Standardizing Ontology Workflows Using ROBOT

Rebecca C Tauber¹, James P Balhoff², Eric Douglass³, Christopher J Mungall³, James A Overton¹

¹Knocean, Toronto, Ontario, Canada

²University of North Carolina, Chapel Hill, North Carolina, USA

³Lawrence Berkeley National Laboratory, Berkeley, California, USA

Abstract— Building and maintaining ontologies can be challenging due to the need to automate a number of common tasks, such as running quality control checks, automatic classification using reasoners, generating standard reports, extracting application-specific subsets, and managing ontology dependencies. These workflows are in some aspects analogous to workflows used in software engineering as part of the normal product lifecycle. However, in contrast to software development, there is a lack of easy to use tooling to support the execution of these workflows for ontology developers.

ROBOT is a tool for working with Open Biomedical Ontologies. It provides a way to execute standard ontology processing operations on the command line. These can be chained together, or coordinated via a separate task execution system such as UNIX Makefiles. These workflows can also be executed within continuous integration systems such as Travis, allowing ontologies to be managed in social coding version control systems such as GitHub, with all the attendant features such as validation of pull requests. We have also created an Ontology Starter Kit that can set up a GitHub repository with standard layout and workflows in place. This makes it simple for an ontology development group to configure the needed processing pipelines.

Keywords—ontology development; automation; ontology release; reasoning; workflows; quality control; import management

I. INTRODUCTION

Ontology engineering (OE) is in many ways analogous to software engineering (SE), yet tool support for OE lags behind tools for SE. Graphical interfaces such as Protégé [1] provide a rich interactive Ontology Development Environment (ODE), but there is a lack of standardized tools for running ontology release workflows. 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 [2] was created in part to develop such standards.

Noy, Tudorache, Nyulas, and Musen (2010) documented a general ontology life cycle with a focus on bio-ontologies [3]. First, requirements for the ontology are gathered. Then, the ontology is collaboratively developed in Protégé. Once the requirements have been fulfilled, the ontology is published and feedback is solicited. Feedback is integrated back into development, and the ontology is continuously updated and published. At any point after the initial publication, the ontology may be deployed in other applications.

While this general workflow has been widely accepted in the bio-ontology community, the publication process still requires much manual effort and review. Various bio-ontologies, while using similar tools, have vastly different release processes. Additionally, much of the collaborative development requires prior knowledge of Protégé, limiting the number of domain experts who are able to work on an ontology.

Our tool, ROBOT (ROBOT is an OBO Tool), provides a standardized yet configurable way to support the ontology development lifecycle. We previously described the basic architecture of the tool [4], which we summarize here.

ROBOT is written in the Java programming language and consists of two parts: 'robot-core' and 'robot-command'. 'robot-core' is a library supporting common ontology development tasks, which we call "operations". 'robot-command' provides a command-line interface divided into "commands", each of which makes some use of 'robot-core' operations. The source code is available from the GitHub repository at <u>https://github.com/ontodev/robot/</u> and released on Maven Central at <u>https://mvnrepository.com/artifact/org.obolibrary.robot</u>.

Any programming language that runs on the Java Virtual Machine (JVM) can import the 'robot-core' library and use the full set of operations. Many of the core operations use OWLAPI [5], which is also used by Protégé and other systems using OWL. Some core operations use Apache Jena [6]. In both cases, the core operations are designed to be higher-level, and usually more specialized, than OWLAPI or Jena for the relevant tasks.

The key advantage of the division between operations and commands (i.e., between 'robot-core' and 'robot-command') is a separation of concerns between working with Java objects, for the former, and working with the terminal and file system, for the latter. Operations work with Java objects representing OWL ontologies, OWL reasoners, OWL classes, etc., while commands work with command-line option strings, files, and perform various conversion and validation steps. The result is cleaner, more focused code on both sides.

II. COMMANDS AND OPERATIONS

ROBOT currently provides 15 operations (in the Java library) and 19 commands (for the command line). While some commands are quite specialized, here we provide an overview of the most common and general commands.

ROBOT is compatible with all OWL ontology formats, including RDF/XML, Turtle, Manchester, OBO, and more. To enable further interoperability, a 'convert' command is included to change between supported ontology formats. A complete list of supported formats can be found in the 'convert' documentation (http://robot.obolibrary.org/convert).

A. Reasoning

Reasoning is one of the most fundamental operations in ROBOT. This has two uses: *logical validation* of an ontology, and *automatic classification*, both covered by the 'reason' command. In both cases, an OWL reasoner is used to perform inference. Users can choose which reasoner to use. Large ontologies such as the Gene Ontology typically use ELK [7], which scales well but supports a smaller OWL profile. Smaller ontologies with richer axiomatization such as the Relations Ontology typically use a complete DL reasoner such as HermiT [8].

When the 'reason' command is invoked on an input ontology, ROBOT will initiate a reasoner using the OWLAPI Reasoner interface. After that, the resulting inferences are checked to ensure the ontology is *logically coherent*: the ontology must be consistent, and have no unsatisfiable classes (i.e., classes that cannot be instantiated without introducing an inconsistency). If the ontology is incoherent then this is reported, and execution halts. ROBOT also performs additional checks, ensuring that no two classes are inferred to be equivalent post-reasoning, as this is usually unintentional.

If the ontology is consistent, then ROBOT will perform automatic classification. All direct inferred subClassOf axioms are asserted into the ontology. We are also exploring the possibility of allowing inference of other axiom types, such as domain and range axioms.

The 'reason' command has additional "helper" commands. The 'relax' operation asserts entailed subClassOf axioms according to a simple structural rule: an expression 'A equivalentTo (R some B) and ...' entails 'A subClassOf R some B'. This can be useful as consumers of bio-ontologies often expect to navigate these expressions, e.g., partonomy in GO and Uberon. The 'relax' command relieves the ontology developer from the need to assert these in addition to the equivalence axioms, and as such it is also often included in release workflows. Additionally, it means that the SubClassOf axioms can form a complete graph for traversal of all relationship types in the ontology, which is a common requirement for ontologies with part-of relationships, such as GO and Uberon. Finally, the 'reduce' operation removes redundant subClassOf axioms, and can be used after 'relax' to remove duplicate axioms that were asserted in that step.

The 'materialize' command uses an Expression Materializing Reasoner (EMR) to assert inferred expressions of the form "A subClassOf R some B" (for more information see https://github.com/owlcollab/expression-materializingreasoner). Where the 'reason' command asserts inferred *named* superclasses, 'materialize' asserts anonymous superclasses. This is not part of the standard release cycle but can be beneficial for creating complete ontology subsets.

B. Working with External Ontologies

The OBO Foundry project aims to coordinate ontologies in a modular fashion, such that parts of some ontologies can be used as building blocks for other ontologies. For example, the ChEBI chemical entities ontology is used to construct OWL definitions for metabolic processes and activities in the Gene Ontology [9].

There are a variety of different strategies for leveraging external ontologies, depending on use cases.

Extract

The 'extract' command creates a module based on a set of entities to extract (a "seed"). There are four different extraction methods (as specified by the '--method' option): MIREOT, TOP, BOT, and STAR.

The MIREOT extraction method is based on the principles of the same name: Minimum Information to Reference an External Ontology Term [10]. For this method, one or more "bottom" entities must be specified. Optionally, one or more "top" entities can also be specified. The command extracts all the "bottom" level entities and their dependencies up to the "top" level from the input ontology. If no "top" entities are provided, dependencies up to the top-level entity ('owl:Thing') will be included.

The TOP, BOT and STAR methods make use of the OWLAPI Syntactic Locality Module Extraction (SLME) implementation, which is guaranteed to capture all information logically relevant to the seed set [11].

Remove and Filter

The ROBOT team is currently developing a pair of complementary commands for fine-grained control over ontology imports and extraction: 'remove' and 'filter'. The 'remove' command will allow users to choose which sets of axioms they wish to remove from a target ontology. The 'filter' command will offer similar functionality, with the difference being that only selected axioms are kept, and all other axioms are removed. These two commands work by starting with the seed set of entities, then applying various selectors to find related entities, and finally selecting which axiom types to remove or filter. We expect only a small number of "power users" will use this feature directly, but these commands will eventually provide a foundation for other higher-level commands.

C. Querying and Reporting

Ontology workflows typically include query operations over the ontology, producing reports which may be informative for both editors and users of the ontology - for example, a table of all classes plus their textual definitions. Query operations can also be used for validation checks. The SPARQL query language provides a universal and declarative way for ontology maintainers to create ontology reports or validation checks. ROBOT provides a convenient way to perform queries over ontologies with the 'query' command, or validation checks using 'verify'. Additionally, the 'report' command provides a powerful and convenient package of standard OBO queries that can be used or configured in any ontology workflow, without requiring the maintainer to be familiar with SPARQL.

Query

ROBOT's 'query' command runs SPARQL queries on loaded ontologies (or other RDF resources). This can be used by an ontology maintainer to either perform interactive queries, or more typically to include standard queries into an ontology workflow.

SELECT queries output a table of results. ASK queries output a file with a boolean value. CONSTRUCT queries output an RDF file, which can be further processed by ROBOT or merged back into the loaded ontology. CONSTRUCTs provide a convenient way of performing "macro" style expansion [12]. ROBOT does not currently support SPARQL UPDATE queries.

Verify

The 'verify' command is a variation on SPARQL SELECT, and is used to ensure that an ontology conforms to a predetermined set of conditions; for example, ensuring that no class has multiple textual definitions. Given a SELECT query, 'verify' will succeed (i.e., exit with status code 0) if NO results are returned. It will fail (i.e., exit with a non-zero status code) if ANY results are return from the query. So given a SPARQL query that SELECTs for invalid data, the 'verify' command will verify that the ontology (or other resource) does not contain such invalid data.

Report

The 'report' command is an extension of 'query' and 'verify' that provides a series of configurable quality control checks for an ontology and returns a spreadsheet or YAML output of the violations. The spreadsheet is in TSV format and easy for a user to read, while the YAML output can be easily parsed with scripts and other programs.

The QC checks include annotation checks, logical checks, and metadata checks. Annotations are important to facilitate human comprehension, so the 'report' command finds cases where missing or duplicate annotations could cause issues. Logical checks look at the structural coherency and consistency of the ontology. Finally, 'report' identifies missing ontology metadata, as dictated by OBO Foundry recommendations.

There are three levels of reporting violations: ERROR, WARN, and INFO. An ERROR-level is the most severe, such as a missing or duplicate label. These types of violations must be fixed before publishing an ontology. WARN-level violations should be fixed as soon as possible, such as inferred one-to-one class equivalencies, which are typically unintended within OBO ontologies. The last level, INFO violations, are recommended fixes for maintaining consistency across OBO Foundry ontologies, such as beginning a definition with an uppercase letter and ending with a period.

A default "profile" with report levels for each QC check is provided by ROBOT, but users are also able to create their own profiles. In these profiles, they can change the reporting levels, choose to exclude certain checks, and add their own checks as SPARQL queries. For example, some ontologies may categorize a class lacking a textual definition as being an error, while others may categorize this as a warning. One of our goals is to converge on a standard profile that is maximally useful for the set of all ontologies in the OBO library, encouraging adoption of common quality control checks.

D. Templated Ontology Development

ROBOT comes with a template-driven ontology term generation system. Users can still plug in their own system into their workflow, such as DOS-DP patterns [13].

A huge amount of data is stored in spreadsheets and databases, and tabular formats are well suited to many sorts of data. ROBOT's 'template' command allows users to convert tabular data into RDF/OWL format. A ROBOT template is simply a tab-separated values (TSV) or comma-separated values (CSV) file with some special conventions.

The first row of a template file must contain the column headers and the second row contains a "template string" for each column to be converted. Template strings have a special syntax that is outlined in the ROBOT 'template' documentation (located at http://robot.obolibrary.org/template), where '%' characters are replaced by cell contents to generate both logical and annotation axioms.

Each subsequent row corresponds to an RDF subject to be generated. The subject rows require an Internationalized Resource Identifier $(IRI)^1$ or Compact URI (CURIE)² to identify them, and the cells of the row contain the objects of various Annotation Properties, Object Properties, and Data Properties. Term labels ('rdfs:label's) can be used instead of IRIs and CURIEs, increasing readability, as long as they are defined in the input ontology. Class expressions can be provided in Manchester syntax with labels, as used in Protégé.

III. WORKFLOWS

A workflow consists of a set of tasks coordinated by some workflow system. Ontology workflows consist of tasks such as executing QC checks, building import modules, reasoning over ontologies, and generating various ontology release products. Although ROBOT allows multiple commands to be chained together, it is not itself a workflow manager. Because ROBOT commands can be executed on the command line, a number of different systems can be used. We highlight the use of Unix Makefiles, although other systems are possible.

A. Makefiles

A Makefile is a set of instructions that is typically used to compile programs. In ontology development, the Makefile is used for automated tasks, such as releasing the ontology. A Makefile consists of a set of rules used to make objects (the "targets" of the rule). In this case, the objects are usually ontology modules and files. The "recipes" for the rules are Unix-style system commands, carried out by the 'make' command.

¹ An IRI is an extension of a Universal Resource Identifier (URI) to support more characters. URIs are intended to unambiguously identify resources, such as ontology classes, by using a namespace (like a URL) and a resource name within that namespace (like an ID).

² A CURIE is an abbreviated version of an IRI or URI. A prefix for the namespace is provided (e.g., http://purl.obolibrary. org/OBI_becomes 'OBI') and the ID is appended to that prefix (e.g., OBI:0000070).

ROBOT commands can be used as the "recipes" to make the "targets". Fig. 1 shows a standard release workflow using ROBOT commands.

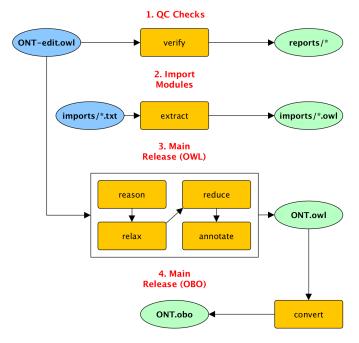


Fig. 1. The ROBOT release workflow

First, quality control checks are run over the editor ontology with 'verify'. These look for equivalent classes, trailing whitespace in annotations, self-references, cross reference syntax, and missing labels. The results are saved to a 'reports/' directory. (This step will be replaced with the updated 'report' command once the new features are complete.) If there are any violations, the 'make' command will fail so that these can be fixed before releasing.

Assuming the QC check step has completed successfully (the process will fail and exit if not), the next step is to create the import modules. The ROBOT 'extract' command runs over a list of import names, which have corresponding term files (for the seed set) in the 'imports/' directory. This step creates all the import modules in the same directory.

Finally, the main release products are created: the OWL file and the OBO file. To create the OWL release, the editor file is passed through a series of chained ROBOT commands: 'reason', 'relax', 'reduce', and 'annotate'. If any of these commands fail, the Make process will terminate with the error message. For example, if an ontology is incoherent, the 'reason' step will fail; if an ontology fails a hard check in 'report' then this step will fail.. The 'annotate' command adds the version IRI to the ontology metadata. This OWL file is then converted to OBO format, at which point all targets are copied to a dated release directory.

Creating a Makefile to coordinate all these steps can be challenging. We make this easier for ontology developers by providing an "Ontology Starter Kit" (available from https://github.com/INCATools/ontology-starter-kit/). This can be used to create a GitHub repository following a standard layout, with a standard Makefile following the workflow detailed above. The resulting GitHub repository will also be configured to be able to run the workflow via the Travis-CI Continuous Integration tool. The workflow can also be executed using Docker via ontology starter kit containers released on Dockerhub (https://hub.docker.com/r/obolibrary/). This allows easy execution of workflows on either the local laptop/desktop computer of an ontology developer.

4

IV. CASE STUDY: USE OF ROBOT IN THE ONTOLOGY FOR BIOMEDICAL INVESTIGATIONS

The Ontology for Biomedical Investigations (OBI) is an OBO Foundry ontology that aims to describe the processes, agents, devices, inputs, and outputs of scientific investigations [14]. When the project began more than a decade ago, development was done in Protégé, without any automation, and hosted on SourceForge. Today OBI uses ROBOT to implement an automated workflow, supported by GitHub Pull Requests and Travis continuous integration testing.

OBI has always imported a range of terms from other OBO ontologies, and developers have maintained a number of separate OWL files to facilitate concurrent development by different groups of users. When it comes time to make a new release of OBI, the various OWL files must be merged, tested and reasoned over.

In the early days of the OBI project. Alan Ruttenberg developed a series of scripts for quality control and common operations, but the process of building a merged and reasoned release file, and testing it, was a manual process that often took many hours. In 2013 James Overton developed an automated build tool in Java, using OWLAPI and Apache Ant, to automate most of the build, test, and release process. This drastically reduced the number of hours required to make a release, and allowed for more frequent releases. While this code was specific to OBI workflows, some of it was used in early versions of ROBOT.

In 2017 OBI moved from SourceForge to GitHub and the release workflow was updated to use ROBOT throughout. OBI currently includes a Makefile that defines a range of tasks for managing imports, converting templates, merging, reasoning, testing, and releasing new versions of OBI. The key steps are:

- 1. Update imports (currently using OntoFox [15])
- 2. Normalize XML for cleaner version history ('robot convert')
- 3. Convert template files (TSV) to OWL modules ('robot template')
- Merge imports and templates with 'obi-edit.owl' ('robot merge')
- 5. Use SPARQL CONSTRUCT to update various annotations ('robot query')
- 6. Run an automated test suite ('robot verify')
- 7. Run the HermiT reasoner ('robot reason')
- 8. Update annotations for release ('robot annotate')
- 9. Extract the OBI Core subset ('robot extract')
- 10. Create a list of terms ('robot query')

ROBOT has been developed to streamline ontology release workflows, while also making it easier for domain experts (who may not be software developers) to collaborate in the ontology development process. Standardizing verification checks and reporting frees the ontology developer from performing routine tasks and allows them to spend more time performing biological knowledge validation on each release. As more bio-ontologies move to include ROBOT in their pipelines, more features will be added to support the community. This will also make it easier for the community to move towards standard ontology workflows and quality control checks.

ACKNOWLEDGMENT

We thank David Osumi-Sutherland for requirements and testing, and Nomi Harris for manuscript edits.

This work and related resources are supported by the following grants:

- From the Deparatment of Health and Human Services (NIH) for "Immune Epitope Database and Analysis Resource Program" (HHSN272201200010C).
- From the National Insitute of Allergy and Infectious Diseases (NIH) for "Human immune signatures of Dengue virus and Mycobacterium Tuberculosis exposure in infection, disease and vaccination" (1-U19-AI-118626-01).
- From the National Human Genome Research Institute (NIH) for "Services to support the OBO foundry standards" (R24-HG010032).

References

 M. Horridge, D. Tsarkov, and T. Redmond, "Supporting early adoption of OWL 1.1 with Protégé-OWL and FaCT++," OWLED, 2006.

- [2] B. Smith, et al., "The OBO Foundry: Coordinated evolution of ontologies to support biomedical data integration," Nature Biotechnology, vol. 25, pp. 1251-1255, November 2007.
- [3] N. Noy, T. Tudorache, C. Nyulas, and M. Musen, "The ontology life cycle: Integrated tools for editing, publishing, peer review, and evolution of ontologies," AMIA Annu. Symp. Proc., pp. 552-556, 2010.
- [4] J.A. Overton, H. Dietze, S. Essaid, D. Osumi-Sutherland, C.J. Mungall, "ROBOT: A command-line tool for ontology development," Proceedings of the International Conference on Biomedical Ontology (ICBO) Lisbon: CEUR Workshop Proceedings, pp. 131-132, 2015.
- [5] M. Horridge, S. Bechhofer, and O. Noppens, "Igniting the OWL 1.1 touch paper: The OWL API," OWLED, 2007.
- [6] J.J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K.J. Wilkinson, "Jena: Implementing the semantic web recommendations," WWW Alt., pp. 74-83, 2004.
- [7] Kazakov Y., Krotzsch M., and F. Simancik, "The incredible ELK," Journal of Automated Reasoning, vol. 53, pp. 1-61, June 2014.
- [8] R. Shearer, B. Motik, and I. Horrocks, "HermiT: A highly-efficient OWL reasoner," OWLED, 2008.
- [9] D.P. Hill, et al., "Dovetailing biology and chemistry: Integrating the Gene Ontology with the ChEBI chemical ontology," BMC Genomics, vol. 14, July 2013.
- [10] M. Courtot, F. Gibson, A.L. Lister, J. Malone, D. Schober, R.R. Brinkman, and A. Ruttenberg, "MIREOT: The minimum information to reference an external ontology term," Applied Ontology, vol. 6, pp. 23-33, 2011.
- [11] B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler, "Modular reuse of ontologies: Theory and practice," Journal of Artificial Intelligence Research, vol. 31, pp. 273-318, 2008.
- [12] C.J. Mungall, A. Ruttenberg, D. Osumi-Sutherland, "Taking shortcuts with OWL using safe macros," Nature Publishing Group, 2010.
- [13] D. Osumi-Sutherland, M. Courtot, J.P. Balhoff, and C. Mungall, "Dead simple OWL design patterns," Journal of Biomedical Semantics, vol. 8, June 2017.
- [14] A. Bandrowski, et al., "The Ontology for Biomedical Investigations," PLoS One, vol. 11, April 2016.
- [15] Z. Xiang, M. Courtot, R.R. Brinkman, A. Ruttenberg, and Y. He, "OntoFox: Web-based support for ontology reuse," BMC Res. Notes, vol. 3, June 2010.