Creating Annotations for Web Ontology Language Ontology Generated from Relational Databases

Matthew Wagner and Dalia Varanka(✉)

U.S. Geological Survey, Rolla, MO 65401, USA
matthewwagner57@gmail.com, dvaranka@usgs.gov

Abstract. Many approaches that have been proposed that allow users to create a Web Ontology Language (OWL) ontology from a relational database fail to include metadata that are inherent to the database tables. Without metadata, the resulting ontology lacks annotation properties. These properties are key when performing ontology alignment. This paper proposes a method to include relevant metadata through annotation properties to OWL ontologies, which furthers the ability to integrate and use data from multiple unique ontologies. The described method is applied to geospatial data collected from The National Map, a data source hosted by the U. S. Geological Survey. Following that method, an ontology was manually created that used the metadata from The National Map. Because a manual approach is prone to human error, an automated approach to storing and converting metadata into annotation properties is discussed.

Keywords: Semantic Web · OWL ontology · Geographic information systems (GIS) · Geospatial data · Metadata

1 Introduction

The next generation of the World Wide Web, named the Semantic Web, focuses on increasing the accessibility and readability of data for machines. The Semantic Web uses concepts and data standards, such as Resource Description Framework (RDF) and Web Ontology Language (OWL) to accomplish that goal. By following Semantic Web guidelines and practices, data from multiple different sources can be accessed, leveraged, and integrated for any application. The research documented in this paper explores the process of adding annotation properties to an OWL ontology generated from a preexisting relational database.

The U.S. Geological Survey (USGS) publishes free geospatial data that cover the United States. Data are published in layers, each referring to a specific geospatial feature set. These data layers include structures, transportation, hydrography, geographic names, boundaries, elevation, land cover, and orthographic images. Whereas the USGS generates some data, a significant portion is collected from third-party sources including individual States, the U.S. Census Bureau, and other organizations. The USGS collects these data, standardizes them, and publishes the data as a single dataset. This process results in a major challenge since the data are not USGS created. In some cases, constraints are imposed by the standards of the collecting agencies. Thus, the USGS acts as a data steward and publisher for data acquired from a variety of different sources.

The data published by the USGS provide an opportunity to bring a large-scale data source to the Semantic Web. Multiple other data sources such as Wikidata, DBPedia, and Geonames.org have brought geospatial data to the Semantic Web [3,5,24]; however, this work examines a process to align data with Semantic Web standards, unlike other data sources in which data are directly in the RDF format.

Various research projects have proposed techniques and workflows that generate an OWL ontology and convert data contained in a relational database into RDF. A major problem with past approaches is the lack of useful annotation properties in an OWL ontology. An annotation property is a property that has a data literal, URI reference, or an individual as the object [15]. These properties fully define entities in an ontology such as classes, properties, datatypes, and even the ontology itself. By adding information to these entities, users can determine the relationships among them. Five different annotation properties are predefined by OWL: owl:versionInfo, rdfs:label, rdfs:comment, rdfs:seeAlso, and rdfs:isDefinedBy. This basic information allows classes, properties, and instances to be compared and contrasted resulting in ontology alignment. Ontology alignment is an important feature of OWL ontologies since they are rarely uniform. The Semantic Web encourages users to build ontologies to fit their needs, resulting in the lack of a complete standard for ontologies to exist. Unique classes, properties, and instances from different ontologies must be linked if their data are going to be used together. Annotation properties provide one method for doing so.

Since the Semantic Web is designed with machines in mind, this problem needs to be examined from a machine perspective. A machine would only be able to use the data at hand to match instances. Common data properties for geospatial data include human readable names and coordinate information. Both Wikidata and DBpedia use the widely accepted rdfs:label[1] datatype property to describe entities. However, the GeoNames ontology uses the geonames:name[2] annotation property. The issue for a machine is not comparing the data from those two properties, but rather knowing which two properties to compare. Since the same attribute exists in both Wikidata and DBpedia, a machine would have no problem finding similarities on an instance level for these two data stores. With the introduction of the GeoNames' ontology, a problem arises. For datatype properties, the machine could use annotation properties to draw a comparison between two or more instances. Common properties such as rdfs:comment, rdfs:range, and rdfs:seeAlso can be used to look for similarities. However, the

[1] The rdfs namespace is http://www.w3.org/2000/01/rdf-schema#.
[2] The geonames namespace is http://www.geonames.org/ontology#.

geonames:name property only possesses the rdfs:domain field, something entirely dependent on the ontology of Geonames. Thus, a machine would be unable to make any real comparison this way and would be forced to compare the instances of each datatype property to align any instances.

A similar situation occurs if the machine examines the description of an object in the form of a geometry. These geometries take the form of a point, polyline, and polygon. In all three data stores, geometries are represented with coordinate points of latitude and longitude. GeoNames and DBpedia uses geo:lat and geo:long[3] whereas Wikidata uses custom datatype properties wikibase:geoLatitude[4] and wikibase:geoLongitude. The geo:lat and geo:long datatype properties have annotations such as rdfs:domain, rdfs:label, and rdfs:comment. The wikibase:geoLatitude and wikibase:geoLongitude have an rdfs:label, rdfs:comment, rdfs:domain, and rdfs:range. Plain text comparison of both sets of rdfs:label and rdfs:comment would be able to provide some links for a machine. However, the lack of an rdfs:range annotation for the geo namespace becomes a hurdle since coordinates can be represented in a variety of formats. The machine would have to again rely on comparing the instances to affirm the contents rather than being able to draw this link from the ontology alone.

A machine needs to accurately analyze datatype and object properties to align different ontologies. Whereas some annotation properties are present, many ontologies lack complete descriptions. Many authors have previously proposed methods for converting relational databases to OWL ontologies. However, a method for converting metadata to annotation properties and adding these to OWL ontology has not been proposed. Without annotation properties to fully define entities within an OWL ontology, Semantic Web data sources will struggle to be used in conjunction with other data sources.

Section 2 of this paper presents background information on the research area being presented. Section 3 discusses past proposed solutions to converting relational database to Semantic Web formats and cases that are inadequate. In Sect. 4, the approach used to manually create an OWL ontology for USGS data is presented. Section 5 presents the resulting ontology created after applying the proposed approach. Challenges faced and the verification of the ontology are also discussed. Section 6 discusses the proposed approach and potential automated conversion solutions. Lastly, in Sect. 7, conclusions from this work are presented.

2 Background

An OWL ontology is a formal method to describe the taxonomy and relationships that exist within data. It consists of classes, properties, and datatypes, and instances that are comparable to the table and key structure for a relational database and instance data that are comparable to individual records. When converting a relational database to an OWL ontology, the structure of the

[3] The geo namespace is http://www.w3.org/2003/01/geo/wgs84_pos#.

[4] The wikibase namespace is http://wikiba.se/ontology#.

database is stored in an OWL ontology whereas the data records are converted into instances.

Classes are an abstract method for grouping instances with similar characteristics. This grouping is the same as a table in a relational database. All of the records in the table have similar attributes, with some characteristics being common among them. Labeling an instance with a class in an OWL ontology allows that instance to inherit the attributes and additional properties of that class. In a relational database, the records in a table are constrained by any overriding rules of that table. This results in those records inheriting the rules of the table.

Properties for OWL ontologies consist of three types: object properties, data properties, and annotation properties. Object properties define relationships between objects, each with a unique Universal Resource Identifier (URI). These properties are comparable to foreign keys in a relational database. For example, geo:hasGeometry is an object property that connects a geographic feature (the subject) to an object describing its coordinate representation (the object). Data properties define relationships between objects and literals. Literal data, which are classified using datatypes, are the plain text data that connect additional information to an object. In a relational database, literal data are the data that are not foreign keys attached to an individual record. A popular example of a data property is geo:asWKT, which links a geometry object (the subject) to a geo:wktLiteral representation of the geometry (the object). Annotation properties are akin to the metadata within a relational database. They describe the restrictions placed on attributes and allow users to understand the contents of that attribute. For example, rdfs:label links a URI (the object) to a human-readable label describing the entity (the subject).

Instance level semantics are the overwhelming majority of links between ontologies [10]. They generally take the form of object properties, such as owl:sameAs. The owl:sameAs relationship indicates that the two instances it links refer to the same thing. Whereas the owl:sameAs relationship is an effective method for drawing connections between instance data, two issues currently exist. First, these relationships must be determined ahead of time and the triples must be stored. Typically, this process is done manually by experts with in-depth knowledge on those ontologies. Secondly, some authors argue that owl:sameAs is often inaccurate when comparing instances [8].

Ontology alignment examines the different ontologies and determines relationships between classes and properties. With these relationships present, direct comparisons of the instance-level data can be performed using string comparisons to determine instance-level relationships. Properties used in ontology alignment can bring in relevant information without needing to state the exact relationship between the instance. This on-the-fly process can avoid the argument about the semantics of such relationships and remove the need to manually link ontologies. However, to perform this task, annotation properties must be defined and used to provide enough depth of information for different ontologies to be aligned successfully. With this background information, previous research in this area is discussed in the next section.

3 Related Work

Significant research has examined approaches to convert a relational database to the Semantic Web standards. Proposed methods to convert a relational database to an OWL ontology include [1,9,12,18]. Each of these references suggests slight variations to achieve similar results. Li et al. [12] proposed a set of 12 rules that were grouped into five categories: rules for learning classes, rules for learning properties and property characteristics with rules for learning hierarchy, cardinality, and instances.

Similarly, Sequeda et al. [18] present a solution to mapping relational databases to RDF that focuses on maintaining information preservation and query preservation during their conversion process. Information preservation refers to the ability to recreate the relational database from the OWL ontology after the mapping process has occurred. Query preservation is the notion that every query that can be performed on the relational database can be translated into an equivalent query on the data post mapping to OWL.

Hu et al. [9] propose an approach to discovering simple mappings between a relational database schema and an ontology. The authors construct a special type of semantic mapping called a contextual mapping, which holds the subsumption relationships existing within the relational database.

The most intuitive of these approaches is one proposed by Astrova et al. [1] for automatic transformation of relational databases to ontologies in which the quality of the transformation is also considered. They created a set of rules for mapping tables, columns, datatypes, constraints, and rows.

The major drawback of these previous approaches is that annotation properties are not discussed. Without annotation properties, the OWL ontologies created by the proposed approaches suffer from the drawbacks discussed in Sect. 1 and Sect. 2.

Other work examined the inverse relationship; converting an OWL ontology to a relational schema. Gali et al. [4] present a set of techniques to provide a lossless mapping of an OWL ontology to a relational schema and the corresponding instances to data. They presented a set of mapping rules for converting an OWL ontology to a relational schema. Similar to the approaches that developed mappings from a relational database to an OWL ontology, the work does not discuss annotation properties.

4 Approach

The Protege tool was used to manually create an OWL ontology for USGS The National Map (TNM) data [14]. Data dictionaries describing the metadata properties of different layers were referenced to build the OWL ontology. These references are not downloaded or incorporated with the datastores. Instead, the USGS is in the process of building a specifications library, SpecX, to store detailed data dictionaries about the USGS products [23]. The Domestic Geographic Names layer was added to the ontology even though its data dictionaries are not currently available in SpecX. Instead, the file format for the data can be accessed online [22].

Data from the USGS TNM is downloaded in datasets that consist of multiple layers grouped. For example, the Transportation dataset contains layers including trails, roads, railways, and airways. In each dataset, each layer is stored as a table. Since all data are downloaded from TNM, a class called topo:TNM[5] was created with two subclasses: topo:Attribute and topo:Feature. Topo:Attribute refers to data belonging to static tables in the database. These include tables containing feature names, feature codes, resolution types, and more. This is separate from the topo:Feature class since the data are referenced across different datasets and layers. By making these static tables a set of static classes, redundant information regarding each instance of these classes can be removed and replaced with object properties referring to these classes, reducing data storage requirements.

Topo:Feature refers to the dynamic tables in the database that contain instance data. The topo:Feature was made equivalent to the geosparql:Feature class. GeoSPARQL was used to leverage preexisting query capabilities that exceed SPARQL, the current standard for performing queries on Semantic Web compliant data [16]. GeoSPARQL is the Open Geospatial Consortium standard for representing and querying geospatial data in the Semantic Web. It adds additional query capabilities that allow users to perform comparisons between geometries. This added equivalence relationship allows the ontology created in this work to leverage the feature geometry relationship and the set of geometry properties, including geosparql:asWKT and geosparql:asGML. This relationship removed the need to create custom definitions of the attributes. It also increased the set of operations that can be performed on the ontology by users.

4.1 Creating Class

The following rules were used to generate classes for the OWL ontology.

– Each database will have its own class. The database's class will be a subclass of topo:Feature.
– Each dynamic table in the database will have its own class. This class will be a subclass of the class for its database.
– Each class referring to a dynamic table belonging to the same database is disjoint from all other classes referring to a dynamic table in the same database.
– Each static table will have its own class. This class will be a subclass of topo:Attribute. Static tables shared across multiple databases shall not be duplicated.

[5] TNM is a spatial data infrastructure for topographic data. The namespace http://data.usgs.gov/lod/topo/ was used for the ontology describing TNM data.

- Each class representing a static table will be disjoint from all other subclasses of topo:Attribute.
- Each record in a static table will have its own class. This class will be a subclass of the class for its table.
- Each record in a static table will be disjoint from all other classes resulting from records in the same static table.

The tables in the Structures data dictionary is split into four categories: Feature Classes, NonSpatial Tables, Feature Code (FCode) Domains, and Non-FCode Domains. The Feature Class tables are dynamic and contain instance data. The NonSpatial Tables contain data that refer to the internal workflows used to publish the data. They are purposely excluded from this work since that information, whereas important, cannot be linked easily to other ontologies. Both the FCode and NonFCode categories refer to sets of static tables. FCode tables are used to explicitly label geographic features.

4.2 Creating Object Properties

The following rules are implemented to generate object properties. The major difference between an object property and a data property is the link to another class. In TNM data, a large volume of data are located in a static table. Thus, object properties are leveraged to decrease the number of instances of that information, overall decreasing the space required to store all the data.

- If an attribute in a table links to another table, that is, it is a foreign key, then it is an object property. In SpecX, this can be seen as the existence of a value in the Domain field. The FCode and FType fields are also considered object properties regardless of the lack of data in the domain field.
- The class representing the table that contains the foreign key becomes the object property's domain.
- The class representing the table to which the foreign key links becomes the object property's range.
- If the foreign key exists in multiple tables, the domain of the object property is the union of those tables.
- If the foreign key has a one-to-one relationship, the object property is a functional property.

The rdfs:subPropertyOf relationship was leveraged to reduce the size of the ontology.

4.3 Creating Data Properties

The following rules are implemented to generate data properties. These properties contain the bulk of the data published by the USGS TNM.

- If an attribute does not link to another table, then it is a data property. In the SpecX database, this means that the domain field is blank.

- The table that contains the attribute becomes the data property's domain.
- The data type of the attribute becomes the data property's range.
- Any unique datatypes that are not defined in geosparql, xsd, rdf, or rdfs become a unique data type.
- If the attribute exists in multiple tables, the domain of the data property is the union of those tables.
- If the attribute has a one-to-one relationship, the data property is a functional property.

Only one unique datatype, topo:objectID, was created for the TNM ontology. The rest of the datatypes were mapped to preexisting datatypes in the rdfs ontology.

4.4 Creating Annotation

While changes were added to previous approaches to create classes, object properties, and data properties, creating annotation properties is an entirely new aspect.

- Any description of the table located in the data dictionary becomes an rdfs:comment for the class representing that table.
- Any metadata attribute located in the data dictionary description become an annotation property.
- The description of the metadata attribute becomes an rdfs:comment of the annotation property.
- The datatype for the metadata attribute becomes the range of that annotation property.
- Any metadata attribute located in the data dictionary description for an attribute becomes an annotation property for that class, object property, or data property.
- If attributes contain different metadata, they must be considered different, unique properties and labeled accordingly.

The results of applying these rules can be seen in the example RDF/XML shown below generated by Protege from applying the rules to create annotation properties.

```
<owl:AnnotationProperty rdf:about="http://data.usgs.gov/lod/topo/description">
    <rdfs:comment>Description of a property.</rdfs:comment>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:AnnotationProperty>
<owl:Class rdf:about="http://data.usgs.gov/lod/topo/Structure/Struct_Point">
    <rdfs:subClassOf rdf:resource="http://data.usgs.gov/lod/topo/Structure"/>
    <rdfs:comment>A feature class representing the location of a building or
        other structure as a point.</rdfs:comment>
</owl:Class>
<rdf:Description rdf:about="http://data.usgs.gov/lod/topo/objectID">
    <comments></comments>
    <default_Value></default_Value>
    <definition>Internal feature or event number.</definition>
    <precision></precision>
    <allows_Nulls rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">
        false</allows_Nulls>
    <length></length>
</rdf:Description>
```

4.5 Creating Geometries

Geometries are unique geospatial data. When coupled with GeoSPARQL, geometries allow users to find unique relationships among instances that otherwise would not exist.

- The geometry field for all tables is not converted to a data property or object property. Instead, it is converted to a geosparql:Geometry.

Since this paper does not cover converting instance data in depth, example geometry data are not included.

4.6 URI Naming Conventions

Due to the size of the datasets published in the USGS TNM, a formal naming convention was created. For classes representing a dataset or attribute table, the name of the table was appended to the namespace for the TNM ontology. For the classes representing dynamic tables, the name of the table was appended to the URI of the class representing the dataset of which it is a part. Similarly, for the content of the static table, the name of the record is appended to the class representing the table in which it is contained. For object properties, data properties, annotation properties, and datatype properties the name of the attribute was appended to the URI for the namespace. The only exception to the rule is properties incorporated from the Geographic Names dataset. Instead, those were appended to the URI for the class representing the Geographic Names dataset. The conventions can be seen in Table 1.

Table 1. URI naming conventions for TNM ontology

Entity	Convention
Dataset classes Static table classes Object properties Data properties Annotation properties Datatype properties	Topo namespace + **Entity name**
Dynamic table classes Static tables contents	Topo namespace + **Dataset name** + “/” + **Entity name**
All geographic name entities	topo namespace + “GNIS/” + **Entity name**

5 Results

The resulting metrics produced by the Protege tool can be seen in Table 2 through Table 6. Protege produces five different metric tables for various parts

Table 2. Protege ontology metrics

Metric	Count
Axiom	6682
Logical axioms	2135
Declaration axioms	1241
Classes	909
Object properties	87
Data properties	225
Annotation properties	33

Table 4. Protege ontology object property axioms

Metric	Count
SubObjectPropertyOf	59
EquivalentObjectProperties	0
InverseObjectProperties	4
DisjointObjectProperties	0
FunctionalObjectProperty	43
InverseFunctionalObjectProperty	0
TransitiveObjectProperty	3
SymmetricObjectProperty	4
AsymmetricObjectProperty	0
ReflexiveObjectProperty	0
IrrefexiveObjectProperty	0
ObjectPropertyDomain	74
ObjectPropertyRange	70
SubProprtyChainOf	0

Table 3. Annotation axioms

Metric	Count
AnnotationAssertion	3288
AnnotationPropertyDomain https://www.overleaf.com/project/5f85d900cbe2ca0001b11f7b	0
AnnotationPropertyRagneOf	9

of the ontology including general metrics, class metrics, object property metrics, data property metrics, and annotation metrics. In looking at these results, it is important to note that some entities are added automatically to an ontology created using Protege. Additionally, the GeoSPARQL ontology was imported into the ontology as a result of the approach discussed in Sect. 4. Thus, some of the relationships not mentioned in the approach were inherent within that ontology.

Table 4 and Table 6 both show interesting results in terms of the Disjoint-ObjectProperties and the DisjointDataProperties. Both tables show that none of those relationships were generated during ontology creation. In several of the data layers that were incorporated into the system, there were multiple places where multiple fields for a single entity related to a static type with multiple object properties. A primary example of this phenomenon is the trails layer in the transportation dataset. In this trails layer table, a majority of the attributes tell a user whether a certain mode of transportation is allowed on a trail. All of the fields then link to the TrailYesNoDomain. Thus, these relations break the formal definition of DisjointObjectProperty [7]. For the DisjointDataProperty, there is no way to determine if one data instance would be linked to an entity with multiple relationships. This is especially true if multiple data objects share a common DataPropertyRange. Thus, no DisjointDataProperties were added to the system (Tables 3 and 5).

Table 5. Protege ontology class axioms

Metric	Count
SubClassOf Properties	921
Equivalent Classes	1
Disjoint Classes	62
GCI Count	0
Hidden GCI Count	2

Table 6. Data property axioms

Metric	Count
SubDataPropertyOf	219
EquivalentDataProperties	0
DisjointDataProperties	0
FunctionalDataProperty	211
DataPropertyDomain	229
DatatPropertyRange	235

5.1 Challenges

When manually creating this ontology, two major challenges were faced. The first challenge to create an OWL ontology from preexisting data is the reuse of attribute labels. Across all datasets published by the USGS, attribute names are shared whereas their meaning and attached metadata are changed. One primary example is the use of the attribute "Name". Working exclusively with the five layers mentioned earlier, twelve different sets of metadata were associated with this attribute. These attributes have a variety of different definitions, field lengths, and some nullable values. Whereas some may consider generalizing the attribute metadata and associating all of these with the same data property in an OWL ontology, that would be a mistake. Each attribute fundamentally has a different definition and could be referring to semantically different things. Manually expanding the ontology, careful attention was paid to ensure all unique pairings of attribute names and metadata were found. A unique entity was created for each pairing. However, this resulted in the second major challenge—naming conventions.

Whereas all of these unique pairings have the same attribute name, each pairing must result in a unique URI. This resulted in the creation of a custom naming convention. For naming these attributes, a combination of the namespace and the metadata difference was used. A shorthand for the table containing the unique pairing was appended if the only major difference between pairings is the description metadata field. Otherwise, the different metadata attribute was appended to the original attribute.

5.2 Validation

The plugin OntoDebug was used to validate the formal logic contained within the OWL ontology [17]. This plugin uses the preexisting reasoners to determine whether the ontology is coherent and consistent. For an ontology to be coherent, all classes must be satisfiable. For an ontology to be consistent, one of its classes must be satisfiable.

To validate the logic contained within the ontology, three different reasoners were used: Pellet, ELK, and HermiT.

Pellet is a popular reasoner created to support OWL Description Logic (OWL-DL) [19]. It is implemented in Java and is free to download as a plugin for the Protege tool. Version 2.2.0 was used within this work. When OntoDebug is run with the Pellet reasoner on the created ontology, it is both coherent and consistent.

ELK is another reasoner available for use through the Protege tool [11]. Version 0.4.3 was used in this work. Although efficient, it does not support a significant number of axioms including DataPropertyRange, ObjectPropertyRange, FunctionalDataProperty, SubDataPropertyOf, and more. With the axioms it does support, ELK produces coherent and consistent results when run on the generated ontology.

HermiT version 1.4.3 is the reasoner shipped with the installation of Protege [6]. This was the only reasoner out of the three that produced an error. However, this error was the result of a restriction on the set of acceptable datatypes allowed in the reasoner. The reasoner only allows datatypes of the OWL 2 datatype map. The xsd:date and topo:objectID datatypes both resulted in errors since they are not present in the standard. A test version of the ontology without either of these datatypes was created and validated using this reasoner. HermiT produced a coherent and consistent validation result.

6 Discussion

The only requirement for applying the manual approach proposed in this work is the existence of metadata information for the database. For the USGS, these are currently presented in the SpecX database and previously described using large data model posters [21]. The proposed rules could be used to create an OWL ontology for all datasets from the USGS. Furthermore, it could be extended to any database outside of the USGS as long as metadata exists. Without the metadata, the created ontology would match previously proposed approaches.

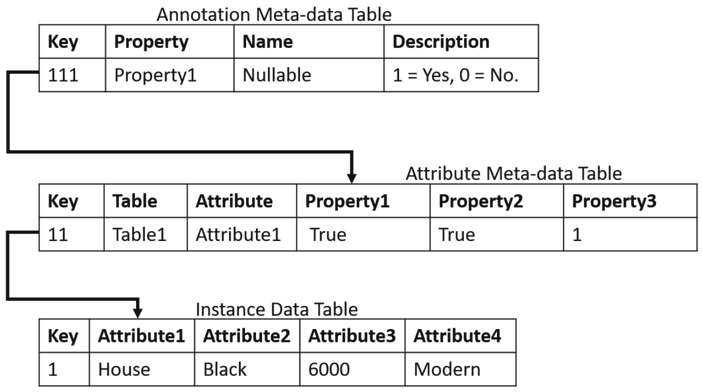
Applying the approach manually is time consuming. The human error involved in the process required multiple rounds of revisions to ensure that all the proposed rules were followed correctly for the example OWL ontology created. To overcome this challenge, an automated approach to generating the metadata needs to be created. A significant amount of research has been done that proposed automated approaches for generating OWL ontologies from relational databases [2,13,20,25].

Cullot et al. presented a tool called DB2OWL that automatically generates ontologies from database schemas [2]. They break the mapping process into six steps that cover mapping classes, subclasses, object properties, and datatype properties. Additionally, they include how additional relationships such as inverse, domain, and range are determined. In [20], Trinh et al. propose RDB2ONT, a tool that describes a formal algorithm to use relational database metadata and structural constraints to construct an OWL ontology preserving the structural constraints of the underlying relational database system. A tool called Ontology Automatic Generation System based on Relational Database (OGSRD) was proposed in [25] as a method for automatic ontology building using the relational database resources. Lastly, Mogotlane & Fonou-Dombeu

[13] performed a study applying two different tools, DataMaster and OntoBase, that automatically construct ontologies from a relational database. Definitions of a class, object properties, datatype property, and instances are given. Nine different rules are drawn from previous work to map a relational database to an OWL ontology.

All of these works present methods to correctly create OWL ontologies that avoid the time-consuming and error-prone process of manually creating them. However, all of the reviewed works fail to discuss the implementation of annotation properties. One possible reason for this is the lack of metadata for the databases used. The existence of metadata describing the data model is often known and maintained by the data provider; however, these metadata generally are not attached to the database. Whereas this does not present a challenge for manual approaches, it does for automated approaches. The previous approaches look exclusively at the relational database and its structure. Requiring the process to examine a website, such as SpecX, and get the requisite information provides its own challenges.

One potential solution to overcoming this challenge is to create metadata tables as part of a relational database. These tables would contain all metadata information for the classes, data property, object properties, and even annotation properties. Rather than creating a new approach, this could leverage the technology already in use in these relational database systems. Additionally, this extension could be added to previously proposed automated ontology creation approaches. Example metadata tables are shown in Fig. 1.


Annotation Meta-data Table

Key	Property	Name	Description
111	Property1	Nullable	1 = Yes, 0 = No.

Attribute Meta-data Table

Key	Table	Attribute	Property1	Property2	Property3
11	Table1	Attribute1	True	True	1

Instance Data Table

Key	Attribute1	Attribute2	Attribute3	Attribute4
1	House	Black	6000	Modern

Fig. 1. Example automated metadata database

In Fig. 1, there are three tables: instance data, attribute metadata, and annotation metadata. Attributes refer to the columns in the instance data table and are converted to object and data properties based on the approach used in this work. Annotations refer to the annotation properties used to describe the attribute metadata. This second level of annotations is important to describe custom metadata used in a system. Many of the properties for the USGS TNM datasets can be seen in the SpecX database including name, definition, type, allow nulls, domain, default value, comments, and many more. Whereas some of these annotations may seem intuitive, fields such as precision and length which are generated by third-party software are not. Thus, exact definitions need to be provided to allow for accurate comparisons of classes and properties between ontologies.

7 Conclusion

This work highlights the need for annotation properties to properly align and use instance data from different ontologies. Additional rules to create an OWL ontology directly from a relational database that addresses the need for these properties is proposed. An ontology was manually created using the Protege tool to show the results of implementing the proposed approach. It incorporated multiple different datasets and data layers produced by the USGS TNM. The ontology excluded instance data due to the size of the datasets. Furthermore, the OntoDebug plugin was used to validate the formal logic present within the OWL ontology. The results from three different reasoners prove that the formal logic in the results of the proposed approach is coherent and consistent.

An automated solution for storing metadata attributes within the database with the instance data was introduced, which addresses the inherent issues of a manual approach. If metadata attributes are stored within the database, they could be converted to annotation properties. This approach could serve as an extension to previously created solutions instead of requiring new solutions to be generated. However, the actual creation of this automated tool is left to future research.

Any use of trade, firm, or product names is for descriptive purposes only and does not imply endorsement by the U.S. Government.

References

1. Astrova, I.: Rules for mapping SQL relational databases to OWL ontologies. In: Sicilia, M.A., Lytras, M.D. (eds.) Metadata and Semantics, pp. 415–424. Springer, Boston (2007). https://doi.org/10.1007/978-0-387-77745-0_40
2. Cullot, N., Ghawi, R., Yétongnon, K.: DB2OWL: a tool for automatic database-to-ontology mapping. SEBD **7**, 491–494 (2007)
3. DBpedia (2020). https://wiki.dbpedia.org/. Accessed July 2020
4. Gali, A., Chen, C.X., Claypool, K.T., Uceda-Sosa, R.: From ontology to relational databases. In: Wang, S., et al. (eds.) ER 2004. LNCS, vol. 3289, pp. 278–289. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30466-1_26
5. GeoNames: Geonames ontology (2019). http://www.geonames.org/ontology/documentation.html. Accessed July 2020
6. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: HermiT: an OWL 2 reasoner. J. Autom. Reasoning **53**(3), 245–269 (2014). https://doi.org/10.1007/s10817-014-9305-1

7. Golbreich, C., Wallace, E.K., Patel-Schneider, P.F.: Owl 2 web ontology language new features and rationale. W3C working draft, W3C (2009). http://www.w3.org/TR/2009/WD-owl2-new-features-20090611
8. Halpin, H., Hayes, P.J.: When owl:sameAs isn't the same: an analysis of identity links on the semantic web. In: LDOW (2010)
9. Hu, W., Qu, Y.: Discovering simple mappings between relational database schemas and ontologies. In: Aberer, K., et al. (eds.) ASWC/ISWC -2007. LNCS, vol. 4825, pp. 225–238. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76298-0_17
10. Jain, P., Hitzler, P., Sheth, A.P., Verma, K., Yeh, P.Z.: Ontology alignment for linked open data. In: Patel-Schneider, P.F., et al. (eds.) ISWC 2010. LNCS, vol. 6496, pp. 402–417. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17746-0_26
11. Kazakov, Y.: Elk (2016). https://protegewiki.stanford.edu/wiki/ELK. Accessed July 2020
12. Li, M., Du, X.Y., Wang, S.: Learning ontology from relational database. In: 2005 International Conference on Machine Learning and Cybernetics. IEEE (2005). https://doi.org/10.1109/icmlc.2005.1527531
13. Mogotlane, K.D., Fonou-Dombeu, J.V.: Automatic conversion of relational databases into ontologies: a comparative analysis of protege plug-ins performances. Int. J. Web Semant. Technol. **7**(3/4), 21–40 (2016). https://doi.org/10.5121/ijwest.2016.7403
14. Noy, N., Sintek, M., Decker, S., Crubezy, M., Fergerson, R., Musen, M.: Creating semantic web contents with protege-2000. IEEE Intell. Syst. **16**(2), 60–71 (2001). https://doi.org/10.1109/5254.920601
15. Parsia, B., Patel-Schneider, P., Motik, B.: Owl 2 web ontology language structural specification and functional-style syntax. W3C, W3C Recommendation (2012)
16. Perry, M., Herring, J.: GeoSPARQL - a geographic query language for RDF data. Open Geospatial Consortium (2012). https://www.ogc.org/standards/geosparql
17. Schekotihin, K., Rodler, P., Schmid, W.: OntoDebug: interactive ontology debugging plug-in for protégé. In: Ferrarotti, F., Woltran, S. (eds.) Foundations of Information and Knowledge Systems. Lecture Notes in Computer Science, vol. 10833, pp. 340–359. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-90050-6_19
18. Sequeda, J.F., Arenas, M., Miranker, D.P.: On directly mapping relational databases to RDF and OWL. In: Proceedings of the 21st International Conference on World Wide Web - WWW 2012. ACM Press (2012). https://doi.org/10.1145/2187836.2187924
19. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: a practical OWL-DL reasoner. SSRN Electron. J. (2007). https://doi.org/10.2139/ssrn.3199351
20. Trinh, Q., Barker, K., Alhajj, R.: RDB2ONT: a tool for generating OWL ontologies from relational database systems. In: Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW 2006). IEEE (2006). https://doi.org/10.1109/aict-iciw.2006.159
21. U. S. Geological Survey: The Best Practices Data Model-Governmental Units (2006). https://services.nationalmap.gov/bestpractices/model/acrodocs/Poster_BPGovtUnits_03_01_2006.pdf. Accessed July 2020
22. U. S. Geological Survey: File Format for Domestic Geographic Names (2020). https://geonames.usgs.gov/docs/pubs/Nat_State_Topic_File_formats.pdf. Accessed July 2020

23. U. S. Geological Survey: Spec-X - Making Information Accessible (2020). https://usgs-mrs.cr.usgs.gov/SPECX/treeview/index. Accessed July 2020
24. Wikidata (2020). https://www.wikidata.org/wiki/Wikidata:Main_Page. Accessed July 2020
25. Zhang, L., Li, J.: Automatic generation of ontology based on database. J. Comput. Inf. Syst. **7**(4), 1148–1154 (2011)