OWL Paths: A library for processing SPARQL-like property paths over OWL classes

Melissa D. Clarkson[1] and Landon T. Detwiler[2]

[1] *University of Kentucky, Lexington, Kentucky, USA*
[2] *Bend, Oregon, USA*

Abstract
Ontologies representing knowledge of biomedical domains are often constructed primarily of classes, rather than individuals. Classes are related to one another using OWL property restrictions which represent the complex network of relationships that would hold between individuals. The SPARQL query language is limited in its ability to represent the complexity needed for some types of queries over ontology class structures. We present the OWL Paths library, which uses a grammar syntactically similar to SPARQL and allows the user to write path expressions in relation to OWLClass objects in the OWL API. We demonstrate the need for this library using an example from the Foundational Model of Anatomy (FMA) ontology.

Keywords
Ontology, Ontology pattern, Anatomy, Knowledge representation

1. Introduction

Ontologies are used to represent knowledge about biological and medical domains. These ontologies are built primarily or exclusively of classes (rather than individuals) and often make use of a large number of types of relations (object properties) between classes [1]. Examples of biomedical ontologies can be found at the Open Biological and Biomedical Ontologies (OBO) Foundry website [2]

Our work is in the domain of human anatomy. The Foundational Model of Anatomy (FMA) ontology is a representation of canonical human anatomy developed by the Structural Informatics Group at the University of Washington. The FMA asserts that anatomy can be represented using a series of organizing units at different levels of granularity. These units include “Cell”, “Portion of tissue”, “Organ”, and “Organ system”. The FMA serves as a reference ontology of canonical human anatomy and has been incorporated into the Unified Medical Language System (UMLS). It consists of over 100,000 classes and 130 types of relations between classes—making it one of the largest biomedical ontologies in existence and the most comprehensive ontology for adult human gross anatomy [3,4].

Our recent work with the FMA has focused on developing methods of auditing the FMA to detect incomplete content and inconsistencies in modeling [5,6]. Anatomical structures repeated throughout the body (such as joints, muscles, and bones) should be modeled in similar and predictable ways, but the changes in modeling schemes during the 20-year development of the FMA and inter-author variation have resulted in inconsistencies. Our goal is to remodel the FMA using a pattern-based strategy that will support the next generation of intelligent biomedical applications.

Anatomical representation is a useful case for exploring how patterns can be applied within complex ontologies. The patterns in our work (which we refer to as “knowledge representation patterns”) are specific to the ontology content, rather than being generalizable patterns as those available through OntologyDesignPatterns.org.

Proceedings Acronym: Proceedings Name, Month XX–XX, YYYY, City, Country
mclarkson@uky.edu (M. Clarkson); detwiler1@gmail.com (L. Detwiler)
0000-0001-9979-176X (M. Clarkson); 0000-0003-2699-2338 (L. Detwiler)

The structure of the FMA is determined by both the naturally occurring structure of human anatomy and medical conventions for dividing anatomical structures into parts. The various organ systems of the body (including the musculoskeletal system, nervous system, lymphatic system) are represented as parts that are related to each other through sets of constraints. For example, each lymph node is continuous with at least one lymphatic vessel.

We are working toward representing the anatomical structure of the human body as machine-readable patterns to aid in authoring and curation of the next iteration of the FMA. To accomplish this work, we must query over classes using a method that is analogous to path expressions. In this paper we (1) demonstrate the need to query over ontologies using property path expressions and (2) present our work in developing OWL Paths, a library that enables SPARQL-like processing over ontology classes. We expect this work will be applicable to many large and complex biomedical ontologies.

2. The need for SPARQL-like path expressions over OWL classes in ontologies

We first introduce a fictional example of a medical records ontology to explain the rational for OWL Paths. We then show a more complex example using a representation of human anatomy from the FMA.

2.1. Representing an instance of a medical record and a medical record ontology

Figure 1 shows a diagram of the graph representation of part of a fictional medical record for patient number 001. One of the subsections of “MedicalRecord_001” is “PersonalData_001”, and that section has subsections “DemographicData_001” and “ContactInfo_001”.

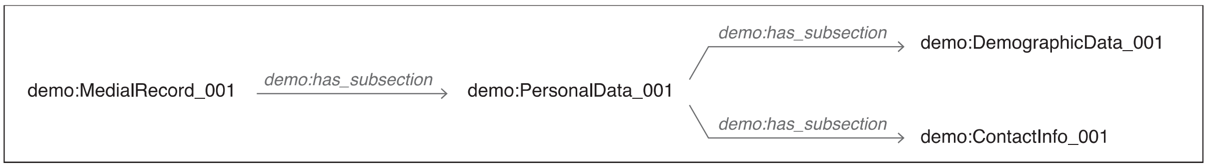


Figure 1: Instance of a fictional medical record

This data structure can be queried by SPARQL. For example, all subsections of the medical record would be retrieved (would bind to ?x) with the following path pattern tuple:

```
    demo:MedicalRecord_001 demo:has_subsection+ ?x
```

Figure 2 shows a diagram of the graph representation of the class structure of the ontology that specifies the structure of medical records in this system. This ontology tells us that each medical record has at least one “PersonalData” section, and each “PersonalData” section has at least one “DemographicData” section and one “ContactInfo” section. Notice that the representation of classes in Figure 2 is more complex than the instance shown in Figure 1 and requires the use of anonymous classes. Suppose we wanted to query the ontology to retrieve all classes that are reachable, recursively, from the class “MedicalRecord” via existential restrictions (owl:someValuesFrom) on the property demo:has_subsection. A SPARQL query can be written only if we know how many levels deep we need to follow the path of demo:has_subsection relations. SPARQL does not have a method of recursively searching a network of this complexity to an arbitrary depth.

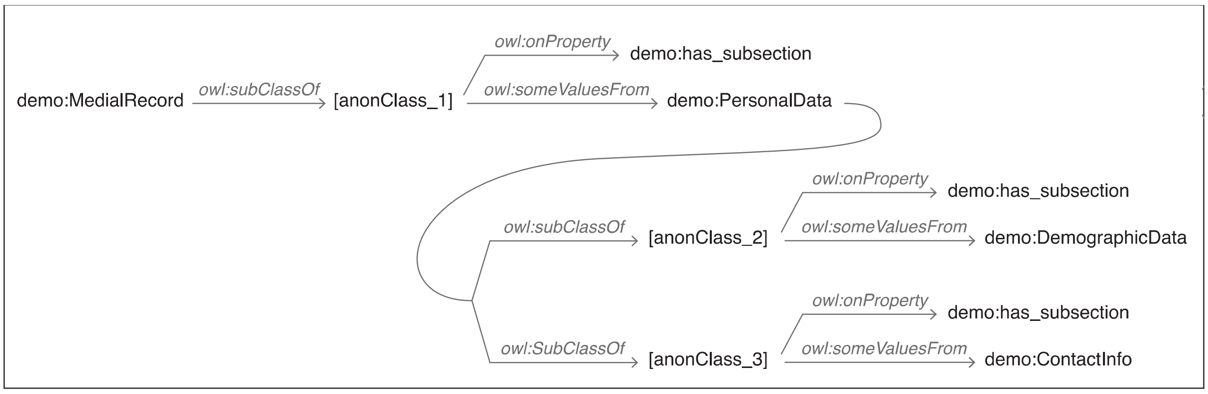


Figure 2: Ontology of a fictional medical record

In practice, this limitation is often overcome by a method called punning, in which an example or model individual is created for each class of an ontology. These individuals can be directly connected via object properties such as demo:has_subsection, thereby simplifying the structure shown in Figure 2 to that in Figure 1. But this is a pre-processing step that that must be done across the entire ontology. The purpose of OWL Paths is to enable SPARQL-like path processing over the classes in an ontology as if the pun individuals were present, without the need for pre-processing.

2.2. Representing human anatomy

Because the FMA represents only classes (not individuals), following paths for any relation other than owl:subClassOf requires following assertions that use anonymous classes. For simplicity, the example in Figure 3 diagrams the relationships between a type of muscle (biceps brachii) and bones (scapula, radius) as an anatomist would conceptualize these paths.

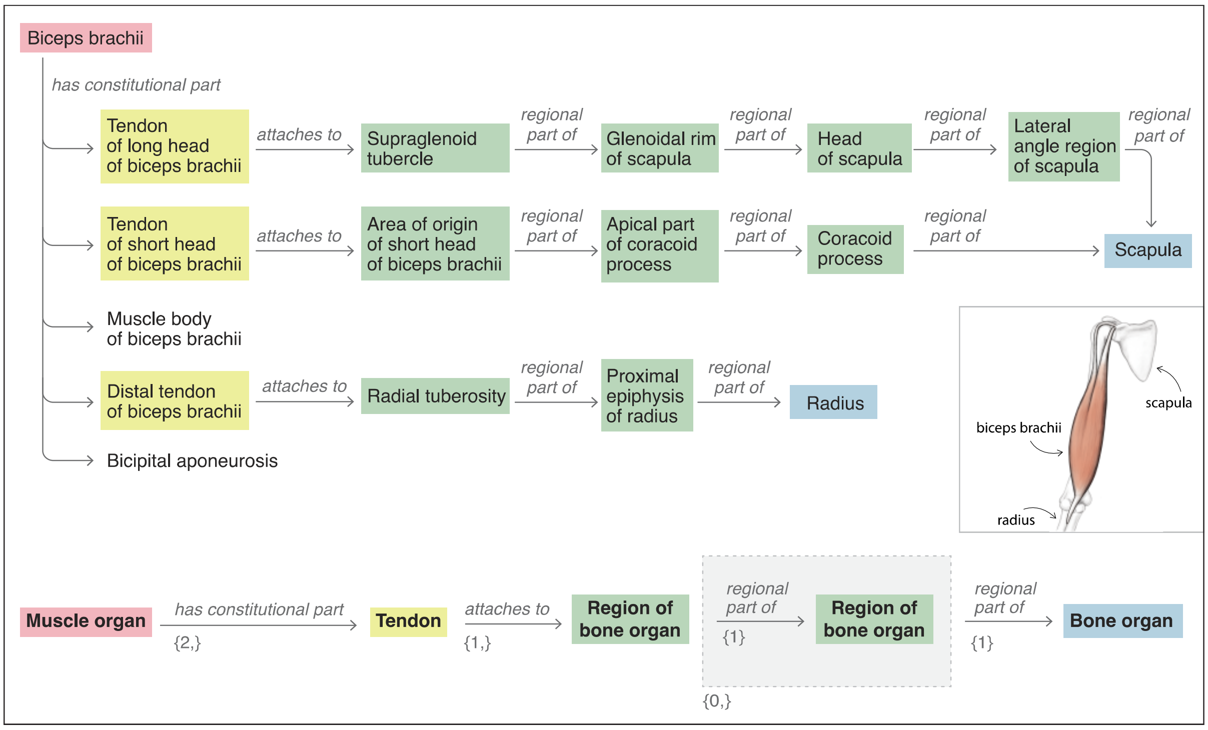


Figure 3: Example of anatomy represented in the FMA as conceptualized by an anatomist, with illustration of anatomy on the right. The generalized pattern is shown at the bottom.

The example in Figure 3 shows that a tendon attaches to a region of a bone, and that region is transitively part of a bone. But the modeling has a variable number of “Region of bone organ” classes traversed between a class “Tendon” and class “Bone organ”. Therefore this path traverses a variable number of regional_part_of relations, and queries that traverse the paths between “Tendon” and “Bone organ” must account for this recursive path. The OWL Paths library was developed to provide the ability to perform this type of complex query.

3. Development of the OWL Paths library

Ontologies are often created or edited in GUI based development tools, such as the Protégé ontology editor and framework [7]. However, when ontologies are used to empower smarter tools there is often a need to consume them via APIs. The OWL API is an API and reference implementation of the OWL 2 standard [8,9]. The OWL Paths tool presented here is a Java library built on the OWL API for processing object property path expressions over OWL classes.

Because most developers in ontology or semantic web fields are familiar with the SPARQL query language for RDF(S), the OWL Paths grammar is syntactically similar to SPARQL. At present, many but not all SPARQL constructs are supported in OWL Paths. OWL Paths also adds two additional expression types that are not a part of SPARQL. Table 1 is adapted from the SPARQL property path specification [10] and compares the expressivity of OWL Paths to that of SPARQL.

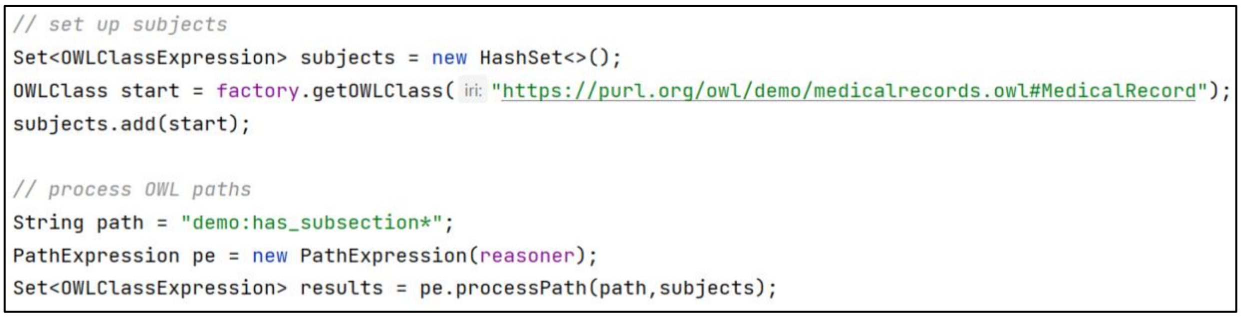
Table 1
Syntax of SPARQL path expressions and OWL Paths

Syntax form	Matches	SPARQL	OWL Paths
`uri`	A URI or a prefixed name. A path of length one.	✓	✓
`^elt`	Inverse path (object to subject).	✓	
`(elt)`	A group path `elt`, brackets control precedence.	✓	✓
`elt1 / elt2`	A sequence path of `elt1`, followed by `elt2`	✓	✓
`elt1 ^ elt2`	Shorthand for `elt1` / ^`elt2`, that is `elt1` followed by the inverse of `elt2`.	✓	
`elt1 | elt2`	A alternative path of `elt1`, or `elt2` (all possibilities are tried).	✓	✓
`elt*`	A path of zero or more occurrences of `elt`.	✓	✓
`elt+`	A path of one or more occurrences of `elt`.	✓	✓
`elt?`	A path of zero or one `elt`.	✓	
`elt{n,m}`	A path between n and m occurrences of `elt`.	✓	
`elt{n}`	Exactly n occurrences of `elt`. A fixed length path.	✓	
`elt{n,}`	n or more occurrences of `elt`.	✓	
`elt{,n}`	Between 0 and n occurrences of `elt`.	✓	
`uri[INV=uri]`	A path constraint that the inverse path must also hold.		✓
`elt[SUP=uri]`	A path constraint on the superclass of all classes on the path.		✓

`uri` is either a URI or a prefixed name and `elt` is a path element, which may itself be composed of path syntax constructs.

The OWL Paths library supports programmatically processing path expressions in relation to OWLClass objects in the OWL API. The OWL Paths library adds a Java class PathExpression which has exactly one relevant method, processPath. processPath accepts two arguments: (1) a path expression which describes what sort of paths to follow, and (2) a list of subject classes to follow such paths from. Paths are followed, in OWL Paths, as though traversing a non-materialized network of punned (model or example) individuals. Presently these non-materialized puns are presumed to exist everywhere for which there is an existential restriction (owl:someValuesFrom) on an object property. Therefore, the owl:someValueFrom restrictions in Figure 2 are processed as if the graph in Figure 1 had already been generated.

The code snippet in Figure 4 would return four OWLClass objects in the results, "demo:MedicalRecord", "demo:PersonalData", "demo:DemographicData", and "demo:ContactInfo". That is because, in the demo ontology, the class "demo:MedicalRecord" is connected via restriction on the property demo:has_subsection to each of those classes recursively (including itself as the '*' operator indicates zero or more).



```
// set up subjects
Set<OWLClassExpression> subjects = new HashSet<>();
OWLClass start = factory.getOWLClass( iri: "https://purl.org/owl/demo/medicalrecords.owl#MedicalRecord");
subjects.add(start);

// process OWL paths
String path = "demo:has_subsection*";
PathExpression pe = new PathExpression(reasoner);
Set<OWLClassExpression> results = pe.processPath(path,subjects);
```

Figure 4: Example of calling processPath.

Like paths in SPARQL, path expressions in OWL Paths are compositional. Encapsulating a path expression in parenthesis results in a new path element, to which further operators can be applied. For example, consider the following path expression:

```
demo:has_subsection|demo:has_coding_system
```

The above path expression matches paths that are either composed of a single demo:has_subsection property or a single demo:has_coding_system property. If we were to encapsulate the above expression in parenthesis, we could then apply additional operators, for example the unary operator '+'. It would then match paths with one or more edges all of which are either demo:has_subsection or demo:has_coding_system. That expression would look like this:

```
(demo:has_subsection|demo:has_coding_system)+
```

The source code for OWL Paths, as well as a pre-build archive, are available at https://gitlab.com/endless-forms-studio/owl_paths.

3.1. Extensions

Although we have compared OWL Paths to SPARQL, it is important to note that our path expressions need not be limited to the expressivity of SPARQL. In this section we describe two (beta) extensions to the path language that are not available in SPARQL property paths but have been implemented in OWL Paths.

For property paths in SPARQL the types of the intervening nodes along the path are not considered, but when querying ontologies this information can be relevant. In SPARQL, demo:has_subsection* specifies that matching paths have zero or more such edges but it does not specify anything else about the nodes that are connected via those edges. In OWL Paths, one extension provides the ability to specify the type for nodes along a path, restricting to only nodes with a given superclass. The syntax for this constraint uses the suffix [SUP=<IRI>]. For example, the following expression would return results connected to the subject by a restriction on demo:has_contact_record only if the superclass of the connected class is "demo:PhoneRecord":

```
demo:has_contact_record[SUP=demo:PhoneRecord]
```

The other extension in OWL paths has a similar syntax, but it restricts results to those that have an additional property restriction in the other direction, from the target to the source:

```
demo:has_contact_record[INV=demo:contact_record_of]
```

4. Conclusion

This work introduces OWL Paths, a Java library built on the OWL API for processing object property path expressions over OWL classes. This was developed in response to our need to perform queries over the FMA to advance our work implementing patterns. We anticipate that this library will be of use to additional groups authoring and auditing other large ontologies that model complex domains.

References

[1] B. Smith, W. Ceusters, B. Klagges, et al. Relations in biomedical ontologies. Genome Biology 66 (2005) R46. doi: 10.1186/gb-2005-6-5-r46.

[2] Open Biological and Biomedical Ontology Foundry. URL: https://obofoundry.org/.

[3] C. Rosse, J.L.V. Mejino, A reference ontology for biomedical informatics: The Foundational Model of Anatomy. Journal of Biomedical Informatics 6 (2003) 478–500. doi: 10.1016/j.jbi.2003.11.007.

[4] C. Rosse, J.L.V. Mejino, The Foundational Model of Anatomy ontology, in: A.G. Burger, D. Davidson, R. Baldock (Eds.), Anatomy Ontologies for Bioinformatics: Principles and practice, Springer, London, 2008, pp. 59–117. doi: 10.1007/978-1-84628-885-2_4

[5] M.D. Clarkson, S. Roggenkamp, Employing knowledge patterns for auditing the Foundational Model of Anatomy, in: Proceedings of the 2nd International Workshop on Quality Assurance and Enrichment of Biological and Biomedical Ontologies and Terminologies at the 2019 IEEE International Conference on Bioinformatics and Biomedicine (IEEE BIBM 2019), SanDiego, CA, November 18–19, 2019. doi: 10.1109/BIBM47256.2019.8983410.

[6] M.D. Clarkson, L.T. Detwiler, K.M. Platt, S. Roggenkamp, Assessing the consistency of modeling in complex ontologies: A study of the musculoskeletal system of the Foundational Model of Anatomy, in: Proceedings of the 2021 International Conference on Biomedical Ontologies (ICBO 2021), Bozen-Bolzano, Italy, September 16–18, 2021. URL: https://ceur-ws.org/Vol-3073/paper2.pdf.

[7] M.A. Musen, The Protégé project: A look back and a look forward. AI Matters (1) 2015: 4–12 doi: 10.1145/2757001.2757003.

[8] OWL API main repository. URL: https://github.com/owlcs/owlapi.

[9] OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition), W3C Recommendation 11 December 2012. URL: https://www.w3.org/TR/owl2-syntax/

[10] SPARQL 1.1 Query Language, W3C Recommendation 21 March 2013. Section 9, Property Paths. URL: https://www.w3.org/TR/sparql11-query/#propertypaths