OPPL-Galaxy: Enhancing ontology exploitation in Galaxy with OPPL

Mikel Egaña Aranguren
Ontology Engineering Group
School of Computer Science
UPM, Spain
megana@fi.upm.es

Jesualdo Tomás Fernández-Breis
School of Computer Science
UM, Spain
jfernand@um.es

Erick Antezana
Department of Biology
NTNU, Norway
erick.antezana@bio.ntnu.no

ABSTRACT
Biomedical ontologies are key to the success of Semantic Web technologies in Life Sciences; therefore, it is important to provide appropriate tools for their development and further exploitation. The Ontology Pre Processor Language (OPPL) can be used for automating the complex manipulation needed to devise biomedical ontologies with richer axiomatic content, which in turn pave the way towards advanced biological data analyses. We present OPPL-Galaxy, an OPPL wrapper for the Galaxy platform, and a series of examples demonstrating its functionality for enriching ontologies. As Galaxy provides an integrated framework to make use of various bioinformatics tools, the functionality delivered by OPPL to manipulate ontologies can be combined along with the tools and workflows devised in Galaxy. As a result, those workflows can be used to perform more thorough analyses of biological information by exploiting extant biological knowledge codified in (enriched) biomedical ontologies.

Categories and Subject Descriptors
J.3 [**Computer Applications**]: Life and medical sciences—*Biology and genetics*

General Terms
Experimentation

Keywords
OWL, Galaxy, OPPL, Bio-ontologies

1. INTRODUCTION
As part of the current, typical life sciences research activities, information is extracted from raw data and shared on the web. A new biological insight is generated by combining that information and the scientist's expertise. Nevertheless, integrating information and generating knowledge out of it is still a challenging task, as the information is frequently codified in computationally opaque formats and dispersed over the web in resources that present different conceptual schemas.

The W3C standards for a prospective Semantic Web[1] provide a means to tackle that issue: RDF[2], SPARQL[3] and OWL[4] are increasingly used by the Life Sciences community to integrate information (RDF), to query it (SPARQL) and to encode consensus knowledge about such information using ontologies (OWL), in the so-called Life Sciences Semantic Web (LSSW).

Biomedical ontologies constitute one of the pillars of the LSSW, since they offer a computationally processable and web-oriented representation of agreed-upon domain knowledge. Projects like the Open Biological and Biomedical Ontologies (OBO) foundry [22] offer free access to curated ontologies like the Gene Ontology (GO) [8]. The functions that biomedical ontologies perform range from intense reasoning [24] to light vocabularies for Linked Data [20]. In order to fulfill such functions, biomedical ontologies should be manipulated to fit scientists' requirements, specially when reusing already existing ontologies: addition or removal of axioms and entities, inference in relation to external ontologies, selective materialisation of inferred axioms, and so forth. Manipulating biomedical ontologies is a laborious task since they are growing in terms of size [16] and contents, that is, axiomatical richness [17]. The Ontology Pre Processor Language[5] (OPPL) offers the possibility of automating ontology manipulation and hence make it more efficient, powerful and less error-prone.

OPPL is beneficial but, like other ontology tools, its functionality cannot be exploited within the bioinformatics processes themselves. Galaxy, a web server for combining different genomics tools in workflows [10], offers the ideal platform for making OPPL part of bioinformatics analyses. Therefore, we have developed OPPL-Galaxy, a tool to execute OPPL scripts from within Galaxy. OPPL-Galaxy enhances OPPL's functionality, *i.e.* automated ontology manipulation, by providing the possibility of directly sending OPPL's output, an improved ontology, to other Galaxy tools. By embedding tools like OPPL in genomic science frameworks like

[1]http://www.w3.org/standards/semanticweb/
[2]http://www.w3.org/standards/techs/rdf
[3]http://www.w3.org/standards/techs/sparql
[4]http://www.w3.org/standards/techs/owl
[5]http://oppl.sf.net

Galaxy the user base of semantic technologies in life sciences increases, since more sophisticated analyses of biomedical information can be performed.

This paper describes OPPL-Galaxy: an overview of its design, implementation and availability is provided in Section 2. Section 3 goes through tested use cases that should be useful for life scientists. Finally, Section 4 discusses OPPL-Galaxy's benefits and foreseen features.

2. OVERVIEW OF OPPL-GALAXY

2.1 OPPL

OPPL offers the possibility of automating the manipulation of ontologies: the OPPL user defines, in an OPPL script, a series of changes to be performed in a concrete ontology when the script is executed. The changes are the addition or removal of axioms according to criteria defined by the OPPL user.

OPPL is based on the OPPL syntax, an extension of the Manchester OWL Syntax (MOS) [11] that includes keywords like `ADD` (to add an axiom), `REMOVE` (to remove an axiom), `SELECT` (to select entities) *etc.* An OPPL script defines an OWL query and some actions that should be performed against the retrieved entities (see Section 3.1). A query can mix variables to be bound by a set of named entities and actual named entities of the target ontology. There are three types of OPPL queries: OWL queries that exploit the automated reasoner, syntactic OWL queries that only work with the asserted axioms, and queries that use a regular expression to match annotation values like `rdfs:label`. The actions are based on the addition or removal of axioms of any complexity to/from the entities retrieved by the query. Once an OPPL script has been defined, the OPPL script and the ontology that will be changed by it are passed to the OPPL engine and it changes the ontology according to the changes described in the OPPL script, generating a new ontology (Figure 1).

OPPL has already shown its utility: it has been used to build an ontology transformation service [23] and for applying Ontology Design Patterns (ODPs) [6, 7, 12, 13]. Also, it is part of Populous, an application for populating ontologies from spreadsheets through the use of ODPs [15].

2.2 Galaxy

Galaxy offers an open, web-based platform for performing genomic analyses. By using Galaxy, different tools can be combined, ranging from simple data manipulations (*e.g.* text parsing) to complex analyses (*e.g.* statistical analysis of Next-Generation Sequencing data). Such combinations can be persistently executed within a single web interface [10]; the output of a tool can be sent to other tools as input, easing the construction of workflows based on recurrent tasks. Those complex analyses can be performed by scientists who are not necessarily computationally skilled. Moreover, a history of all performed actions is saved, so analyses can be reproduced at any time and shared by different users of the system. Galaxy workflows can be built from the users' history and shared by different means, including uploading to myExperiment [9].

Developing Galaxy tools is straightforward, since only a tool definition XML file must be created, containing a description of the tool's web interface and inputs and outputs.

2.3 OPPL-Galaxy

OPPL can be used through the graphical interface of Protégé[6], as part of Populous, or by writing a Java program. Despite those possible means of manipulating ontologies with OPPL, OPPL cannot be used as part of a workflow that might include other bioinformatics analysis tools (*e.g.* gene prediction tools), unless a tailored Java program is written. OPPL-Galaxy fills that niche by offering a version of OPPL that can be used in combination with other bioinformatics Galaxy tools.

In order to create the OPPL-Galaxy tool, an OPPL wrapper was developed to act as an interface between Galaxy, the OPPL 2 API[7] and the OWL API[8], adding a layer on top of both APIs to fit in the Galaxy input and output requirements (Figure 2).

OPPL-Galaxy takes as inputs a target ontology and an OPPL script, both uploaded to Galaxy by the user (or produced as output by another Galaxy tool), and generates a new ontology that has been changed according to the OPPL script, by adding or removing axioms. The OPPL-Galaxy web interface presents the following options (Figure 3):

- Imports (optional): if the input (OWL) ontology imports other ontologies, a flat file that maps the ontology URIs to physical URIs must be uploaded to Galaxy by the user.
- Target ontology: the input ontology that will be modified by the OPPL script. Since OPPL-Galaxy relies on the OWL API for loading and saving ontologies, it can load ontologies in the following formats: OBO flat file, OWL (RDF/XML, OWL/XML, Functional OWL Syntax, MOS), turtle, and KRSS.
- OPPL script: a flat file containing the OPPL script that, when executed, will perform the desired changes in the target ontology (see Section 3.1). This file may be created by using the Protégé OPPL plugin *via* the OPPL text editor (with autocompletion), the OPPL script builder, or the OPPL macros tab.
- Output format: the format that the output ontology should have, OBO or OWL (RDF/XML).
- Add inferred axioms (optional): this option adds the inferred subsumption axioms to the output ontology as asserted axioms.
- Choose reasoner: Pellet[9], HermiT[10] or FaCT++[11] can be used.
- Merge ontologies (optional): if imported ontologies have been used, they can be merged in a single new ontology.

The output ontology can be downloaded from the web interface so that it could be used outside Galaxy, for example with Protégé or OBO-Edit[12], or reused as input for other Galaxy tools like ONTO-toolkit [1].

[6]`http://protege.stanford.edu/`
[7]`http://sourceforge.net/projects/oppl2/files/OPPL\%20API/`
[8]`http://owlapi.sf.net`
[9]`http://clarkparsia.com/pellet/`
[10]`http://www.hermit-reasoner.com/`
[11]`http://code.google.com/p/factplusplus/`
[12]`http://oboedit.org`

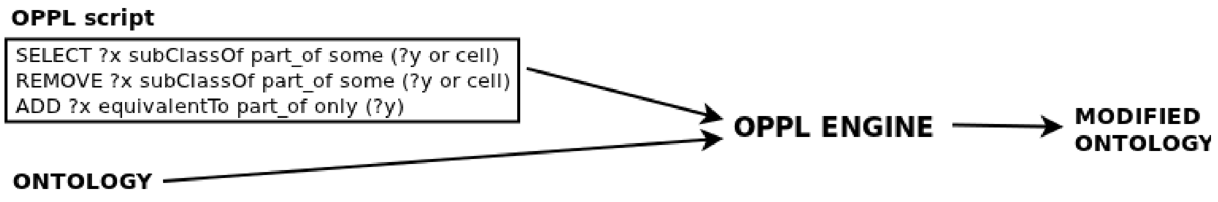

OPPL script
SELECT ?x subClassOf part_of some (?y or cell)
REMOVE ?x subClassOf part_of some (?y or cell)
ADD ?x equivalentTo part_of only (?y)
ONTOLOGY
OPPL ENGINE
MODIFIED ONTOLOGY

Figure 1: OPPL process, including a toy OPPL script (OPPL syntax simplified for the sake of clarity). The OPPL engine takes an ontology and an OPPL script as inputs, and performs the changes defined in the OPPL script in the input ontology, generating a new output ontology. This OPPL script will select any class (`?x`) that is a `subClassOf part_of some (?y or cell)`: `part_of` and `cell` are actual named entities of the ontology, and `?y` can be any class that fits in the OWL expression. Next, it will remove an axiom and add another one to the retrieved entities, generating a new ontology containing the new axioms.

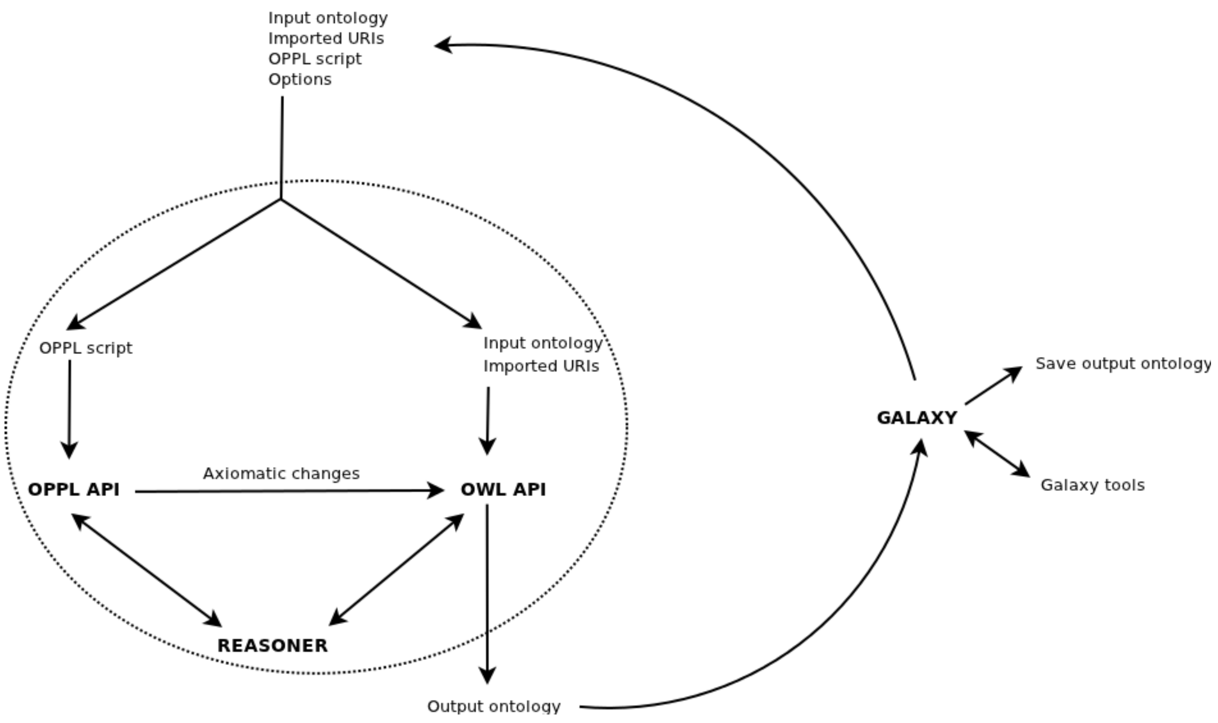
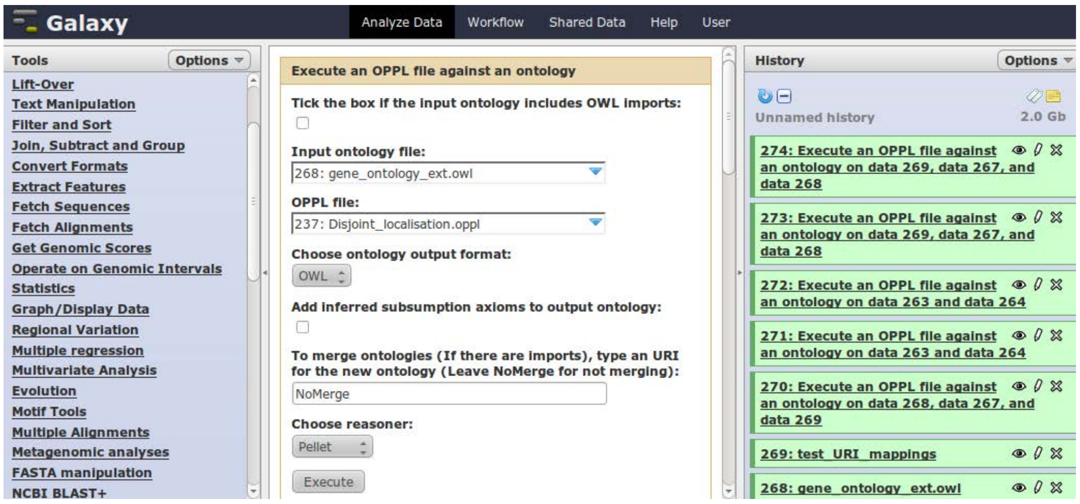


Figure 2: OPPL-Galaxy implementation. The circle represents the OPPL wrapper. Galaxy deals with the data and the parameters that will be passed to the OPPL wrapper. In order to pass, for instance, an ontology to the OPPL wrapper, the ontology must be first uploaded to Galaxy (or passed to it as the output of another Galaxy tool). Also, Galaxy deals with the output of the OPPL wrapper: the output can be redirected to other Galaxy tools or downloaded and saved as an standalone file. The OPPL wrapper coordinates the OPPL API (to parse the OPPL script and execute it), the OWL API (to load ontologies, make changes and save ontologies) and the chosen reasoner (to perform inference).


Galaxy

Analyze Data | Workflow | Shared Data | Help | User

Tools | Options

Lift-Over
Text Manipulation
Filter and Sort
Join, Subtract and Group
Convert Formats
Extract Features
Fetch Sequences
Fetch Alignments
Get Genomic Scores
Operate on Genomic Intervals
Statistics
Graph/Display Data
Regional Variation
Multiple regression
Multivariate Analysis
Evolution
Motif Tools
Multiple Alignments
Metagenomic analyses
FASTA manipulation
NCBI BLAST+

Execute an OPPL file against an ontology

Tick the box if the input ontology includes OWL imports: ☐

Input ontology file: 268: gene_ontology_ext.owl

OPPL file: 237: Disjoint_localisation.oppl

Choose ontology output format: OWL

Add inferred subsumption axioms to output ontology: ☐

To merge ontologies (If there are imports), type an URI for the new ontology (Leave NoMerge for not merging): NoMerge

Choose reasoner: Pellet

Execute

History | Options

Unnamed history 2.0 Gb

274: Execute an OPPL file against an ontology on data 269, data 267, and data 268

273: Execute an OPPL file against an ontology on data 269, data 267, and data 268

272: Execute an OPPL file against an ontology on data 263 and data 264

271: Execute an OPPL file against an ontology on data 263 and data 264

270: Execute an OPPL file against an ontology on data 268, data 267, and data 269

269: test_URI_mappings

268: gene_ontology_ext.owl

Figure 3: OPPL-Galaxy web interface, displayed in the middle. On the left pane, a list of Galaxy tools is shown; on the right pane, the history of the executed Galaxy tools.

2.4 Requirements and availability

In order to use OPPL-Galaxy, Java and Galaxy[13] must be installed in a UNIX machine (GNU/Linux or Mac OS X), since OPPL-Galaxy uses standard UNIX redirection. MS Windows^{TM} is not officially supported by Galaxy and since Galaxy is used as a server (except for tool development), it is recommended that Galaxy be deployed in a UNIX-based machine.

OPPL-Galaxy can be found at the Galaxy Tool Shed[14], under the 'Ontology manipulation' category: the OPPL-Galaxy bundle includes the software itself (along with third-party libraries and XML tool file), sample scripts and ontologies, and instructions on installation and usage. OPPL-Galaxy is open source[15] and is distributed under the General Public License[16].

A public instance of Galaxy with OPPL-Galaxy installed is also available[17].

3. USING OPPL-GALAXY

The following use cases[18] provide some useful examples of how to use OPPL-Galaxy. More OPPL examples can be found at the OPPL scripts site[19].

3.1 Basic usage

The OPPL-Galaxy bundle includes a simple OPPL script for testing purposes. It is described as follows to help the reader understand the rest of the use cases:

```
?whole:CLASS,
?part:CLASS
SELECT
?part SubClassOf part_of some ?whole
WHERE ?part != Nothing
BEGIN
ADD ?part SubClassOf part_of only ?whole
END;
```

[13]http://galaxy.psu.edu/
[14]http://toolshed.g2.bx.psu.edu/
[15]http://toolshed.g2.bx.psu.edu/repos/mikel-egana-aranguren/oppl
[16]http://www.gnu.org/copyleft/gpl.html
[17]http://sele.inf.um.es:8080/
[18]http://miuras.inf.um.es/OPPL-Galaxy
[19]http://oppl2.sourceforge.net/taggedexamples/

First, the variables that will be used and their type is declared: `?whole` and `?part` should be bound by OWL classes (`CLASS`). The script queries the reasoner (`SELECT` clause) for the classes that are subclasses of `part_of some ?whole` (note the use of MOS, `SubClassOf part_of some`). The retrieved classes should be satisfiable (`!= Nothing`). Then, the axiom `SubClassOf part_of only ?whole` will be added to them. `part_of` is a named entity in the ontology; `?part` and `?whole` are variables defining groups of classes.

3.2 Ontology debugging and evaluation

Ontology debugging can be a daunting activity, specially if the ontology the scientist is working with has not been developed in-house and/or if it presents a complex axiomatisation over many entities. OPPL-Galaxy can be used for detecting and fixing certain structures that are considered bad practices (antipatterns) or at least 'suspicious'. The detection of antipatterns also offers a 'picture' of the ontology: it can be used to evaluate the overall structure of the ontology as a further element to judge the ontology's quality. OPPL-Galaxy offers a means of defining antipatterns as 'test units' that can be run against ontologies automatically, as part of Galaxy workflows.

The notion of antipatterns in ontologies has already been defined [4, 21]. For example [21] mentions using the OWL universal restriction (`only`) without any other restriction like `some` on the same property (exclusive universal) as a potential antipattern. The OWL universal restriction, on its own, can be trivially satisfied by an empty class, *e.g.* `A subclassof p only (B and C)` can be satisfiable even when `B disjointWith C`, since the semantics of `only` state that *if* there is a relation, it must be to `(B and C)`, or *none*: `(B and C)` is empty and therefore would be the *none* case.

The exclusive universal structure can be detected in, for example, BioPAX[20], by the following OPPL script:

```
?target:CLASS,
?prop:OBJECTPROPERTY,
?filler:CLASS
SELECT ASSERTED ?target SubClassOf ?prop only ?filler
WHERE FAIL ?target SubClassOf ?prop some ?filler
BEGIN
ADD ?target SubClassOf !OnlyBadPracticeResult
END;
```

This script detects the exclusive universal structure and adds all the classes that present it as subclasses of `OnlyBadPracticeResult`[21], a class created on the fly if it does not exist in the ontology (`!` symbol). Note the use of the `ASSERTED` keyword (the reasoner is deactivated for querying in order to improve performance) and the `FAIL` keyword (negation as failure is used to detect *absent* existential restrictions, something out of the scope of OWL semantics).

More antipatterns can be found in the collection presented in [4]:

- Logical Antipatterns (LAP): detectable by an automated reasoner, *e.g.* unsatisfiable classes.
- Non-Logical Antipatterns (NLAP): modelling errors that are not detectable by a reasoner, usually due to a misunderstanding of the language semantics, *i.e.* the logical consequences of the statements made.
- Guidelines (G): alternative, simpler axiomatic expressions of the same knowledge.

SynonymeOfEquivalence (SOE) is an example of a NLAP. Such antipattern describes the situation in which two classes are declared equivalent and both pertain to the same ontology (*i.e.* they have not been imported). Generally that means that the developer intends to model a synonym, which should be an `rdfs:label` string, as a whole class. Such structure can be found, for example, in the NIF Gross Anatomy ontology[22], using the following script (it also removes the structure):

```
?target:CLASS,
?filler:CLASS
SELECT ASSERTED
?target equivalentTo ?filler
BEGIN
REMOVE ?target equivalentTo ?filler
END;
```

We do not claim that these structures (exclusive universal in BioPAX and SOE in NIF Gross Anatomy) are erroneous *per se*. We rather state that, according to the experience of the authors of [21], [4] and ours, they are modelling practices that may yield unexpected results when automated reasoning is applied. Therefore, the scientist who is reusing those ontologies should be aware of the existence of the mentioned antipatterns. OPPL-Galaxy is an straightforward, powerful and flexible (any antipattern can be defined by the scientist) tool to detect such antipatterns *en masse*, and, even more useful, as part of automated Galaxy workflows.

[20]`http://www.biopax.org/release/biopax-level3.owl`

[21]This script detects any case in which a universal restriction is used in the absence of an existential restriction. Therefore, it would (wrongly) flag as an instance of the antipattern, for example, a universal restriction and a `exactly` restriction used together. A more thorough script is feasible but out of the scope of this paper.

[22]`http://ontology.neuinfo.org/NIF/BiomaterialEntities/NIF-GrossAnatomy.owl`

3.3 OWL punning

OWL punning is a feature of OWL 2 that makes it possible for different entities to have the same URI[23], acting as different 'views' on the same entity. The entities are differentiated by the reasoner using their axiomatic context. OWL punning can be useful, for example, when adding an OBO-formatted ontology to a Knowledge Base (KB) that will be queried using SPARQL. OBO terms are mapped to OWL by all the existing OBO to OWL mappings as OWL classes[24], but it may be useful to model them also as individuals, for example for performing more succinct SPARQL queries (querying directly for triples rather than for the triple-based serialisation of OWL axioms). Therefore, to have both classes (for OWL queries) and individuals (for more 'comfortable' SPARQL queries), it makes sense to add, for every class, an individual with the same URI, *i.e.* to use OWL punning in the ontology [14]. The following OPPL script can be used for such task:

```
?x:CLASS,
?y:INDIVIDUAL = create(?x.RENDERING)
SELECT ?x SubClassOf Thing
WHERE ?x != Nothing, ?x != Thing
BEGIN
ADD ?y Type ?x
END;
```

By applying this script a 'punned' ontology can be quickly obtained: the script adds an individual as a member of each class, with the same URI as the class (`?x.RENDERING`), except in the case of `owl:Thing` and `owl:Nothing`. Thus, an ontology in which each class has an individual with the same URI is obtained. Triples from existential restrictions can be added to the punned ontology executing the following script (using the punned ontology as input):

```
?x:CLASS,
?y:INDIVIDUAL,
?z:CLASS,
?w:INDIVIDUAL,
?p:OBJECTPROPERTY
SELECT ASSERTED ?x SubClassOf ?p some ?z,
ASSERTED ?y Type ?x, ASSERTED ?w Type ?z
WHERE ?x != Nothing, ?x != Thing
BEGIN
ADD ?y ?p ?w
END;
```

This script will only work for existential restrictions, *i.e.* it will not transform universal restrictions to triples. Therefore, it will completely transform an ontology that only presents existential restrictions, like GO. By using these scripts sequentially in a Galaxy workflow a ready-to-use (OWL) RDF representation of an OBO-formatted ontology is obtained.

3.4 Ontology refactoring

It is often necessary to refactor (*i.e.* change) an axiomatic representation for another one to improve, for instance, querying, maintenance, or inference in a given ontology. Such a

[23]`http://www.w3.org/TR/owl2-new-features/#F12:_Punning`

[24]`http://berkeleybop.org/~cjm/obo2owl/obo-syntax.html`

refactoring can be regarded as an application of ODPs [2]. For example, the following script, taken from [23], transforms an ontology that follows the Entity-Property-Quality ODP[25] into an ontology that presents the Entity-Feature-Value ODP[26] [6]:

```
1  ?x:CLASS,
2  ?y:OBJECTPROPERTY = MATCH("has((\w+))"),
3  ?z:CLASS,
4  ?feature:CLASS = create(?y.GROUPS(1))
5  SELECT ASSERTED ?x subClassOf ?y some ?z
6  BEGIN
7  REMOVE ?x subClassOf ?y some ?z,
8  ADD ?x subClassOf !hasFeature some
9  (?feature and !hasValue some ?z)
10 END;
```

This script retrieves any object property whose URI fragment matches the regular expression defined in line 2. It then creates a class using the first group of the matched string (`?y.GROUPS(1)`) and refactors any existing relation, creating the `hasFeature` object property (`!hasFeature`).

This script changes the ontology at the points that match the query. OPPL can be used to expand complex modelling that has been encapsulated in one script to different parts of the ontology, a process that, if performed manually, is error-prone. By using OPPL-Galaxy, the complex modelling encapsulated in the script can be applied (parameterised with the actual named entities of the ontology), for example, every time an ontology is updated, to ensure that the desired representation is used for analysing biomedical information.

3.5 OPPL-Galaxy as part of Galaxy workflows

OPPL-Galaxy can be combined with other Galaxy tools to build complex workflows like the one shown in Figure 4. The workflow can be used by a scientist interested in proteins that act on GO biological processes involving Hepatocytes that are not localisation processes. The workflow executes two OPPL scripts, an ONTO-toolkit function (get parents), and the Galaxy default tool for comparing two datasets. Thus, it combines three Galaxy tools to retrieve exactly the proteins that the scientist is interested in.

The first OPPL script makes all the siblings of localisation (`GO_0051179`) disjoint to it:

```
?localisation_sibling:CLASS
SELECT
ASSERTED ?localisation_sibling SubClassOf GO_0008150
WHERE ?localisation_sibling != GO_0051179
BEGIN
ADD ?localisation_sibling DisjointWith GO_0051179
END;
```

The second OPPL script gets the output of the first OPPL script (a new version of GO with more disjoint axioms) and queries the resulting ontology for the biological processes that have 'Hepatocyte' as part of their names and are related *via* `part_of` or `regulates` to a biological process that is not localisation. It adds the relation `actsOn` to every resulting class.

```
?hepatocyte_process:CLASS,
?hepatocyte_process_label:CONSTANT
= MATCH(".?hepatocyte.+"),
?has_part_hepatocyte_process:CLASS,
?part_of_or_regulates:OBJECTPROPERTY
SELECT
ASSERTED ?hepatocyte_process.IRI
label ?hepatocyte_process_label, ?hepatocyte_process
subClassOf ?part_of_or_regulates some
(?has_part_hepatocyte_process and not GO_0051179)
WHERE ?hepatocyte_process != GO_0008150
BEGIN
ADD ?hepatocyte_process subClassOf !actsOn some
?has_part_hepatocyte_process
END;
```

The resulting ontology is processed by ONTO-toolkit to get the parents of a term[27], specially the ones through the relationship `actsOn`, to obtain the processes in which the scientist is interested. Then, the Galaxy tool for comparing two datasets is used to extract the proteins involved in the resulting processes of interest, using the GO parent terms as keys against a Gene Ontology Annotation (GOA) file [3].

This workflow shows some of the advantages provided by OPPL-Galaxy:

- Applying disjoint axioms, by executing a script on the fly, each time GO is updated. In this case disjoint axioms involving only a concrete class have been used in order to make the workflow faster. OPPL variables can be used to make the whole GO sibling-wise disjoint.
- Mixing text processing (in this case the regular expression (`".?hepatocyte.+"`)) and automated reasoning (in this case: `subClassOf` transitivity, `subPropertyOf`, `disjointFrom`, and `part_of` transitivity) in the same query.
- Referring to groups of entities *via* variables: `part_of` and `regulates` are represented by the same variable `?part_of_or_regulates`, including the subproperties `negatively_regulates` and `positively_regulates` due to OWL semantics.

These advantages are enhanced by the fact that OPPL is used within Galaxy: the process can be repeated with any new version of GO, it can be shared with other scientists, combined with other tools (in this case manipulation of GOA files), and modified or ran in parallel.

4. DISCUSSION AND CONCLUSIONS

The success of the application of the Semantic Web technologies in Life Sciences not only relies on building ontologies and fine tuning current standards but also on creating tools that can be exploited as part of frequently-used data analysis environments such as Galaxy. Galaxy facilitates the combination of several bioinformatics tools (sequence analysis, phylogenetics tools, *etc.*) in a single web interface. Since OPPL-Galaxy can be used as part of the Galaxy framework as an ontology manipulation tool, it can be exploited in combination with other Galaxy tools.

[25]`http://www.gong.manchester.ac.uk/odp/html/Entity_Property_Quality.html`

[26]`http://www.gong.manchester.ac.uk/odp/html/Entity_Feature_Value.html`

[27]In this case `GO_0048175` is used for querying to illustrate the workflow in a simple way, making the first script redundant. A further OPPL script and ONTO-toolkit function should be used to retrieve all the modified (`actsOn` added) classes, to make the whole process more abstract, without the need for a query including a concrete class.

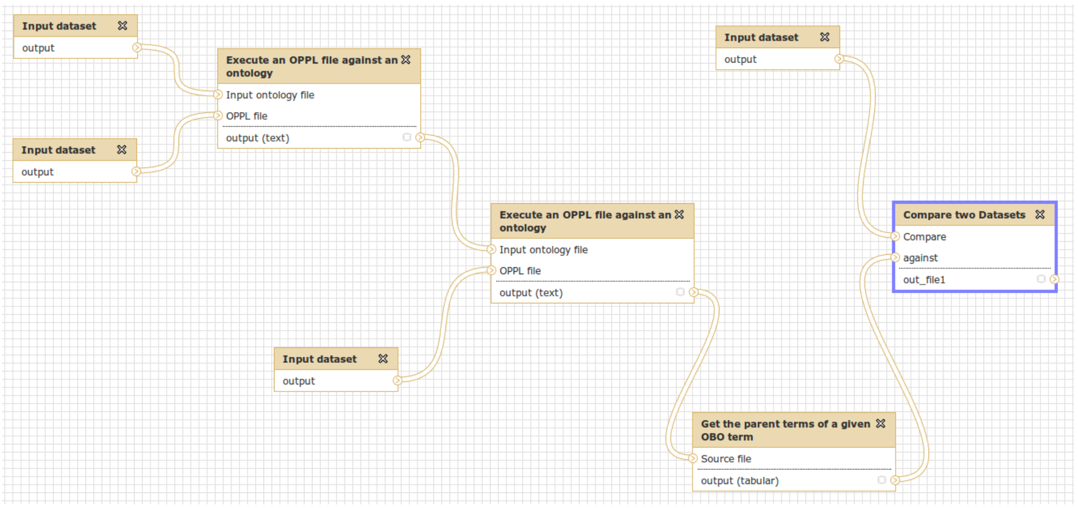


Figure 4: OPPL-Galaxy workflow for exploiting an enriched version of GO against GOA files. This workflow also involves ONTO-toolkit and the Galaxy default tool for data analysis.

That is, precisely, what sets OPPL-Galaxy apart from other ontology tools that offer similar functionality: it can be used with the actual data and tools that life scientists use in a daily basis, instead of in isolation. To the best of our knowledge, there is no Galaxy tool comparable to OPPL-Galaxy except ONTO-toolkit. However, they offer very different functionalities that do not overlap and in fact can be combined to obtain meaningful results, as described in the workflow of Section 3.5.

The OPPL syntax extends the OWL syntax with intuitive keywords, and therefore it is not difficult to learn for a user minimally fluent in OWL. That means that OPPL-Galaxy offers a powerful and familiar tool for automating ontology curation processes that otherwise would need considerable human resources and produce incomplete results. The OPPL scripts described in Section 3 are relatively simple, yet they add benefits to ontology development and exploitation like debugging, refactoring and axiomatic enrichment *via* ODPs. Specially in the case of ODPs, a well-known ontology engineering practice, OPPL-Galaxy offers the ideal setting for their application, since such ODPs can be shared as ready to execute Galaxy workflows, saving time and effort. More complex OPPL scripts would undoubtedly yield even greater benefit, particularly if combined in workflows (*e.g.* debugging and refactoring sequentially and sending the output to other Galaxy tools).

An example of a Galaxy workflow that combines different OPPL scripts with other Galaxy tools is provided in Section 3.5. Such a workflow is simple for the sake of clarity, but it shows what can be achieved with OPPL-Galaxy. More sophisticated analyses can be performed in workflows exploiting OPPL-Galaxy, like more fine-grained axiomatic enrichment of biomedical ontologies [5, 7, 18, 19] or the application of the method described in [1] for extraction of modules from biomedical ontologies, but, in the case of OPPL-Galaxy, additionally applying inference. The diversity and functionality of Galaxy workflows involving OPPL-Galaxy depends only on the user.

In summary, OPPL-Galaxy offers the possibility of automating ontology manipulations in a reproducible, persistent and shareable fashion, in a context in which the result of such manipulations can be directly sent to further tools in order to build powerful workflows. Therefore, OPPL-Galaxy could, on the one hand, be of interest for ontologists that maintain ontologies and, on the other hand, for life scientists that exploit ontologies to analyse biomedical information.

OPPL-Galaxy is a seminal prototype that is continuously improved. Foreseen features include a support for OWLLink[28] and, in the case of adding inferred axioms to the output ontology as asserted axioms, the possibility of choosing other axioms apart from subsumption between named classes (equivalence axioms, type axioms, *etc.*). Performance can be an issue while working with OPPL-Galaxy, since performing inferences on biomedical ontologies is resource demanding, even considering that OPPL-Galaxy will normally work in a server with considerable memory. Such performance depends on automated reasoners, and it is expected to increase in the future, as reasoners become more efficient.

5. ACKNOWLEDGEMENTS

MEA is funded by the Marie Curie Cofund programme (FP7). JTFB is funded by the Spanish Ministry of Science and Innovation (TIN2010-21388-C02-02).

6. REFERENCES

[1] E. Antezana, A. Venkatesan, C. Mungall, V. Mironov, and M. Kuiper. ONTO-ToolKit: enabling bio-ontology engineering via galaxy. *BMC bioinformatics*, (Suppl 12):S8+, 2010.

[2] M. E. Aranguren, E. Antezana, M. Kuiper, and R. Stevens. Ontology Design Patterns for bio-ontologies: a case study on the Cell Cycle Ontology. *BMC bioinformatics*, 9(Suppl 5):S1, 2008.

[3] E. Camon, M. Magrane, D. Barrell, V. Lee, E. Dimmer, J. Maslen, D. Binns, N. Harte, R. Lopez,

[28] `http://www.owllink.org/`

and R. Apweiler. The Gene Ontology Annotation (GOA) Database: sharing knowledge in Uniprot with Gene Ontology. *Nucleic Acids Res*, 32:D262–D266, Jan 2004.

[4] O. Corcho, C. Roussey, L. M. Vilches Blázquez, and I. Pérez. Pattern-based OWL Ontology Debugging Guidelines. In F. S. V. S. Eva Blomqvist, Kurt Sandkuhl, editor, *WOP, ISWC*, pages 68–82, 2009.

[5] A. D. Diehl, A. D. Augustine, J. A. Blake, L. G. Cowell, E. S. Gold, T. A. Gondré-Lewis, A. M. Masci, T. F. Meehan, P. A. Morel, A. Nijnik, B. Peters, B. Pulendran, R. H. Scheuermann, Q. A. Yao, M. S. Zand, and C. J. Mungall. Hematopoietic cell types: Prototype for a revised cell ontology. *Journal of Biomedical Informatics*, 44(1):75 – 79, 2011.

[6] M. Egaña, A. Rector, R. Stevens, and E. Antezana. Applying Ontology Design Patterns in Bio-ontologies. In A. Gangemi and J. Euzenat, editors, *EKAW 2008, LNCS 5268*, pages 7–16, 2008.

[7] J. T. Fernandez-Breis, L. Iannone, I. Palmisano, A. L. Rector, and R. Stevens. Enriching the Gene Ontology via the dissection of labels using the Ontology Pre Processor Language. In *EKAW*, pages 59–73, 2010.

[8] Gene Ontology Consortium. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 23(May):25–29, 2000.

[9] C. A. Goble, J. Bhagat, S. Aleksejevs, D. Cruickshank, D. Michaelides, D. Newman, M. Borkum, S. Bechhofer, M. Roos, P. Li, and D. De Roure. myExperiment: a repository and social network for the sharing of bioinformatics workflows. *Nucleic Acids Research*, 38(suppl 2):W677–W682, 2010.

[10] J. Goecks, A. Nekrutenko, J. Taylor, and Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology*, 11(8):R86+, 2010.

[11] M. Horridge, N. Drummond, J. Goodwin, A. L. Rector, R. Stevens, and H. Wang. The Manchester OWL Syntax. In B. C. Grau, P. Hitzler, C. Shankey, E. Wallace, B. C. Grau, P. Hitzler, C. Shankey, and E. Wallace, editors, *OWLED*, volume 216 of *CEUR Workshop Proceedings*, 2006.

[12] L. Iannone, I. Palmisano, A. L. Rector, and R. Stevens. Assessing the safety of knowledge patterns in owl ontologies. In *ESWC*, pages 137–151, 2010.

[13] L. Iannone, A. Rector, and R. Stevens. Embedding Knowledge Patterns into OWL. In *ESWC*, pages 218–232, 2009.

[14] J.A. Miñarro-Gimenez, M. Egaña Aranguren, R. M. Béjar, J. T. Fernández-Breis, and M. Madrid. Semantic integration of information about orthologs and diseases: The OGO system. *Journal of biomedical informatics*, in press.

[15] S. Jupp, M. Horridge, L. Iannone, J. Klein, S. Owen, J. Schanstra, R. Stevens, and K. Wolstencroft. Populous: A tool for populating ontology templates. *Journal of biomedical semantics*, in press.

[16] J. W. Kim, J. C. Caralt, and J. K. Hilliard. Pruning bio-ontologies. *Hawaii International Conference on System Sciences*, 0:196c, 2007.

[17] A. Masci, C. Arighi, A. Diehl, A. Lieberman, C. Mungall, R. Scheuermann, B. Smith, and L. Cowell. An improved ontological representation of dendritic cells as a paradigm for all cell types. *BMC Bioinformatics*, 10:70+, 2009.

[18] Mikel Egaña Aranguren, C. Wroe, C. Goble, and R. Stevens. In situ migration of handcrafted ontologies to reason-able forms. *Data and Knowledge Engineering*, 66(1):147–162, 2008.

[19] E. Mikroyannidi, A. Rector, and R. Stevens. Abstracting and Generalising the Foundational Model Anatomy (FMA) Ontology. In *Bio-Ontologies*, 2009.

[20] M.-A. Nolin, M. Dumontier, F. Belleau, and J. Corbeil. Building an HIV data mashup using Bio2RDF. *Briefings in Bioinformatics*, 2011.

[21] A. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe. OWL pizzas: Practical experience of teaching OWL-DL: Common errors and common patterns. In E. Motta, N. Shadbolt, A. Stutt, and N. Gibbins, editors, *Engineering Knowledge in the Age of the SemanticWeb*, volume LNAI 3257, pages 63–81, 2004.

[22] B. Smith, M. Ashburner, C. Rosse, J. Bard, W. Bug, W. Ceusters, L. J. Goldberg, K. Eilbeck, A. Ireland, C. J. Mungall, N. Leontis, P. Rocca-Serra, A. Ruttenberg, S.-A. Sansone, R. H. Scheuermann, N. Shah, P. L. Whetzel, and S. Lewis. The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nat Biotech*, 25(11):1251–1255, 2007.

[23] O. Šváb Zamazal, V. Svátek, and L. Iannone. Pattern-based ontology transformation service exploiting oppl and owl-api. In *EKAW'10*, pages 105–119, 2010.

[24] K. Wolstencroft, R. Mcentire, R. Stevens, L. Tabernero, and A. Brass. Constructing ontology-driven protein family databases. *Bioinformatics*, 21(8):1685–1692, 2005.