Biological Ontology Next Generation (BONG) workflow: user's guide

Mikel Egaña Aranguren

2007

Contents

1 Introduction

The Biological Ontology Next Generation (BONG) workflow [1] is able to dissect class names from an ontology, using regular expressions, and define new semantics for each dissected class. The new semantics for each class are defined by the user, when creating the regular expressions that dissect the class names. Following that process the ontology can be semantically enriched, using a simple methodology; with the new semantics, a reasoner can infer new interesting relationships within the ontology.

This document provides an example of the execution of the workflow with a real example; it also describes how to setup and execute a personalized workflow.

2 Installation

Download the files from: `http://gong.man.ac.uk/downloads/`. The dependencies to fulfill are:

- A Java virtual machine (VM) installed in the system, at least version 1.5.
- A DIG compliant reasoner (`http://dig.sourceforge.net/`), preferably FaCT++ (`http://owl.man.ac.uk/factplusplus/`).

The script OBO2OWL (`http://gong.man.ac.uk/downloads/`) may also be of use, as the workflow only accepts ontologies in OWL format and the script converts any OBO ontology into OWL format.

3 Configuration and execution of a BONG workflow

The best way of understanding how to use BONG is with a concrete example that we describe as follows. The user can set up his own workflow by changing the provided examples. This example dissects the whole subtree of terms under *cell differentiation* from the Gene Ontology (`http://geneontology.org/`) and adds semantics from the Cell Type ontology (`http://obo.sourceforge.net/cgi-bin/detail.cgi?cell`).

3.1 Files

The typical BONG workflow execution needs the following files:

BONG file: (see `examples/BONG_exec_GO_cell_differentiation/BONG`). This file describes the basic information the program needs to execute the workflow: the name of the table to store the result (the results are stored in a database, see bellow), and the URIs of the ontologies. It has the following structure:

```
[HypersonicTableName] tablename
[BONGOntology] logicalURI - physicalURI
[CentralOntology] logicalURI - physicalURI
[AccesoryOntology] logicalURI - physicalURI
```

BONG ontology: (see `examples/BONG_exec_GO_cell_differentiation/BONG.owl`). This ontology describes the workflow itself, which is mainly some regular expressions with semantics associated to it. The program reads the ontology and executes the workflow according to the information found on it. This ontology imports the other two (the central and accessory ontology) so the program can work with them as well. The ontology should have the following classes:

- `RegexpN`: each regexp has got its own class: the names (`Regexp1`, `Regexp2`) should reflect the specifity of regexps, as lower numbers are tried first; `(regulation of) (.+?) (development)` should have a lower number than `(.+?) (development)`, as the latter will capture all the terms that should be captured by the former, if executed first. The regexp itself is codified in the `RegexpString` annotation value. The new semantics that the matched classes should have are coded in the equivalent conditions: neccesary conditions are not read by the program.

- `RegexpMatchGroupN`: there should be as many classes as the highest number of groups in a given RegexpN class (RegexpMatchGroup1, RegexpMatchGroup2, ...). This is a convenience class so restrictions defined when creating semantics for RegexpN classes can point to this classes.

- `RootTerm`: this is the term from the central ontology that will be used as root to extract the subtree. It has one annotation value, `ClassURI`, that gives the URI of the class.

- `CentralOntology`: it gives the logical URI of the central ontology in the `OntologyURI` annotation value.

- `AccesoryOntology`: it gives the logical URI of the accessory ontology in the `OntologyURI` annotation value.

- `AccesoryClass`: under this class there should be any class that complements the dissection semantically.

- `MappingN`: the mappings are provided to make sure that there is a syntactic match between the central ontology terms and the accessory ontology terms. For example, if there is a term called `neuron development` in GO, the program will try to find `neuron` in Cell (the class name has matched the regular expression `(.+?) (development)`); if there is a class with the `rdfs:label` value `neuron`, it will use it; otherwise, it will try on the mapping and one of the mappings should map `neuron` to the correspondent class in Cell, e.g. `nervous cell`. If there is no match or no mapping provided, the program skips that class. The `MappingN` class has got two annotation values, `MapsFrom` (the matched token from the central ontology class name) and `MapsTo` (the URI of the corresponding class in the accessory ontology). Mappings are optional.

Central ontology: (see `examples/BONG_exec_GO_cell_differentiation/go.owl`). The ontology where the subtree is extracted and the terms dissected.

Central ontology: (see `examples/BONG_exec_GO_cell_differentiation/cell.owl`). This ontology is used as an accessory for semantic enrichment: normally new restrictions added to the central ontology will point to this ontology.

3.2 Execution

If the needed ontologies are in OBO format, there should be converted to OWL using OBO2OWL (`http://gong.man.ac.uk/downloads/`). Once the central and the accessory ontologies are in OWL format, the BONG file should be edited accordingly (adding the correct paths). Before executing the bong program the reasoner must be already running; the bong program is executed like that:

```
java -jar -Xmx1000M bong.jar BONG
```

`BONG` is the BONG file, available as an example in `examples/BONG_exec_GO_cell_differentiation/BONG`. Redirecting the output of the workflow to a log file is highly recommended:

```
java -jar -Xmx1000M bong.jar BONG > log
```

The workflow can take a lot of time to finish; the results are stored in the Hypersonic database,[1] in the table with the name provided in the BONG file, and a new OWL central ontology will appear in the execution directory, with the new semantics.

The best thing is to keep the results of each execution on the database, with a different table name. However, if one table needs to be dropped from the database, the following program can be used, passing the table name as the only parameter:

```
java -jar db_cleaner.jar bong21
```

The performance of the workflow (measured in the number of new meanigfull is-a relationships) depends on the regular expressions; the example provided here does not yield plenty of new relationships but it is meant to be fast and simple.

The demonstration of the usefullnes of this workflow is appreciated when analyzing the new relationships of the workflow example (either in the log file or reading the file `db/bong.script`):

[1]Hypersonic (`http://hsqldb.org`) is an SQL compliant relational database that is executed in the java VM and uses files on the files system, not needing any installation or configuration from the BONG user. The reason for using the database is to store the results in an easy to access and permanent format, so the user can write other programs to deal with the results.

```
[BONG] New relationship:  GO_0021861 is_a GO_0010001
```

What has happened is that due to the new semantics, the reasoner has inferred that as `radial glial cell` is a (type of) `glial cell`, `radial glial cell differentiation` (differentiation that acts on radial glial cell) is a (type of) `glial cell differentiation` (differentiation that acts on glial cell). This new relationship is absent from the original GO, but it is meanigfull; the aim of BONG is to yield such relationships in big quantities.

4 Things to come in next versions

- GUI, including regexp/BONG ontology creation, in the form of Protégé 4 plugin.
- BONG ontology parser.
- Statistics and logs, including *reasoning traces*.
- Handling of complex added semantics.
- Possibility of using more than one accessory ontology.

5 Contact

If you have any queries contact the author in `mikel.eganaaranguren@cs.man.ac.uk`. If you create a BONG ontology please contact the author, so the ontology is uploaded into the project site to make it possible for everyone to execute that workflow; please check out the project website (`http://www.gong.manchester.ac.uk`).

6 Acknowledgements

The BONG workflow would not be possible without the Wonder Web OWL API and the already mentioned Hypersonic database.

References

[1] C.J. Wroe, R.D. Stevens, C.A. Goble, and M. Ashburner. A Methodology to Migrate the Gene Ontology to a Description Logic Environment Using

DAML+OIL. In *8th Pacific Symposium on biocomputing (PSB)*, pages 624–636, 2003.