Article

Modularization Method to Reuse Medical Knowledge Graphs

Maricela Bravo [1,*], Darinel González-Villarreal [1], José-A. Reyes-Ortiz [1], and Leonardo-D. Sánchez-Martínez [1]

[1] Universidad Autónoma Metropolitana; mcbc@azc.uam.mx, darinelrammstein@gmail.com, jaro@azc.uam.mx, ldsm@azc.uam.mx
[*] Correspondence: mcbc@azc.uam.mx

Abstract: During the creation and integration of a health care system based on medical knowledge graphs, it is necessary to review and select the vocabularies and definitions that best fit the information requirements of the system being developed. This implies the reuse of medical knowledge graphs; however, full importation of knowledge graphs is not a tractable solution in terms of memory requirements. In this paper we present a modularization-based method for knowledge graph reuse. A case study of graph reuse is presented by transforming the original model into a lighter one.

Keywords: Medical knowledge graphs, knowledge graphs reuse, ontology modularization

1. Introduction

Knowledge graphs in the medical area have become enormously relevant to support medical research development and to facilitate exchange of clinical data and scientific advances. That is the case of the well-known NCBO BioPortal , a medical ontology repository, where multiple health institutions publish their ontologies or knowledge graphs. In this article we describe a modularization-based method for reutilization of portions of medical knowledge graphs into a general model maintaining reasoning efficiency.

Reutilization of knowledge graphs occurs during the design and integration of multiple semantic representation models (ontologies or knowledge graphs) with the aim of generating a complete representation model to solve the requirements of intelligent information systems or expert systems. Reutilization of a knowledge graph or ontology is the task of completely or partially importing a model into another. The reuse of a knowledge graph may involve carrying out a process of modularization and adaptation or mapping from one model to another.

The process of reusing a knowledge graph begins with the search and selection of the graphs or ontologies that are considered appropriate to complete the definitions required by the system. Once the knowledge graphs to be reused have been identified, it is necessary to review in detail how this integration will be carried out. To address this problem, the following questions should be considered:

Is it necessary to import the entire model or only a part of the model is required?

Is it necessary to adjust or make adaptations between the imported model and the host model?

What is the most suitable modularization approach??

What is the computational cost of each option?

This article describes a general method for generating and reusing knowledge graph modules. This method arises from the need to integrate a general knowledge graph for managing electronic patient records. During the process of designing and integrating the general knowledge graph, a great disadvantage was observed in reusing some biomedical ontologies. This is mainly because most of these ontologies are very large and complex. Therefore, it was necessary to develop a method that would allow obtaining useful information for the general graph by extracting modules.

As a result, four modules of knowledge graphs were obtained that were imported into the general knowledge graph. The resulting knowledge graph functioned adequately for the information needs required for the project.

The rest of the paper is organized as follows: in Section 2 a review on modularization of knowledge graphs is presented, in Section 3 the method to generate and reuse medical knowledge graphs modules, in Section 4 the application of the method to generate the Medicament knowledge graph module is presented, in Section 5 the application of the method to generate the Disease knowledge graph module is described, in Section 6 the integrated knowledge graph modularization system is presented, in Section 7 two evaluation approaches are described, an finally in Section 8 conclusions are presented.

2. Modularization of Knowledge Graphs

One of the first attempts to address modularization of representation models was reported by Alan Rector in 2003 [1], who stated the importance of modularity as a key requirement for ontologies to achieve reutilization, maintainability, and evolution. Modularization of ontologies was defined as the task of decomposing an ontology into independent disjoint skeleton taxonomies.

In 2006 Schlicht and Stuckenschmidt [2] described a modularization approach aiming at supporting the distribution of knowledge in a P2P network. Authors describe the requirements of reasoning efficiency, robustness and maintainability; and propose the following structural criteria: connectedness of modules, stating that the efficiency of distributed computation is closely related with the degree of interconnectedness of the generated modules; size and number of modules, this criteria has a strong impact on the robustness; and redundancy of representation, the use of redundant representations will improve robustness at the cost of increasing maintenance. The goal of this approach is to apply modularization on a given ontology with the purpose of splitting it considering numerical aspects, such as: size, number of modules and modules interconnectedness.

From a logic-based perspective a series of contributions have been reported by a research group led by Bernardo Cuenca and Ian Horrocks. In 2006 [3] Cuenca et al. described the notion of a module from a model-theoretic perspective as a self-contained unit within the ontology. They stressed additional module requirements such as: the scope of the module, its size, and the correct interpretation of the module. Later in 2007 [4] Cuenca et al. presented an approach to extract modular fragments from ontologies preserving the minimal conditions. They defined a module with respect to a signature S, aiming at reducing the cost of importing external ontologies they proposed to import only a fragment of given ontologies preserving the meaning of the terms imported.

Based on the afore mentioned references a module can be defined as a self-contained knowledge unit within the knowledge graph, where a module has a clearly defined scope, its size, and a correct interpretation of the module.

Whereas modularization can be defined as the task of decomposing a knowledge graph into independent disjoint modules, splitting it considering modules size, number of modules and modules interconnectedness.

Accordingly, knowledge graph modularization should address the following criteria:

a) Facilitate reutilization, and evolution of knowledge graphs.
b) Maintain reasoning efficiency.
c) Keep the balance between robustness and maintainability.
d) Enable distribution of knowledge in open networks.
e) Support connectedness of modules.

The concepts of module and modularization are closely related to the process of reusing knowledge graphs, since the purpose is to extract modules from knowledge graphs so that they can be imported into another model, ensuring that the semantic definitions and relations are not lost while maintaining the efficiency of reasoning.

Therefore, in the rest of this article we will use the concept of Knowledge Graph Module (KGM) as the resulting product of modularization process.

In 2009 Pathak et al. [5] presented a survey on modular ontologies techniques that are based on logical formalisms and graph theories. They state that modular ontologies techniques are crucial for the biomedical domain, since most popular ontologies are large and complex. Therefore, the development of tools for managing multiple distributed ontologies will benefit reasoning performance. Authors stablish that an ontology module should be self-contained and logically consistent. Also defined the goals of ontology modularization: partial reuse, complexity, ownership and customization, efficient reasoning, and tooling support. Finally, authors outlined evaluation criteria and requirements: localized semantics, correct reasoning, transitivity, safe reuse and decidability.

In Courtot M. et al. [6] authors presented the Minimum Information to Reference an External Ontology Term (MIREOT) guidelines to aid the development of the Ontology of Biomedical Investigations (OBI). The purpose of this guide is to import the minimum required reference from an external ontology, for this the authors propose three cases, depending on what is required as a reference, all of them generally require that the identifier or URI (currently IRI) of the class or term reference be included. The method described in this article agrees to keep the unique identifier of each concept or class, but includes more relevant details of each concept, avoiding an overload in the memory of the reasoner.

One of the main trade offs of ontology modularization is to determine the level of expressiveness that you want to maintain from the original knowledge graph. However, the higher the level of expressiveness, the lower the ability to reason with large knowledge graphs and the efficiency will be very poor. Regarding this problem, Algergawy et al. [7] introduced the Ontology Analysis and Partitioning Tool (OAPT), a framework for analyzing and partitioning ontologies. However, the main approach used by the authors is to partition an ontology into modules and then apply evaluation criteria. Instead, in this article we focus on the extraction of specific information or knowledge that needs to be included in another knowledge graph of a particular purpose. The original graphs are not partitioned into modules.

In 2012 Alan Rector et al. [8] presented use cases for modular development of ontologies by using the import mechanism of OWL. According to the authors, ontology modularization concerns two topics: module extraction, which consists of separating existing ontologies into modules; and modular development, which resembles modular software engineering. In this article we present a modularization method that combines aspects of both cases, on the one hand, the general design of the knowledge graph is done with a modular development approach from the beginning; and on the other hand we present a semi-automatic method to perform the extraction of modules for reuse.

In 2021 Shimizu, Hammar & Hitzler [9] described the reasons why ontologies are not frequently reused as follows: differing representational granularity, lack of conceptual clarity in many ontologies, lack and difficulty of adherence to good modeling principles, lack of support in ontology engineering tools.

Another relevant issue that we deal with in the modularization method described in this paper is related to the way biomedical ontologies are developed, most of which do not have A-Boxes, they are mainly designed with T-Boxes. This represents a difficulty for their reuse, in case of establishing an *is_A* relationship between an object and a class, the interpretation is that the object is a member of that class, which generates misleading interpretations.

Although there are various approaches and studies presented for the reuse of ontologies or knowledge graphs, there are still difficulties for this practice to become a reality. In this paper we present a module extraction approach that depends on a careful design of the concepts and relationships that are required to be reused.

3. Method to Generate and Reuse Medical KGM

To describe the analysis and decision-making process that was carried out, this section first presents the system that required a method for reusing knowledge graphs. Specifically, the analysis of information requirements of the system required the search, selection, and reuse of various external knowledge graphs. This system is based on the specification of a comprehensive knowledge graph to support the representation and management of Electronic Health Records (EHR) of patients. We start from a base model that was designed with the aim of representing patient profiles. This base model was first reported in [10].

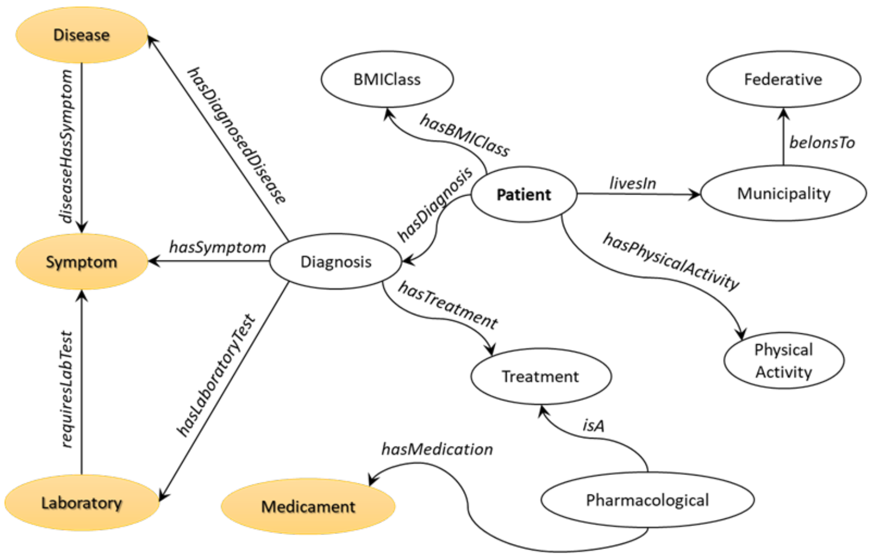


Figure 1. Knowledge graph for the representation and management of Electronic Health Records (EHR).

Figure 1 shows the general EHR model, which depicts in highlighted color the concepts that require reutilization of medical vocabularies: Medicament, Disease, Symptom, and Laboratory.

As a result of the analysis of conceptual requirements of the initial representation model, it was necessary to find reliable and scientifically valid sources of information for the inclusion of concepts about medical knowledge. Therefore, to complete this model, we decided to reuse vocabularies and definitions from external references. A series of searches were executed in BioPortal, resulting in the list of ontologies shown in Table 1.

Table 1. NCBO Ontologies identified for reuse.

Acronym	Description	Ontology
SYMPT	The Symptom Ontology has been developed as a standardized ontology for symptoms of human diseases.	http://purl.obolibrary.org/obo/symp.owl
LOINC	Logical Observation Identifier Names and Codes (LOINC)	http://purl.bioontology.org/ontology/LNC/
NDF-RT	National Drug File - Reference Terminology Public Inferred Edition	http://evs.nci.nih.gov/ftp1/NDF-RT/NDF-RT.owl
DOID	A hierarchical controlled vocabulary for human disease representation	http://purl.obolibrary.org/obo/doid.owl

[1] Tables may have a footer.

To reuse each of these knowledge graphs various methods exist, for instance, importing full models into a general knowledge graph. However, importing a complete model is not a good alternative, this is because the number of axioms that are imported are added, making the integrated representation model intractable in terms of the memory resources that would be required to make use of it. Therefore, a different approach should be defined and implemented.

We considered using a modularization-based method for reutilization of knowledge graphs, this method consists in the transformation of part of the knowledge graph definitions that are relevant for the specific model being integrated. For each of the general concepts highlighted in Figure 1, we defined and implemented the following procedure (method) that allowed the modularization and reutilization of knowledge graphs.

1. Select the knowledge graph or ontology that will be reused. The knowledge graph must be downloaded and analyzed locally.
2. Identify the relevant data or attributes that will be reused to complete the conceptualization into the knowledge graph being integrated.
3. Define and implement data structures using the Object-Oriented Programming paradigm to allow the clearest and most efficient handling of the concepts of interest.
4. Implement a program to query the knowledge graph using SPARQL and automatically obtain the list of concepts of interest.
5. Define and construct the model of the knowledge graph that will be used to contain the concepts and attributes of interest.
6. Develop a program to automatically populate the knowledge graph with the list of concepts obtained.

The application of this method for each of the required concepts is described in the following sections.

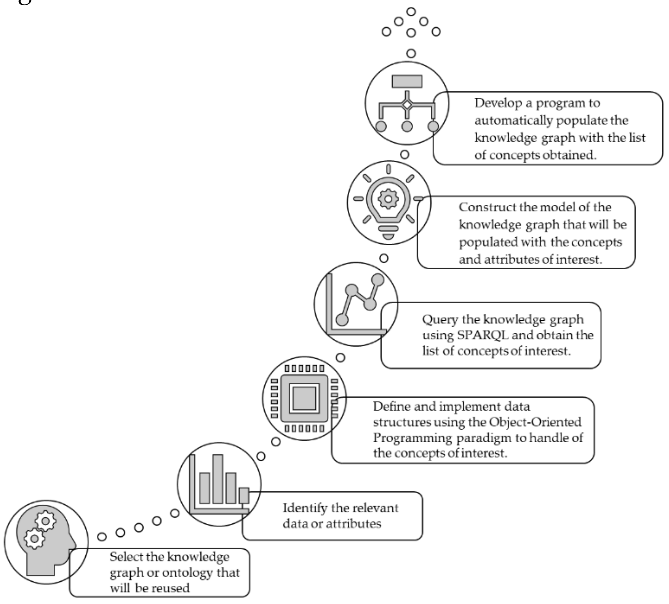

Develop a program to automatically populate the knowledge graph with the list of concepts obtained.
Construct the model of the knowledge graph that will be populated with the concepts and attributes of interest.
Query the knowledge graph using SPARQL and obtain the list of concepts of interest.
Define and implement data structures using the Object-Oriented Programming paradigm to handle of the concepts of interest.
Identify the relevant data or attributes
Select the knowledge graph or ontology that will be reused

Figure 2. Method to generate and reuse medical KGM.

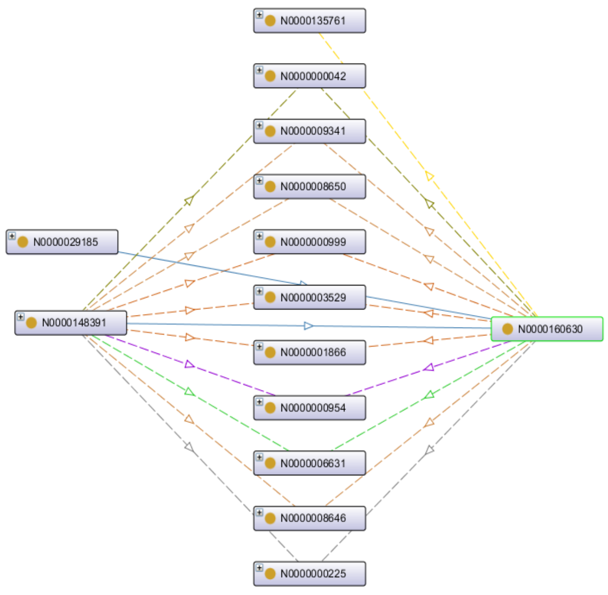
4. Medicament KGM

For the representation of an EHR it is necessary to specify the medications that a patient may be taking, or that will be indicated as part of his pharmacological treatment. Therefore, it is necessary to use a medical vocabulary of drugs.

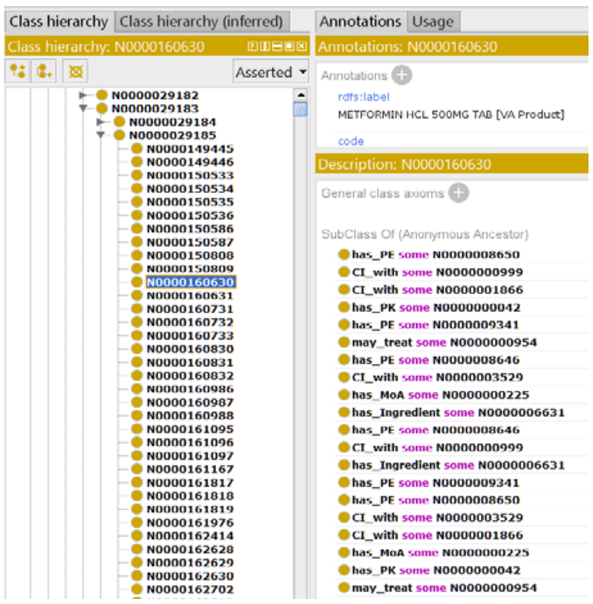
To obtain a Medicament KGM, we implemented the method to generate and reuse medical KGM as follows:

4.1. Selection of the knowledge graph

For the representation of medicaments concepts and relations we selected the NDF-RT knowledge graph. NDF-RT is the National Drug File - Reference Terminology produced by the Veterans Health Administration. Figure 3 shows a snapshot of the NDF-RT model as visualized in Protégé editor.



(**a**)



Class hierarchy | Class hierarchy (inferred)

Class hierarchy: N0000160630

Asserted

```
▸ N0000029182
▾ N0000029183
  ▸ N0000029184
  ▾ N0000029185
      N0000149445
      N0000149446
      N0000150533
      N0000150534
      N0000150535
      N0000150536
      N0000150586
      N0000150587
      N0000150808
      N0000150809
      N0000160630
      N0000160631
      N0000160731
      N0000160732
      N0000160733
      N0000160830
      N0000160831
      N0000160832
      N0000160986
      N0000160987
      N0000160988
      N0000161095
      N0000161096
      N0000161097
      N0000161167
      N0000161817
      N0000161818
      N0000161819
      N0000161976
      N0000162414
      N0000162628
      N0000162629
      N0000162630
      N0000162702
```

Annotations | Usage

Annotations: N0000160630

Annotations

rdfs:label
METFORMIN HCL 500MG TAB [VA Product]

code

Description: N0000160630

General class axioms

SubClass Of (Anonymous Ancestor)

```
has_PE some N0000008650
CI_with some N0000000999
CI_with some N0000001866
has_PK some N0000000042
has_PE some N0000009341
may_treat some N0000000954
has_PE some N0000008646
CI_with some N0000003529
has_MoA some N0000000225
has_Ingredient some N0000006631
has_PE some N0000008646
CI_with some N0000000999
has_Ingredient some N0000006631
has_PE some N0000009341
has_PE some N0000008650
CI_with some N0000003529
CI_with some N0000001866
has_MoA some N0000000225
has_PK some N0000000042
may_treat some N0000000954
```

(**b**)

Figure 3. This figure shows a partial view of the NDF-RT ontology, specifically the N0000160630 class is highlighted which represents "Metformin hydrochloride 500 MG Oral Tablet". (**a**) Presents the relations of N0000160630 with other classes; (**b**) Presents the details of the relations or properties of the N0000160630 concept.

As can be seen in Figure 3, the NDF-RT knowledge graph is very complex and is not an intuitive model to understand. This is in part because the medical information it represents is highly specialized and technical. However, the structure of the ontological model is also difficult to understand. For example, class naming is based on a nomenclature established in the design principles indicated by the NCBO for the publication of ontologies in BioPortal. As a result of this naming convention, class names do not provide any clue of which drugs they represent, it is necessary to expand the classes to identify the class that groups the entire hierarchy of drugs. Additionally, it is necessary to explore all the annotations to know the specific details of each concept.

Another problem that is observed with this knowledge graph is the fact that all the represented concepts are organized mainly using class definitions, that is, there are no individuals as member of the classes, which represents a difficulty in establishing a relationship between objects according to what is established in the base model shown in Figure 1.

Despite the difficulties that this model presents, NDF-RT is quite useful to be reused within the EHR because it defines concepts that relate drugs to diseases, for example, may-treat and may-prevent relationships, which we consider relevant.

4.2. Identification of the data and attributes to be imported

In order to review and analyze the NDF-RT knowledge graph was saved it in Turtle syntax. For the extraction of medicament products, we selected only the "Drug Products by VA Class", which represent clinical pharmaceutical products that can be ordered. As an example, the code shown in Figure 4 describes the concept N0000160630, that identifies the drug product "Metformin hydrochloride 500 MG Oral Tablet". Based on this code we can identify the important attributes that should be considered, for instance: the RxNorm_Name, the RxNorm CUI , units, strength, and NUI, among others.

```
NDF-RT:N0000160630 NDF-RT:RxNorm_Name "Metformin hydrochloride 500 MG Oral Tablet" ;
                   NDF-RT:code "C45762" ;
                   NDF-RT:Level "VA Product" ;
                   NDF-RT:Units "MG" ;
                   NDF-RT:Product_Component NDF-RT:N0000148391 ;
                   NDF-RT:Status "Active" ;
                   NDF-RT:NUI "N0000160630"^^xsd:string ;
                   NDF-RT:RxNorm_CUI "861007" ;
                   NDF-RT:VANDF_Record "
                      <VANDF_Record>50.68^12369^</VANDF_Record>
                      <VA_File>50.68</VA_File>
                      <VA_IEN>12369</VA_IEN>" ;
                   NDF-RT:Print_Name "METFORMIN HCL 500MG TAB" ;
                   NDF-RT:UMLS_CUI "C0978483" ;
                   NDF-RT:Display_Name "METFORMIN HCL 500MG TAB" ;
                   NDF-RT:Product_Component_Qualifier "
                      <Product_Component>METFORMIN HYDROCHLORIDE</Product_Component>
                      <Strength>500</Strength>
                      <Unit>MG</Unit><VA.IEN>3556</VA.IEN>" ;
                   rdfs:label "METFORMIN HCL 500MG TAB [VA Product]" ;
                   metadata:prefixIRI "N0000160630"^^xsd:string ;
                   NDF-RT:VA_National_Formulary_Name "METFORMIN TAB,ORAL" ;
                   NDF-RT:VUID "4012735" ;
                   NDF-RT:Strength "500" .
```

Figure 4. Excerpt from the NDF-RT representation of the N0000160630 class, that identifies the drug product "Metformin hydrochloride 500 MG Oral Tablet".

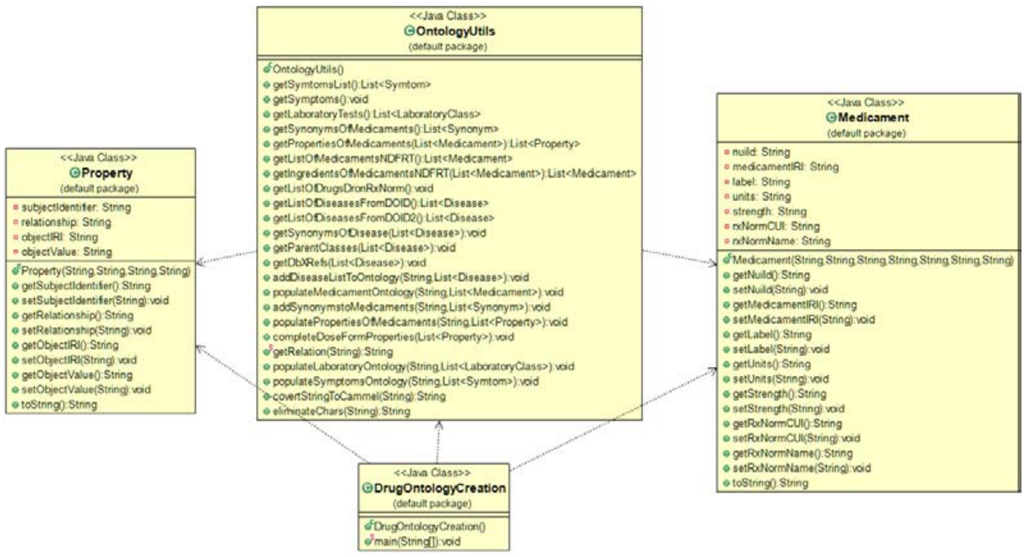
The set of relations or roles that NDF-RT uses can be extracted from a different part of the definitions, the code shown in Figure 5 presents the set of drug relationships with other concepts, for example, mechanism of action, physiological effects, clinical kinetics, among others.

```
###  http://evs.nci.nih.gov/ftp1/NDF-RT/NDF-RT.owl#N0000160630
NDF-RT:N0000160630 rdf:type owl:Class ;
                   rdfs:subClassOf NDF-RT:N0000029185 ,
                                   NDF-RT:N0000148391 ,
                                   [ rdf:type owl:Restriction ;
                                     owl:onProperty NDF-RT: ;
                                     owl:someValuesFrom NDF-RT:N0000135761
                                   ] ,
                                   [ rdf:type owl:Restriction ;
                                     owl:onProperty NDF-RT:CI_with ;
                                     owl:someValuesFrom NDF-RT:N0000000999
                                   ] ,
                                   [ rdf:type owl:Restriction ;
                                     owl:onProperty NDF-RT:CI_with ;
                                     owl:someValuesFrom NDF-RT:N0000001866
                                   ] ,
                                   [ rdf:type owl:Restriction ;
                                     owl:onProperty NDF-RT:CI_with ;
                                     owl:someValuesFrom NDF-RT:N0000003529
                                   ] ,
                                   [ rdf:type owl:Restriction ;
                                     owl:onProperty NDF-RT:has_Ingredient ;
                                     owl:someValuesFrom NDF-RT:N0000006631
                                   ] ,
                                   [ rdf:type owl:Restriction ;
                                     owl:onProperty NDF-RT:has_MoA ;
                                     owl:someValuesFrom NDF-RT:N0000000225
                                   ] ,
                                   [ rdf:type owl:Restriction ;
                                     owl:onProperty NDF-RT:has_PE ;
                                     owl:someValuesFrom NDF-RT:N0000008646
                                   ] ,
```

Figure 5. Excerpt from the NDF-RT representation of, that describes the list of properties of the N0000160630 class.

4.3. Define and implement data structures

To extract and represent medicament information using an object-oriented paradigm, the JavaTM classes shown in Figure 6 were implemented. The Medicament class includes the attributes that were selected for their relevance. It should be noted that a lighter extraction of the concept attributes is performed. On the other hand, the Property class is used as a helper class in extracting all the important relationships that drugs can have with other concepts. The OntologyUtils class is an important class in which all the information extraction methods have been implemented, as well as the ontology population methods. The DrugOntologyCreation class represents the main program in which the data extraction, transformation, and module generation tasks are executed in sequence.



<<Java Class>>
Property
(default package)
subjectIdentifier: String
relationship: String
objectIRI: String
objectValue: String
Property(String,String,String,String)
getSubjectIdentifier():String
setSubjectIdentifier(String):void
getRelationship():String
setRelationship(String):void
getObjectIRI():String
setObjectIRI(String):void
getObjectValue():String
setObjectValue(String):void
toString():String
<<Java Class>>
OntologyUtils
(default package)
OntologyUtils()
getSymtomsList():List<Symtom>
getSymptoms():void
getLaboratoryTests():List<LaboratoryClass>
getSynonymsOfMedicaments():List<Synonym>
getPropertiesOfMedicaments(List<Medicament>):List<Property>
getListOfMedicamentsNDFRT():List<Medicament>
getIngredientsOfMedicamentsNDFRT(List<Medicament>):List<Medicament>
getListOfDrugsDronRxNorm():void
getListOfDiseasesFromDOID():List<Disease>
getListOfDiseasesFromDOID2():List<Disease>
getSynonymsOfDisease(List<Disease>):void
getParentClasses(List<Disease>):void
getDbXRefs(List<Disease>):void
addDiseaseListToOntology(String,List<Disease>):void
populateMedicamentOntology(String,List<Medicament>):void
addSynonymstoMedicaments(String,List<Synonym>):void
populatePropertiesOfMedicaments(String,List<Property>):void
completeDoseFormProperties(List<Property>):void
getRelation(String):String
populateLaboratoryOntology(String,List<LaboratoryClass>):void
populateSymptomsOntology(String,List<Symtom>):void
covertStringToCammel(String):String
eliminateChars(String):String
<<Java Class>>
Medicament
(default package)
nuild: String
medicamentIRI: String
label: String
units: String
strength: String
rxNormCUI: String
rxNormName: String
Medicament(String,String,String,String,String,String,String)
getNuild():String
setNuild(String):void
getMedicamentIRI():String
setMedicamentIRI(String):void
getLabel():String
setLabel(String):void
getUnits():String
setUnits(String):void
getStrength():String
setStrength(String):void
getRxNormCUI():String
setRxNormCUI(String):void
getRxNormName():String
setRxNormName(String):void
toString():String
<<Java Class>>
DrugOntologyCreation
(default package)
DrugOntologyCreation()
main(String[]):void

Figure 6. Class diagram of the Medicament KGM creation.

4.4. Implement a program to query the knowledge graph

To obtain the information from the NDF-RT knowledge graph, we implemented a JavaTM program using the API RDF4J that allowed issuing SPARQL queries through which a list of objects with the previously defined types were generated. The purpose of the first query in Table 2 is to obtain the list of medications and their basic attributes, the second query obtains the entire set of relationships that are all characterized as properties.

Table 2. SPARQL queries executed to retrieve the list of Medicament objects from NDF-RT.

SPARQL Query	Typed Collection Returned
PREFIX NDF-RT: <http://evs.nci.nih.gov/ftp1/NDF-RT/NDF-RT.owl#>	
SELECT distinct ?medicament ?label ?units ?strength ?nui ?cui ?name	
WHERE {	
?medicament rdf:type owl:NamedIndividual .	
?medicament rdfs:label ?label .	List<Medicament> listMed;
?medicament NDF-RT:Level "VA Product" .	
?medicament NDF-RT:Units ?units .	
?medicament NDF-RT:Strength ?strength .	
?medicament NDF-RT:NUI ?nui .	

```
                ?medicament NDF-RT:RxNorm_CUI ?cui .
                ?medicament NDF-RT:RxNorm_Name ?name .          }
```

```
PREFIX NDF-RT: <http://evs.nci.nih.gov/ftp1/NDF-RT/NDF-RT.owl#>
SELECT ?property ?value ?valueLabel
         WHERE {
                NDF-RT:" + medId + " rdf:type owl:Class .                List<Property> listProps;
                NDF-RT:" + medId + " rdfs:subClassOf ?s1 .
                ?s1 owl:onProperty ?property; owl:someValuesFrom ?value
.
                ?value rdfs:label ?valueLabel .      }
```

[1] SPARQL queries that are embedded into java code that produce a list of Medicament objects.

4.5. Define and construct the model of the knowledge graph

For the construction of the Medicament KGM that represents the data about drugs, it is necessary to design and implement the conceptual model of the target module graph, which is, the T-Box of the ontology. Therefore, and considering the types of data (or objects) returned by queries, the ontological model presented in Figure 7 was designed. This model represents a different view than the original NDF-RT knowledge graph. The resulting KGM is a simplified and much lighter version of the original, this model has more intuitive named concepts and relationships that facilitate reuse in another knowledge graph. For instance, it is clearer that a given instance of the Medicament concept may treat, or may diagnose, or may prevent any instance of the Disease concept.

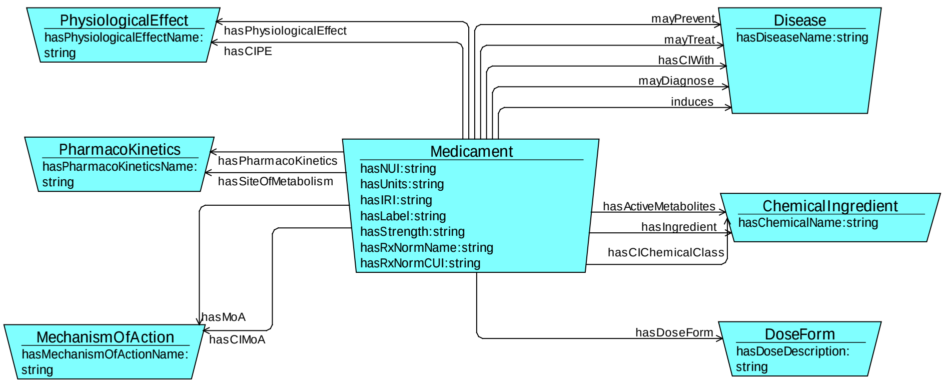


Figure 7. Model of Medicament KGM based on the NDF-RT ontology model.

According with the NDF-RT documentation , the following classes (concepts) are defined as follows:

a) Medicament. This concept is at the core of the hierarchy in NDF-RT. It includes classifications of medications, generic ingredient preparations used in medications, and orderable drug products.

b) Clinical Kinetics. Represents a collection of concepts describing the absorption, distribution, and elimination of drug active ingredients.

c) Chemical Ingredient. Represents chemicals or other drug ingredients, organized into a chemical structure classification hierarchy.

d) Mechanism of Action. Represents molecular, subcellular, or cellular effects of drug generic ingredients, organized into a chemical function classification hierarchy.

e) Physiological Effect. Concerns tissue, organ, or organ system effects of drug generic ingredients, organized into an organ system classification hierarchy.

f) Dose Form. Represents specific hierarchy of administered medication dose.

g) Disease. Represents pathophysiologic as well as certain non-disease physiologic states that are treated, prevented, or diagnosed by an ingredient or drug product. May also be used to describe contraindications.

The following semantic relations (object properties) are defined as follows:

a) CI Chemical class. This relation is used to specify that a medicament is contraindicated with a chemical ingredient.

b) Has ingredient. Relation used to indicate that a medicament has a chemical ingredient.

c) Has active metabolites. Relation used to indicate that a medicament has active metabolites.

d) Has PE. This relation is used to specify that a medicament has a physiological effect.

e) CI PE. Relation used to indicate that a medicament is contraindicated with a physiological effect.

f) Has dose form. This relation correlates the medicament with its form of dose.

g) Has MoA. This relation is used to specify that a medicament has a mechanism of action.

h) CI MoA. Relation used to indicate that a medicament is contraindicated with a mechanism of action.

i) Induces. Relation used to indicate that a medicament induces a disease.

j) May diagnose. Relation used to indicate that a medicament may diagnose a given disease.

k) May prevent. Relation used to indicate that a medicament may prevent a disease.

l) May treat. Relation used to indicate that a medicament may treat a disease.

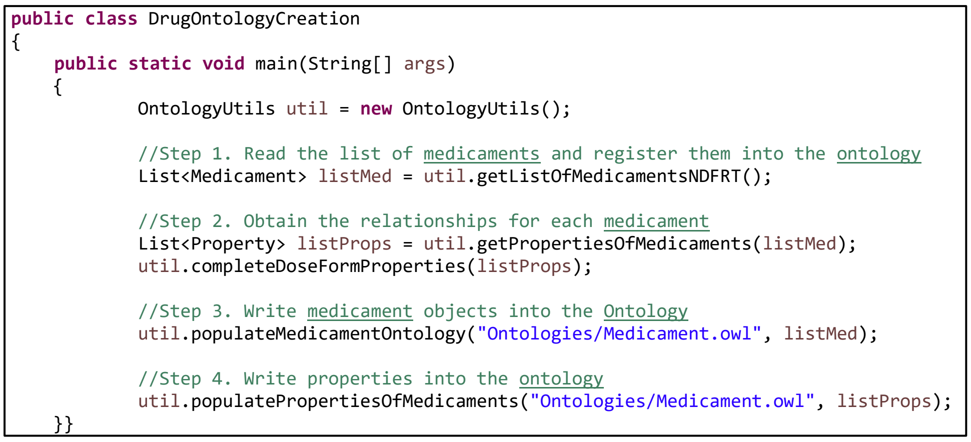
m) CI with. This relation is used to specify that a medicament is contraindicated with a given disease.

n) Has PK. Relation that indicates that a medicament has a clinical kinetics (absorption, distribution, and elimination of drug active ingredients).

o) Site of metabolism. Relation used to indicate that a medicament has a site of metabolism.

4.6. Develop a program to automatically populate the knowledge graph

Finally, a general JavaTM program was developed for the construction of the Medicament KGM, which integrates the information retrieval programs and the ontology population methods. The general procedure consists of the following steps: read the list of medicaments from the NDF-RT knowledge graph, recover the properties of the medicaments, populate the Medicament KGM with the list of medicaments, and finally for each medicament instance, populate the list of properties and update the Medicament KGM.



```
public class DrugOntologyCreation
{
    public static void main(String[] args)
    {
            OntologyUtils util = new OntologyUtils();

            //Step 1. Read the list of medicaments and register them into the ontology
            List<Medicament> listMed = util.getListOfMedicamentsNDFRT();

            //Step 2. Obtain the relationships for each medicament
            List<Property> listProps = util.getPropertiesOfMedicaments(listMed);
            util.completeDoseFormProperties(listProps);

            //Step 3. Write medicament objects into the Ontology
            util.populateMedicamentOntology("Ontologies/Medicament.owl", listMed);

            //Step 4. Write properties into the ontology
            util.populatePropertiesOfMedicaments("Ontologies/Medicament.owl", listProps);
    }}
```

Figure 8. Main program in which the data extraction, transformation, and module generation tasks are executed in sequence.

5. Disease KGM

One of the most important concepts of an EHR is that of diseases. There are various definitions of disease in dictionaries or in the medical literature, here we present some general definitions:

"Disease, any harmful deviation from the normal structural or functional state of an organism, generally associated with certain signs and symptoms and differing in nature from physical injury." (Encyclopedia Britannica)

"(an) illness of people, animals, plants, etc., caused by infection or a failure of health rather than by an accident." (Cambridge Dictionary)

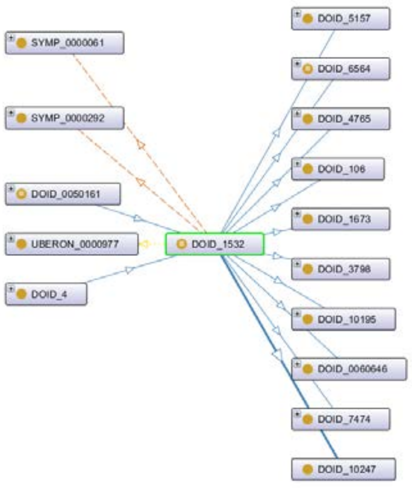
"A disease is an illness which affects people, animals, or plants, for example one which is caused by bacteria or infection." (Collins Dictionary)

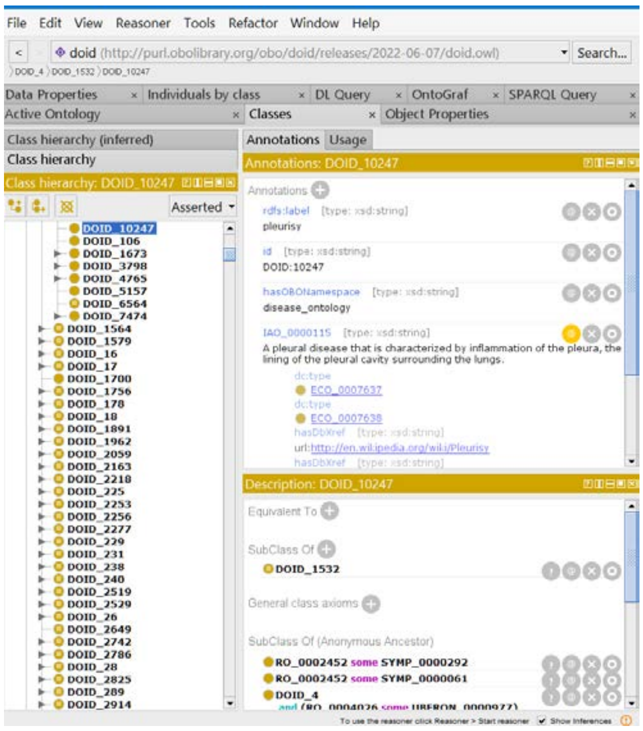
To obtain a Disease KGM we implemented the method to generate and reuse medical KGM as follows:

5.1. Selection of the knowledge graph

Considering the management of information for an EHR, a disease is associated with the person or patient to whom the file refers; likewise, a disease is associated with a set of signs and symptoms. Therefore, we have decided to reuse the DOID knowledge graph for this purpose. The Human Disease Ontology (DOID) was developed with the collaboration between biomedical researchers coordinated by the University of Maryland School of Medicine, Institute for Genome Sciences.

Figure 9 shows a partial view of the DOID ontology, as can be seen, this model presents a complex class structure, and the names are not intuitive either. If there is a requirement for obtaining the information of a particular disease, it is necessary to know the specific id with which it has been classified to do a search and read all the tags added as annotations of the concept. Of course, with a well-developed search tool, the use of this type of model is feasible, but it still requires knowledge and experience dealing with this kind of vocabularies.




File Edit View Reasoner Tools Refactor Window Help
doid (http://purl.obolibrary.org/obo/doid/releases/2022-06-07/doid.owl)
Search...
DOID_4 DOID_1532 DOID_10247
Data Properties
Individuals by class
DL Query
OntoGraf
SPARQL Query
Active Ontology
Classes
Object Properties
Class hierarchy (inferred)
Class hierarchy
Annotations
Usage
Asserted

Class hierarchy: DOID_10247

DOID_10247
DOID_106
DOID_1673
DOID_3798
DOID_4765
DOID_5157
DOID_6564
DOID_7474
DOID_1564
DOID_1579
DOID_16
DOID_17
DOID_1700
DOID_1756
DOID_178
DOID_18
DOID_1891
DOID_1962
DOID_2059
DOID_2163
DOID_2218
DOID_225
DOID_2253
DOID_2256
DOID_2277
DOID_229
DOID_231
DOID_238
DOID_240
DOID_2519
DOID_2529
DOID_26
DOID_2649
DOID_2742
DOID_2786
DOID_28
DOID_2825
DOID_289
DOID_2914

Annotations: DOID_10247

Annotations
rdfs:label [type: xsd:string]
pleurisy
id [type: xsd:string]
DOID:10247
hasOBONamespace [type: xsd:string]
disease_ontology
IAO_0000115 [type: xsd:string]
A pleural disease that is characterized by inflammation of the pleura, the lining of the pleural cavity surrounding the lungs.
dc:type
ECO_0007637
dc:type
ECO_0007638
hasDbXref [type: xsd:string]
url:http://en.wikipedia.org/wiki/Pleurisy
hasDbXref [type: xsd:string]

Description: DOID_10247

Equivalent To
SubClass Of
DOID_1532
General class axioms
SubClass Of (Anonymous Ancestor)
RO_0002452 some SYMP_0000292
RO_0002452 some SYMP_0000061
DOID_4
and (RO_0004026 some UBERON_0000977)

To use the reasoner click Reasoner > Start reasoner ✓ Show Inferences

(a) (b)

Figure 9. This figure shows a partial view of the DOID ontology, specifically the DOID_10247 class which represents "pleurisy" a disease. (**a**) Presents the relations of DOID_10247 with other classes; (**b**) Presents the details of the relations or properties of the DOID_10247 concept.

5.2. Identification of the data and attributes to be imported

To execute this revision the DOID knowledge graph was saved in Turtle syntax. The code shown in Figure 10 describes the concept **DOID_9352**, which corresponds to the diabetes mellitus type 2 disease. Based on this code we can identify important attributes and relations, such as: obo:IAO_0000115 which is an annotation property used for defining and explaining the meaning of a class or property; oboInOwl:hasDbXref is a qualifier that allows cross referencing other data bases, enabling the recovery of information about the disease by querying the included references; oboInOwl:hasExactSynonym is a property used to include synonyms of the disease; and rdfs:label which is used to describe the disease.

```
###  http://purl.obolibrary.org/obo/DOID_9352
obo:DOID_9352 rdf:type owl:Class ;
              rdfs:subClassOf obo:DOID_9351 ;
              obo:IAO_0000115 "A diabetes mellitus that is characterized by high blood sugar,
                            insulin resistance, and relative lack of insulin."^^xsd:string ;
              oboInOwl:hasDbXref "EFO:0001360"^^xsd:string ,
                                 "ICD10CM:E11"^^xsd:string ,
                                 "KEGG:04930"^^xsd:string ,
                                 "MESH:D003924"^^xsd:string ,
                                 "NCI:C26747"^^xsd:string ,
                                 "OMIM:125853"^^xsd:string ,
                                 "OMIM:601283"^^xsd:string ,
                                 "OMIM:601407"^^xsd:string ,
                                 "OMIM:603694"^^xsd:string ,
                                 "OMIM:608036"^^xsd:string ,
                                 "SNOMEDCT_US_2021_09_01:44054006"^^xsd:string ,
                                 "UMLS_CUI:C0011860"^^xsd:string ;
              oboInOwl:hasExactSynonym "NIDDM"@en ,
                                       "insulin resistance"^^xsd:string ,
                                       "non-insulin-dependent diabetes mellitus"@en ,
                                       "type 2 diabetes"@en ,
                                       "type II diabetes mellitus"@en ;
              oboInOwl:hasOBONamespace "disease_ontology"^^xsd:string ;
              oboInOwl:id "DOID:9352"^^xsd:string ;
              oboInOwl:inSubset doid:DO_FlyBase_slim ,
                                doid:NCIthesaurus ;
              rdfs:comment """Xref MGI.
              OMIM mapping confirmed by DO. [SN]."""^^xsd:string ;
              rdfs:label "type 2 diabetes mellitus"^^xsd:string .
```

Figure 10. Excerpt from the DOID ontology describing the DOID_9352 concept which corresponds to the diabetes mellitus disease.

5.3. Define and implement data structures

To extract and represent disease information as a list of objects the JavaTM classes shown in Figure 11 were implemented.

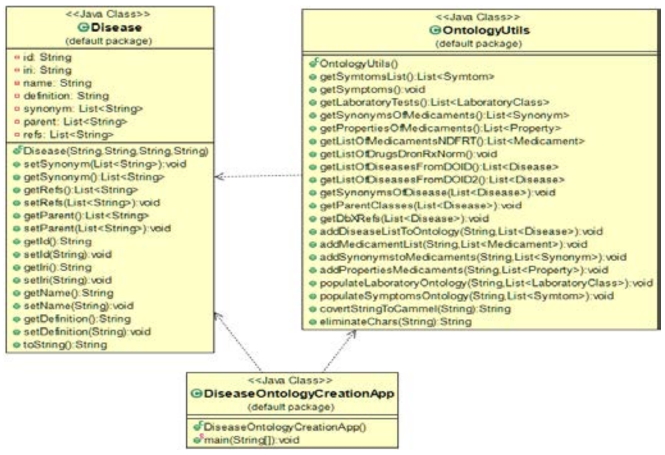

<<Java Class>>
Disease
(default package)
id: String
iri: String
name: String
definition: String
synonym: List<String>
parent: List<String>
refs: List<String>
Disease(String,String,String,String)
setSynonym(List<String>):void
getSynonym():List<String>
getRefs():List<String>
setRefs(List<String>):void
getParent():List<String>
setParent(List<String>):void
getId():String
setId(String):void
getIri():String
setIri(String):void
getName():String
setName(String):void
getDefinition():String
setDefinition(String):void
toString():String
<<Java Class>>
OntologyUtils
(default package)
OntologyUtils()
getSymtomsList():List<Symtom>
getSymptoms():void
getLaboratoryTests():List<LaboratoryClass>
getSynonymsOfMedicaments():List<Synonym>
getPropertiesOfMedicaments():List<Property>
getListOfMedicamentsNDFRT():List<Medicament>
getListOfDrugsDronRxNorm():void
getListOfDiseasesFromDOID():List<Disease>
getListOfDiseasesFromDOID2():List<Disease>
getSynonymsOfDisease(List<Disease>):void
getParentClasses(List<Disease>):void
getDbXRefs(List<Disease>):void
addDiseaseListToOntology(String,List<Disease>):void
addMedicamentList(String,List<Medicament>):void
addSynonymstoMedicaments(String,List<Synonym>):void
addPropertiesMedicaments(String,List<Property>):void
populateLaboratoryOntology(String,List<LaboratoryClass>):void
populateSymptomsOntology(String,List<Symtom>):void
covertStringToCammel(String):String
eliminateChars(String):String
<<Java Class>>
DiseaseOntologyCreationApp
(default package)
DiseaseOntologyCreationApp()
main(String[]):void

Figure 11. Class diagram for the extraction of disease concepts from the DOID ontology, and ontology population of the Disease KGM.

5.4. Implement a program to query the knowledge graph

To obtain the information from the DOID knowledge graph, we implemented various JavaTM programs using the API RDF4J , these programs execute the SPARQL queries shown in Table 3. The first query retrieves the data of diseases from DOID ontology, the rest of the queries are used to complete the definitions with other important data such as the synonyms of each disease, the parent classes, and the set of data base cross references for each disease. It is important to note that the information retrieved and included in the new ontology contains IRI references to enable further queries in case that additional information of a particular disease is required.

Table 3. SPARQL queries executed to retrieve the list of Disease objects from DOID.

SPARQL Query	Typed Collection Returned
PREFIX oboIn: <http://www.geneontology.org/formats/oboInOwl#>	
PREFIX obo: <http://purl.obolibrary.org/obo/>	
SELECT DISTINCT ?s ?label ?def ?synonym ?id	
WHERE {	
?s rdf:type owl:Class .	List<Disease> listDis;
?s rdfs:label ?label .	
?s obo:IAO_0000115 ?def .	
?s oboIn:hasExactSynonym ?synonym .	
?s oboIn:id ?id . }	
PREFIX oboIn: <http://www.geneontology.org/formats/oboInOwl#>	
PREFIX obo: <http://purl.obolibrary.org/obo/>	
SELECT ?syn	List<Disease> listDis;
WHERE {	
obo:" + id + " oboIn:hasExactSynonym ?syn . }	
PREFIX oboIn: <http://www.geneontology.org/formats/oboInOwl#>	
PREFIX obo: <http://purl.obolibrary.org/obo/>	
SELECT ?parent	List<Disease> listDis;
WHERE {	
obo:" + id + " rdfs:subClassOf ?parent . }	
PREFIX oboIn: <http://www.geneontology.org/formats/oboInOwl#>	
PREFIX obo: <http://purl.obolibrary.org/obo/>	
SELECT ?ref	List<Disease> listDis;
WHERE {	
obo:" + id + " oboIn:hasDbXref ?ref . }	

[1] SPARQL queries that are embedded into java code to obtain the list of Disease objects.

5.5. Define and construct the model of the knowledge graph

For the construction of the Disease knowledge graph it is necessary to design the conceptual model, which is, the T-Box of the ontology. Considering the structure of the Disease type returned by SPARQL queries, the ontological model depicted in Figure 12 was developed. This model represents mainly the Disease concepts with the most important references and definitions to other medical data bases, also is a simplified and lighter representation resource compared with the original DOID knowledge graph. The most important difference is the elimination of direct and indirect imported ontologies. The extraction and construction of an ontological representation model without the use of imported ontologies offers benefits in terms of managing memory resources to perform logical inference. Specifically, the representation model developed has the basic attributes that allow diseases to be identified without the need to include a burden of reference models. If required, the concepts can be expanded through queries using the IRIs and the referenced databases.

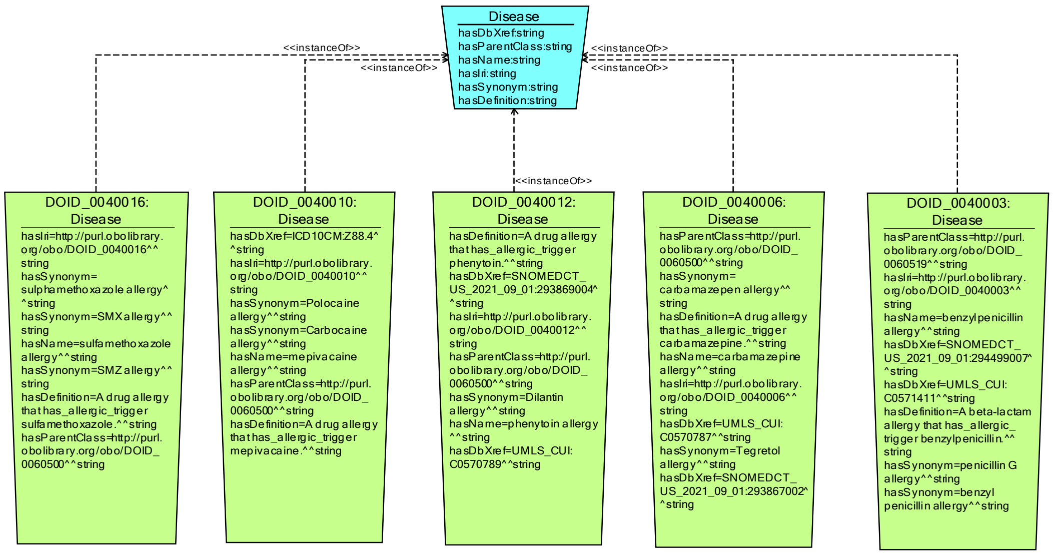

Disease
hasDbXref:string
hasParentClass:string
hasName:string
hasIri:string
hasSynonym:string
hasDefinition:string
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
DOID_0040016: Disease
hasIri=http://purl.obolibrary.org/obo/DOID_0040016^^string
hasSynonym=sulphamethoxazole allergy^^string
hasSynonym=SMXallergy^^string
hasName=sulfamethoxazole allergy^^string
hasSynonym=SMZ allergy^^string
hasDefinition=A drug allergy that has_allergic_trigger sulfamethoxazole.^^string
hasParentClass=http://purl.obolibrary.org/obo/DOID_0060500^^string
DOID_0040010: Disease
hasDbXref=ICD10CM:Z88.4^^string
hasIri=http://purl.obolibrary.org/obo/DOID_0040010^^string
hasSynonym=Polocaine allergy^^string
hasSynonym=Carbocaine allergy^^string
hasName=mepivacaine allergy^^string
hasParentClass=http://purl.obolibrary.org/obo/DOID_0060500^^string
hasDefinition=A drug allergy that has_allergic_trigger mepivacaine.^^string
DOID_0040012: Disease
hasDefinition=A drug allergy that has_allergic_trigger phenytoin.^^string
hasDbXref=SNOMEDCT_US_2021_09_01:293869004^^string
hasIri=http://purl.obolibrary.org/obo/DOID_0040012^^string
hasParentClass=http://purl.obolibrary.org/obo/DOID_0060500^^string
hasSynonym=Dilantin allergy^^string
hasName=phenytoin allergy^^string
hasDbXref=UMLS_CUI:C0570789^^string
DOID_0040006: Disease
hasParentClass=http://purl.obolibrary.org/obo/DOID_0060500^^string
hasSynonym=carbamazepen allergy^^string
hasDefinition=A drug allergy that has_allergic_trigger carbamazepine.^^string
hasName=carbamazepine allergy^^string
hasIri=http://purl.obolibrary.org/obo/DOID_0040006^^string
hasDbXref=UMLS_CUI:C0570787^^string
hasSynonym=Tegretol allergy^^string
hasDbXref=SNOMEDCT_US_2021_09_01:293867002^^string
DOID_0040003: Disease
hasParentClass=http://purl.obolibrary.org/obo/DOID_0060519^^string
hasIri=http://purl.obolibrary.org/obo/DOID_0040003^^string
hasName=benzylpenicillin allergy^^string
hasDbXref=SNOMEDCT_US_2021_09_01:294499007^^string
hasDbXref=UMLS_CUI:C0571411^^string
hasDefinition=A beta-lactam allergy that has_allergic_trigger benzylpenicillin.^^string
hasSynonym=penicillin G allergy^^string
hasSynonym=benzyl penicillin allergy^^string

Figure 12. Ontology model for the representation of Disease information.

5.6. Develop a program to automatically populate the knowledge graph

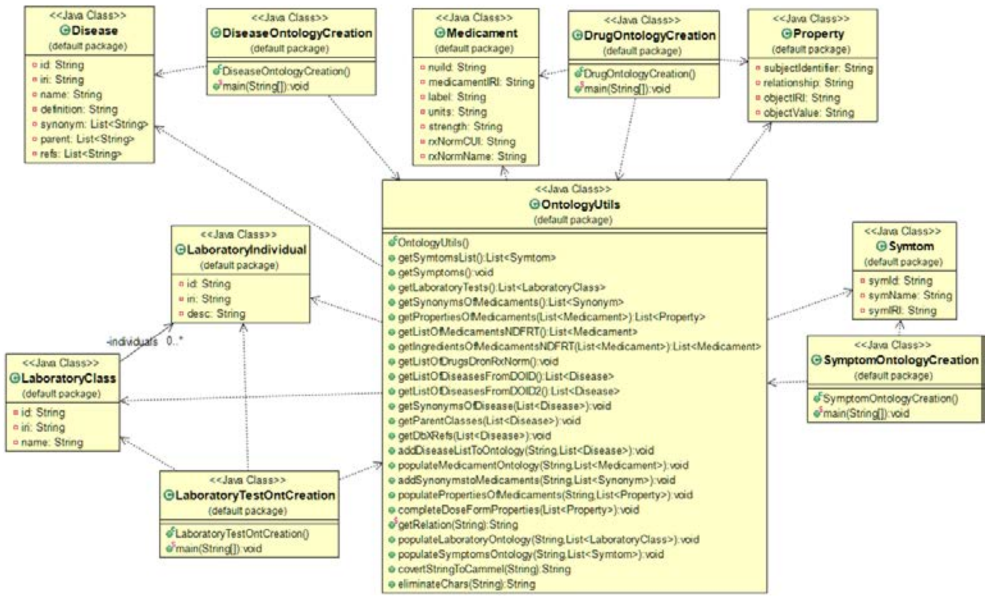
Figure 13 shows the general Java program that was implemented for the construction of the Disease KGM, which starts by executing the program that obtains the list of disease definitions from DOID ontology, then updates the list of disease objects obtaining the synonyms, the parent class, and the data base cross references of each disease. Finally, the program writes the list of disease objects into the Disease ontology file.

```java
public class DiseaseOntologyCreation
{
    //Generate ontology Disease module from DOID ontology
    public static void main(String[] args)
     {
            //Step1. Extract the list of diseases from DOID ontology
            OntologyUtils util = new OntologyUtils();
            List <Disease> listDis = util.getListOfDiseasesFromDOID2();
            //Step 2. Obtain the synonyms of each disease and update the list
            util.getSynonymsOfDisease(listDis);
            //Step 3. Get the parent classes IRIs of each disease and update the list
            util.getParentClasses(listDis);
            //Step 4. Get the data base cross references of each disease and update the list
            util.getDbXRefs(listDis);
            //Step 5. Write the disease individuals into the ontology
            util.addDiseaseListToOntology("Ontologies/Disease.owl", listDis);
     }
   }
```

Figure 13. Main program in which the data extraction, transformation, and module generation tasks are executed in sequence.

6. Knowledge Graph Modularization System

We integrated all the programs developed in an modularization system. This system design is based on the object-oriented programming paradigm which makes it easy to carry out updates or adaptations in any of the tasks of the modularization process. Figure 14 shows the class diagram of the modularization system. The objective of this system is to extract the concepts of interest from diverse medical knowledge graphs, represent these concepts using a object oriented paradigm, and generate specific KGMs that will be reused and usable in the target EHR knowledge graph.



<<Java Class>>
Disease
(default package)
id: String
iri: String
name: String
definition: String
synonym: List<String>
parent: List<String>
refs: List<String>
<<Java Class>>
DiseaseOntologyCreation
(default package)
DiseaseOntologyCreation()
main(String[]):void
<<Java Class>>
Medicament
(default package)
nuiId: String
medicamentIRI: String
label: String
units: String
strength: String
rxNormCUI: String
rxNormName: String
<<Java Class>>
DrugOntologyCreation
(default package)
DrugOntologyCreation()
main(String[]):void
<<Java Class>>
Property
(default package)
subjectIdentifier: String
relationship: String
objectIRI: String
objectValue: String
<<Java Class>>
OntologyUtils
(default package)
OntologyUtils()
getSymtomsList():List<Symtom>
getSymptoms():void
getLaboratoryTests():List<LaboratoryClass>
getSynonymsOfMedicaments():List<Synonym>
getPropertiesOfMedicaments(List<Medicament>):List<Property>
getListOfMedicamentsNDFRT():List<Medicament>
getIngredientsOfMedicamentsNDFRT(List<Medicament>):List<Medicament>
getListOfDrugsDronRxNorm():void
getListOfDiseasesFromDOID():List<Disease>
getListOfDiseasesFromDOID2():List<Disease>
getSynonymsOfDisease(List<Disease>):void
getParentClasses(List<Disease>):void
getDbXRefs(List<Disease>):void
addDiseaseListToOntology(String,List<Disease>):void
populateMedicamentOntology(String,List<Medicament>):void
addSynonymstoMedicaments(String,List<Synonym>):void
populatePropertiesOfMedicaments(String,List<Property>):void
completeDoseFormProperties(List<Property>):void
getRelation(String):String
populateLaboratoryOntology(String,List<LaboratoryClass>):void
populateSymptomsOntology(String,List<Symtom>):void
covertStringToCammel(String):String
eliminateChars(String):String
<<Java Class>>
LaboratoryIndividual
(default package)
id: String
iri: String
desc: String
-individuals 0..*
<<Java Class>>
LaboratoryClass
(default package)
id: String
iri: String
name: String
<<Java Class>>
LaboratoryTestOntCreation
(default package)
LaboratoryTestOntCreation()
main(String[]):void
<<Java Class>>
Symtom
(default package)
symId: String
symName: String
symIRI: String
<<Java Class>>
SymptomOntologyCreation
(default package)
SymptomOntologyCreation()
main(String[]):void

Figure 14. Class diagram of the Knowledge Graph Modularization System.

Applying the described method, the *LaboratoryTest*, *Symptom*, and *Vaccine* KGMs were also generated. As can be seen, the method for generating and reusing medical KGM is semi-automatic. This is because it is necessary to know in detail the knowledge graphs from which the modules will be extracted, in addition to the fact that important decisions must be made regarding what attributes and relationships are required in the target graph.

Another relevant aspect that must be taken into consideration is that the medical knowledge graphs are constantly being updated, for which it is essential to review the changes in the original model and eventually make the pertinent updates in the system.

As a result of the implementation of the method for the generation and reuse of KGMs, four modules were obtained, which are described below.

7. Evaluation

Aiming at evaluating the method reported in this paper, in this section we describe two different approaches to evaluate the resulting modules.

7.1 Evaluation of the usefulness of the knowledge graph modules

According with Duque-Ramos et al. [11] "the quality of an ontology module can be defined as the degree of conformance to functional and non-functional requirements". In the case of the knowledge graph modules generated, they were intended to be used as part of an integral knowledge graph to support management of EHR. With this goal in mind, we review the general knowledge graph and its usefulness to represent patient data with relationships to these graphs. Figure 15 shows the EHR general graph diagram, which presents in light blue the predefined concepts and attributes: *Patient*, *ClinicalDiagnosis*, *StateFederative*, and *Municipality*; this diagram also shows in dark blue the concepts that come from the imported graph modules: *LaboratoryTest*, *NDFRT_Disease*, *PharmacologicalTreatment*, and *COVID-19 Vaccine*.

It is important to mention that this general knowledge graph for the management of EHRs was populated with 500 clinical records of patients. To show the usefulness of the integrated knowledge graph, Figure 16 shows an instance of real data from a female patient who was diagnosed with Coronavirus, this diagnosis is related to the NDFRT_Disease graph.

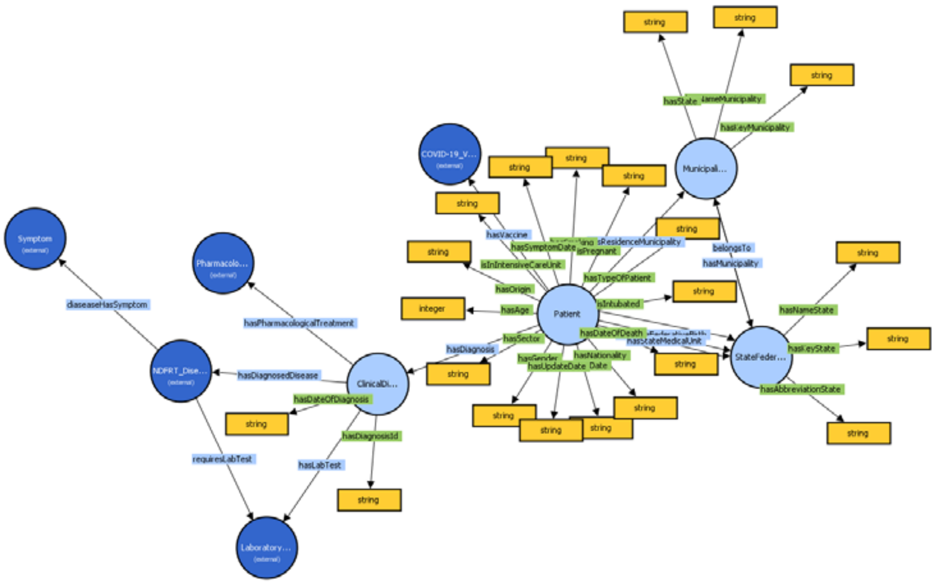
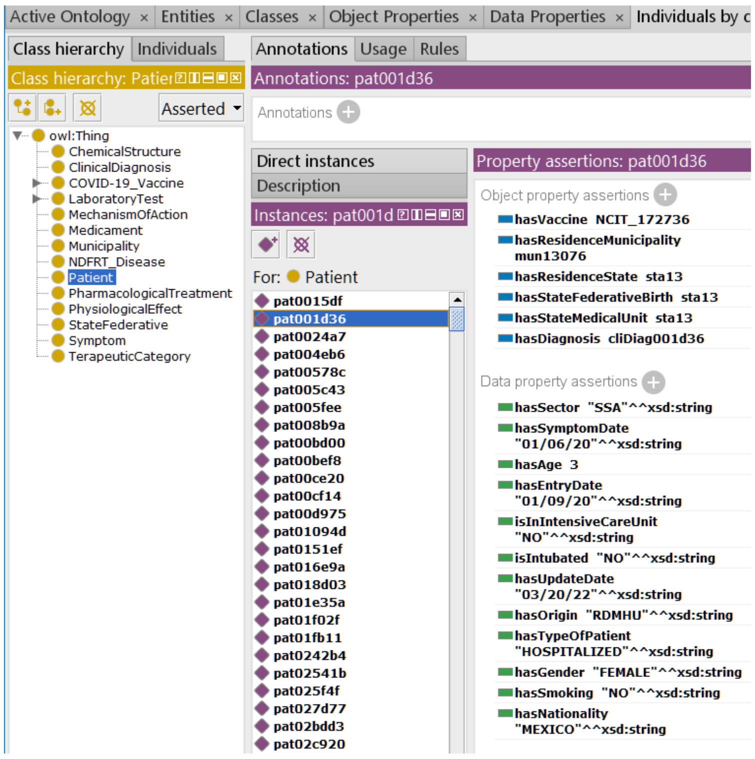


Figure 15. Integrated general knowledge graph.


Active Ontology × Entities × Classes × Object Properties × Data Properties × Individuals by c

Class hierarchy | Individuals

Class hierarchy: Patient

Asserted

- owl:Thing
  - ChemicalStructure
  - ClinicalDiagnosis
  - COVID-19_Vaccine
  - LaboratoryTest
  - MechanismOfAction
  - Medicament
  - Municipality
  - NDFRT_Disease
  - Patient
  - PharmacologicalTreatment
  - PhysiologicalEffect
  - StateFederative
  - Symptom
  - TerapeuticCategory

Annotations | Usage | Rules

Annotations: pat001d36

Annotations

Direct instances

Description

Instances: pat001d

For: Patient

pat0015df
pat001d36
pat0024a7
pat004eb6
pat00578c
pat005c43
pat005fee
pat008b9a
pat00bd00
pat00bef8
pat00ce20
pat00cf14
pat00d975
pat01094d
pat0151ef
pat016e9a
pat018d03
pat01e35a
pat01f02f
pat01fb11
pat0242b4
pat02541b
pat025f4f
pat027d77
pat02bdd3
pat02c920

Property assertions: pat001d36

Object property assertions

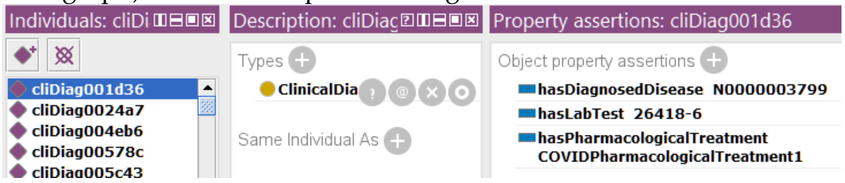
hasVaccine NCIT_172736
hasResidenceMunicipality mun13076
hasResidenceState sta13
hasStateFederativeBirth sta13
hasStateMedicalUnit sta13
hasDiagnosis cliDiag001d36

Data property assertions

hasSector "SSA"^^xsd:string
hasSymptomDate "01/06/20"^^xsd:string
hasAge 3
hasEntryDate "01/09/20"^^xsd:string
isInIntensiveCareUnit "NO"^^xsd:string
isIntubated "NO"^^xsd:string
hasUpdateDate "03/20/22"^^xsd:string
hasOrigin "RDMHU"^^xsd:string
hasTypeOfPatient "HOSPITALIZED"^^xsd:string
hasGender "FEMALE"^^xsd:string
hasSmoking "NO"^^xsd:string
hasNationality "MEXICO"^^xsd:string

Figure 16. Example of an instance of a patient registered in the general graph.

Figure 17 shows the specific diagnosis related to the disease (NDFRT_Disease imported graph), with the laboratory test required for the diagnosis (LaboratoryTest imported graph) and with the pharmacological treatment.


Individuals: cliDi

cliDiag001d36
cliDiag0024a7
cliDiag004eb6
cliDiag00578c
cliDiag005c43

Description: cliDiag

Types

ClinicalDia

Same Individual As

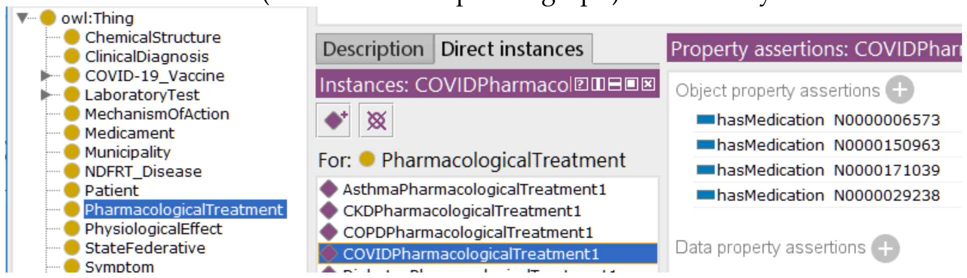
Property assertions: cliDiag001d36

Object property assertions

hasDiagnosedDisease N0000003799
hasLabTest 26418-6
hasPharmacologicalTreatment COVIDPharmacologicalTreatment1

Figure 17. Example of an instance of a diagnosis related with NDFRT_Disease and LaboratoryTest.

Figure 18 shows an example of pharmacological treatment for COVID-19, which is related to the medications (Medicament imported graph) indicated by the doctor.


owl:Thing
ChemicalStructure
ClinicalDiagnosis
COVID-19_Vaccine
LaboratoryTest
MechanismOfAction
Medicament
Municipality
NDFRT_Disease
Patient
PharmacologicalTreatment
PhysiologicalEffect
StateFederative
Symptom

Description | Direct instances

Instances: COVIDPharmaco

For: PharmacologicalTreatment

AsthmaPharmacologicalTreatment1
CKDPharmacologicalTreatment1
COPDPharmacologicalTreatment1
COVIDPharmacologicalTreatment1

Property assertions: COVIDPhar

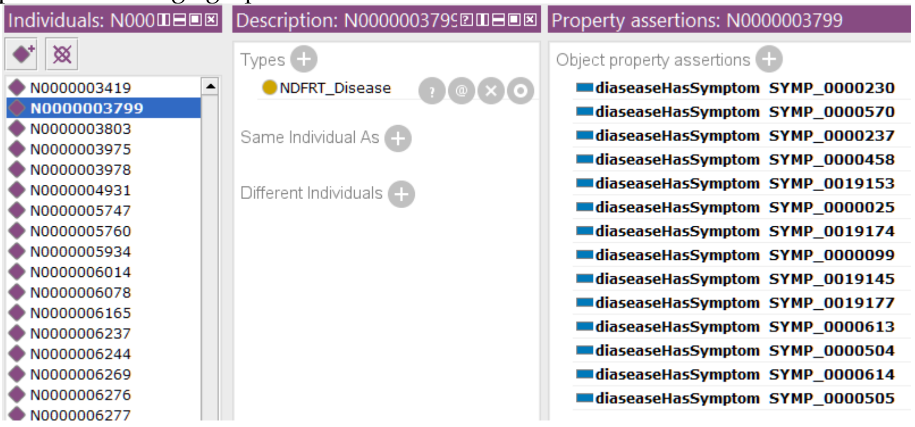
Object property assertions

hasMedication N0000006573
hasMedication N0000150963
hasMedication N0000171039
hasMedication N0000029238

Data property assertions

Figure 18. Example of a Pharmacological treatment related with specific medications.

Figure 19 shows the coronavirus disease related signs and symptoms from the Symptom knowledge graph.


Individuals: N000

N0000003419
N0000003799
N0000003803
N0000003975
N0000003978
N0000004931
N0000005747
N0000005760
N0000005934
N0000006014
N0000006078
N0000006165
N0000006237
N0000006244
N0000006269
N0000006276
N0000006277

Description: N0000003799

Types
NDFRT_Disease

Same Individual As

Different Individuals

Property assertions: N0000003799

Object property assertions

diaseaseHasSymptom SYMP_0000230
diaseaseHasSymptom SYMP_0000570
diaseaseHasSymptom SYMP_0000237
diaseaseHasSymptom SYMP_0000458
diaseaseHasSymptom SYMP_0019153
diaseaseHasSymptom SYMP_0000025
diaseaseHasSymptom SYMP_0019174
diaseaseHasSymptom SYMP_0000099
diaseaseHasSymptom SYMP_0019145
diaseaseHasSymptom SYMP_0019177
diaseaseHasSymptom SYMP_0000613
diaseaseHasSymptom SYMP_0000504
diaseaseHasSymptom SYMP_0000614
diaseaseHasSymptom SYMP_0000505

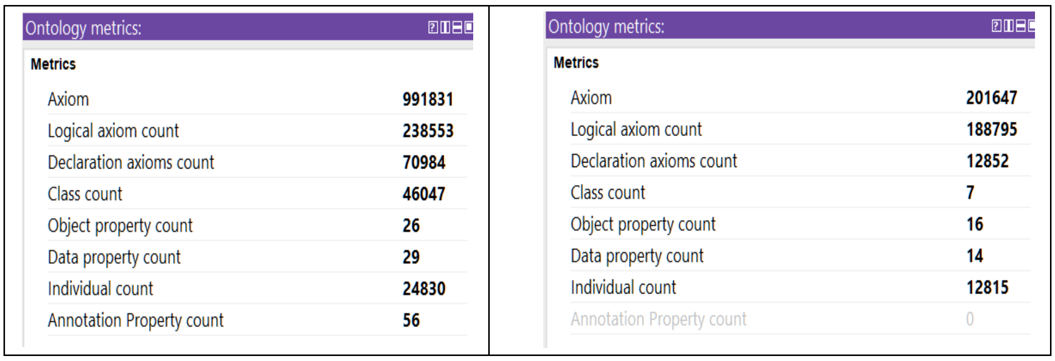
Figure 19. Integrated general knowledge graph.

Based on the initial requirements of the EHR management system, it can be determined that the imported graph modules satisfactorily meet the objective of the model. Therefore, from the application and utility point of view, the graph modules generated by the method described in this article are correctly designed and meet the required information needs.

7.2 Evaluation based on graph metrics

To carry out a quantitative evaluation, a comparison of the metrics of the original knowledge graphs versus the generated knowledge graph modules is described in this subsection. The set of metrics used to make the comparison are those generated by the Protégé ontology editor. Among all the metrics presented, the most important is the number of axioms, which despite being very general, offers a good measure of the reduction achieved with modularization., the general idea is to have an estimation of the size of the module extracted and the reduction achieved. Figure 20 shows on the left side the original NDF-RT metrics, whereas on the right side the metrics of the Medicament KGM generated. As can be seen the extracted Medicament KGM represents the 20.33% of the complete NDF-RT. Other important difference is the number of classes used un the Medicament KGM which is 7, while the original has 46047 classes. This is mainly because the original graph does not define individuals, it handles everything with classes.

NDF-RT original KG metrics	Medicament KGM metrics



Ontology metrics:

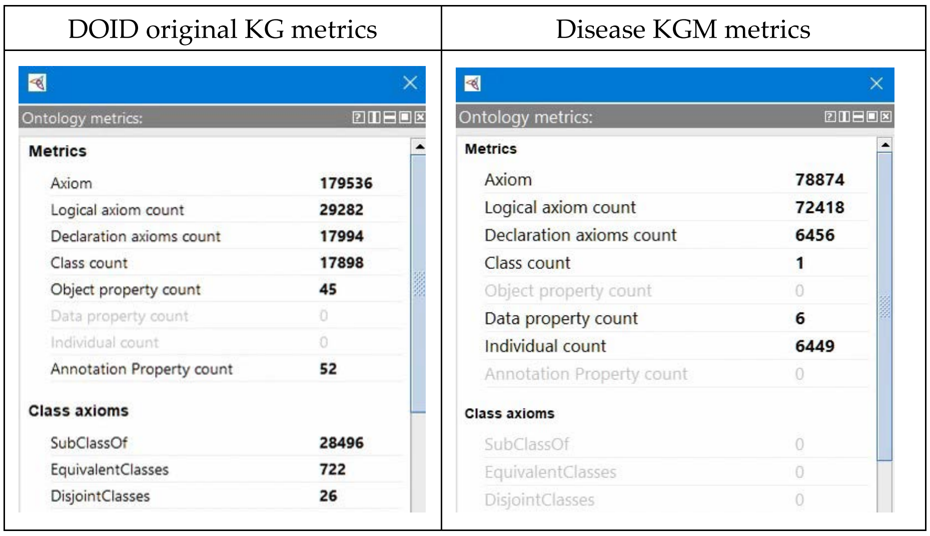
Metrics	
Axiom	991831
Logical axiom count	238553
Declaration axioms count	70984
Class count	46047
Object property count	26
Data property count	29
Individual count	24830
Annotation Property count	56

Ontology metrics:

Metrics	
Axiom	201647
Logical axiom count	188795
Declaration axioms count	12852
Class count	7
Object property count	16
Data property count	14
Individual count	12815
Annotation Property count	0

Figure 20. Comparison of NDF-RT metrics with the Medicament KGM obtained

Figure 21 on the left shows the metrics of the original DOID knowledge graph, and on the right the metrics of the Disease KGM obtained form the modularization process. Disease KGM metrics show 56% reduction in the number of axioms of the original knowledge graph.



DOID original KG metrics

Ontology metrics:

Metrics	
Axiom	179536
Logical axiom count	29282
Declaration axioms count	17994
Class count	17898
Object property count	45
Data property count	0
Individual count	0
Annotation Property count	52
Class axioms	
SubClassOf	28496
EquivalentClasses	722
DisjointClasses	26

Disease KGM metrics

Ontology metrics:

Metrics	
Axiom	78874
Logical axiom count	72418
Declaration axioms count	6456
Class count	1
Object property count	0
Data property count	6
Individual count	6449
Annotation Property count	0
Class axioms	
SubClassOf	0
EquivalentClasses	0
DisjointClasses	0

Figure 21. Comparison of DOID metrics with the Disease KGM obtained

8. Conclusions

One of the difficult problems to solve during the reuse of medical knowledge graphs is that they directly or indirectly import other ontologies or graphs. This generates an overload of models in RAM memory, which prevents reasoning and inference from being carried out more efficiently.

In this article we have described a modularization method applicable to large knowledge graphs; in particular, in graphs from the biomedical area that, in addition to being large in volume, are highly complex for the user. The described modularization method differs from methods that seek to divide a graph into parts, each of which is self-contained. It is also different from fully automatic methods that extract a module considering logically complete modules.

The modularization method described is semi-automatic and is more oriented towards having support tools to solve very specific needs of information or specialized knowledge that must be incorporated into larger and complex knowledge graphs.

The examples of application of the method allow to observe and analyze the specific decision-making that must be done during the design of the knowledge modules, the selection of the necessary attributes to meet the requirements, and that once the design is complete, a mechanism for automatic population is programmed.

It should be noted that the reuse of knowledge graphs entails many challenges, one of the most complicated being that the way in which the structure of the original graph may not be the most appropriate for the destination graph. Therefore, in the examples shown in this article, we have made the decision to make changes in the way of representing the classes and class hierarchies. We have chosen to handle many concepts or classes as individuals in the destination graph.

Author Contributions: Conceptualization, Maricela Bravo and Darinel González-Villarreal; Formal analysis, Maricela Bravo and José-A. Reyes-Ortiz; Investigation, Maricela Bravo, José-A. Reyes-Ortiz and and Leonardo-D. Sánchez-Martínez; Methodology, Maricela Bravo; Software, Darinel González-Villarreal; Validation, Maricela Bravo, José-A. Reyes-Ortiz and and Leonardo-D. Sánchez-Martínez; Writing – original draft, Maricela Bravo, José-A. Reyes-Ortiz and and Leonardo-D. Sánchez-Martínez; Writing – review & editing, Maricela Bravo, Darinel González-Villarreal, José-A. Reyes-Ortiz and Leonardo-D. Sánchez-Martínez.

Funding: "This research received no external funding".

Data Availability Statement: In this section, please provide details regarding where data supporting reported results can be found, including links to publicly archived datasets analyzed or generated during the study. Please refer to suggested Data Availability Statements in section "MDPI Research Data Policies" at https://www.mdpi.com/ethics. If the study did not report any data, you might add "Not applicable" here.

Conflicts of Interest: "The authors declare no conflict of interest." "The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results".

References

1. Rector, A., L. Modularisation of Domain Ontologies Implemented in Description Logics and related formalisms including OWL. Proceedings of the 2nd International Conference on Knowledge Capture, pp. 121-128, 2003.
2. Schlicht, A., Stuckenschmidt, H. Towards Structural Criteria for Ontology Modularization. Proceedings of the ISWC Workshop on Modular Ontologies, 2006.
3. Cuenca, B., Parsia, P., Sirin, E., Kalyanpur, A. Modularity and Web Ontologies. Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006.
4. Grau, Bernardo Cuenca, et al. "A Logical Framework for Modularity of Ontologies." IJCAI. Vol. 2007.
5. Pathak, J., Johnson, T. M., & Chute, C. G. (2009). Survey of modular ontology techniques and their applications in the biomedical domain. Integrated computer-aided engineering, 16(3), 225-242.
6. Courtot, M., Gibson, F., Lister, A. L., Malone, J., Schober, D., Brinkman, R. R., & Ruttenberg, A. (2011). MIREOT: The minimum information to reference an external ontology term. Applied Ontology, 6(1), 23-33.
7. Algergawy, A., Babalou, S., Klan, F., & König-Ries, B. (2020). Ontology modularization with OAPT. Journal on Data Semantics, 9(2), 53-83.
8. Rector, A., Brandt, S., Drummond, N., Horridge, M., Pulestin, C., & Stevens, R. (2012). Engineering use cases for modular development of ontologies in OWL. Applied Ontology, 7(2), 113-132.
9. Shimizu, C., Hammar, K., & Hitzler, P. (2021). Modular ontology modeling. Semantic Web, (Preprint), 1-31.
10. Bravo, M., González, D., Ortiz, J. A. R., & Sánchez, L. "Management of diabetic patient profiles using ontologies". Contaduría y administración, 65(5), 12 (2020).
11. Duque-Ramos, A., Fernández-Breis, J. T., Iniesta, M., Dumontier, M., Aranguren, M. E., Schulz, S., ... & Stevens, R. (2013). Evaluation of the OQuaRE framework for ontology quality. Expert Systems with Applications, 40(7), 2696-2703.