Part I
Modeling of High School’s Information System

Introduction
For the elaboration of the work was used WebProtege, a web application of collaborative writing and development of OWL ontologies.

OWL is an ontologies’ language that is based on the well-known RDF and RDFS, and its initials mean Web Ontology Language, and was created, just like RDF, in order to be interpreted by computers [4]. The difference with RDF is that it is a much richer language with much greater vocabulary and much better capability of interpretation by computers.

There are three versions of OWL: OWL Lite, OWL DL (contains Lite) and OWL Full (contains DL) [1]. The composition OWL is based on RDF and RDFS languages which are based on the “triple” of RDF/XML composition [2]. The key features of an OWL ontology’s composition are:

- The header
- The class
- The instances of the classes
- The properties

Header
The header is essentially the root of the ontology and is defined as a rdf:RDF element that specifies a number of namespaces. The namespaces used in the rdf:RDF tags exist to identify the vocabulary of other tags in order to be used later in the ontology. To understand this, we will use an example of rdf:RDF header

```xml
<rdf:RDF
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
  <owl:Class rdf:ID="Object">
    <rdfs:subClassOf rdf:resource="#Object"/>
  </owl:Class>
</rdf:RDF>
```

```
In our example, the xmlns = “http://example.org/
Example#” and xmlns:exd = “http://example.org/Example#” are our default namespaces.
```

Class
An OWL class is expressed in RDF/XML through an owl:Class element. The owl:Class tag contains the statement of an rdf:ID element that locally identifies the class name in this ontology file.

```xml
<owl:Class rdf:ID="Object">
  <owl:oneOf rdf:resource="#Spring"/>
  <owl:oneOf rdf:resource="#Summer"/>
  <owl:oneOf rdf:resource="#Fall"/>
  <owl:oneOf rdf:resource="#Winter"/>
</owl:Class>
```

```
An owl:oneOf element combined with rdfs:subClassOf can be used to define a class by derogating instances that belong to this class.
```

In the example shown below the owl:Class “Season” may be just one of the ‘Spring’, ‘Summer’, ‘Fall’, and ‘Winter’ instances.
Let us note that classes can also be anonymous. In the example above, the 'Season' owl:Class element is anonymous. Also the owl:Restriction element also creates anonymous classes.

Also the owl:equivalentClass declares that this class is the same as another. Example: ('PoliticalDivision' is the same as 'AdministrativeBoundary')

Instances
Class instances are defined by specifying the class of which they are instances. For example, the following statement sets an instance with ID 'George' of the Person class.

Properties
Of course we could not create a meaningful ontology by simply associating classes. So using the ontology properties we display elements and specific facts about class members. It is a binary relationship and is expressed by two types of properties:

• datatype properties
• object properties

Owl Datatype Properties
An owl:DatatypeProperty element expresses the relationship between an instance and a given value. As the example shows, the 'hasAge' property is declared with a value for the 'Person' instance.

Owl Object Properties
An owl:ObjectProperty element expresses the relationship between two instances. As shown in the example, the definition of ObjectProperty as 'hasWife' is used to declare a 'Male' value.

WebProtege
The process of writing ontologies in OWL language is greatly simplified by WebProtege [3]. Having created an interface through an open source web application enables you through collaborative writing to create the classes and properties with just a few clicks.

In this way you can collaboratively write an ontology with your team, leaving comments or notes to other members of the group as well as leaving your project free for use by others.

WebProtege offers you a multitude of options when it comes to creating ontologies. Some of these support OWL 2, a default working environment that provides access to frequently used OWL structures, full change and history tracking for all team members, customizable interface and support of multiple formats for downloading ontology or uploading another in the web environment.

Ontologies Introduction
In the present work we attempted to implement an ontology for modelling a high school/secondary school information system. We used webProtégé as an ontology development tool. Then we introduced the ontology in Protégé 4.3.0 to get a better graphic representation.

Ontology Description
For the modelling of the high school/secondary school information system we considered that:

• The classrooms belong to the school.
• A class corresponds to each classroom.
• Students belong to a class.
• Teachers have a specialty.
• Teachers teach lessons.
• The lessons correspond to a class.
• Students attend-belong to classes.

Class Hierarchy
At the top of our hierarchy are nine classes: Course, Punishment, Person, TeacherFaculty, SchoolTrip, ClassRoom, School, ClassGrade and Accolade which were implemented by Protégé as Thing class subclasses.

Next, we have developed the Parent, Staff and Student classes as subclasses of the Person class. A person can be either a student or parent, or belong to the school staff. From the Staff class we created two new classes that inherit its attributes, the OtherStaff and Teacher classes. Of which school staff can be either an educator or have another capacity.
The Classroom class inherits the features of the other two classes, LabClassRoom and LectureClassRoom. In which a room can be either a lab or a lecture room.

**Object Properties**

Object properties in Protégé refer to the characteristics of the classes that receive instances of other classes as values, through which the classes are linked. In the ClassRoom class we have two object properties, classInSchool and classroomHasClass (Figure 3).

The classSetInClassroom property accepts an object of the Classroom class that indicates which classroom the class is housed in. The going School Trip property is associated with the School Trip class and indicates that a class participated in an excursion. Finally, with the has Courses property, Course class objects are declared that show the lessons a class has.

The Parent class refers to the guardians of the students. It has a guardian Of property (Figure 6) that is associated with the Student class and indicates of which students they are guardians.

The Teacher class has two properties: isSpecialize and teaches (Figure 8). The isSpecialize property points to objects of TeacherFaculty class from which the teacher receives his specialty. The teaches property points to courses of Course class that are the lessons he teaches.
Finally we have the Course class with taughtBy and taughtInGrade properties (Figure 9). The taughtBy property accepts Teacher class objects that show which teacher teaches the lesson. The taughtInGrade property accepts ClassGrade objects that show which class each lesson belongs to.

![Object Properties Class Course.](image)

**Data Properties**

Data properties are the properties of the classes that associate objects with data values (string, integer, date, etc.) and not attributes of other objects such as object properties.

In this section we will describe some of the data properties of the classes we created for our ontology. The data properties we have developed are shown in Figure 10:

![Data properties.](image)

- The accoladeDescription and punishmentDescription attributes are alphanumeric (string) belonging to the Accolade and Punishment classes respectively and contain a description of the type of praise and punishment respectively. In Figure 11 we see that the attribute belongs to the Accolade class and is of string type.

![Figure 11. accolade_description](image)

- The address, firstName, and lastName attributes are alphanumeric (string) belonging to the Person class and contain values for the address, name and surname of the persons. Parent, Student, Staff, OtherStuff, and Teacher classes inherit these attributes from the Person class. Figure 12 shows that address attribute belongs to the Person class and is a string type.

![Figure 12. address](image)

- The classroom_projector, classroom_smartBoard, classroom_sound, labClassRoom_chemistry, labClassRoom_info and labClassRoom_phyisc attributes are boolean, so they can take true or false values and describe how a class can be used. In Figure 13 we see the Classroom_projector attribute that belongs to the LectureClassRoom class and is a boolean type.

![Figure 13. Classroom_projector.](image)

**Individuals**

Individuals are instances of classes. For our own ontology, we will present some indicative instances to describe how the modelling of a high school/secondary school information system works. Figure 14 shows a part of the instances we have created.

![Figure 14. Individuals.](image)
We originally created an instance of the school class called 1st_High_school_Thessalonikis. The characteristics of the instance are shown in Figure 15.

Figure 15. 1st_High_School_Thessaloniki.
We see that the instance we created belongs to the School (Types) class. It has schoolHasClasses as object properties, in which it accepts two instances: LabClassRoom and LectureClassRoom. Also, as data properties, it is given two string values for school_name and for school_address.

The instances we have created from the ClassGrade class are 1st_grade_High_School, 2nd_grade_High_School, 3rd_grade_High_School, 1st_grade_Secondary_School, 2nd_grade_Secondary_School, and 3rd_grade_Secondary_School. The features of the 1st_grade_High_School instance are shown in Figure 16.

Figure 16 .1st_grade_High_School.
We see in the 1st_grade_High_School instance features that it has as object properties the hasCourses in which it receives instances of the Courses class, and the classSetInClassRoom in which it receives instances of the ClassRoom class. To classGradeName data properties which is alphanumeric (string), has been given the value “1st grade high school”. One of the highlights of the Course class we have created is Biology (image 17).

Figure 17. Biology
This instance contains attributes of the taughtInGrade type in which we have instances of GlassGrade class and in addition we have the courseName attribute which is alphanumeric and contains the name of the course.

From the Student class we have created multiple instances. Specifically the instance Student1 (image 18).

Figure 18. Student.
We see that it has GuardianBy features that contain a Parent class instance with parent1 value which is the pupil’s guardian, hasAccolade instance of Accolade class, that is, the student has been awarded, a study instance of classGrade class showing which class the pupil attends. We also have values for the student’s name, surname and code.

Finally, from the Teacher class instances in Figure 19 we see the instance teacher1.

Where it has teaches as attributes that are instances of the Courses class and show us what lessons the particular teacher teaches, the isSpecialized attribute is of the TeacherFaculty type and contains the teacher’s specialty.

Reasoner
We have used Pellet for the control of ontology. An automated OWL-DL reasoning program, written in java. It collaborates with the Jena framework and is the primary solution for applications written in java. It has the ability to classify ontologies together with their relationships. It also has a built-in engine for SPARQL.

We first executed the pellet with the classify parameter image which shows the class hierarchy as a result.

Figure 20 . Pellet classify.
With the consistency parameter we checked the consistency in our ontology image.

**Figure 21. Pellet consistency.**

With the info parameter we see information for ontology (Figure 23).

**Figure 22. Pellet info.**

Finally, with the lint parameter we check if there are problematic classes (Figure 23).

**Figure 23. Pellet lint.**

Part II SPARQL

**SPARQL (Simple Protocol and RDF Query Language)**

SPARQL (Simple Protocol and RDF Query Language) is the official query language for RDF. It was standardized by the W3C RDF Data Access Working Group and in 2008 the W3C was formally established for SPARQL 1.0. SPARQL is for RDF precisely what SQL is for relational databases, or what XQuery is for XML. And because the Semantic Web is based on RDF knowledge representation, SPARQL and its related protocols are paramount in it.

Trying to use Semantic Web without SPARQL is like trying to use a relational database without SQL [6].

SPARQL queries are based on triple patterns. A triple pattern is the same as a RDF triple, except that one or more of its resources-components are variable. A variable is denoted by ?name or $name, where name is its name. For variables the prefix ? or $ is the same [8].

The SPARQL engine that executes a query searches from all resources, those that verify the query triple patterns, in accordance with the RDF suggestions in the knowledge base with which the SPARQL engine is connected.

SPARQL has many similarities to SQL. A query in SPARQL can contain conjunctions, disjunctions, optional limitations, limitations on the number of results, and generally almost all of the basic fundamental elements that SQL has (Figure 24).

There are four SPARQL query formats: SELECT, CONSTRUCT, ASK and DESCRIBE.

- **SELECT**: These are queries that return the objects in a table format (resources and verbal) that verify the desired triple patterns
- **CONSTRUCT**: These are queries that return an RDF graph, according to a graph template included in the query.
- **ASK**: These are queries that answer whether or not there is a solution for some triple patterns, without stating what that is, if it exists, eg.

```sparql
ASK
WHERE {
  FILTER ( lang(?property) = "el")
}
```

- **DESCRIBE**: These are queries that return an RDF graph containing data about some of the resources declared in the query, e.g.

```sparql
DESCRIBE <http://el.dbpedia.org/resource/Teacher>.
```

There are SPARQL implementations for many programming languages and tools such as:

- the connection to a SPARQL interface and the semiautomatic construction of a query, and
- the translation of SPARQL queries into other query languages (eg. SQL, XQuery, etc.).

SPARQL 1.1 was designated as W3C Recommendation on March 21, 2013 [7, 9].

**Εικόνα 24. Τυπική σύνταξη ερωτήματος SPARQL.**

**ANNEX A - SPARQL QUESTIONS**

On all queries, the Ontology School was used because the Greek version of dbpedia (el.dbpedia.org) is out of order.

In the exercise it is mentioned that the examples are executed at a DBpedia endpoint, unfortunately there are some restrictions to these, for example: SERVICES outputs “Virtuoso 42000 Error SQ200: Must have privileges on view DB.DBA.SPARQL_SINV_2”. We therefore used also different endpoints for the exercise needs.

**SPARQL query # 1**

**Endpoint**: http://dbpedia.org/snorql

**SPARQL Query**

```sparql
SELECT DISTINCT ?property ?hasValue ?isValueOf
WHERE {
  SELECT DISTINCT ?property ?hasValue ?isValueOf
  WHERE {
    <http://el.dbpedia.org/resource/Σσολείο> ?hasValue
    UNION
    <http://el.dbpedia.org/resource/Σσολείο> ?isValueOf
  }
  LIMIT 300
}
```

**Description**

Displays the ontology names in languages other than Greek (Ontology School).

**Results**

**Ερώτημα SPARQL # 2**

**Endpoint**: http://dbpedia.org/snorql

**SPARQL Query**

```sparql
SELECT distinct ?property ?desc
WHERE {
  FILTER ( lang(?desc) = "en")
}
```

**Description**

Displays the related information - properties such as abstract, commentary for English.

**Results**
SPARQL query #3
Endpoint: http://dbpedia.org/snorql
SPARQL Query
PREFIX : <http://dbpedia.org/resource/>
SELECT distinct ?Category ?Concept
WHERE {
  FILTER (?Category IN (:School, :Gymnasium, :Lyceum)) .
  FILTER ( lang(?Concept) = "en" ) .
} LIMIT 10

Description
Displays the related information - properties such as abstract, commentary for English. The difference with the previous query is that it looks for Gymnasium and Lyceum beyond the School ontology.

Results

SPARQL query #4
Endpoint: http://dbpedia.org/snorql
SPARQL Query
SELECT distinct ?Category ?Concept
WHERE {
  FILTER ((?Category IN (:Teacher, :Paraprofessional))
  && ( lang(?Concept) = "en" )) .
} order by ?Concept

Description
Displays related information - properties such as label, comment for English for Teacher and Paraprofessional ontologies sorted by ?Concept variable.

Results

References