RESEARCH

ROBOT: A tool for managing and releasing Open Biological Ontologies

James A Overton[??*]
, James Balhoff[??]
, Eric Douglass[??]
, David Osumi-Sutherland[??]
and Christopher John Mungall[??]

*Correspondence:
james@overton.ca
[??]Knocean, TODO, Toronto, Canada
Full list of author information is available at the end of the article

Abstract

ROBOT is a tool for working with Open Biomedical Ontologies

Keywords: sample; article; author

1 Background

Ontology engineering (OE) is in many ways analgous to software engineering (SE), yet tool support for OE lags behind tools for SE. Tools such as Protege[CITE] provide a rich interactive Ontology Development Environment (ODE), but there is a lack of standardized tools for the process of cutting an ontology release. Indeed there is a lack of broad agreement on standards for what should be included or excluded in a release, with different projects making different decisions independently. The Open Biological Ontologies (OBO) Foundry [CITE] was created to develop such standards.

TODO: Figure showing ontology development lifecycle

%% could show http://www.cs.man.ac.uk/~stevensr/onto/node13.html

We present here a tool ROBOT (ROBOT is an OBO Tool) that provides a standardized yet configurable way of supporting the ontology development lifecycle.

2 Implementation

ROBOT is implemented as a java library, with command line abilities. It depends on the java OWLAPI and the Jena API. [CITE]

The core concept in ROBOT is an *Operation*. All operations take one or more ontologies as input, and optionally additional input, such as term files. Most operations create a new ontology as output, some produce reports or term lists. Each operation is implemented as a stateless (static) Java class.

Each operation corresponds to a *Command*, which can be executed on the command line.

Operations can be *composed* into sequences of operations, `S = T1 o T2 ... Tn`.

We draw a distinction between the *source* for an ontology (sometimes called the "edit" file, and the *target* (or targets), derived from that source via a flow of operations. The source is edited by ontology developers, and the target(s) is what is released to consumers of the ontology.

We provide here a formal description of each ROBOT operation. We also provide practical examples of how to perform each one of these on the command line on the ROBOT website.

2.1 Convert Operation

The convert operation takes an ontology file in one format, and produces an ontology file in a different one (alternatively, the output format may be the same, in which case this is called "round-tripping" ").

Formats supported:

- OWL-RDF/XML (default)
- OWL-RDF/Turtle
- OWL-XML
- OWL-Functional Syntax
- OWL-Manchester Syntax
- OBO Format [CITE]
- OBO Graphs JSON [CITE] (output only)

By convention, the main release file is OWL-RDF/XML. Some ontologies also produce an ancillary OBO Format file, which is less expressive, but widely consumed by various bioinformatics pipelines. The new OBO JSON format is designed to supplant this.

Ontologies vary in the format used for source. Some choose formats that work better with Version Control Systems (VCSs) [CITE].

In all cases other than the OBO Graph JSON format, the implementation is handled by the OWL API.

2.2 Reason Operation

The reason operation takes an ontology as input and uses a user-specified OWL Reasoner to (1) perform logical validation on the ontology (2) generate inferred axioms that are asserted in the output ontology.

The validation step aims to detect if an ontology is *incoherent*. Formally, we define as ontology as being incoherent if any of the following conditions hold.

1. There exists some class `C` in the ontology, such that `C` is unsatisfiable (equivalent to the empty set)
2. There exists an object property (relation) `R` such that no facts can be stated with that relation
3. A contradiction is entailed by the ontology

Note that 1 and 3 are part of the standard definition of incoherency used in Description Logics, but as far as we are aware condition 2 has not been previous identified.

If an incoherency is detected then an error is thrown and the sequence of operations is terminated. If the ontology is incoherent on account of having unsatisfiable classes, the an "unsatisfiable module" is generated. This contains a minimal ontology that can be used by the ontology developers to investigate the issue using ontology development environments such as Protege. This can be seen as analogous to a "core dump" produced when execution of programs fail catastophically.

Note that incoherencies are generally only possible if the ontology includes (directly, or via imports) *constraining* axioms such as disjointness axioms or domain/range restrictions on object properties. Many ontologies lack these axioms, meaning some problems may go undetected.

After validation, the reasoner is interrogated for inferred axioms, and these are added to (asserted in) the ontology. Formally, the semantics of the ontology does not change (the operation is entailments-preserving), the assertion of the axioms is there for the convenience of consumers.

The operation does not *saturate* the ontology - only *direct* axioms are added.

For example, in the following ontology:

```
Class: A SubClassOf: B
Class: B SubClassOf: C
Class: XA EquivalentTo: X and R some A
Class: XB EquivalentTo: X and R some B
Class: XC EquivalentTo: X and R some C
```

Three axioms are entailed:

1. XA SubClassOf XB
2. XB SubClassOf XC
3. XA SubClassOf XC

However, only 1 and 2 (the *direct* SubClassOf) axioms are asserted back into the ontology.

The terminology for distinguishing between ontologies in the prior and posterior state can be confusing: sometimes "pre-reasoned" is taken to mean the source ontology, before the reasoning step has been applied; sometimes it is taken to mean "this ontology has been reasoned in advance". Here we use `R-` to indicate that inferred axioms have not been added.

2.3 Materialization

The Materialize operation is an extension to the Reason operation, which will assert the most direct parent, where the parent is either a named class or an *existential expression*, i.e. an expression of the form `R some C`. Normal OWL reasoning only produces the former.

To explain the utility of this operation, we introduce the concept of a Basic Existential Graph (BEG), which is the graph formed from the set of `SubClass` axioms in an ontology, where the subclass is a named class, and the superclass is either a named class or a simple existential expression of the form `R some C`. Many applications, especially in bioinformatics, ignore all axioms in an ontology other than these, and map these axioms onto a graph, where edges are labeled either with is-a/SubClassOf, or with `R`. It is therefore important for an ontology to have a BEG that is complete, with respect to direct edges. The BEG formed by the OWLAPI Reason operation is not guaranteed complete, as can be seen in the following example:

`TODO`

To implement this operation, we extended the OWLAPI Reasoner interface to create an `ExtendedReasoner`, and created a default implementation which finds

direct existential parents. This is avaialable as a standalone java library from GitHub and maven central.[CITE]

This can be combined with filter and reduce to create an ontology subset that is complete

```
robot materialize --reasoner ELK --input emr_example.obo --term BFO:0000050 filter -t BFO:00000
```

2.4 Relaxing equivalence axioms

Robot can be used to relax Equivalence Axioms to weaker SubClassOf axioms. The resulting axioms will be redundant with the stronger equivalence axioms, but may be useful for applications that only consume SubClassOf axioms

```
robot relax  --input ribosome.owl --output results/relaxed.owl
```

2.5 Reducing Graph

Robot can be used to 'reduce' (i.e. remove redundant subClassOf axioms), independent of this previous step.

```
robot reduce --reasoner ELK --input ribosome.owl --output results/reduced.owl
```

2.6 Diffing two ontologies

To compare two text files and check for differences, you can use the Unix `diff` command on Linux or Mac OS X:

```
diff file1.txt file2.txt
```

or on Windows the `FC` command:

```
FC file1.txt file2.txt
```

Any number of graphical diff tools are also available, for example FileMerge is part of Apple's free XCode tools.

Although OWL ontology files are text, it's possible to have two identical ontologies saved to two files with very different structure. ROBOT provides a `diff` command that compares two ontologies while ignoring these differences:

```
robot diff --left edit.owl --right release.owl
```

If an `output` is provided then a report will be written with any differences between the two ontologies:

```
robot diff --left edit.owl --right release.owl --output results/release-diff.txt
```

See `release-diff.txt` for an example.

2.7 Merging

OWL ontologies are often divided into several `.owl` files, with `owl:imports` statements to bring them together. Sometimes you want to take all those imports and merge them into a single ontology with a single `.owl` file.

```
robot merge --input edit.owl --output results/merged.owl
```

You don't need `owl:import` statements: you can merge any number of ontologies by using multiple `--input` arguments. All the ontologies and their imports are merged into the first ontology.

```
robot merge --input edit.owl --input foo.owl --output results/merged2.owl
```

2.8 Filtering

Some ontologies contain more axioms than you want to use. You can use the `filter` command to keep only those axioms with ObjectProperties that you specify. For example, Uberon contains rich logical axioms, but sometimes you only want to keep the 'part of' and 'has part' relations. Here we start with a fragment of Uberon and filter for parthood relations:

```
robot filter --input uberon_fragment.owl --term obo:BFO_0000050 --term obo:BFO_0000051 --output
```

2.9 Module Extraction

The reuse of ontology terms creates links between data, making the ontology and the data more valuable. But often you want to reuse just a subset of terms from a target ontology, not the whole thing. Here we take the filtered ontology from the previous step and extract a "STAR" module for the term 'adrenal cortex' and its supporting terms:

```
robot extract --method STAR
    --input filtered.owl \
    --term-file uberon_module.txt \
    --output results/uberon_module.owl
```

The `--method` options fall into two groups: Minimal Information for Reuse of External Ontology Term (MIREOT) and Syntactic Locality Module Extractor (SLME).

- MIREOT: extract a simple hierarchy of terms
- STAR: use the SLME to extract a fixpoint-nested module
- TOP: use the SLME to extract a top module
- BOT: use the SLME to extract a bottom module

For MIREOT, both "upper" (ancestor) and "lower" (descendant) limits must be specified, like this:

```
robot extract --method MIREOT
    --input uberon_fragment.owl \
    --upper-term "obo:UBERON_0000465" \
    --lower-term "obo:UBERON_0001017" \
    --lower-term "obo:UBERON_0002369" \
    --output results/uberon_mireot.owl
```

For more details see:

- MIREOT
- SLME
- ModuleType

2.10 Annotating

It's important to add metadata to an ontology before releasing it, and to update the ontology version IRI.

```
robot annotate --input edit.owl \
  --ontology-iri "https://github.com/ontodev/robot/examples/annotated.owl" \
  --version-iri "https://github.com/ontodev/robot/examples/annotated-1.owl" \
  --annotation rdfs:comment "Comment" \
  --annotation rdfs:label "Label" \
```

```
  --annotation-file annotations.ttl \
  --output results/annotated.owl
```

2.11 Mirroring

Many ontologies make use of `owl:imports` to bring in other ontologies, or portions of other ontologies. Large import chains involving multiple large ontologies are more prone to run-time failure due to network errors or latency. It can therefore be beneficial to mirror or cache an external ontology's import chain locally. This can be though of as analogous to what happens with standard dependency management tools for software development.

The following command will mirror a local

```
robot mirror -i test.owl -d results/my-cache -o results/my-catalog.xml
```

This will generate a directory `results/my-cache/purl.obolibrary.org/obo/ro/` with the imported ontologies as files. The file my-catalog.xml is a generated XML catalog mapping the source URIs to local files.

2.12 Templating

ROBOT can also create ontology files from templates. See template.md for details. Here's a quick example:

```
robot template --template template.csv \
  --prefix "ex: http://example.com/" \
  --ontology-iri "https://github.com/ontodev/robot/examples/template.owl" \
  --output results/template.owl
```

TODO: CITE QTT, DOSDP[**?**]

2.13 Validating Profiles

OWL 2 has a number of profiles that strike different balances between expressive power and reasoning efficiency. ROBOT can validate an input ontology against a profile (EL, DL, RL, QL, and Full) and generate a report. For example:

```
robot validate-profile -profile EL \
  --input merged.owl \
  --output results/merged-validation.txt
```

2.14 Prefix Management

Terms in OBO and OWL are identified using IRIs (Internationalized Resource Identifiers), which generalize the familiar addresses for web pages. IRIs have many advantages, but one of their disadvantages is that they can be long. Additionally, by convention, ontology classes are frequently identified by a shortform identifier, e.g. `GO:0008150`

So we have standard ways to abbreviate IRIs in a particular context by specifying **prefixes**. For example, Turtle files start with `@prefix` statements, SPARQL queries start with `PREFIX` statements, and JSON-LD data includes a `@context` with prefixes.

For robot we use the JSON-LD format. See `robot-core/src/main/resources/obo_context.jsonld` for the JSON-LD context that is used by default. It includes common, general linked-data prefixes, and prefixes for all the OBO library projects.

If you do not want to use the defaults, you can use the `--noprefixes` option. If you want to replace the defaults, use the `--prefixes` option and specify your JSON-LD file. Whatever your choice, you can add more prefixes using the `--prefix` option, as many times as you like. Finally, you can print or save the current prefixes using the `export-prefixes` command. Here are some examples:

```
robot --noprefixes --prefix "foo: http://foo#" \
  export-prefixes --output results/foo.json

robot --prefixes foo.json -p "bar: http://bar#" -p "baz: http://baz#" \
  export-prefixes
```

The various prefix options can be used with any command. When chaining commands, you usually want to specify all the prefix options first, so that they are used "globally" by all commands. But you can also use prefix options for single commands. Here's a silly example with a global prefix "foo" and a local prefix "bar". The first export includes both the global and local prefixes, while the second export includes only the global prefix.

```
robot --noprefixes --prefix "foo: http://foo#" \
  export-prefixes --prefix "bar: http://bar#" \
  export-prefixes
```

3 Results and Discussion

3.1 Use as part of Continuous Integration

Travis

Jenkins

[CITE]

3.2 Releases with GitHub

3.3 Ontology Starter Kit

3.4 Releasing to OBO

3.5 Case Study: Relation Ontology

The Relation Ontology (RO) differs from most other ontologies, in that the majority of content is in the *RBox* (i.e. axioms about relations), whereas for most ontologies the majority of content is in the TBox (i.e. axioms about classes or terms).

We extended the concept of an incoherent ontology to include incoherent RBoxes

3.6 Case Study: GO

We have previously written about use of OWL in GO [CITE: OWLED], this predates ROBOT.

Multiple editors. Edits are made via Pull Requests. Triggers a travis job that runs

The need for Basic OBO Graphs

3.7 Case Study: Planteome

3.8 Case Study: OBA

GCIs

3.9 Case Study: OBI

TODO: James to write

3.10 Templating

DOSDPs, QTT

3.11 Use of SPARQL

3.12 Use of Makefiles

On Unix platforms (including Mac OS X and Linux) you can use the venerable Make tool to string together multiple `robot` commands. Make can also handle dependencies between build tasks.

`TODO make`

`TODO make release`

When working with Makefiles, be careful with whitespace! Make expects tabs in some places and spaces in others, and mixing them up will lead to unexpected results. Configure your text editor to indicate when tabs or spaces are being used.

BioMake[CITE]

3.13 Missing features

Here are some other commands we should provide examples for:

- import, update imports
- add metadata
- package for release
- get term hierarchy
- convert formats

Uberon - has a complex pipeline with many features implemented in OWLTools. For example, species subsetting.

3.14 See Also

CITE: Tawny

CITE: Scowl

blah

3.15 foo

dd

- ROBOT: ROBOT is an OBO Tool
- OWL: Web Ontology Language
- RDF: Resource Description Format
- DOSDP: Dead Simple OWL Design Patterns
- ODE: Ontology Development Environment

FOO

Competing interests

The authors declare that they have no competing interests.

Author's contributions

JAO designed and implemented the framework. CJM provided requirements and implemented the materialize and reduce operations. DOS provided requirements and specifications and performed testing for GO.

Acknowledgements

Text for this section ...

References

Figures

Tables

Figure 1 Sample figure title. A short description of the figure content should go here.

Figure 2 Sample figure title. Figure legend text.

Table 1 Sample table title. This is where the description of the table should go.

	B1	B2	B3
A1	0.1	0.2	0.3
A2	...	..	.
A3	..	.	.