Avestia Publishing
Journal of Machine Intelligence and Data Science (JMIDS)
Volume 2 Year 2021
ISSN: 2564-3282
DOI: 10.11159/jmids.2021.001

A Hands-on Project for Teaching Semantic Web Technologies in an Undergraduate AI Course

Neli P. Zlatareva
Central Connecticut State University
Department of Computer Science
1615 Stanley Street, New Britain, CT 06050, USA
zlatareva@ccsu.edu

Abstract - *The latest advances in Semantic Web technologies suggest an accelerating emergence of new exciting Artificial Intelligence applications that are expected to dramatically extend and improve current web services. Yet, these new technologies are outside the scope of undergraduate computer science curriculum. This paper presents our experience with introducing a hands-on project intended to teach Linked Data and Semantic Web as part of an undergraduate Artificial Intelligence course. The project is intended to achieve the following: 1.) Demonstrate the evolution of Knowledge Engineering into Ontological Engineering; 2.) Introduce students to Semantic Web technologies and tools such as ontology editor Protégé, Web Ontology Language (OWL), Semantic Web Rule Language (SWRL), and query language SPARQL; 3.) Extend the topic on reasoning into Description Logics and demonstrate the advantages of their inferencing capabilities; 4.) Use OWL and SWRL to compare descriptive and rule-based reasoning frameworks and show how their integration can improve the efficiency and the semantic adequacy of applications; 5.) Illustrate the Linked Data principles in a practical setting. Limited assessment of the pedagogical value of this project based on student learning outcomes suggests that it enhances students' understanding of the core AI topics, boosts their engagement and interest in the course, but more importantly introduces them to the newest advances in web application development.*

Keywords**:** Computer Science Education, Artificial Intelligence, Ontological Engineering, Description Logics, Semantic Web.

1. Introduction

Semantic Web (SW) is envisioned to extend and dramatically improve current web services by providing a universal language for information exchange allowing data to be shared and reused by applications. Since Tim Berners-Lee coined the term in late 1990s, the enthusiasm for implementing his vision has grown exponentially, and nowadays the theory and practice of the Semantic Web is mature enough to make a difference in how to utilize the enormous amount of information available on the web. Yet, these new technologies are outside the mainstream of undergraduate CS curriculum. The hands-on project presented in this paper aims to introduce students to the Semantic Web and Linked Data technologies in practical terms and at the same time extend their understanding of knowledge engineering to include ontological modelling and semantic mark-up.

The SW project has the following learning objectives:

LO1: Demonstrate the evolution of Knowledge Engineering into Ontological Engineering.

LO2: Introduce students to SW technologies and tools such as ontology editor *Protégé*, Web Ontology Language (*OWL*), Semantic Web Rule Language (*SWRL*), and query language *SPARQL*.

LO3: Extend the topic on reasoning into Description Logics (*DLs*) and demonstrate the advantages of their representation and inferencing capabilities.

LO4: Use OWL and SWRL to compare descriptive and rule-based reasoning and show how their integration helps improve semantic adequacy of applications.

Date Received: 2020-08-24
Date Accepted: 2020-11-19
Date Published: 2021-02-26

LO5: Illustrate the Linked Data principles in a practical setting by utilizing Apache Jena API to build a prototype SW application.

We briefly elaborate on these objectives next.

Knowledge engineering is a core topic in any undergraduate AI course, which introduces students to the application site of AI. Although the field has evolved considerably over the years and now offers well-established methodologies for building Knowledge-Based Systems (KBSs) [1], it does not fully demonstrate the underlying principles of any engineering discipline, namely knowledge sharing and reuse, as it is a common practice to build knowledge bases from scratch. These typically reflect the view of a domain expert or a group of experts without imposing any restrictions on the vocabulary used to represent domain knowledge. KBSs are built as stand-alone problem solvers intended to provide the best advice according to that view, which might not be a consensus view on that domain. Ontological engineering, on the other hand, emphasizes the consensus knowledge of the community which is expressed by precisely defined terms and thus, as advocated in [2], is seen a successor of knowledge engineering.

Ontological engineering as a field has a long history dating back to early 1980s. It was inspired by Newell's AAAI presidential address [3], where he advocated that it is not sufficient to describe knowledge at the "symbol level" (the *physical-symbol system hypothesis* formulated by Newell and Simon [4] is still the underlying principle of modern AI), but at a more abstract "knowledge level" to emphasize generic definitions and reusable reasoning patterns. This resulted in a paradigm shift from "production rules" technology to "knowledge modelling" which led to the realization that "…we can build sharable knowledge bases for wider usability than that of a conventional knowledge base" [2]. Semantic Web is the perfect domain to demonstrate the critical role of ontological engineering in ensuring semantic interoperability between applications utilizing such sharable knowledge bases, or *ontologies*.

The term *ontology* has its roots in philosophy, but in the context of knowledge representation it is "… an explicit, formal specification of a shared conceptualization" [5]. That is, the ontology defines fundamental concepts in the domain of interest, as well as their properties and relations, and explicates the agreed upon domain assumptions allowing for a unique interpretation of that domain by any agent, human or machine. Building an ontology is similar to building a data model in a relational database application with one fundamental difference, namely, ontologies implicitly define formal rules of inference thus allowing new information to be derived about objects and their relations. Languages for building ontologies, therefore, must have a well-defined formal semantics to ensure that such inferences are sound. A lot of research was devoted to developing ontology languages for the Semantic Web [6]. Currently, the Web Ontology Language *(OWL)* is the official recommendation of Web Ontology Working Group of W3C [7]. Building ontologies directly in OWL, however, is an extremely difficult task, which is why a number of tools were developed to facilitate this process. Protégé [8, 9] is the most widely used open source ontology editor, because of the variety of features offered including DLs reasoners. Two of the reasoners, *Pellet* [10] and *HermiT* 1.4 [11], support SWRL allowing for easy comparison of descriptive and rule-based reasoning within the same framework.

DLs are not typically covered in an undergraduate AI course but they are becoming increasingly important with the widespread need for open access digital libraries of various information resources and databases residing on the Linked Open Data Cloud [12]. These are decidable fragments of first-order logic intended to achieve favorable trade-offs between expressivity and scalability. Introducing DLs allows us to stress the importance of reasoning that is both decidable and expressive. It also brings the discussion on semantic networks and frame-based representations, which are the origins of DLs, to a more practical level and illustrates how these alternative knowledge representation languages were extended and linked together. Because DLs is a family of logics which defer by their expressivity depending on the constructors employed to build complex descriptions, we choose to cover the simplest logic, *Attributive Language with Complement (ALC)*, in detail and introduce the more expressive constructors available in OWL and Protégé, respectively, and corresponding to the *SROIQ(D)* logic as we progress throughout the Semantic Web project. *SROIQ(D)*, although very expressive, is NP-hard, and if fully utilized is extremely slow even for a small-scale application like ours. We show that combining *SROIQ(D)* reasoning with SWRL rules can improve run-time efficiency and avoid some of the pitfalls of DLs reasoning caused by the *Open World Assumption (OWA)*. We demonstrate the limitations of rule-based reasoning due to the *Closed World Assumption (CWA),* and show that combining

SWRL rules and DLs allow us to improve semantic adequacy of the obtained results.

Finally, we demonstrate how all these technologies are utilized to build artificial intelligence applications for the Semantic Web. As long as we have the data in a serializable RDF format (a huge number of such datasets is currently available on the Linked Open Data Cloud, https://www.lod-cloud.net/), we can use tools such as Apache Jena to set-up an application in a fast and convenient way. Brief introduction to Jena API and query language SPARQL is intended to demonstrate the development of such applications by using a simplified version of the ontology presented in this paper.

The rest of the paper is structured as follows. Section 2 discusses the motivation and formal preliminaries of the project. In Section 3, we present the lesson plan, activities and assignments intended to evaluate student progress towards project learning objectives. Section 4 introduces the SW project in some detail with a reference to the web site where the actual code can be found. We conclude with some assessment results and reflect on some challenges that we plan to address in future course offerings.

2. Motivation and Formal Preliminaries

The main goal of the Semantic Web project is to illustrate the evolving understanding of AI from a stand-alone problem solver to a network of intelligent agents working in cooperation and serving as equal partners to humans in a variety of applications built on the top of the Semantic Web. The best example of this transition is the Linked Open Data Cloud, which can be viewed as a huge library of compatible datasets that SW applications can easily access, interpret and integrate utilizing a common reasoning framework based on DLs. As part of the project, we introduce students to the underlying representation, *Resource Description Framework (RDF),* its derivatives (RDFS and OWL) and serializations, and discuss how it changes established knowledge engineering practices.

Assuming that students are already familiar with the foundations of knowledge representation and reasoning, introducing them to OWL and DLs as the latest advances in the field should not be a challenge. In fact, an ontology is a formal representation of a semantic network defined as a set of triples <*Subject, Predicate, Object*>, where $\in$, $\sqsubseteq$, and $\equiv$ are special predicates for describing membership, subsumption, and equivalence relations, respectively.

In DLs context, a KBS is a triple <*TBox, ABox, RBox*>, where:

- The *TBox* defines the agreed upon domain terminology expressed as a hierarchy of classes (concepts) and formally described by subsumption and equivalence relationships between classes, $C \sqsubseteq D$ and $C \equiv D$, respectively. The latest version of OWL implemented in Protégé, OWL 2, also includes disjunction constructor, a special class expression Self: $\exists S.Self$, and allows for qualified number restrictions $\geq n\ S.C$ and $\leq n\ S.C$ to express statements such as "a family with at least/at most 3 children".
- The *ABox* contains facts about the domain expressed as class membership of domain entities/individuals ($a \in C$, or equivalent $C(a)$), property relations between domain entities (<a, R, b>, or equivalent R(a, b)), and equality relations between individuals (a = b, or sameAs(a, b)).
- The *RBox* defines complex properties such as inverse properties, symmetry, reflexivity, irreflexively and disjunctiveness of properties, as well as combination of properties (property chains), $R_1 \circ R_2 \sqsubseteq S$, allowing statements such as "my father's brother is my uncle".

Choosing an appropriate domain for the project was a major challenge since it was supposed to be accessible to students, inference rich and easy to navigate and evaluate inference results. Our initial choice was the "university domain" [13], but it did not offer a broad variety of inference patters although it allowed for experimentation with a number of reusable inference tasks such as web search, data integration and personalized recommendation. Other domains considered were "home design", "car buying", "choosing a movie", and a few more (some of these were explored by students as independent research projects). These domains made great Semantic Web applications but did not serve well as inference test beds for the project. We finally decided on the most "trivial" choice – the "family" domain, because it is inference rich, can be easily described in both procedural and declarative terms thus allowing us to illustrate advantages and disadvantages of both frameworks and show how their integration ensures better semantic adequacy. Another advantage is

that students are already familiar with this domain as it is used in our textbook [14] to illustrate FOL.

3. Lesson Plan, Activities, and Assignments

The Semantic Web project is designed as a final four-weeks module of our undergraduate AI course. Students are expected to have already acquired knowledge on various knowledge representation frameworks (Propositional Logic, First-order Logic, Default Logic, Semantic Networks) and have practiced associated reasoning techniques.

The four-weeks lesson plan, activities and assignments helping assess student progress towards achieving project learning objectives is as follows:

Week 1:

- Introduction to the Semantic Web and Protégé. After a brief 40-minutes lecture outlining the limitations of current web technologies and traditional knowledge engineering, students are assigned several motivational on-line presentations about the SW [15, 16, 17, 18], and a tutorial on Protégé [8] for independent work. Students are asked to write a brief essay on SW and ontological engineering to assess their progress towards LOs 1 and 2. The remaining half of the lecture introduces the tableaux algorithm for PL to allow for easier transition to DLs. We found that students quickly grasp this new reasoning technique after being previously introduced to Wang's algorithm [19].
- Second lecture is devoted to DLs. We cover *ALC* syntax, model-theoretic semantics and modified tableaux algorithm for *ALC* in detail. An assignment on the latter (proofs) is given as an assessment instrument towards LO 3 and also to ensure that students are prepared to tackle more advanced DLs implemented in different versions of OWL.

Week 2:

- First lecture tackles the foundations of the SW: *RDF* and its serializations, *RDFS* and its axiomatic and Direct Inference System semantics. These are graph-based data models which makes it easy for students to connect them to semantic networks. However, it is important to emphasize the major difference between RDF/RDFS and semantic networks, namely the lack of well-defined semantics for the latter.
- In the second lecture, we expand on *ALC* to introduce more DLs constructors in transitioning to OWL which latest version, *OWL 2*, is based on *SROIQ(D)* logic. As we discuss different versions of OWL 2, we stress on the need for a reasonable balance between expressivity of the language and efficiency of its inference procedure. Students can assess this balance in practical terms (experimenting with different types of inference tasks) as they work through the hands-on project as described in the next section.

Week 3:

- Students are expected to have completed the introductory Protégé tutorial and be ready to begin hands-on practice with the Semantic Web project. First lecture introduces project objectives and the "family" ontology (see next section for details). Based on this example domain and student essays, we further advance the discussion on similarities and differences between knowledge engineering and ontological engineering. Students are asked to expand the initial domain (the *A-box*) with a new "related" family to familiarize themselves with the ontology and use Protégé reasoners *HermiT* and *Pellet* to validate the extended ontology and compare inference and performance results. They are expected to report these results on the project discussion board explaining noticed semantic discrepancies. They also should notice that *HermiT* run-time performance is better even on a small application like ours (due to a more efficient version of the tableaux algorithm that it employs, called *hypertableau*, but we were not able to go into details and interested students were referred to [11]). Students should also catch some unintuitive inferences due to the underlying assumptions, *Open World Assumption (OWA)* employed by underlying DL and *Closed World Assumption (CWA)* which is behind SWRL.
- Second lecture is split between a discussion about CWA and OWA explaining unintuitive inference results reported by the students, and introduction of the Semantic Web Rule Language (SWRL) supported by both Pellet and HermiT. SWRL rules are Horn clauses applied in forward chaining manner and thus

easily comprehensible by students. In Protégé, students can experiment with both procedural and declarative reasoning on the same ontology. For that, students are given multiple queries phrased in English (see next section) which they must translate to OWL and run in DL Query tab in Protégé. To see the difference between the two reasoning frameworks, students are also asked to substitute some of the SWRL rules with property chains (i.e. expand the *R-box* of the ontology), and finally to use the Drools rule engine (embedded in Protégé) to convert SWRL rules and all relevant OWL knowledge into OWL2 RL (the rule version of OWL2) which should result in a considerably faster run time on the same queries. This assignment aims to assess student progress towards LOs 2, 3 and 4.

Week 4.

- First lecture reviews the results of student experiments with an emphasis on semantic limitations of both descriptive and procedural frameworks. We discuss how to manage the consequences of OWA in the T-box to improve the adequacy of OWL reasoning and why CWA is instrumental in achieving computational efficiency in rule-based reasoning (SWRL is strictly monotonic) at the expense of some semantic inadequacy. Interestingly, some non-monotonicity can be "simulated" by utilizing OWL negation constructor in more advanced DL queries. Students are asked to experiment with different versions of such queries to ensure that only valid results are returned and explain run-time differences due to property restrictions involved. This assignment aims to assess LO 4.
- Final lecture summarizes the results of the project and discusses how family ontology can become part of the Linked Open Data Cloud. For that, we have created a small application using Apache Jena (http://jena.apache.org) which provides extensive Java libraries for processing RDF files, as well as allows for SPARQL queries to be integrated into the application code. It should be noted that the original family ontology developed in Protégé and processed using DLs reasoners cannot be efficiently handled by Jena reasoners including Jena OWL reasoner. The letter uses rules to reason about instances while reasoning about classes is performed by utilizing prototype instances of classes. For example, if a prototype instance of class A is proved to be a member of class B then it is concluded that A is a *subClassOf* B. This type of reasoning cannot handle expressive ontologies such as our family ontology which was intended to illustrate the richness of descriptive representations. On the other hand, Jena API allows for rapid application development and easy deployment on the SW. To introduce students to Jena software, we have created a smaller less expressive version of the family ontology which can be efficiently processed by Jena reasoners. Various types of SPARQL queries were demonstrated and students were encouraged to use this prototype framework to create their own applications utilizing RDF files on Linked Open Data Cloud.

4. Project Description

As stated above, the project is about building an open, distributed "family" library, where people can input information about their families to discover various inter-family and cross-family relations between individuals. The ultimate goal is to deploy this library on the Linked Open Data Claud by configuring a W3C-standard SPARQL endpoint using Apache Jena Fuseki. Unfortunately, limited time for the project does not allow us to discuss this important practical matter. At this point, development activities as well as ontology processing are performed in Protégé, and a small Java application created with the help of Jena API is used to exercise a variety of query patterns.

Students begin their work on this project with a fully functional ontology and their first assignment is to study and expand it as suggested in section 4.1. Next step is to experiment with a reasoner to study various *SROIQ(D)* constructors (section 4.2.) followed by a more detailed assessment of OWL 2 property chains and comparing them to SWRL rules built into the original ontology (section 4.3.). The final part of the project illustrates how Jena API (http://jena.apache.org) can be used to build and query Semantic Web applications (section 4.4.).

4.1. Exploring and extending family ontology

The objective of this project module is to demonstrate the ontology development process.

Once the domain and the intended purpose of the ontology are defined, the next step in the development process is to decide on the common terminology describing domain knowledge. In our case, these are the terms and relations referring to the entities in the family domain. We have chosen the classification given at freepages.rootweb.com as a starting point for building our family ontology, which was subsequently modified to better fit the needs of the project. It suggests a very limited number of classes describing people and their sex, *Person* and *Sex*, with subclasses *Parent*, *Male* and *Female*. In addition, we want to have a class, *Family*, to represent a group of individuals that belong to the same family, and *Man/Woman* as defined classes for more convenient specification of individuals. Such "shortcuts" are common in knowledge engineering for improving efficiency. Currently, we have four defined *Family* subclasses, *BennettFamily, BrownFamily, RichardsFamily* and *SmithFamily* (see http://www.cs.ccsu.edu/~neli/FamilyProject.owl).

Deciding on basic properties is the next step. In OWL (and Protégé, respectively) properties are divided into *object properties* and *data properties*. The former describe relations between domain entities, while the later define object attributes.

We have chosen the following set of basic properties: *hasLastName, hasFirstName, hasBirthday* (data properties describing individuals), *hasMother, hasFather, hasSpouse, hasFormerSpouse* (object properties describing relations between individuals) and *hasSex* (also an object property associating individuals with *Female/Male* objects). Other properties depicted at freepages.rootweb.com and some additional ones are shown on Figure 1.

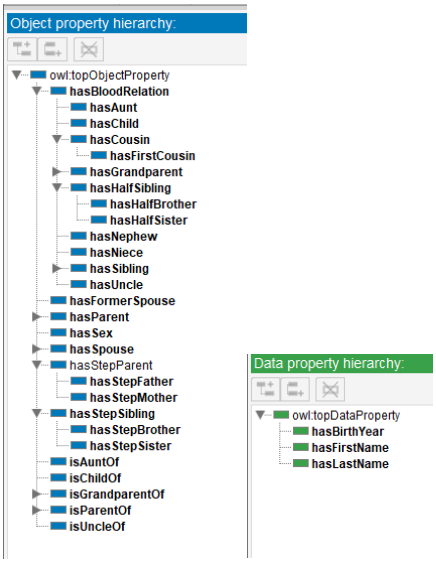

Figure1. Property hierarchy in family ontology.

After familiarizing themselves with the basic ontology design, students are asked to expand the ontology with a new "related" family to be used in subsequent processing.

4.2. Utilizing reasoners

This part of the project is intended to familiarize students with DL reasoners such as Pellet and later compare them to the rule-based reasoners such as Drools. To illustrate some of Pellet features, consider individual *BR1972* initially defined as:

FamilyProject:BR1972 rdf:type
owl:NamedIndividual ,
FamilyProject:Man;
FamilyProject:hasFather FamilyProject:IR1940 ;
FamilyProject:hasMother FamilyProject:IG1945 ;
FamilyProject:hasBirthYear 1972 ;
FamilyProject:hasFirstName "Boris" ;
FamilyProject:hasLastName "Richards" .

After running a reasoner (see Figure 2), some of the facts derived about BR1972 are counter-intuitive (BR1972 is his own sibling, halfsibling, brother) and students are asked to explain why, and how this can be revised (at this point, students are working with the original ontology which combines knowledge in OWL and SWRL). They are expected to notice that the problem cannot be resolved by making *hasSibling* property reflexive (which causes logical inconsistency), nor the corresponding SWRL rule can be modified due to the monotonic nature of the rule-based formalism. Students are also asked to create DL queries to extract specific

subsets of derived results. For example, *hasSibling value BR1972* returns BR1972 (counter-intuitive), SR1970, and VR1965. If asked *hasSibling min 3*, BR1972 is one of the instances returned. But if asked *hasSibling max 3*, no result is returned. Students are asked to explain this and similar results, obviously caused by the OWA under which DLs operate.

Classification is a major task of a DL reasoner. In our ontology, if we want to find all members of a given family, we must appropriately define the class. The description of *BennettFamily* as well as the derived instances of the class are shown on Figure 3.

Students are asked to modify *Family* subclasses' definitions to include only immediate family members or only in-laws, and create new classes including members of more than one family (for example class *BrownSmithFamilies* to include members of both families).

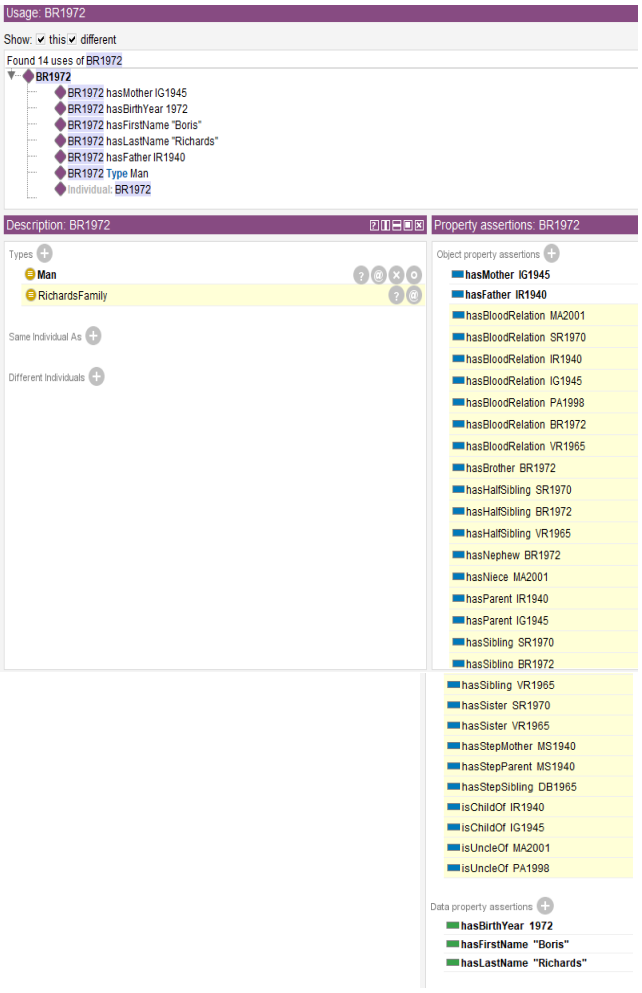

Figure 2. Derived facts about BR1972.

4.3. SWRL rules *versus* OWL property chains

One of the main emphases of this project is to demonstrate the difference between descriptive and procedural knowledge representations. The original ontology utilizes SWRL rules to express the procedural component of the domain knowledge. Students are asked to substitute rules 1 through 7 (http://www.cs.ccsu.edu/~neli/FamilyProject.owl) with property chains to compare the expressiveness of the two formalisms. Some rules can be easily converted into property chains, such as *hasParent(?x, ?y) ^ hasParent(?y, ?z) -> hasGrandParent(?x, ?z).*

Other rules, however, are impossible to convert within the selected representation framework. For example, consider the rule deriving *hasUncle* (*isUncleOf* is declared as its inverse):
hasParent(?x, ?y) ^ hasBrother(?y, ?z) -> hasUncle(?x, ?z)
It seems natural to express *hasBrother(?y, ?z)* as *hasSibling o (hasSex value Man)*. However, *hasSex value Man* is not expressible in OWL and the only way to resolve this problem is to extend the A-box by declaring sisterhood/brotherhood relations explicitly.

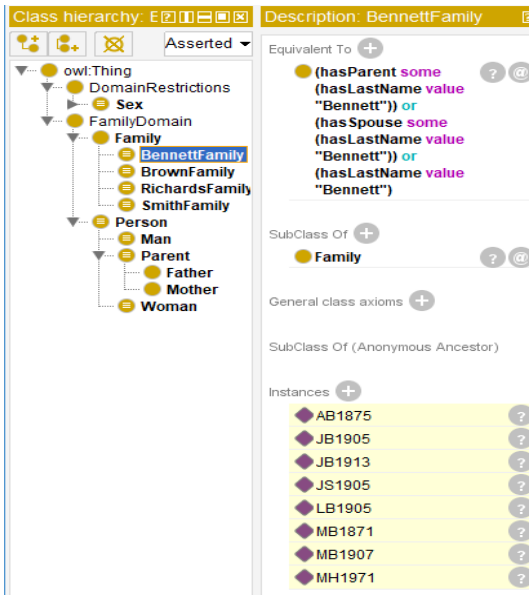

Figure 3. *BennettFamily* class description and its instances.

After "hard-wiring" sisterhood/brotherhood relations in the A-box, students will encounter the same problem each time *hasSex* property is involved and applying the same solution would result in an unreasonable extension of the A-box.

A different problem is illustrated by the rule

hasStepMother(?x, ?m) ^ isMotherOf(?m, ?c) ^ hasStepFather(?c, ?y) ^ hasFather(?x, ?y) ->

hasStepSibling(?x, ?c).

Note that *hasFather(?x, ?y)* is not part of the property chain *hasStepMother o isMotherOf o hasStepFather* and thus *hasStepSibling* cannot be defined by a property chain. Similarly, if *hasParent(?x, ?p) ^ hasSpouse(?p, ?s) ^ hasFormerSpouse(?p, ?f) ^ isParentOf(?f, ?x) -> hasStepParent(?x, ?s)* is "converted" to *hasParent o hasSpouse o hasFormerSpouse o isParentOf,* it would result in deriving the triple *DB1965 hasStepParent SR1970* which is incorrect.

These experiments demonstrate differences in rules' and DL statements' expressivity. Clearly, OWL property chains are limited in the type of causal relationships they can express to what is referred in [20] as "limited transitivity". At the same time, rules alone also produce semantically incorrect results due to their monotonicity. For example, BR1972 is derived to be his own brother, sibling, half sibling, and nephew. However, we can create a DL query that takes the result returned by the rule *hasMother(?x, ?m) ^ hasFather(?x, ?f) ^ hasMother(?y, ?m) ^ hasFather(?y, ?f) -> hasSibling(?x, ?y)* and filters incorrect instances by using the DLs negation constructor, i.e. *hasSibling value BR1972 and not (hasFirstName value "Boris").* Notice that this is very different from the "negation as failure" rule in non-monotonic reasoning.

Asking for JS1959 cousins can be done by means of the following queries:

hasCousin value JS1959 and not (hasParent some (isParentOf value JS1959))

or

hasCousin some (JS1959) and not (hasParent some (isParentOf value JS1959))

It is interesting to notice the difference in execution times of these semantically equivalent queries. The former takes 67.372 seconds, while the latter takes 89.556 seconds (with Pellet). Clearly, using nominals (i.e. searching in a class *versus* referring to a particular individual) is computationally less efficient. Yet another version of the same query, *hasCousin some (hasFirstName value "Jacob") and not (hasParent some (isParentOf value JS1959)),* takes more than 7 minutes to return the result (times vary depending on computer speed, but their correlation stays the same). As part of their last assignment, students are asked to create similar equivalent queries and explain their vastly different execution times. Obviously, constructors used in each class expression define OWL profile supporting query execution and OWL profiles are based of different DLs which belong to different complexity classes.

Finally, as part of this module, students are assigned to study and experiment with *Drools*, a rule-based inference engine embedded in Protégé. It is interesting to note that Drools and Pellet process rules differently. Students will notice that conclusions derived by SWRL rules using Pellet are highlighted in yellow (see Figures 2 and 3) and treated as any other conclusion by the reasoner. This is because for Pellet, SWRL rules are an "extension" of OWL representation. Drools, on the other hand, implements OWL 2 RL (the rule version of OWL 2) and works as a traditional rule-based engine. Once a conclusion is derived, it is treated as an axiom and thus prevents Protégé explanation capability to be initiated for that conclusion. This, however, is not a bad thing, because if properly used it may improve the computational efficiency. When Drools is called, all OWL axioms and SWRL rules are transferred to the rule engine for processing and all inferred axioms are transferred back to OWL. The resulting ontology is now permanently extended, and all subsequent DL queries are executed much faster. For example, the last version of the query about JS1959 cousins takes 3.771 seconds compared to more than 7 minutes previously.

We have seen so far that neither OWL nor SWRL allow for building semantically correct extensions of the underlying dataset. In both frameworks, semantic inadequacies result from overgeneralization errors. If we can envelop those extensions in a rule-based application intended to identify and process such overgeneralization errors, then we can provide the user with semantically adequate results. The last module of the project describes a small application built with the help of Jena API (http://jena.apache.org). It also introduces students to *SPARQL*, the query language for the Semantic Web.

4.4. Building applications for the Semantic Web

It should be noted that our original ontology was intended to demonstrate the richness of descriptive representations, but such ontologies cannot be efficiently handled by Jena reasoners. This is why we created a smaller, simplified version of our ontology (see http://www.cs.ccsu.edu/~neli/FamilyProjectTiny.owl) to allow students to experiment with different Jena reasoners (RDFS-based, OWL-based, transitive). It should be noted that Pellet can be easily integrated with Jena which allows for efficient processing of the original ontology, but for the purposes of this project we feel it is

more beneficial for the students to experiment with conventional built-in Jena reasoners.

The following snippet of Java code creates and loads the ontology model, and instantiates and runs Jena OWL reasoner (see [21] for more on Jena API):

```
Model schema =
ModelFactory.createOntologyModel();
schema.read("https://cs.ccsu.edu/~neli/FamilyPr
ojectTiny.owl", null, "RDF/XML");
Reasoner reasoner =
     ReasonerRegistry.getOWLReasoner();
reasoner = reasoner.bindSchema(schema);
InfModel owlSchema =
ModelFactory.createInfModel(reasoner, schema);
```

The extended dataset, *owlSchema*, is now ready

```
------------
| member   |
============
| "Tony"   |
| "Mark"   |
| "Marie"  |
| "Leo"    |
| "John"   |
| "Joyce"  |
| "Archie" |
------------
```

A custom implementation of the *ResultSetFormatter* method allows the result to be returned in a desired format rather than in a standard table form and can be further processed by the application as appropriate.

The *CONSTRUCT* query allows the application to extract triples directly from the dataset or build new triples out of existing ones. Consider the following *CONSTRUCT* query intended to create sibling relations between children of *AB1875* (same prefixes as above marked here as …):

```
String queryQuestion = new String ("Retrive the
children of :AB1875 and establish sibling
relation between them");
String queryString = … +
"CONSTRUCT {?name1 :hasSibling ?name2}  " +
"WHERE {?child1 :hasFather :AB1875 . " +
       "?child2 :hasFather :AB1875 . "   +
       "?child1 :hasFirstName ?name1 . " +
       "?child2 :hasFirstName ?name2 . }";
```

The query is instantiated the same way as shown above. Here is the snippet for executing the query:

```
System.out.println(queryQuestion);
try {Iterator<Quad> triples =
qexec.execConstructQuads();
while (triples.hasNext()) {
   Quad quad = triples.next();
   System.out.println(quad.getSubject() + " " +
   quad.getPredicate() + " " +
                    quad.getObject()); }
} finally {qexec.close();}
```

The result with abbreviated links to resource *hasSibling* is shown next:

```
     "Leo" http://...#hasSibling "Joyce"
     "John" http://...#hasSibling "Mark"
     "John" http://...#hasSibling "Leo"
     "John" http://...#hasSibling "John"
     "John" http://...#hasSibling "Joyce"
     "Joyce" http://...#hasSibling "Mark"
     "Joyce" http://...#hasSibling "Leo"
     "Joyce" http://...#hasSibling "John"
    "Joyce" http://...#hasSibling "Joyce"
```

These newly created triples can be added to the dataset thus further extending the model. That is, *CONSTRUCT* queries can act similarly to "if-then" production rules.

The *ASK* query allows the application to ask if a particular triple or a set of triples is present in the dataset. Assume, we ask if Bennett family has a member named Mark or Marie:

```
String queryQuestion = new String ("Is there a
Bennett with first name Mark or Marie?");
String queryString = … +
"ASK WHERE {{?m a :BennettFamily . " +
            "?m :hasFirstName 'Mark' .}" +
            "UNION  {?m a :BennettFamily ." +
            "?m :hasFirstName 'Marie' .}}";
```

The output is "Yes". It would be "No" if neither Mark, nor Marie is a member of the family.

Here is the snippet executing the query:

```
System.out.println(message);
try {System.out.println(qexec.execAsk() ?
                                "Yes" : "No"); }
finally {qexec.close();}
```

The *DESCRIBE* query returns an RDF file in a *TURTLE* format containing all triples in the dataset related to a particular resource. Here is the one about Marie Bennett:

```
String queryQuestion = new String ("Tell me
everything you know about Marie Bennett.") ;
String queryString = … +
     "DESCRIBE ?member " +
    "WHERE {?member a :BennettFamily . " +
           "?member :hasFirstName 'Marie' . }";
```

The output for this query is very long but here is the beginning of it:

```
Tell me everything you know about Marie
Bennett.
```

```
@prefix :
<http://www.cs.ccsu.edu/~neli/FamilyProjectTiny
.owl#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-
syntax-ns#> .
@prefix xml:
<http://www.w3.org/XML/1998/namespace> .
@prefix xsd:
<http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs:<http://www.w3.org/2000/01/rdf-
schema#> .
_:b0 a rdfs:Resource , rdfs:Class , owl:Class ;
rdfs:subClassOf :Person , _:b1 , _:b0 ,
                    owl:Thing , rdfs:Resource ;
                     owl:equivalentClass _:b0 ;
owl:intersectionOf [a rdfs:Resource, rdf:List ;
                            rdf:first :Person ;
        rdf:rest [ a rdfs:Resource , rdf:List ;
                   rdf:first _:b1 ;
                   rdf:rest ()] ] .
```

The returned RDF file can be used on its own and queried if specific information about that resource is sought. The snippet executing the query is the following:

```
System.out.println(message);
try {qexec.execDescribe().write(System.out,
"TTL");}
finally {qexec.close();}
```

Introducing students to Apache Jena and SPARQL allows them to build their own SW applications utilizing the enormous volume of datasets available on the Linked Open Data Cloud.

5. Conclusion

In this paper, we presented a hands-on project introducing students to Semantic Web technologies and at the same time allowing us to expand and revisit some of the core topics of an undergraduate AI curriculum. Our experience so far suggests enhanced student understanding of the course material, increased engagement, and interest in the course. Overall, students did well on project assignments, however some weakness was noticed when asked to explain experimental results. For example, not all students were able to correctly explain why *hasChild max 4* returns no result, although OWA which causes the problem was discussed in length.

It should be noted that the timeframe allocated for this project did not allow for a thorough review of all project components. Many advanced features of OWL were only demonstrated on family ontology, and complexity results of different DLs (OWL profiles) were not discussed. Also, students were only briefly introduced to Jena API and SPARQL by working mostly with the provided code. In future course offerings, we would like to include an independent project component into the course to encourage students to build their own application utilizing datasets from the Linked Open Data Cloud.

Overall, we believe that this project is a valuable component of our AI course and we plan to further fine-tune it to maximize its pedagogical value.

Acknowledgement. This work was partially supported by the School of Engineering, Science & Technology at CCSU.

References

[1] Schreiber G., Akkermans H., Anjewierden A., de Hoog R., Shadbolt N., de Velde V., Wielinga B. Knowledge Engineering and Management: The CommonKADS Methodology. MA: MIT Press, 2000.

[2] Mizoguchi R. Tutorial on Ontological Engineering: Part 1. *New Generation Computing*, vol. 21, no. 4, pp. 365-384, December 2003.

[3] Newell A. The Knowledge Level (presidential address). *AI Magazine*, vol. 2, 1980.

[4] Newell A. and Simon H. Computer Science as Empirical Inquiry: Symbols and Search. *Communications of the ACM*, vol. 19, no. 3, 1976.

[5] Gruber, T. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, vol. 5, no. 2, pp. 199-220, 1993.

[6] Kalibatiene, D. & Vasilecas, O. Survey on Ontology Languages. In *Lecture Notes in Business Information Processing* 90, pp. 124-141, 2011.

[7] [Online] Available: https://www.w3.org/OWL/

[8] Horridge M. A Practical Guide to Building OWL Ontologies Using Protégé 4 and CO-ODE Tools,n [Online],Available:http://mowl- power.cs.man.ac.uk/ protegeowltutorial/resources/ProtegeOWLTutorialP4 _v1_3.pdf

[9] Musen, M. A., The Protégé project: A look back and a look forward. AI Matters. *ACM Special Interest Group in Artificial Intelligence* vol. 1, no. 4, June 2015.

[10] Sirin E., Parsia B., Cuenca Grau B., Kalyanpur A., Katz Y., Pellet: A Practical OWL-DL Reasoner, [Online]. Available: http://www.cs.ox.ac.uk/people/ bernardo.cuencagrau/publications/PelletDemo.pdf

[11] B. Motic, Shearer B., and Horrocks I., Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, vol. 36, pp. 165 – 228, 2009.

[12] [Online] Available: https://lod-cloud.net/
[13] Zlatareva N., Swaim N., Fitzgerald M., Bagheri-Marani, S., and J. Watkins, Building an Application Agent for the Semantic Web. In *Proceedings of 9th International Multi-Conference on Complexity, Informatics and Cybernetics (IMCIC'18),* March, Orlando, FL, 2018.
[14] Russell S. and P. Norvig, Artificial Intelligence. A Modern Approach, 3-rd edition, Prentice Hall, 2010.
[15] Berners-Lee T., Hendler J., Lassila O. The Semantic Web. *Scientific American*, May 2001.
[16] Berners-Lee, T. The Next Web. TDC 2009 conference speech.[Online].Available:http://www.ted.com/talks/tim_berners_lee_on_the_next_web
[17] Heath T., Linked Data? Web of Data? Semantic Web? WTF? [Online]. Available: http://tomheath.com/blog/2009/03/linked-data-web-of-data-semantic-web-wtf/
[18] [Online].Available: https://www.cambridgesemantics.https://www.cambridgesemantics.com/blog/semantic-university/intro-semantic-web/
[19] Wang, H. "Towards mechanical mathematics", *IBM Journal of Research and Development*, vol.4, 1960.
[20] Stevens R., Stevens M., Matenzoglu N., Jupp S., Manchester Family History Advanced OWL Tutorial, 2015,[Online].Available:http://owl.cs.manchester.ac.uk/publications/talks-and-tutorials/fhkbtutorial/
[21] [Online] Available: http://jena.apache.org/tutorials/
[22] DuCharme, B. Learning SPARQL, O'Reilly Publ., 2013.