FINAL ASSIGNMENT - MCBPO

Pedro Bueso-Inchausti García, Ignacio Taguas Garzón

1. INTRODUCTION

The goal of this project is to develop an OWL ontology. Our ontology is called *Master in Computational Biology Protein Ontology* (MCBPO, from now on), and its purpose is to provide a consensual knowledge model of proteins. It will be implemented on Protégé.

Proteins are large molecules that have many critical functions in the body. They are encoded by *genes* in the DNA; each gene can encode multiple proteins, but a single protein can only be encoded by one gene. Proteins are made up of hundreds or thousands of smaller units called amino acids, which are attached to one another in long chains. The length of a protein is measured by how many amino acids it contains. There are 20 different types of amino acids that can be combined to make a protein. The sequence of amino acids determines each protein's unique three-dimensional structure, as well as its biological function. Depending on such function, the proteins participate in different biological processes (the different biological tasks performed by living organisms). There are over 20.000 different proteins in the body, each with a specific role.

How do proteins have so much functional diversity, if they are comprised of just 20 amino acids? The reason is that proteins are really complex molecules, with four kind of structures (*Figure 1*):

- The primary structure consists of the amino acid sequence of the protein.
- The secondary structure consists of the small-scale 3D structures caused by the interactions between amino acids. The major classes are α-helixes and β-sheets.
- The tertiary structure is the large-scale 3D conformation of the protein, caused by interactions between secondary structures.
- The quaternary structure is the arrangement of several protein subunits (tertiary structures) in a complex. Different proteins may interact and act as a whole, being protein complexes.

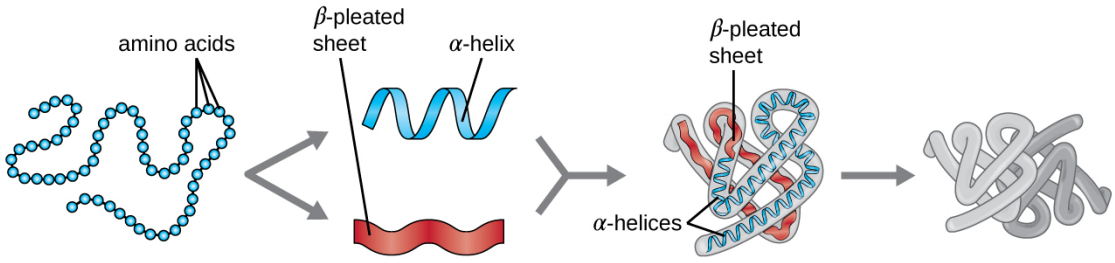

Figure 1. **Schematic representation of the four structures of a protein.** The amino acidic chain folds upon itself to form secondary structures, which in turn fold to form tertiary structures (protein subunits). Lastly, protein subunits join together to form a protein. Taken from https://www.ptglab.com/news/blog/the-complexity-of-proteins/.

As mentioned above, proteins are so complex that it is really difficult to make a clear classification. Thus, there is not one clear way to classify proteins, and the main protein sources often have very different ways of doing it. UniProt, for instance, classifies proteins according to the family they belong to (a protein family is a group of evolutionarily related proteins). Protein Ontology, on the other side, has defined a great number of functions, and classifies proteins according to which of those is its function. Other protein-related sites have many different classification methods.

We believe that the easiest way to classify proteins is by their function; however, the classification used in Protein Ontology is way too complex, since the functions that define the groups are very concrete; therefore, it may be quite difficult to understand for people that are not specifically working on protein-related topics. We have developed an ontology with wider groups, so that a more straightforward classification is available for users without much knowledge in proteins. At the same time, we want our ontology to also be useful to obtain more protein in-depth knowledge, so each protein instance has very specific data regarding its sequence, the gene that encodes it, its function at the most detailed level…

2. REUSED SOURCES

Before going into depth with the design in our ontology, we will briefly talk about the ontological and non-ontological resources we have used to create the ontology, which have highly impacted the final design of MCBPO.

2.1. UNIPROT

The Universal Protein Resource (UniProt — https://www.uniprot.org/) is the most widely used database of protein sequences and functional information. The UniProt Knowledgebase (UniProtKB) provides the central database of protein sequences with accurate, consistent, rich sequence and functional annotation (Morgat *et al.*, 2019). UniProt data is licensed under the Creative Commons Attribution 4.0 Unported License.

UniProt has an RDF model, available at https://www.uniprot.org/core/, and also allows its exploration through a SPARQL endpoint available at https://sparql.uniprot.org/. The RDF file is extremely extensive, so we could not open it on Protégé or any other ontology viewer. Therefore, we decided to use the database in txt format, extract useful data and then convert it programmatically to RDF format.

The only thing we have imported from the UniProt ontology are the URIs for our protein instances (each of the proteins in MCBPO has the same URI as in the UniProt ontology), as well as the URIs for our species instances.

2.2. PROTEIN ONTOLOGY

Protein Ontology (PRO — https://proconsortium.org/pro.shtml) provides an ontological representation of protein-related entities by explicitly defining them and showing the relationships between them (Adams *et al.*, 2011); it is the reference ontology for proteins in the OBO foundry. Unlike UniProt, PRO is a source completely focused on the ontology format, being probably the most useful ontology source in the protein's domain. PRO provides a SPARQL endpoint (https://proconsortium.org/pro_sparql.shtml), as well as Virtuoso SPARQL endpoint server (https://sparql.proconsortium.org/virtuoso/sparql), very clear explanations on how the ontology is organized, tutorials and documentation.

Once again, this ontology is far too big to be opened on Protégé. However, in order to further inspect the ontology, we downloaded OBO Edit 2.3.1 for Linux. OBO-Edit is an open source ontology editor written in Java. It is optimized for reading and writing ontologies in the OBO biological ontology file format, which is a biology-oriented language for building ontologies (Day-Richter *et al.*, 2007). PRO data is licensed under the Creative Commons Attribution 4.0 Unported License.

We have not directly imported any information contained in PRO to our ontology; however, we have gone through the different functions defined as classes in this ontology, grouping them into the final seven functional groups our ontology has.

2.3. GENE ONTOLOGY

Gene Ontology (GO — http://geneontology.org/) describes the gene products (proteins) with respect to their molecular function, cellular component and biological role (Ashburner *et al*., 2000; "The Gene Ontology Consortium", 2019). Gene Ontology Consortium data and data products are licensed under the Creative Commons Attribution 4.0 Unported License.

We have imported the following data from GO: the cellular locations, the biological processes and the functions; all these instances have the same URI in our ontology as they do on GO.

2.4. NCI THESAURUS OBO EDITION

NCI Thesaurus (NCIt — https://ncit.nci.nih.gov/ncitbrowser/) is a reference terminology that includes broad coverage of the cancer domain. The Thesaurus currently contains over 34.000 concepts, and it is used to provide terminology support for numerous portals in the biology field. The NCI Thesaurus is released under the Creative Commons Attribution 4.0 Unported License.

We have imported from this ontology the following classes:

- *Species* (http://purl.obolibrary.org/obo/NCIT_C45293).
- *Protein* (http://purl.obolibrary.org/obo/NCIT_C17021).
- *Gene* (http://purl.obolibrary.org/obo/NCIT_C16612).
- *Subcellular Structure* (http://purl.obolibrary.org/obo/NCIT_C13282, *CellularLocation* in ours).
- *Protein Function* (http://purl.obolibrary.org/obo/NCIT_C18967, *Function* in ours).
- *Biological Process* (http://purl.obolibrary.org/obo/NCIT_C17828, *Process* in ours).
- *Unit* (http://purl.obolibrary.org/obo/NCIT_P355).
- *Value* (http://purl.obolibrary.org/obo/NCIT_C25712).

3. MCBPO SPECIFICATION

3.1. PURPOSE

As was previously mentioned, the purpose of MCBPO is to provide a simple classification model for proteins according to their function, while containing at the same time very specific data of each protein introduced. For this purpose, it will have the following characteristics:

- Proteins will be divided in seven disjoint sets according to their function.
- For each protein, the ontology will give information (if available) for its cellular location, the species it belongs to, in which biological processes it is involved and by which gene is it encoded. MCBPO will also contain other data such as the protein ID, its sequence…

3.2. SCOPE

In order to achieve the previously mentioned goals, the ontology will contain vocabulary related to the concepts of proteins, genes, species and cells. We intend MCBPO to be part of a network of ontologies, being connected with other protein or gene-related ontologies such as UniProt or GO. In order to eventually achieve these purposes, we have reutilized URIs from other ontologies, as explained on *Section 2*.

The expressivity is intended to be high, so the ontology aims to be a heavyweight ontology. This means that the expressiveness is high, with classes, relations, instances and formal axioms, with a detailed classification of the instances introduced and relations between all the instances.

Finally, ours is a domain ontology, which means it has a high usability, but a not high reusability.

3.3. IMPLEMENTATION LANGUAGE

The implementation language of MCBPO will be OWL.

3.4. INTENDED END-USERS

MCBPO is aimed to be used by the following types of users:

1. Users that lack deep knowledge on the protein's domain and look for a general schema containing the main characteristics and groups of proteins.
2. Students
3. Researchers working on genomic analysis, phylogeny and protein engineering…
4. Researchers studying a specific organism or a set of organisms, as in metagenomics.
5. Companies that use proteins for their applications such as food or pharmaceutic companies that are looking for which organisms produce a certain protein (which is highly important, because depending on the organism it is produced by, a protein might be efficiently retrieved or not) or which modifications could be made on the structure.

3.5. INTENDED USES

MCBPO is aimed to have the following uses:

1. Serve as an introduction into the protein's world and functions.
2. Serve as a more in-depth knowledge base.
3. Find a protein's functions, coding gene, amino acid sequence… in order to perform functional analyses.
4. Find whether a protein is produced by a concrete organism, as well as whether an organism has a protein that performs a particular function.
5. Act as a connection for already existing ontologies of the same domain.

3.6. ONTOLOGY REQUIREMENTS

3.6.1. NON-FUNCTIONAL REQUIREMENTS

MCBPO has the following non-functional requirements:

- The ontology must be written in English.
- The ontology must be connected to other ontologies of the same domain.
- The performance of the system must be fast. For such purpose, the ontology will redirect to other ontologies, instead of having redundant information.
- The ontology must be compatible with all operating systems.
- The way of introducing new data must be automatic.
- The ontology must be released under Creative Commons Attribution 4.0 Unported License.

3.6.2. FUNCTIONAL REQUIREMENTS

The ontology functional requirements are defined by a series of competency questions (CQs) and the answers the ontology is expected to give.

<u>CQs related to general metrics</u>
Number of protein classes → A number
Number of proteins → A number
Number of genes → A number
Number of species → A number
Number of locations → A number
Number of functions → A number
Number of processes → A number

<u>CQs related to accessibility</u>
What is the license of MCBPO? → A license

CQs related to identification
What is the URI for a given protein class name? → A protein class URI
What is the URI, name and ID for a protein accession number? → A protein URI and two strings
What is the URI and name for a gene ID? → A gene URI and a string
What is the URI and name for a given species ID? → A species URI and a string
What is the URI and name for a given subcellular location ID? → A location URI and a string
What is the URI and name for a given biological function ID? → A function URI and a string
What is the URI and name for a given biological process ID? → a process URI and a string

CQs related to taxonomies
What class of protein does a given protein belongs to? → A string

CQs related to properties
What gene codifies for a given protein? → A string
Which are the cellular locations in that a given protein is found? → A list of strings
Which are the biological functions that a given protein carries out? → A list of strings
Which are the biological processes in that a given protein is involved? → A list of strings
What is the length and unit of measure of a given protein? → A number and a string
Which proteins are coded by a given gene? → A list of strings
Which proteins appear in a given cellular location? → A list of strings
Which proteins have a given biological function? → A list of strings
Which proteins are involved in a given biological process? → A list of strings

CQs related to interconnectivity
Which proteins have the same name as a given protein? → A list of strings
Which genes have the same name as a given gene? → A list of strings
Which cellular locations are shared by two given proteins? → A list of strings
Which biological functions are shared by two given proteins? → A list of strings
Which biological processes are shared by two given proteins? → A list of strings

3.7. PRE-GLOSSARY OF TERMS

The following terms are used in the ontology:

- Protein
- Catalytic protein
- Immunological/infectious protein
- Regulatory protein
- Signaling/receptor protein
- Storage protein
- Structural protein
- Transport protein
- Species
- Gene
- Subcellular location
- Biological process
- Protein function
- Length
- UniProt

4. MCBPO INGINEERING

Ontological engineering refers to the activities that concern the ontology development process. The most important methodology for ontological engineering is *NeOn methodology*. It considers 55 activities that are carried out when ontology networks are collaboratively built.

4.1. NEON ACTIVITIES

The activities from the NeOn glossary of terms needed for the construction of MCBPO were:

Management activities

The scheduling activity was necessary to organize the work that had to be done in order to construct the ontology.

Development activities

For the development of the ontology, we needed the activities of ontology specification, ontology conceptualization, ontology formalization and ontology implementation.

Support activities

In order to make MCBPO usable by others, we performed ontology summarization, ontology annotation, ontology documentation and ontology integration.

Reuse related activities

We will reuse different ontologies regarding proteins, genes, functions… Therefore, activities like ontology search, ontology selection, ontology reuse, ontology comparison, ontology matching, ontology aligning and ontology merging will be necessary. We will also use information from other sources, so non-ontological resource reuse will be needed. Finally, we also consulted bibliography, so ontology elicitation was also used.

4.2. NEON SCENARIOS

The scenarios that we had to apply where the following:

- S2: Building ontology networks by reusing and reengineering non ontological resources.
- S6: Building ontology networks by reusing, merging, reengineering ontologies or modules.
- S8: Building ontology networks by restructuring ontologies or modules.

5. ONTOLOGY DESIGN

On this section, we will talk about how the ontology was design and the decisions that were made during its construction. *Figure 2* shows a schematic representation of our ontology. We will be reviewing the model in the following sections. The sections of this part follow a chronological order of how we designed the ontology.

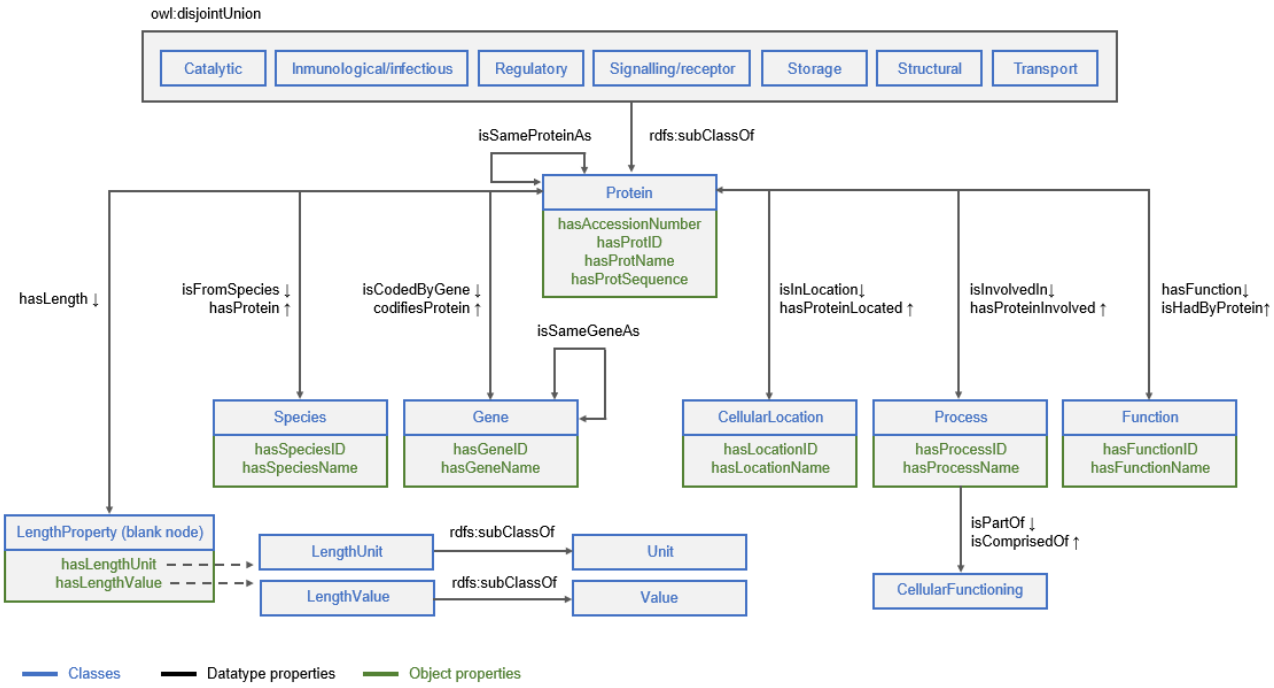

***Figure 2*. Schematic representation of MCBPO.** Classes are represented in blue, data type properties in black and object properties in green. This figure depicts only the model, instances are not included.

5.1. PROTEIN SUBCLASSES

One of the main purposes of this ontology was the construction of a simple classification of proteins. Therefore, we started by searching for a way to make such classification. This proved to be a difficult task, since there is no consensus in the way to classify proteins; some examples are shown on Wolstencroft *et al.*, 2006; Petrey and Honig, 2009; Andreeva *et al.*, 2020.

The first thing we decided was to make a classification by function. Which groups should be used, however, was again a tough task, due to the high diversity of protein functions. We could not find one clear and simple classification of proteins by function, but rather many diffuse and complicated classifications (Aharoni *et al.*, 2005; Jones *et al.*, 2014).

These classifications did not meet our goals, so we decided to do one ourselves. In order to do this, we blend what we had read from the literature and what we learned from Protein Ontology.

As previously explained, to explore the Protein Ontology we used OBO-Edit, which proved to be a really useful tool for ontology visualization. On *Figure 3*, we show a screenshot of how the PRO reasoned ontology in OBO format (taken from https://proconsortium.org/download/current/) can be visualized in OBO Edit. The editor shows four different widows:

- Classes and relationships are shown on the top-left window.
- A query window that allows browsing the ontology is shown on the top-right window. Using it, we can easily access any information we wish (in a much simpler way than using SPARQL).
- The ontology graphical display is shown on the bottom-left window. The class or instance we have currently selected is highlighted, and the path to the highest class is shown.
- The bottom-right window shows the data related to the currently selected class or instance.

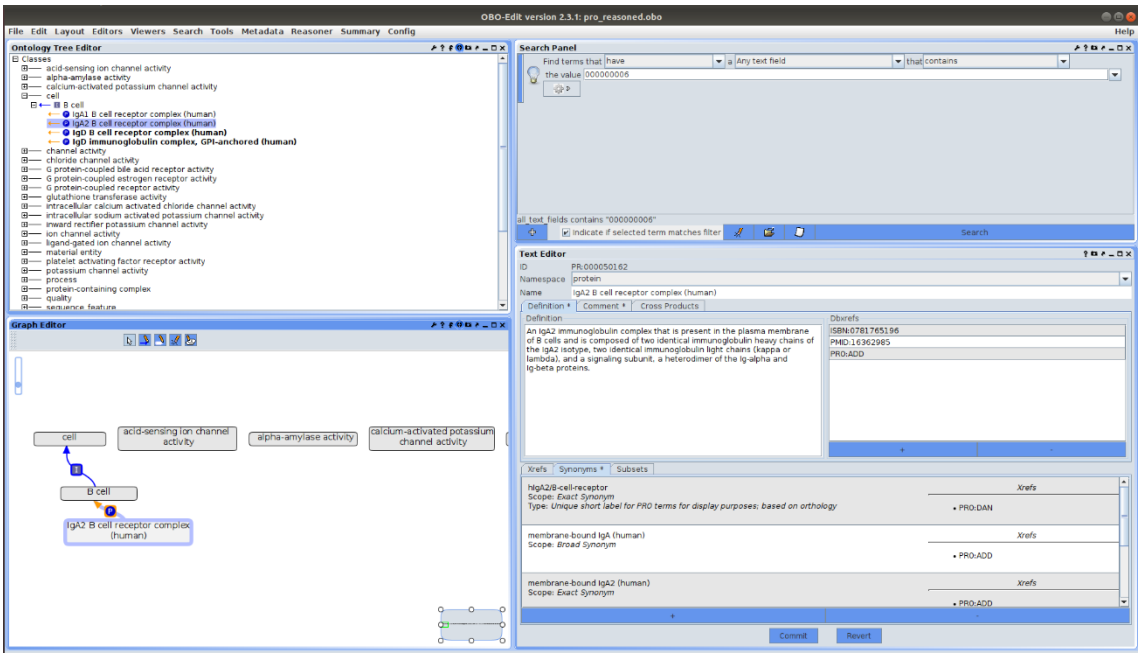

Figure 3. **OBO Edit software visualization.** The Protein Ontology OBO reasoned file is loaded.

Thanks to this, we could explore the ontology. As expected, the protein classification was rather messy. They were mainly classified according to function, but these functions were very specific. As a result, proteins were classified in many different groups. On *Figure 3,* we can see, in the top-left window, multiple examples of these activities: *channel activity*, *G-protein-coupled receptor activity*, *glutathione transferase activity*…

This classification was far from what we wanted to achieve, but it served us as a starting point. We went through all the activities included in PRO and, gathering this information with what we learned from the literature, we reduced the protein functions to just seven (very diverse) groups. These groups are shown and explained on *Table 1*.

Table 1. **Functional groups proteins are divided in on MCBPO.**

Protein group	Basic information
Structural	They provide a structural framework for cells and organs
Storage	They serve as biological reserves for metal ions and amino acids
Transport	They transport molecules and metabolites inside or across the cell
Catalytic	They catalyze metabolic reactions (enzymes)
Regulatory	They regulate the genetic expression
Signaling/receptor	They are involved in the communication between cells
Immunological/infectious	They provide immunity or cause an immune response

Once we had these groups decided, we could start the design of our ontology. The core class of MCBPO is *Protein*. *Protein* has the seven subclasses written in *Table 1*; these classes are disjoint (one protein cannot belong to two classes of proteins), and there can be proteins that belong to neither protein subclass. We originally did not plan to allow unclassified proteins, because the classification is the main purpose of our ontology, but due to a problem with the reasoner (explained on *Section 6.2*) we had to change it.

5.2. DATA RETRIEVAL AND INCORPORATION

Once we had the groups we wanted, we needed to decide what information of the proteins we wanted to include. In order to do this, we accessed the biggest protein database, UniProtKB. We downloaded the whole database as a text file (.dat) from https://www.uniprot.org/downloads. The file occupied more than 3 Gb, so to explore it we had to use a special text editor, EmEditor.

The first thing we had to do was to learn how the information is structured in this file. To achieve this, we used a UniProt help link (https://www.uniprot.org/docs/userman.html).

UniProt has a computer-annotated section, called TrEMBL, that derives from different databases entries. Each entry is composed of various line types. Every line begins with a two-character code, which indicates the type of data contained, and follows with the information itself (in a format that differs from type to type). The line types and their order of appearance is displayed in *Table 2*.

Table 2. **Structure of UniProtKB database entries.**

Line code	Content	Occurrence in an entry
ID	Identification	Once; starts the entry
AC	Accession number(s)	Once or more
DT	Date	Three times
DE	Description	Once or more
GN	Gene name(s)	Optional
OS	Organism species	Once or more
OG	Organelle	Optional
OC	Organism classification	Once or more
OX	Taxonomy cross-reference	Once
OH	Organism host	Optional
RN	Reference number	Once or more
RP	Reference position	Once or more
RC	Reference comment(s)	Optional
RX	Reference cross-reference(s)	Optional
RG	Reference group	Once or more (Optional if RA line)
RA	Reference authors	Once or more (Optional if RG line)
RT	Reference title	Optional
RL	Reference location	Once or more
CC	Comments or notes	Optional
DR	Database cross-references	Optional
PE	Protein existence	Once
KW	Keywords	Optional
FT	Feature table data	Once or more in Swiss-Prot, optional in TrEMBL
SQ	Sequence header	Once
(blanks)	Sequence data	Once or more
//	Termination line	Once; ends the entry

5.2.1. DATA SELECTION AND IMPLEMENTATION OF THE MODEL

After an initial exploration, we defined what we wanted to include, and this conditioned the design of the ontology. For each protein, the information we would keep is:

- Its accession number, its ID and its name.
- Its sequence and the length of the sequence.
- The species where that protein is present (name and ID).
- The gene that codifies the protein (name and ID).
- The subcellular location(s) of the protein (name and ID).
- The biological process(es) the protein is involved in (name and ID).
- The biological function(s) the protein has (name and ID).

To contain this information, the classes *Species*, *Gene*, *CellularLocation*, *Process* and *Function* were created. As previously mentioned, this classes were recycled from the NCIt ontology. All these classes will have as instance name their ID, and then as datatype properties their name. *Protein* will have as instance name its accession number, and as datatype properties its ID, name and sequence.

The length of the protein is harder to define, because it contains both a value (integer) and a unit (amino acids). To define it we created the artificial class *LengthProperty*, which acts as a blank node. We then created a relationship between *Protein* and *LengthProperty* called *hasLength*. We also created the classes *Value* and *Unit* (both taken from NCIt), and their subclasses *LengthValue* and *LengthUnit*. We finally created two datatype properties: *hasLengthValue* and *hasLengthUnit*.

The way proteins will be related to their length is the following: when a protein *A*, with length 400 amino acids, is included as an instance, another instance called *ALength* is created. *ALength* will have the corresponding values for *hasLengthValue* and *hasLengthUnit* (400 and 'amino acids'). *A* and *ALength* will be associated by the object property assertion *A hasLength ALength*.

All datatype properties have a domain (to which class they belong) and a range (the format of the data; for instance, *float* of *string*).

Finally, we created the object properties, which represent relations between classes. All the object properties initially defined were asymmetric and had their inverse, as well as defined domains and ranges.

We also added the class *CellularFunctioning* and added the object properties *Process isPartOf CellularFunctioning* and *CellularFunctioning isComprisedOf Process*. This was done so that all processes are considered as part of the cellular functioning (it is just a symbolic relation, because *CellularFunctioning* has no instances).

Once we had the design ready, the next step was to incorporate data in our ontology. This process required the following steps:

1. Parse the UniProt database to retrieve the information from the database.
2. Convert it to tabular format.
3. Write it in RDF format.

5.2.2. DATA RETRIEVAL AND CONVERSION TO TABULAR FORMAT

The first step in the process consists on parsing the information and including it in a dictionary. Most of the information we wanted to retrieve could be obtained through Biopython parser *SeqIO*, which recognizes the format of the database we had downloaded as '*swiss'*. To help us use it, we used the Biopython documentation (http://biopython.org/DIST/docs/tutorial/Tutorial.html).

Using *SeqIO* we were able to retrieve protein accession, protein identifier, protein name, protein sequence, protein length, protein keyword, gene identifier, gene name, organism identifier and organism name. This parser, however, does not return relevant information like the protein subcellular locations, protein functions and processes in which the protein participates. To have access to it, we had to parse the file ourselves, without automatic parsers.

There was one thing, however, that we were incapable of retrieving because it just does not exist in Uniprot database. This is, in which group of the 7 we had defined can a protein be classified. In order to solve this problem, we designed a lexical classifier based on information that Uniprot gave us like the keywords, locations, functions and processes. The way in which this classifier functions is the following: each group of proteins, from the 7 we had defined, receives a set of terms which are characteristic (there selection is based on previous knowledge from the field); if the studied protein includes one of these terms within its information, it is assigned to such group. This is obviously a naïve classifier; a list of terms is not the most precise approach for prediction and the fact that a protein can appear in many groups or in none is clearly undesired. Nonetheless, the idea looks quite promising. There is not a clear functional prediction of proteins and doing it based on already available annotation could actually be possible.

Once we had the parsed information, we had to convert it to tabular format. From all the selected proteins, we just included in the tabular file those for which the gene identifier is known and which are classified in only one group. The first filter keeps only well-known proteins; the second keeps only proteins that are clearly classified in one group and so appear as appropriate for our ontology. We keep ~120.000 proteins from a total of 560.000. In our ontology we only included a group of 1.198 proteins because using them all would require a ton of time and it would generate a far too big ontology for us to handle. The process, however, is completely automatic, so that all proteins could be included if we wanted.

To sum up, the data retrieval and conversion of information to tabular format included four steps:

- Parse the database using *SeqIO*.
- Parse the database manually.
- Assign proteins a group using our classifier.
- Create the tab-separated file.

All of these is done by a single python file, the script *MCBPO_data.py*.

5.2.3. CONVERSION TO RDF FORMAT

The second step in this process consist on converting the tabular file into RDF. This is usually done with Open Refine, a powerful tool for converting formats. Although Open Refine can be used for manipulating dataframes, we had already done this work in Python. Therefore, we just had to define the RDF schema; this is, how columns are related between them.

However, the use of Open Refine had one inconvenient: we were unable to generate individual dynamical URIs that had a prefix different from the base URI (which impeded us to generate URIs linked to other ontologies). This was a great problem, because one of the main purposes of our ontology was for it to belong to a network of ontologies in the domain. Therefore, we decided to dispose of the Open Refine-generated RDF file and create our own programmatically.

This, again, was a challenge. The objective was to generate the RDF data corresponding to our instances in a separate file, and then paste all the information in the RDF file created by Protégé of our model. The process involved the following steps:

1. Visualization of what the RDF data we were going to generate had to look like.
2. Creation of individual tabular files for the instances of each class in Rstudio.
3. Creation of the RDF data in bash.

The first step was easy: we opened our model in Protégé and created an example instance of each class, with its corresponding properties and relations. Then, we changed the prefixes of the URIs of the instances from the ones given by our ontologies to the following:

- *Protein* instances had the UniProt URI http://purl.uniprot.org/uniprot/*prot_accesion*
- Species instances had the UniProt URI http://www.uniprot.org/taxonomy/*org_identifier*.
- Instances of the classes *CellularLocation*, *Function* and *Process* had the GO URI http://geneontology.org/go#*identifier*.

The second step did not present any difficulties either: we created the script *instances_files.R*, which took as input the file previously generated with Python, and for each class retrieved the relevant information. For the class *Function*, for example, a tab-separated file containing the columns corresponding to the function names and the function IDs was generated. The rows with the same ID (the same instance) were deleted.

This was done for the classes *Function, CellularLocation*, *Process*, *Gene* and *Species*. For the *Protein* instances, we directly used the file generated by python.

The third step was the arduous one. We created a separate script for each class. The scripts for the instances of *Function, CellularLocation, Process, Gene* and *Species* were all very similar. The script for the instances of *Protein* was larger, given that *Protein* instances have many relations.

An important characteristic on how the RDF was generated is how we handled the missing data. For all the instances, for both datatype properties and object properties, if some data was missing, the relationship was included with a *NULL* value. For this purpose, in the range of the datatype properties that were supposed to be numbers, were included as ranges both *int* and *string*.

Once we had the file generated, we pasted it in the RDF file of the model created in Protégé. That way, we already had our model with some instances.

We then proceeded to do a manual model revision. We checked all the classes and relations, as well as some example instances of every class, and we noticed several changes that should be made.

We first noticed that there were species and genes with the same name that had different IDs, and proteins with the same name that had different accession numbers.

First, clearly two species with the same name are the same species. Therefore, we decided that we would fix this by adding to the *Species* instances that had the same name a *Same Individual As* property. To do this, we changed the R script so that it would generate a third column including the IDs of the species that had the same name as the species of the current row. Then, we updated the *bash* script so that it would add the feature *SameIndividualAs* when needed. This achieved us the desired result: *Species* with IDs 10254 and 126794, for example, both have the *SpeciesName "Vaccinia virus"*, and are therefore considered the same individual in our ontology.

As for genes, we were not so sure that we could consider genes with the same name the same individual. We did a little of research and found out that the reason was that those genes were the same, but on different organisms. Therefore, we decided not to use the *Same Individual As*, as we had done with *Species*. Instead, we added a symmetric object property, *isSameGeneAs*, that has *Gene* as both domain and range. Then, we changed the R and bash scripts in a similar manner as we had done with the species, and again got the desired output: *Gene* 1185340, for example, shares name with 9 other genes (*"acdS"*).

Finally, the case for proteins is analog to that of genes, so we added the symmetrical property *isSamePropertyAs*. Protein B2SAT5, for instance, shares name with eight other proteins ("Lectin-like protein BA14k").

6. ONTOLOGY VALIDATION

Once we had finished our ontology, the next step was validation. To validate our ontology, we used three tools:

1. OOPS!
2. HermiT 1.4.3.456
3. RDFlib 4.2.2

6.1. OOPS!

OOPS! (ObtOlogy Pitfall Scanner! — http://oops.linkeddata.es/index.jsp) searches for some of the most common pitfalls in ontologies (Poveda-Villalón *et al.*, 2014). To use it, we just have to access the web page and copy our RDF code. We cannot load the full ontology, so we just add the model without any instances.

The first OOPS! report showed the following pitfalls:

- *P04: Creating unconnected ontology elements*. This pitfall was shown for the *Classes Unit* and *Value*. This, however, is not important, because these classes are there just to represent the blank node, and there is no need for them to be connected to any other class. Therefore, this pitfall was not corrected.
- *P08: Missing annotations.* We had not commented the ontology, so we did. We added annotations to all classes and properties. However, this pitfall appears for every element that does not have both comment and label. We have only defined labels for the classes whose URI was imported (the class *Function* name, for example, is NCIT_C18967, automatically given when we set the URI, so we added the label to see it as *Function*).
- *P11: Missing domain or range in properties*. We had forgotten to set the domain and range of some properties. We do, and afterwards we are only missing the properties *hasName* and *hasID* (their subproperties are the ones with domain and range defined).
- *P13: Inverse relationships not explicitly declared.* The only relationship without inverse is *hasLength*, which is a relationship that goes to an artificial class; therefore, we do not consider it necessary.
- *P19: Defining multiple domains or ranges in properties.* As explained on *Section 5.2.3*, we had defined two ranges for some properties, having into consideration the *NULL* values. This generates the error. We fixed it by setting all property values as strings, because we thought that a number could be considered a string (which was a mistake, as shown on the next section).
- *P22: Using different naming conventions in the ontology*. We are not sure where this is coming from: we have used the CameBack notation for class names and the CammelCase notation for properties. The only thing that might be causing the error are the classes names imported from other ontologies such as NCIT_C18967 for *Function*.
- *P24: Using recursive definitions*. The relationships *isSameGeneAs* and *isSameProteinAs* are recursive. This, however, is a design decision, not a mistake.
- *P41: No license declared*. We declare MCBPO under the Creative Commons Attribution 4.0 Unported License.

6.2. HermiT

HermiT is an ontologies' reasoner written in OWL. It comes by default in Protégé, and can be found at (https://mvnrepository.com/artifact/net.sourceforge.owlapi/org.semanticweb.hermit).

We ran it on Protégé, and our ontology was inconsistent. There were two main reasons for this.

First, we had property values that did not match its range. We had set, for example, the range of *hasLengthValue* as a string instead of an integer because of the *NULL* values. Therefore, all the instances that had a value were raising an inconsistency.

After seeing this, we decided that maybe it was better to just not add the relations when the information was missing (if a gene has no name, show no relation instead of showing the relation *Gene hasGeneName NULL*), so we changed the scripts that generated the instances to obtain this result. This fixed the error.

The other issue was one that we do not understand. We got an error from the disjoint union of the protein classes, even though no protein instance was unclassified. To fix it, we just changed the disjoint union for a disjoint classes relationship.

After this changes, our ontology became consistent.

6.3. RDFlib

As a last evaluation method, we tried to answer the competency questions. To do this, we used RDFlib (https://rdflib.readthedocs.io/en/stable/), which is a Python package working with RDF with a SPARQL 1.1 implementation. All the SPARQL queries and their results are available in the script *MCBPO_queries.py*. *Table 3* also shows the answers we obtained.

Table 3. **Answers to the competency questions.**

Queries	Data used as example	Answers
Number of protein classes	-	7
Number of proteins	-	1.198
Number of genes	-	1.169
Number of species	-	384
Number of locations	-	103
Number of functions	-	236
Number of processes	-	335
What is the license of MCBPO?	-	http://creativecommons.org/licenses/by/4.0/
What is the URI for a given protein class name?	-	http://www.semanticweb.org/a/ontologies/2020/0/MCBPO#Structural
What is the URI, name and ID for a protein accession number?	A0A344	http://purl.uniprot.org/uniprot/A0A344 ACCD_COFAR
What is the URI and name for a gene ID?	4421772	http://www.semanticweb.org/a/ontologies/2020/0/MCBPO/4421772 accD
What is the URI and name for a given species ID?	13443	http://www.uniprot.org/taxonomy/13443 Coffea arabica
What is the URI and name for a given subcellular location ID?	GO:0009317	https://www.ebi.ac.uk/QuickGO/term/GO:0009317 acetyl-CoA carboxylase complex
What is the URI and name for a given biological function ID?	GO:0003989	https://www.ebi.ac.uk/QuickGO/term/GO:0003989 acetyl-CoA carboxylase activity
What is the URI and name for a given biological process ID?	GO:0006633	https://www.ebi.ac.uk/QuickGO/term/GO:0006633 fatty acid biosynthetic process
What class of protein does a given protein belongs to?	P63115	Transport
What gene codifies for a given protein?	P63115	Slc7a10
Which are the cellular locations in that a given protein is found?	P63115	apical dendrite neuronal cell body integral component of plasma membrane
Which are the biological functions that a given protein carries out?	P63115	L-amino acid transmembrane transporter activity
Which are the biological processes in that a given protein is involved?	P63115	D-alanine transport D-serine transport neutral amino acid transport
What is the length and unit of measure of a given protein?	P63115	530 amino acids
Which proteins are coded by a given gene?	79777	Acyl-CoA-binding domain-containing protein 4

Figure 3. Continuation

Which proteins appear in a given cellular location?	GO:0005618	Aspartate aminotransferase, cytoplasmic isozyme 1 Probable cell wall protein ARB_06477 Diacylglycerol acyltransferase/mycolyltransferase Ag85A 34 kDa antigenic protein homolog 35 kDa protein
Which proteins have a given biological function?	GO:0008289	Acyl-CoA-binding domain-containing protein 6 Acyl-CoA-binding protein homolog 3 Putative acyl-CoA-binding protein Acyl-CoA-binding domain-containing protein 5 Acyl-CoA-binding domain-containing protein 4 Acyl-CoA-binding domain-containing protein 2 Virion membrane protein A16 homolog Acyl-CoA-binding protein homolog 1 Acyl-CoA-binding protein
Which proteins are involved in a given biological process?	GO:0010468	Ataxin-7-like protein 3B Putative regulator AbrB
Which proteins have the same name as a given protein?	P63115	Q9NS82
Which genes have the same name as a given gene?	5382663	24153212 34178287 998201
Which cellular locations are shared by two given proteins?	P16909 P14196	chromatin nucleus
Which biological functions are shared by two given proteins?	Q88FY2 D4B1Y1	monooxygenase activity FAD binding
Which biological processes are shared by two given proteins?	A2QT85 A2Q7E0	hemicellulose catabolic process arabinan catabolic process

The process of doing the queries made us conscious about a couple of things that were missing in the ontology. Although we had used the protein accession number and the other classes IDs for building the instances URIs, there was no property that allowed us to retrieve an instance from its ID (which is crucial from the user point of view). This was easily fixed by creating the data type properties: *hasAccessionNumber, hasSpeciesID, hasGeneID, hasLocationID, hasFunctionID* and *hasProcessID*, and adapting the scripts. Another thing we encountered was that our functional groups had a URI but not a label, which is what the user would like to use. This was fixed directly from *Protégé.* Finally, there were some object properties that were using the base prefix incorrectly, which we had to fix programmatically. The fact that we still had some mistakes at so advance stages of our ontology development comes to show that creating ontologies is a clear iterative process that constantly requires from evaluation and rethinking steps.

7. ONTOLOGY DOCUMENTATION

To document our ontology, we used Widoco 1.4.13. Widoco (https://github.com/dgarijo/Widoco/) is a generator of HTML templates to document ontologies. This documentation is attached as *MCBPO_documentation.html*.

8. CONCLUSION AND FUTURE IMPROVEMENTS

As final result, MBCPO is an ontology containing 4.623 instances, 19 classes, 14 object properties and 18 data properties. It also has 57.916 axioms defined.

We are very satisfied with the resulting ontology. We think it achieves all our objectives: it provides a simple protein classification, it is connected to other widely used protein ontologies, we are able to add any number of instances automatically from UniProt… Nevertheless, there are many

implementations that could be made. In this section, we review what we think would be the future steps to improve the ontology.

The first thing that could be improved is the classifier we used in *Section 5.2.2.* As explained on that section, the classifier we developed to divide proteins in our seven functional groups is very simple, but we think it has great potential. Here are some suggestions on how this classifier could be sophisticated:

- Assign weights to the terms so to elucidate between groups.
- Consider combination of terms instead of single terms.
- Include dynamical behavior so the classifier learns new terms associated to each group.

Using this, we could retrieve many more proteins (we obtained ~120.000 proteins from a total of 560.000 with our previous classifier) and add them all of it to our ontology without filters needed.

The contents of the ontology itself could be improved in many ways. The first logical step would be to add interactions between proteins. Proteins often act as complexes, and that is critical for their function. Therefore, it would be useful to add an object property from *Protein* to *Protein* such as *interactsWithProtein* that indicates which proteins interact.

Another useful implementation would be the creation of a class *Pathway*. Biological pathways are series of interactions among molecules in a cell that lead to a certain product or change in the cell. We would define the relation *Pathway isComprisedOf Process*, and then we could see which processes are part of the same pathway (and, indirectly, which proteins are also part of each pathway). To do this, we could use MICV, which is a structured controlled vocabulary for the annotation of experiments concerned with protein interactions.

One last improvement in the content of the ontology would be to add a classification of *Catalytic* proteins, also known as enzymes. Enzymes are the most important type of protein, and there are many stablished classifications for them (contrary to what we found for proteins in general). To help us develop this task, there are already many enzyme ontologies:

- Enzyme Mechanism Ontology (EMO) describes enzymes and their reaction mechanisms. From this ontology we could use classes, properties and axioms specific to enzymes.
- Enzyme Structure Function Ontology (ESFO) includes a hierarchical classification of enzymes that relates sequence, structure and function.
- BRENDA tissue / enzyme source (BTO) is a structured controlled vocabulary for the source of an enzyme comprising tissues, cell lines, cell types and cell cultures.

Finally, the last improvement we think could be done is using the ontology to create a database, which might be more accessible for common people than an ontology.

BIBLIOGRAPHY

Adams N, Hoehndorf R, Gkoutos GV, Hansen G, Hennig C. PIDO: the primary immunodeficiency disease ontology. Bioinformatics. 2011 Nov 15;27(22):3193-9.

Aharoni A., Gaidukov L., Khersonsky O., Gould S. M., Roodveltdt C., Tawfik D. S., The 'evolvability' of promiscuous protein functions, Nature genetics, Volume 37, Issue 1, 1 Jan 2005.

Andreeva A., Kulesha E., Gough J., Murzin A. G., The SCOP database in 2020: expanded classification of representative family and superfamily domains of known protein structures, *Nucleic Acids Research*, Volume 48, Issue D1, 08 January 2020, Pages D376–D382.

Day-Richter J, Harris MA, Haendel M, Gene Ontology OBO-Edit Working Group, Lewis S. OBO-Edit–an ontology editor for biologists. Bioinformatics. Aug 2007;23(16):2198-200.

Jones P., Binns D., Chang H. Y., Fraser M., Li W., McAnulla C., McWilliam H., Maslen H., Mitchell A., Nuka G., Pesseat S., Quinn A. F., Sangrador-Vegas A., Scheremetjew M., Yong S. Y., Lopez R., Hunter S., InterProScan 5: genome-scale protein function classification, *Bioinformatics*, Volume 30, Issue 9, 1 May 2014, Pages 1236–1240

K. Wolstencroft, P. Lord, L. Tabernero, A. Brass, R. Stevens, Protein classification using ontology classification, *Bioinformatics*, Volume 22, Issue 14, 15 July 2006, Pages e530–e538

Morgat A, Lombardot T, Coudert E, Axelsen K, Neto TB, Gehant S, Bansal P, Bolleman J, Gasteiger E, de Castro E, Baratin D, Pozzato M, Xenarios I, Poux S, Redaschi N, Bridge A, UniProt Consortium.

Petrey D, Honig B. Is protein classification necessary? Toward alternative approaches to function annotation. *Curr Opin Struct Biol*. 2009;19(3):363–368.

Poveda-Villalón, M., Gómez-Pérez, A., & Suárez-Figueroa, M. C. (2014). OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation. *International Journal on Semantic Web and Information Systems (IJSWIS), 10*(2), 7-34.

The Gene Ontology Consortium. The Gene Ontology Resource: 20 years and still GOing strong. Nucleic Acids Res. Jan 2019;47(D1):D330-D338.