the process of return was initiated, so we had the "Return001", which came from "Loan001", but now the staff that registered the transaction was "S002". Furthermore, in our graph, it is shown this is a copy of a book, and we have an extra copy of that book. It is also presented that this book was written by author "A0002", who is a "male" and has also written a paper.

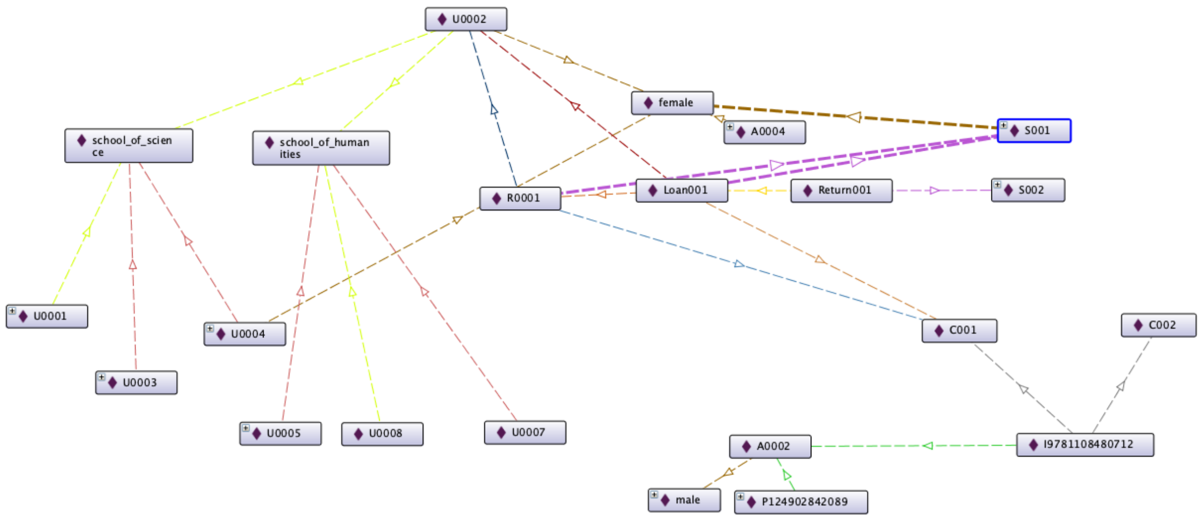
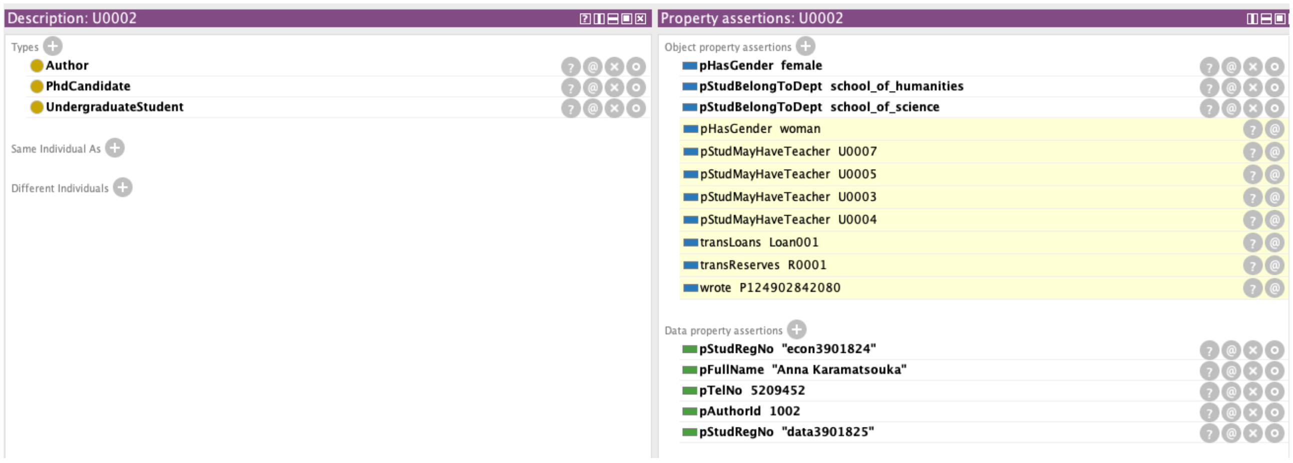


Figure 10: User U0002 Ontology Graph

What is not depicted in Figure 10, but we can see in Figure 11 is that Anna wrote a paper as well. But, this happens for two reasons. Firstly, we have only defined that this paper was written by Anna, so since wrote is an inverse property it was inferred that she wrote it. Secondly, it was not declared what type was the instance of that Paper. But, since there is a necessary condition, as presented on Figure 5 and we have declared that “C008” “isCopyOf” that paper, it was inferred that it belongs to the class “Document”, as presented in Figure 12.


Description: U0002

Types
- Author
- PhdCandidate
- UndergraduateStudent

Same Individual As

Different Individuals

Property assertions: U0002

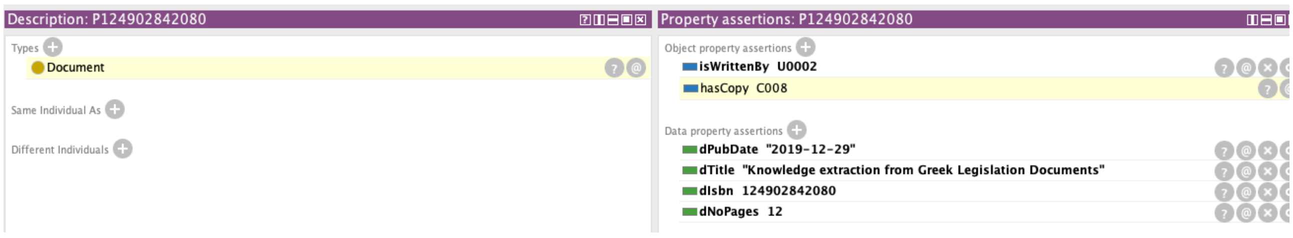
Object property assertions
- pHasGender female
- pStudBelongToDept school_of_humanities
- pStudBelongToDept school_of_science
- pHasGender woman
- pStudMayHaveTeacher U0007
- pStudMayHaveTeacher U0005
- pStudMayHaveTeacher U0003
- pStudMayHaveTeacher U0004
- transLoans Loan001
- transReserves R0001
- wrote P124902842080

Data property assertions
- pStudRegNo "econ3901824"
- pFullName "Anna Karamatsouka"
- pTelNo 5209452
- pAuthorId 1002
- pStudRegNo "data3901825"

Figure 11: Inferences for U0002

Worth mentioning is that in the case of the paper we had “dPaperJournal”, which is in which journal this paper was published, the inference ruler would set as a type “Paper” since the domain of the data property is the class “Paper”.

Another interesting inference presented in Figure 11 and also in the graph of Figure 10, is the one that comes up from the chain between properties presented in Figure 8, based on which if a student belongs to a department he/she can have as a teacher all the academic staff that works on it. This is the reason why it was inferred Anna may have four teachers.


Description: P124902842080

Types
- Document

Same Individual As

Different Individuals

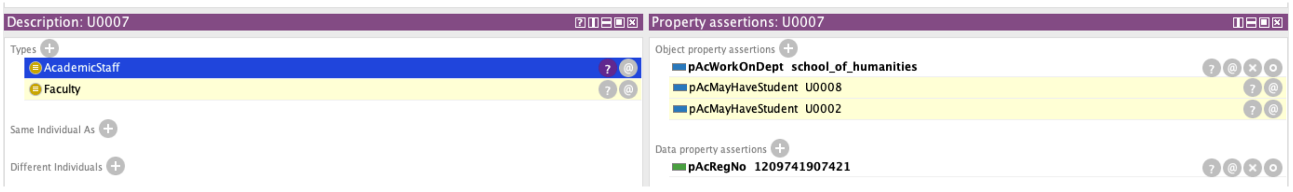
Property assertions: P124902842080

Object property assertions
- isWrittenBy U0002
- hasCopy C008

Data property assertions
- dPubDate "2019-12-29"
- dTitle "Knowledge extraction from Greek Legislation Documents"
- dIsbn 124902842080
- dNoPages 12

Figure 12: Inferences for paper “P124902842080”

In Figure 13, we present inferences generated for the user “U0007”, who when we registered in the system we only filled in that he/she works in the department of “school_of_humanities” and that has an academic registration number. Both of these details drove the reasoner to end up in the conclusion that this user is an “AcademicStaff” and also a “Faculty” since these two classes are inverse of each other. The reason why these inferences come up is that the data property “pAcRegNo” has as a domain the “AcademicStaff” and for the object property “pAcWorkOnDept” we have defined, as presented in Figure 3, a necessary and sufficient relationship. Lastly, since this user is an “AcademicStaff”, then it may have as students the ones that belong to that department, as explained above.



Description: U0007

Types
- AcademicStaff
- Faculty

Same Individual As

Different Individuals

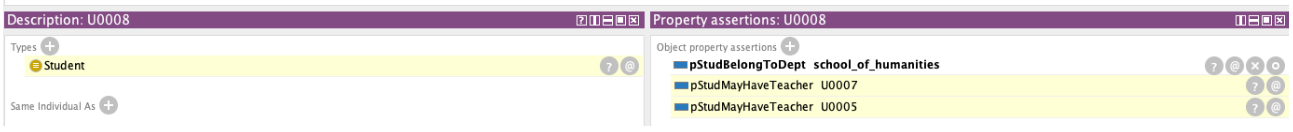
Property assertions: U0007

Object property assertions
- **pAcWorkOnDept school_of_humanities**
- pAcMayHaveStudent U0008
- pAcMayHaveStudent U0002

Data property assertions
- **pAcRegNo 1209741907421**

Figure 13: Inferences for User “U0007”

The same logic applies for the User “U0008”, but this time the user is a “Student” since we have defined that this user belongs to the department of “school_of_humanities. And since it belongs to this department he/she may have as a teacher the “U0007” we have introduced previously.



Description: U0008

Types
- Student

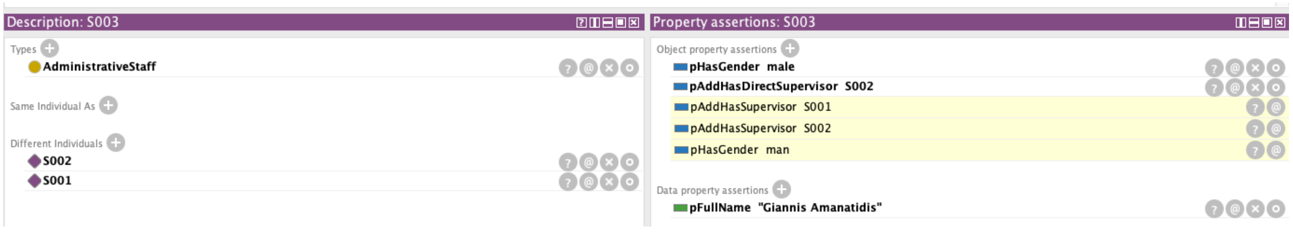
Same Individual As

Property assertions: U0008

Object property assertions
- **pStudBelongToDept school_of_humanities**
- pStudMayHaveTeacher U0007
- pStudMayHaveTeacher U0005

Figure 14: Inferences for User “U0008”

Another interesting conclusion came up after running the inference ruler on the administrative staff “S003” (Figure 15). This staff member has as “pAdHasDirectSupervisor” the member “S002”. But, since it is a sub-property of “pAdHasSupervisor” it is inferred that “S002” also belongs to that super property concerning staff member “S003”. Furthermore, “S002” has as direct supervisor “S001”, which consequently means that “pAdHasSupervisor” is the staff member “S001”. And since the “pAdHasSupervisor” is a transitive object property it was inferred that the staff member “S003” has also as supervisor “S001”. But “S002” is his/her direct supervisor. Lastly, the instance “male” is equivalent to “man”, so “S003” is also a “man”.


Description: S003

Types
AdministrativeStaff

Same Individual As

Different Individuals
S002
S001

Property assertions: S003

Object property assertions
pHasGender male
pAddHasDirectSupervisor S002
pAddHasSupervisor S001
pAddHasSupervisor S002
pHasGender man

Data property assertions
pFullName "Giannis Amanatidis"

Figure 15: Inferences for Administrative Staff "S003"

Inconsistency Examples

While constructing our ontology model our target is to conclude in valuable outcomes, and for this reason, we set many restrictions. If we are not careful when we register the documents we might end up with some inconsistencies. Below we will present some examples of inconsistencies, which we have removed from our final deliverable. Of course, there are much more that could be presented, but could not all be included in our project deliverable.

In Figure 16, we represent an inconsistency that came up after setting for the "U0001" both the type as "PostgraduateStudent" and as "UndergraduateStudent". But, these two classes are disjoint with each other so a student can not be in both classes.

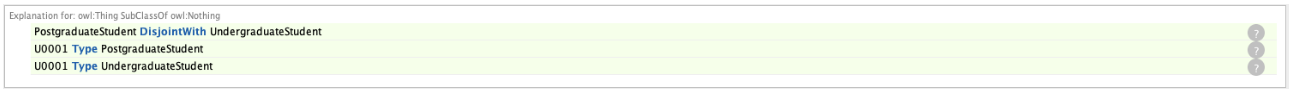

```
Explanation for: owl:Thing SubClassOf owl:Nothing
PostgraduateStudent DisjointWith UndergraduateStudent
U0001 Type PostgraduateStudent
U0001 Type UndergraduateStudent
```

Figure 16: Disjoint classes inconsistency

Inconsistency presented in Figure 17, might seem not completely logical but could be made due to a mistake in the registration process. In this example, we say that copy "C007" is a copy of the instance "male". Since we have set that a "Copy" can be a copy of only the class "Document" this means that it can not take another class instance and an inconsistency arouses.

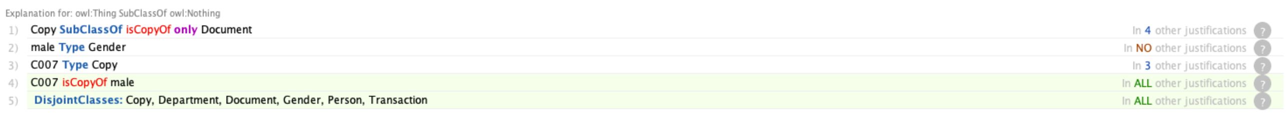
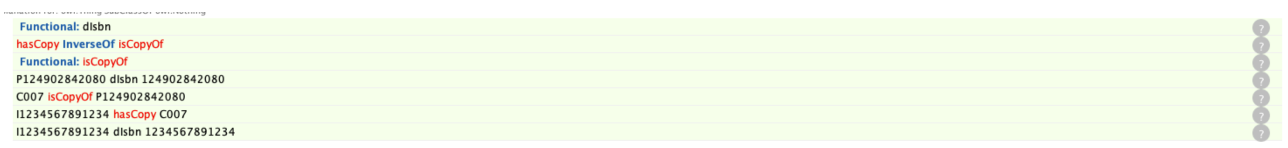

```
Explanation for: owl:Thing SubClassOf owl:Nothing
1) Copy SubClassOf isCopyOf only Document                          In 4 other justifications
2) male Type Gender                                                In NO other justifications
3) C007 Type Copy                                                  In 3 other justifications
4) C007 isCopyOf male                                              In ALL other justifications
5) DisjointClasses: Copy, Department, Document, Gender, Person, Transaction   In ALL other justifications
```

Figure 17: Necessary condition inconsistency

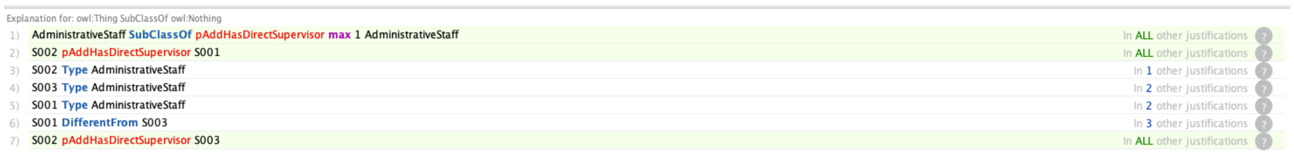
In the same copy during the registration we made another mistake and we set that is a copy of two documents. But, since the object property is functional this means that it can not happen. So the reasoner returned an inconsistency.



```
Functional: dIsbn
hasCopy InverseOf isCopyOf
Functional: isCopyOf
P124902842080 dIsbn 124902842080
C007 isCopyOf P124902842080
I1234567891234 hasCopy C007
I1234567891234 dIsbn 1234567891234
```

Figure 18: Functional object property inconsistency

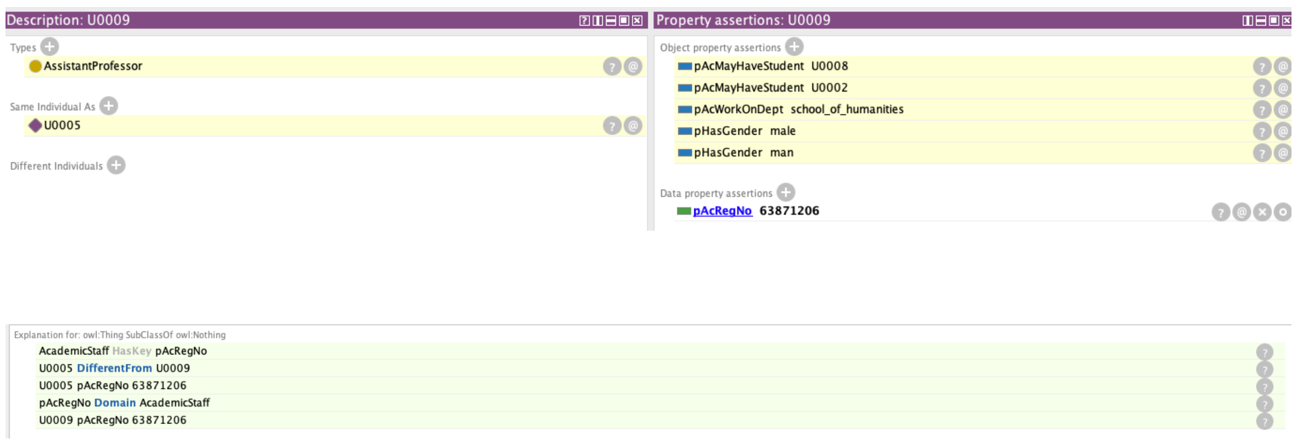
In Figure 19, a cardinality restriction inconsistency is presented. An administrative staff can have at most one direct supervisor. In this case we set up two direct supervisors for the staff member "S003", so the reasoner concluded that there is an inconsistency.



```
Explanation for: owl:Thing SubClassOf owl:Nothing
1) AdministrativeStaff SubClassOf pAddHasDirectSupervisor max 1 AdministrativeStaff   In ALL other justifications
2) S002 pAddHasDirectSupervisor S001                                                   In ALL other justifications
3) S002 Type AdministrativeStaff                                                       In 1 other justifications
4) S003 Type AdministrativeStaff                                                       In 2 other justifications
5) S001 Type AdministrativeStaff                                                       In 2 other justifications
6) S001 DifferentFrom S003                                                             In 3 other justifications
7) S002 pAddHasDirectSupervisor S003                                                   In ALL other justifications
```

Figure 19: Cardinality restriction inconsistency

Finally, we introduced a new user "U0009" and the only details added was the "pAcRegNo". Since it had the same with "U0005" the ruler ended up that these users are the same person as presented in Figure 20. But then, we set that these two individuals are different, so an inconsistency arouse.



```
Description: U0009
Types
    AssistantProfessor
Same Individual As
    U0005
Different Individuals

Property assertions: U0009
Object property assertions
    pAcMayHaveStudent  U0008
    pAcMayHaveStudent  U0002
    pAcWorkOnDept  school_of_humanities
    pHasGender  male
    pHasGender  man
Data property assertions
    pAcRegNo  63871206
```

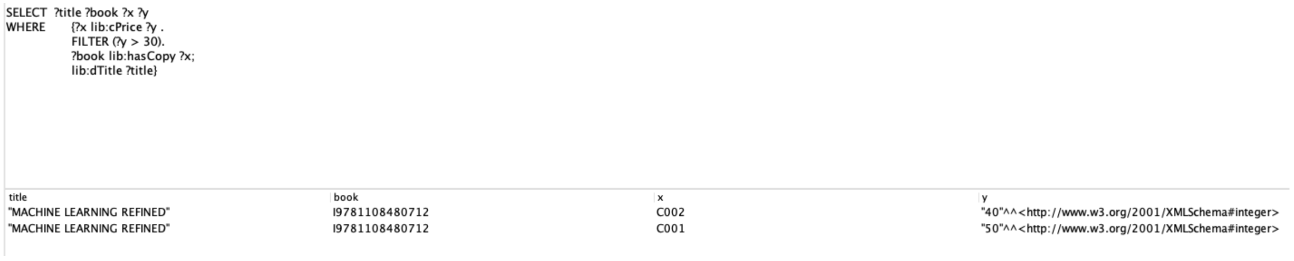
```
Explanation for: owl:Thing SubClassOf owl:Nothing
AcademicStaff HasKey pAcRegNo
U0005 DifferentFrom U0009
U0005 pAcRegNo 63871206
pAcRegNo Domain AcademicStaff
U0009 pAcRegNo 63871206
```

Figure 20: Different individuals inconsistency

SPARQL Queries

SPARQL is the standard query language and protocol for Linked Open Data and RDF databases. In this section we will present three SPARQL query examples and their results.

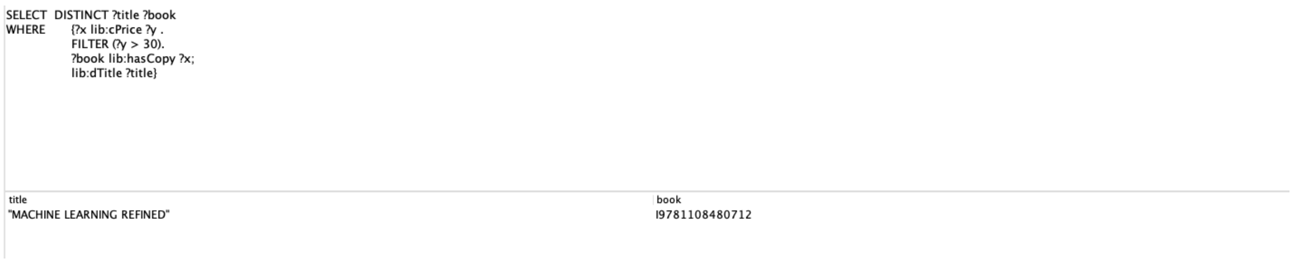
In the first query, we wanted to find the title of the books that their copies cost more than 30€. So, we had first to find copies that have a price over 30€ and after that find the books that correspond to that copies. The result revealed that we have two copies and these copies are in the same book, as presented in Figure 21. But, we can limit the results find the unique books by using the DISTINCT keyword if we want to proceed so (Figure 22).



```
SELECT  ?title ?book ?x ?y
WHERE     {?x lib:cPrice ?y .
          FILTER (?y > 30).
          ?book lib:hasCopy ?x;
          lib:dTitle ?title}
```

```
title                         book              x       y
"MACHINE LEARNING REFINED"    I9781108480712    C002    "40"^^<http://www.w3.org/2001/XMLSchema#integer>
"MACHINE LEARNING REFINED"    I9781108480712    C001    "50"^^<http://www.w3.org/2001/XMLSchema#integer>
```

Figure 21: Title of books that its copies costed more than 30€



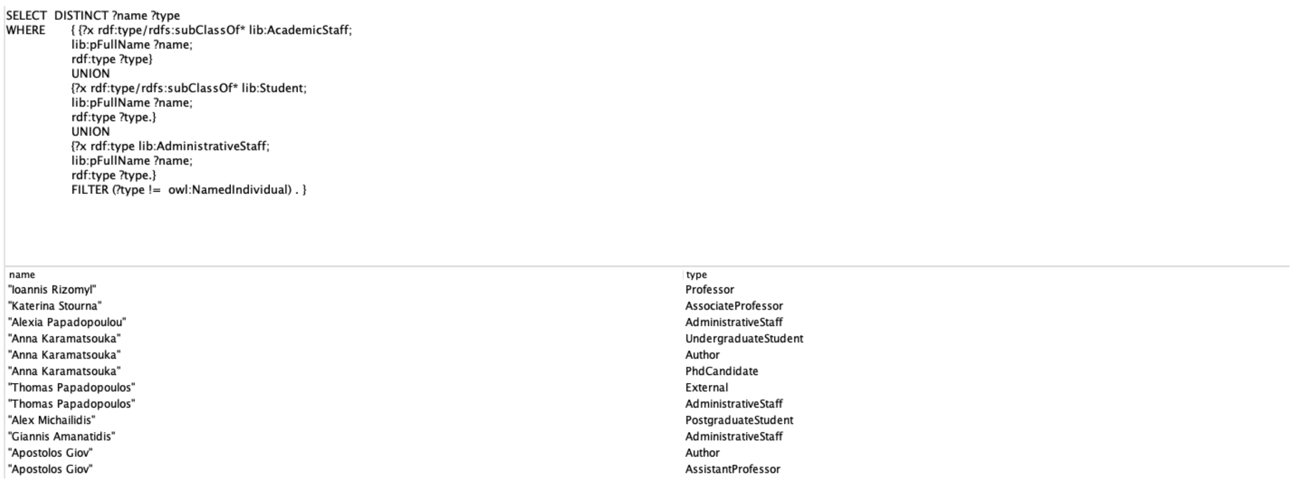
```
SELECT  DISTINCT ?title ?book
WHERE     {?x lib:cPrice ?y .
          FILTER (?y > 30).
          ?book lib:hasCopy ?x;
          lib:dTitle ?title}
```

```
title                                          book
"MACHINE LEARNING REFINED"                     I9781108480712
```

Figure 22: Title of unique books that its copies costed more than 30€

In this query, we would like to extract all the people that are members of our university and in which classes they belong to. So, the people at our university are the union of the classes "AcademicStaff", "Student" and "AdministrativeStaff". For the first two classes, we had to extract all the members that belong to its subclasses, but of the "AdministrativeStaff" this could not be the same case, since the class "External" is a subclass. So, we only took the members that are in this class. Finally, we filter our results since the results returned belong also to the "owl:NamedIndividual" class, which we wanted to exclude. As one can see in Figure 23, some people belong to more than one class, but all these are people that belong to our university.

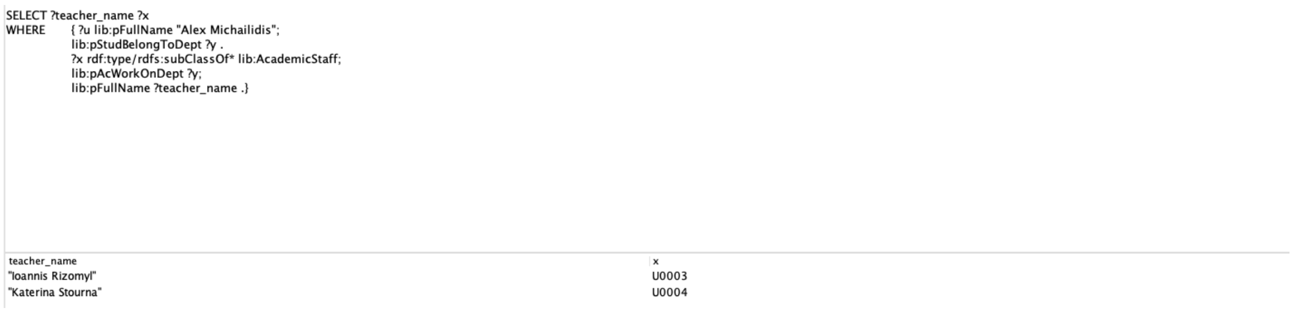
Figure 23: People of the University and the classes they belong to



```
SELECT  DISTINCT ?name ?type
WHERE      { {?x rdf:type/rdfs:subClassOf* lib:AcademicStaff;
           lib:pFullName ?name;
           rdf:type ?type}
           UNION
           {?x rdf:type/rdfs:subClassOf* lib:Student;
           lib:pFullName ?name;
           rdf:type ?type.}
           UNION
           {?x rdf:type lib:AdministrativeStaff;
           lib:pFullName ?name;
           rdf:type ?type.}
           FILTER (?type !=  owl:NamedIndividual) . }
```

name	type
"Ioannis Rizomyl"	Professor
"Katerina Stourna"	AssociateProfessor
"Alexia Papadopoulou"	AdministrativeStaff
"Anna Karamatsouka"	UndergraduateStudent
"Anna Karamatsouka"	Author
"Anna Karamatsouka"	PhdCandidate
"Thomas Papadopoulos"	External
"Thomas Papadopoulos"	AdministrativeStaff
"Alex Michailidis"	PostgraduateStudent
"Giannis Amanatidis"	AdministrativeStaff
"Apostolos Giov"	Author
"Apostolos Giov"	AssistantProfessor

In our last query which is demonstrated in Figure 24, we try to find the teachers that the user Alex Michailidis may have. To achieve so, we first found the department to which Alex belongs and after that find the academic staff that works on that department and their names. After running the SPARQL query our system returned two academic staff that may be also teachers of Alex.



```
SELECT ?teacher_name ?x
WHERE      { ?u lib:pFullName "Alex Michailidis";
           lib:pStudBelongToDept ?y .
           ?x rdf:type/rdfs:subClassOf* lib:AcademicStaff;
           lib:pAcWorkOnDept ?y;
           lib:pFullName ?teacher_name .}
```

teacher_name	x
"Ioannis Rizomyl"	U0003
"Katerina Stourna"	U0004

Figure 24: The teachers that Alex Michailidis might have

SWRL Rules

SWRL is a Semantic Web Language that allows Horn-like rules to be combined with OWL 2 DL ontologies, allowing this way the user to expand the rules and inferences that can be extracted from the ontology. In the examples above we will represent three plus one bonus rules created using the SWRL language.

In the first example, we have created a rule that aims the reclassification of some instances. More specifically, since we have the information that some of our instances are either a male or a female, we have created two classes, which are subclasses of the Person class. These classes were named "Man" and "Woman", and they were generated by the following rules:

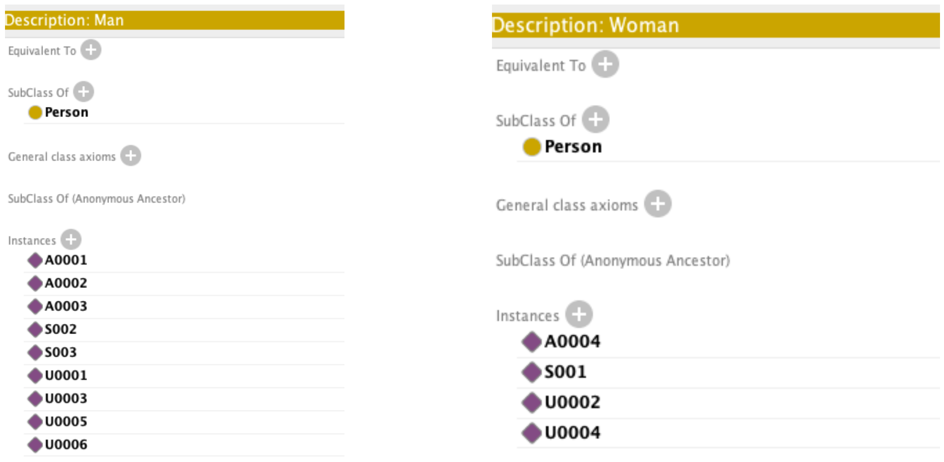
SWRL Rule 1

lib:Person(?m) ^ lib:pHasGender(?m, lib:male) -> lib:Man(?m)

SWRL Rule 2

lib:Person(?m) ^ lib:pHasGender(?m, lib:female) -> lib:Woman(?m)

After running the rules the result was to insert all the instances that was marked as male to the class "Man", while the ones marked as female to the class "Woman", as seen in Figure 25.



Description: Man

Equivalent To

SubClass Of
- Person

General class axioms

SubClass Of (Anonymous Ancestor)

Instances
- A0001
- A0002
- A0003
- S002
- S003
- U0001
- U0003
- U0005
- U0006

Description: Woman

Equivalent To

SubClass Of
- Person

General class axioms

SubClass Of (Anonymous Ancestor)

Instances
- A0004
- S001
- U0002
- U0004

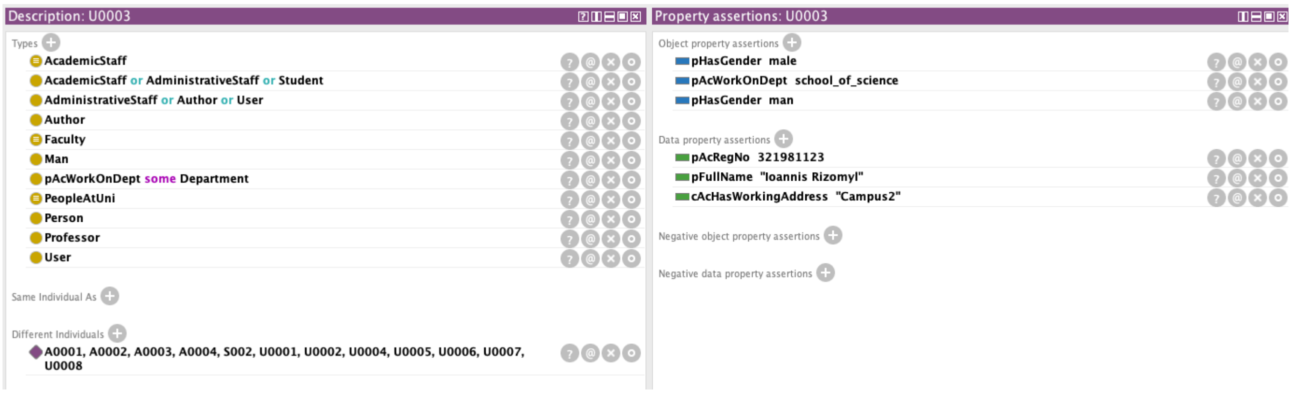
Figure 25: Results of Reclassification Rules

In the next example we assigned some values to a new property. More specifically, the rule created had as a target to find the working address of the "Academic Staff", which is the same as the the address of the "Department" they work on. In order to be able to extract this information, we had first to create an object property, which is called "dAddress", which stores the address of the department. We also created a data property named as “cAcHasWorkingAddress”, which will be used in our rule to extract the working address of an academic staff. The rule is presented below.

SWRL Rule 3

lib:AcademicStaff(?x) ^ lib:pAcWorkOnDept(?x, ?d) ^ lib:dAddress(?d, ?a) -> lib:cAcHasWorkingAddress(?x, ?a)

As presented in Figure 26, after running the ruler, Ioannis Rizomyl, who belongs to the class “AcademicStaff” and works in the department “school_of _science” takes on the data property “cAcHasWorkingAddress” the value “Campus2”, which is the address of the department.



```
Description: U0003
Types
  AcademicStaff
  AcademicStaff or AdministrativeStaff or Student
  AdministrativeStaff or Author or User
  Author
  Faculty
  Man
  pAcWorkOnDept some Department
  PeopleAtUni
  Person
  Professor
  User
Same Individual As
Different Individuals
  A0001, A0002, A0003, A0004, S002, U0001, U0002, U0004, U0005, U0006, U0007,
  U0008

Property assertions: U0003
Object property assertions
  pHasGender  male
  pAcWorkOnDept  school_of_science
  pHasGender  man
Data property assertions
  pAcRegNo  321981123
  pFullName  "Ioannis Rizomyl"
  cAcHasWorkingAddress  "Campus2"
Negative object property assertions
Negative data property assertions
```

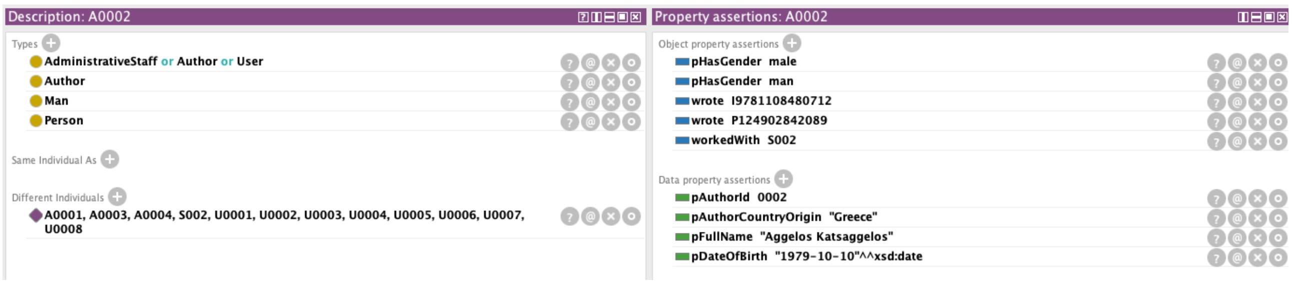
Figure 26: Results of Data Property Value Assignment based on Rule

In the same direction, we inserted a rule in our ontology that has as a target to find the authors that have worked together. These authors are those who wrote the same document. We had to insert a new object property in our ontology to capture this relationship, called “workedWith”. More specifically the rule is presented below.

SWRL Rule 4

lib:Document(?d) ^ lib:isWrittenBy(?d, ?p1) ^ lib:isWrittenBy(?d, ?p2) ^ differentFrom(?p1, ?p2) -> lib:workedWith(?p1, ?p2)

In an indicative example after the implementation of the rule, in the table of author “A0002” the object property attribute “workedWith” has been included and took the value “S002”. This was inferred because they have both written the document “P124902842089”.


Description: A0002

Types
- AdministrativeStaff or Author or User
- Author
- Man
- Person

Same Individual As

Different Individuals
- A0001, A0003, A0004, S002, U0001, U0002, U0003, U0004, U0005, U0006, U0007, U0008

Property assertions: A0002

Object property assertions
- pHasGender male
- pHasGender man
- wrote I9781108480712
- wrote P124902842089
- workedWith S002

Data property assertions
- pAuthorId 0002
- pAuthorCountryOrigin "Greece"
- pFullName "Aggelos Katsaggelos"
- pDateOfBirth "1979-10-10"^^xsd:date

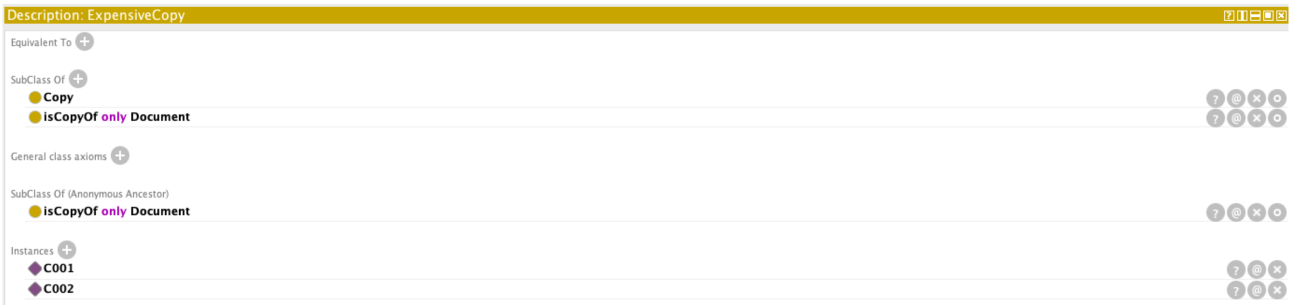
Figure 27: Results of Object Property Value Assignment based on Rule

Finally, we created an extra rule that was based on the built-in functions of the SWRL and had a reclassification impact after running it. More specifically, we assign the copies that have a greater than 30 to the class “ExpensiveCopy”, which is a subclass of “Copy”.

SWRL Rule 5

lib:Copy(?c) ^ swrlb:greaterThan(?price, 30) ^ lib:cPrice(?c, ?price) -> lib:ExpensiveCopy(?c)

By running this rule two of our copies were inserted in this class, as presented in Future 28.


Description: ExpensiveCopy

Equivalent To

SubClass Of
- Copy
- isCopyOf only Document

General class axioms

SubClass Of (Anonymous Ancestor)
- isCopyOf only Document

Instances
- C001
- C002

Figure 28: Results of Object Property Value Assignment based on Rule

Conclusion

The model of a library is quite complicated including a lot of variables that are interconnected to each other so that a model to be built exhaustively. In our case, we created an ontology that describes some transaction processes to a satisfactory degree, but not completely.