Title

Ontology Development Kit: a toolkit for building, maintaining and standardizing biomedical ontologies

Permalink

https://escholarship.org/uc/item/76f427fs

Authors

Matentzoglu, Nicolas
Goutte-Gattat, Damien
Tan, Shawn Zheng Kai
et al.

Publication Date

2022-10-08

DOI

10.1093/database/baac087

Database, 2022, 1–8
DOI: https://doi.org/10.1093/database/baac087
Original article

Ontology Development Kit: a toolkit for building, maintaining and standardizing biomedical ontologies

Nicolas Matentzoglu[1], Damien Goutte-Gattat[2], Shawn Zheng Kai Tan[3], James P. Balhoff[4], Seth Carbon[5], Anita R. Caron[3], William D. Duncan[5,6], Joe E. Flack[7], Melissa Haendel[8], Nomi L. Harris[5], William R. Hogan[6], Charles Tapley Hoyt[9], Rebecca C. Jackson[10], HyeongSik Kim[11], Huseyin Kir[3], Martin Larralde[12], Julie A. McMurry[8], James A. Overton[13], Bjoern Peters[14], Clare Pilgrim[2], Ray Stefancsik[3], Sofia MC Robb[15], Sabrina Toro[8], Nicole A Vasilevsky[8], Ramona Walls[16], Christopher J. Mungall[5] and David Osumi-Sutherland[3,*]

[1]Semanticly, Spaces Ermou Ermou 56, Athens 10563 ΓEMH 160976003000, Greece
[2]Department of Physiology, Development and Neuroscience, University of Cambridge, Downing Street, Cambridge, CB2 3DY, UK
[3]Samples Phenotypes and Ontologies Team (SPOT), European Bioinformatics Institute (EMBL-EBI), Wellcome Genome Campus, Hinxton, Cambridgeshire, CB10 1SD, UK
[4]RENCI, University of North Carolina, Chapel Hill, NC, North Carolina 27517, USA
[5]Berkeley Bioinformatics Open-source Projects (BBOP), Lawrence Berkeley National Laboratory (LBNL), 1 Cyclotron Road, Mailstop 977-0257, Berkeley, CA 94720, USA
[6]College of Dentistry; Health Outcomes and Biomedical Informatics, College of Medicine, University of Florida, William D. Duncan: 1395 Center Dr, Gainesville, William R. Hogan: 1600 SW Archer Rd, Gainesville, FL 32610, USA
[7]School of Medicine, Johns Hopkins University, 733 N Broadway, Baltimore, Baltimore, MD 21205, USA
[8]University of Colorado Anschutz Medical Campus, 13001 E 17th Pl, Aurora, CO 80045, USA
[9]Laboratory of Systems Pharmacology, Harvard Medical School, 200 Longwood Avenue Armenise Building Room 109, Boston, MA 02115, USA
[10]Bend Informatics LLC, 5305 RIVER RD NORTH, STE B, KEIZER, OR 97303, USA
[11]Robert Bosch LLC, Sunnyvale, CA 94085, USA
[12]Structural and Computational Biology Unit, European Molecular Biology Laboratory, Meyerhofstraße 1, Heidelberg 69117, Germany
[13]Knocean Inc., Toronto, Ontario, ON M6P 2T3, Canada
[14]Institute for Allergy & Immunology, La Jolla Institute for Immunology, 9420 Athena Circle, La Jolla, CA 92037, USA
[15]Stowers Institute for Medical Research, 1000 E. 50th St., Kansas City, MO 64110, USA
[16]Critical Path Institute, 1730 E River Road, Tucson, AZ 85718, USA

[*]Corresponding author: Tel: +44 1223 494 144; Fax: +44 1223 48 46 96; Email: davidos@ebi.ac.uk

Citation details: Matentzoglu, N., Goutte-Gattat, D., Tan, S.Z. *et al.* Ontology Development Kit: a toolkit for building, maintaining and standardizing biomedical ontologies. *Database* (2022) Vol. 2022: article ID baac087; DOI: https://doi.org/10.1093/database/baac087

Abstract

Similar to managing software packages, managing the ontology life cycle involves multiple complex workflows such as preparing releases, continuous quality control checking and dependency management. To manage these processes, a diverse set of tools is required, from command-line utilities to powerful ontology-engineering environmentsr. Particularly in the biomedical domain, which has developed a set of highly diverse yet inter-dependent ontologies, standardizing release practices and metadata and establishing shared quality standards are crucial to enable interoperability. The Ontology Development Kit (ODK) provides a set of standardized, customizable and automatically executable workflows, and packages all required tooling in a single Docker image. In this paper, we provide an overview of how the ODK works, show how it is used in practice and describe how we envision it driving standardization efforts in our community.

Database URL: https://github.com/INCATools/ontology-development-kit

Introduction

In a time of increasing biomedical data output, ontologies have become crucial in research, playing an important role in making data findable, accessible, interoperable and reusable (FAIR) (1) by providing standard identifiers (2–5), vocabulary, metadata and machine-readable axioms (6). Developing high-quality and scalable ontologies requires reusing parts of other ontologies, the use of reasoning to automate classification and extensive quality control (QC) testing. Managing development while following this approach can be a complex process

Received 7 July 2022; Revised 19 August 2022; Accepted 23 September 2022

involving tasks such as import management, release file compilation, integration testing and QC. This difficulty is compounded by the fact that ontologies in the biomedical domain are generally under-resourced and that biologists, who need to be an integral part of their development, are often not trained in software engineering and therefore lack exposure to standard best practices for software development. Furthermore, the complexity of the ontology development process is a huge barrier to entry for the community to contribute, limiting the democratization (and arguably the quality) of these ontologies. Over the years, some ontology communities developed their own workflows for managing the ontology life cycle using a variety of tools and technical approaches. However, given the complexity of the technologies involved, it is very difficult for even the most experienced ontology pipeline developers to maintain and extend these workflows.

The Open Biomedical Ontologies (OBO) Foundry aims to unify ontologies in the biomedical domain through an evolving set of shared principles governing ontology development, allowing interoperability between ontologies (7, 8). However, to effectively achieve this, tools that enable standardization of these shared principles are needed (i) to support ontology developers (ODs) to conform to the principles (e.g. through standardized QC as well as standardised release pipelines) and (ii) to allow less technical ontology curators to abide by these standards without the need for intensive engineering training.

Ontology engineering is a complex task involving many different workflows such as:

(i) Running releases: transforming the ontology through a variety of pipelines involving reasoning, removing redundant content, adding versioning information and exporting to the variety of ontology formats consumed by users such as RDF-XML (https://www.w3.org/TR/owl2-xml-serialization/), OBO (https://owlcollab.github.io/oboformat/doc/GO.format.obo-1_4.html), Turtle (https://www.w3.org/TR/2014/REC-turtle-2014 0225/), and OBO-graphs JSON (https://github.com/geneontology/obographs)s.

(ii) Requesting changes in the form of issues on an issue tracker and discussing the merit of the proposed changes.

(iii) Applying changes to the ontology: adding or editing terms, removing logical axioms or changing labels.

(iv) Reviewing change requests, usually in the form of pull requests.

(v) QC checking: ensuring that the ontology conforms to a variety of integrity checks, such as logical coherence, label uniqueness and provenance standards.

(vi) Dependency management: providing methods to import terms from other ontologies and keep those terms up-to-date in the light of changes.

(vii) Documentation management: providing methods to keep documentation current in the light of changes to the ontology and the ontology workflows.

In this paper, we present the Ontology Development Kit (ODK), a tool for managing the ontology life cycle. ODK is currently used to maintain more than 70 ontologies, mostly in the biomedical domain, such as the widely used Human Phenotype Ontology (9), the Cell ontology (CL) (10, 11), Uberon (12), PATO (13), the Brain Data Standards Ontology and Provisional Cell Ontology (PCL) (14).

The ODK comprises two major components: a set of executable ontology-engineering workflows and a toolbox. It delivers these workflows, which reflect standard best practices recommended by the OBO Foundry, as a customizable git repository set-up including all the different files and scripts necessary to run, for example, releases and QC tests, and import terms from other ontologies. The toolbox is delivered as a Docker image and includes all tools necessary to execute these workflows, from command-line utilities (sed, git and rsync) to ontology pipeline tools such as ROBOT (15) and dosdp-tools (16).

The ODK simplifies the process of maintaining an ontology, allowing ODs to focus on content rather than technical aspects of maintenance. It also allows ODs to fully leverage modern 'social coding' open-source development practices exemplified by many GitHub repositories, such as allowing community contributions via GitHub pull requests and using cloud-based continuous integration (CI) tools to help with QC.

Motivation

Sharing best practices

Best practices for ontology engineering evolve over time. For example, it took years of discussions and collective learning to define the OBO Foundry principles, a set of best practices for open, FAIR and interoperable ontology development in the biomedical domain (8) and their refinement is ongoing. While those practices are slowly adopted through a mix of community engagement activities and improved tooling such as the OBO Dashboard (8), the need for extending those practices never stops. For example, there is currently no agreed-upon metadata schema for reflecting contributions to ontology terms, which is critical not only for attribution (grant proposals and individual editors) but more generally important for provenance-related questions (Who wrote that definition? Who suggested that term to be added?). To drive this forward, a group of organizations decided to collect this information using a specific property (http://purl.org/dc/terms/contributor) and uniquely identifying Internationalized Resource Identifiers (IRIs) for contributors,such as (ORCID) Open Researcher and Contributor IDentifier, Wikidata or (ROR) Research Organization Registry identifiers. To ensure that these metadata are captured correctly, a schema check needs to be defined. This is typically realized using SPARQL (https://www.w3.org/TR/rdf-sparql-query/) in ODK and then shared across all participating ontology repositories.

Having a centralized infrastructure like the ODK means that when one ontology faces such an issue, tests (and fixes) can be rolled out via the executable workflows defined by the ODK to all ontologies using it, not just to that particular ontology. This reduces the overhead needed to fix multiple ontologies and provides a more collaborative environment for problem-solving.

Standardized repository architecture and release products

Ontologies, even if built with OBO principles in mind, vary in the forms in which they are made available. For example, should the ontology be published with import statements or should the imports be merged in? Should the ontology be published with or without the logical inferences computed by a reasoner? Furthermore, users frequently want ontology files in alternative formats, like RDF/XML, OBO Flatfile format, OBO Graphs JSON or Turtle. Another problem is that the ontology repositories are not usually standardized. Editors who edit more than one ontology have to adjust to the idiosyncrasies of each repository: Which files to edit? How to run tests? How to provide versioned releases? How to add new terms? To address some of these issues, the ODK can automatically generate a standardized file and directory structure that is delivered as a git repository.

Git has become the most widely used version control system in the biomedical ontologies domain. Git repository hosting providers such as GitHub and GitLab have become powerful tools beyond simple version control that cover most aspects of modern software (and ontology) project management, including code review, issue tracking, CI testing, discussions and milestone planning. While not entirely tied in with git and git hosting tools such as GitHub, the ODK is designed in a way that leverages their capabilities. The idea is to encourage best practices promoted by these platforms such as creating and reviewing pull requests that are automatically tested before applying a change, separating source files from release files and deploying documentation pages side-by-side with the code for ontology development.

The ODK promotes a 'convention-over-configuration' model by imposing a standardized repository structure where all files are stored in predictable paths within the repository. This ensures that ontology editors are on familiar ground even if they work on multiple different ontologies. The standardized structure includes a strict separation between 'source files', which are manipulated by ontology editors and from which the ontology is built (OWL files, files containing DOSDP patterns or SPARQL queries, helper scripts, etc.), and 'release files', which result from running the release workflow and are intended for downstream users.

The released version of an ontology can take several forms, depending for example on whether the ontology has been reasoned over or whether it contains imported axioms from foreign ontologies. To facilitate interoperability and modular reuse of ontologies, the ODK defines a few standardized release products, such as the 'base' product, which contains only native axioms and the 'full' product, which in addition also includes imported axioms and axioms inferred by logical reasoning.

An example of implementation of the approach described above can be found in the ODK ontologies CL and Uberon—both ontologies have release files in OWL, OBO and JSON, in full, base and simple formats, which are located in the main directory and are never edited directly. Source files, which are the files that get edited, are located separately, but in the same location for both ontologies (e.g. the main edit file is located in src/ontology for both ontologies).

The ODK toolbox

The ODK can be divided into two principal architectural components:

(i) A toolbox containing everything needed to develop, build and maintain ontologies, from Unix command-line development tools (e.g. rsync and git) to specialized ontology pipeline programs (e.g. ROBOT and fastobo-validator).

(ii) A set of executable ontology-engineering workflows, delivered as a directory of scripts, build rules (e.g. to prepare releases or refresh imports) and source files. These workflows are described in the next section.

The goal of the ODK toolbox is to provide ontology editors with all the tools they need to build, test and release their ontologies. Tools are chosen for their ability to support the core workflows for managing the ontology life cycle, such as running releases and QC. A selection of tools (15–21) included in the ODK can be found in Supplementary Table S1. As those tools are very diverse and use different technologies, we cannot merely provide installation instructions that work reliably across the operating systems and computer architectures routinely used by ontology editors. We also lack the resources to provide customized installation packages for all those systems and architectures.

We, therefore, decided to use a Docker image (22) as a software distribution mechanism. Docker is a tool that automates the deployment of applications inside software containers. While it originated on GNU (https://www.gnu.org/)/Linux systems, it is now available on Windows and MacOS as well. The ODK Docker image is based on the Ubuntu base image, in which all the tools listed in Supplementary Table S1 are already installed. All of the core ontology tools and most of the python dependencies are explicitly versioned by the ODK developers, and upgrading them has to be done explicitly, requiring extensive testing. Ontology editors just need to install Docker itself on their platform and fetch the ODK image from the Docker Hub (https://hub.docker.com/r/obolibrary/odkfull). This saves a great deal of time for both the developers/system administrators and the editors, since the Docker image is effectively a 'plug and play' application that can run on any major operating system.

We provide two distinct Docker images. The ODK-Lite (obolibrary/odklite) image contains only the minimal set of tools needed by the standard workflows described in the next section. The ODK-Full (obolibrary/odkfull) image includes additional tools that an OD may need for some customized, ontology-specific workflows. With one of the ODK images available to the local Docker daemon, ontology editors can invoke any of the provided tools inside the container, without needing to do any additional set-up.

Executable ontology development workflows

The executable ontology-engineering workflows are delivered as an ODK-generated Makefile. Targets in that Makefile can roughly be divided into those that provide the recipe for generating a specific file (such as the release file of an ontology) and those that provide simple workflows, such as 'clean' to delete temporary files, 'prepare_release' to execute the release workflow or 'refresh-imports' to update the terms in all imported ontologies. In the rest of this section, we will discuss some of the workflows prevalent in the biomedical ontology community and describe how they are supported by ODK.

The initialization and update workflows

The initialization workflow is performed once in the lifetime of an ontology to create a new ODK set-up. Unlike the other workflows, it is launched not from a Makefile but from a small wrapper script that uses the ODK Docker image to:

(i) create a new directory which contains all files necessary for editing and managing the new ontology (importantly, this includes the automatically generated Makefile that will pilot all the other standard workflows described in the rest of this section);
(ii) make a Git repository of the newly created directory and
(iii) generate an initial release based on the empty ontology.

This initialization process can be parameterized using either command-line arguments to the wrapper script or a small, YAML (https://yaml.org/)-formatted configuration file. Once the repository is set up, it can be pushed to a git hosting service such as GitHub. The choice of the hosting provider is left to the user, but when GitHub is used, the ODK provides special support for automatically triggering a few server-side workflows (as GitHub Actions) upon certain conditions, e.g. to run a CI test suite whenever a pull request is submitted.

The ODK is continually being updated with new functions, better support and updated tooling. This allows us to be highly adaptive to the ever-changing landscape of ontology development and maintain relevance to the community. ODK updates are semi-automated, with a three-step process:

(i) Update the ODK Docker image.
(ii) Run the 'update_repo' command.
(iii) Commit the changes into the ontology repository.

We have decided to leave the process of updating the repository to the user rather than folding the workflow into the ODK Docker image itself. This way, the dissemination of new features is a bit slower, but it also gives more control to the ontology engineer to postpone the implementation of potentially breaking changes, such as QC checks added to the default set-up.

The editors' workflow

Editors frequently change the contents of an ontology by adding or obsoleting terms, revising logical axioms or updating the metadata. While there are many variants of the editors' workflow, i.e. the sequence of actions that lead to the final application of a change, it can be roughly divided into the following steps:

1. The OD opens the editor's file in their preferred ontology development environment (e.g. Protégé) and makes a change (e.g. adding a term and changing a label). Alternatively, a template file (like a ROBOT template) is edited that first needs to be transformed into OWL.
2. The OD creates a new git branch locally, commits the change and opens a pull request on the ontology's public repository on GitHub.
3. A CI test suite job configured by the ODK is executed automatically once a pull request is created. This job executes a series of standard and customizable tests, such as looking for unsatisfiable classes, malformed cross-references or missing labels.
4. If the test fails, the developer can inspect the execution log and proceed to fix the problem.
5. Once the test passes, another member of the ontology's editorial team reviews the change. Depending on the ontology, one or more approvals may be required before a change is merged in, after which the changes will appear in the release products when the release pipeline is run.

The ODK plays two major roles in the workflow, applying changes from templates and coordinating and executing the test suite that ensures that the edit did not 'break anything', i.e. violate one or more of the QC rules. Templating systems are integrated into the ODK and will be covered in detail in the 'Support ODK Features' section later in the manuscript. The test suite provides built-in QC and CI that can be customized to the individual users' needs.

QC and CI

Even the most experienced ontology curators make mistakes when editing an ontology, from simple ones such as introducing unwanted whitespace in an ontology term label (i.e. a trailing or leading space character), to more complicated errors that lead to unintended logical consequences (e.g. axioms that render a class unsatisfiable or that cause two distinct classes to be wrongly inferred to be equivalent). To avoid adding such 'breaking changes' to the ontology, the ODK uses continuous integration to automatically run QC tests when a pull request is created or updated. These tests can easily be adopted by other version control providers such as Bitbucket and GitLab, as long as they provide a way to run Docker-based workflows. The QC tests are on the ODK Docker image via a Makefile target, 'make test'.

The ODK comes with a wide range of standard QC checks that utilize ROBOT (15), including SPARQL-based validation and logic-based validation. ROBOT incorporates a customizable validation framework for ontologies (called ROBOT report) that performs checks like 'illegal trailing whitespace', 'illegal cross-reference syntax', 'missing license' and others. These checks reflect the best practices of the OBO Foundry—if any are not applicable to a specific ontology, they can be skipped and other checks can be added. Custom SPARQL-based validation provided by the ODK in addition to the standard ROBOT checks follows the same general idea as ROBOT report: an 'anti-pattern', e.g. an undesirable situation like a non-obsolete term without a label, is specified as a SPARQL select query and then executed using ROBOT verify. Logical checks involve running the reasoner using the reason function in ROBOT and ensuring logical coherency (i.e. the absence of unsatisfiable classes) and the absence of unintended logical equivalencies (i.e. cases where a change to the logical axioms lead to two classes being inferred as logically equivalent that are conceptually distinct). The validation framework is made to be easily customizable. For example, an organization might have the requirement that new terms are always signed with a (valid) ORCID (23), so they could implement a SPARQL query which looks for terms in the ontologies' namespace that do not have the respective annotation present and further checks that if the annotation is present, the annotation value is a valid ORCID.

An example of QC implemented in ODK can be found in CL, which has both standard QC checks that come with the

ODK like that mentioned above and also custom checks like 'no label check', 'pmid not in dbxref', 'illegal annotation property' and others. A list of custom checks is included in the ODK yaml set-up file, with corresponding SPARQL query files of the same name in a separate folder called 'sparql'.

The release workflow

Generating release versions of the ontology, including different syntaxes (e.g. JSON, OBO and RDF/XML) and variants (see Section X) is entirely automated as part of an ODK workflow called 'prepare release'. To execute this workflow, the OD simply runs a single command on the command line (sh run.sh make prepare_release). The release workflow executes the following steps:

1. Ensure that any automatically generated components [such as portions of the ontology managed as ROBOT or Dead Simple Ontology Design Patterns (DOSDP) templates] are converted into OWL.
2. Ensure that dynamically imported terms are up-to-date.
3. Generate all release variants, such as the 'base', 'full' and 'simple' variants using standardized ROBOT pipelines. This includes serializing these release variants into all configured formats such as RDF/XML and OBO format and adding versioning information.
4. Execute all QC tests and generate QC reports.

Under the hood, 'make prepare_release' builds all release targets (release file variants which are configured as Makefile targets) by loading the editors file and performing (mostly ROBOT-based) transformation pipelines such as merging, reasoning and adding version information. These release targets in turn depend on others, such as up-to-date imports, which are executed as part of the pipeline. After all the release files (also called release assets) are generated, the OD will usually commit the release files to a branch and optionally ask for a review to ensure that all the changes to the release files are intended. Depending on how releases are being managed, which differs from ontology to ontology, the last step in the release workflow is to publish the release, which usually involves merging the release to the 'main' or production branch and publishing a release (e.g. GitHub release). There are some experimental workflows in ODK that automate even this last part of the release process, but in our experience, ontology curators value the opportunity for a 'final check' before an ontology release is published.

Dependency management: importing and reusing existing ontologies

Ontologies, just like software, can be developed in a modular fashion. Many ontologies make use of an external ontology to provide logical definitions. Previously the ontology community has had a wide range of practices for managing these kinds of inter-ontology dependencies, ranging from copying-and-pasting external terms into an ontology, making duplicative terms in their own ID space or using the owl:imports mechanism. Using imports is considered best practice, but even here there is a range of different practices. Some ontologies import an external ontology in its entirety, while others import subsets of external ontologies, with a diverse range of methods for creating these subsets. Importing an entire ontology can lead to scalability issues, especially when the external ontology is large (e.g. CHEBI(24)). Importing subsets of external ontologies can also be problematic, since these may be transitively imported by other ontologies. Additionally, external subsets can get stale.

The ODK supports what is considered best practice in OBO and aims to make it easy for ontology editors to manage imports. An OD can list the external terms they require in a text file managed in git. They can then trigger an ODK workflow that uses ROBOT to generate an 'import module' using the appropriate ontology modularization method, such as syntactic-locality-based module extraction (SLME) (25).

ODK will also take care of making special releases of ontologies that avoid the stale subset problem. A special reusable component release called a base ontology is created. This component includes any terms belonging to the ontology natively and all their axioms, but does not include any of the imported axioms. This base module can then be used as a modular component by other ontologies. Examples of ontologies that import base files are CL, PATO and PCL. The ODK will continue to evolve with best practices in this area.

Support ODK features

Using template-based workflows for ontology editing

The ODK supports templating systems such as DOSDP (16) and ROBOT templates (15), which allow ontology content to be curated in the form of spreadsheets, which could include new terms, axioms or annotations. These spreadsheets are compiled by the ODK into their OWL representation using a simple pattern. Many such patterns have been developed in recent years and are available to be reused. Increasing commitment to patterns will ensure consistent axiomatization in a scalable manner and increase interoperability and reuse between open ontologies. DOSDP configuration files and templates can be placed in fixed folders in the standard ODK set-up, which will then be automatically integrated into the ontology during the build process.

Auto-generated documentation

Given the complexity of the entire development life cycle of an ontology, it is important to carefully document all workflows. This documentation includes instructions on how to contribute, how to run a release and how to refresh an import. The ODK repository generation process generates a template for such a documentation system and auto-generates documentation of the most important ontology workflows tailored to the ontology. For example, rather than providing generic examples of which files to edit during the editors workflow, the specific files used by the ontology (e.g. cl-edit.owl for the Cell Ontology) are mentioned. The documentation system is based on mkdocs (26) and is easily extensible to accommodate the documentation of custom workflows. When using GitHub as the git hosting provider, updates to the documentation can be automatically deployed using GitHub actions. In addition to auto-generated documentation, we have written a number of tutorials on how to set up a new repository or update an existing one (https://oboacademy.github.io/obook/).

Governance, community requests, QC and releases

The core ODK team selected and centralized the tools required to optimally support the executable workflows described above. In addition to the core tools required (such as ROBOT (15)), we include a wide range of other tools that are useful for processing of ontologies. The ODK is set up as a community-driven resource in which tool requests and suggestions are encouraged on the issue tracker (https://github.com/INCATools/ontology-development-kit). The possible addition of tools is assessed by the core team. In addition to being open source and free to use, tools should be demonstrably useful for managing some part of the ontology life cycle.

The workflows available in the ODK and reported here were designed with OBO principles in mind. We are also working with members of the FAIR semantics (27) community to incorporate practices beyond the OBO standards (for example, by including the owl:versionInfo property as part of the ontology metadata).

Any change to the ODK Docker image, in particular upgrading the tools in the ODK toolbox, is carefully evaluated by a large set of integration tests, which are executed every time a new feature or upgrade is proposed in a pull request and just before every ODK release. These tests include building and running a large variety of different configurations of ontologies, as well as testing the integrity of some of the support tools directly using a shell script which is executed as part of the build process.

The ODK is scheduled for a new release every 3 months, but occasionally, an additional release is required to update a tool that has implemented a critical bug fix. Overall, more than 30 releases were created in the last 3 years. All new tool additions are documented in the changelog of each release. Users can subscribe to be informed about new releases through GitHub's 'Notification' feature.

Use across OBO and beyond

The ODK is designed to allow use at different levels of 'buy in'. Some ontologies are entirely automated using ODK. Others have partial adoption, such as using the ODK container to run custom-built workflows/checks.

We evaluated the use of ODK using a number of methods, but we do not claim to provide a full account of the usage, just a lower bound. We first gathered some of the ontologies that we already knew were using ODK. We also performed GitHub searches to see which additional repositories were using ODK (using search terms like 'ontology development kit' or 'ontology starter kit'). Lastly, we performed an informal user survey targeting the OBO ontology community (distributed via the obo-discuss mailing list), which got 35 responses, 23 of which had used the ODK. Among the surveyed ontologies, 61.3% were in the OBO Foundry.

The ODK Docker image was pulled 74 611 times from Docker Hub at the time of this writing (22.06.2022). Note that this merely establishes a lower bound, as the Docker image is cached locally, and therefore does not need to be pulled more than once per 3 months by most developers. Most of these pulls are probably generated by various continuous integration tools, which are difficult to differentiate from human users. A table of all ODK-based ontology repositories and a summary of ODK versions used can be found in the supplemental materials (Supplementary Table S2, Supplementary Figure S1).

Related work

The ODK is not the only ontology development toolkit used by ODs. In this section, we will highlight some popular tools that are currently being used and how the ODK compares with them.

Firstly, arguably the most popular ontology development tool is Protégé. Protégé (28) is a visual ontology editor that is free and open source. Protégé has a few key features that make it a highly useful tool. Firstly, it has a graphical user interface that is relatively easy to use and navigate, making it highly accessible to non-technical users. It also has a library of plugins that can further extend its functionality, and since it is open source, it is possible to develop new custom plugins. The ODK was developed not as a replacement for Protégé, but rather a complement to it. The ODK is not aimed at the actual process of ontology editing and therefore lacks any support for data entry (such as visual editors). Protégé also has a lightweight cloud-based version (webProtégé) (29) that can be accessed either through their or a hosted server, allowing better support for simultaneous collaborative editing of OWL ontologies. The ODK does not currently integrate with WebProtégé editing workflows, but exploring this integration is on our list for the future.

The OntoAnimal set of tools in conjunction with the eXtensible ontology development (XOD) principles (30) is another toolkit that is widely used across OBO. Our work builds upon the general XOD principles, such as 'reuse' and 'bulk-import', and we hope to reconcile some of our workflows over the coming years. There are some differences to overcome. Firstly, the OntoAnimals toolkit does not provide standardized executable workflows (releases workflows and dynamic imports for an entire project), which is at the heart of the ODK design. Some of the OntoAnimals best practices could therefore simply be integrated into the ODK workflow system. Additionally, the preferred way to extract modules in the wider OntoAnimals culture is to use (MIREOT) Minimum Information to Reference an External Ontology Term (31) modules through a web service. This is different from the ODK culture in two ways: (i) one key design decision of ODK is to go straight to a source ontology from which to import terms, download it and extract a module from the file, while the OntoAnimals approach tends to use the OntoFox web service to extract terms. We prefer our approach because of the lack of intermediary (the imports are less often out-of-date when not using an intermediary) and less pronounced network dependency (the web service might become unavailable or a network could be down). For example, we can extract a module from a 'mirror' over and over without downloading the mirror again. However, there are downsides to our approach: the often quite large ontologies require large amounts of resources (memory and Central Processing Unit (CPU) usage). The best approach has not been definitively determined, but it should be mentioned that OntoAnimals module extraction workflows can be easily integrated into the ODK through its customization features. (ii) A second conceptual 'conflict' between OntoFox and ODK is the use of MIREOT modules vs. SLME (syntactic-locality-based modules). MIREOT modules are often more intuitive for downstream consumers of the ontology, because they only expose the terms that are deliberately imported by the OD, while SLME-based modules import all terms that are necessary to preserve the semantics of the source ontology (which is usually a lot). While we believe that SLME modules are necessary to ensure that our ontology is logically compatible with our dependencies at development time, we believe that neither MIREOT nor SLME modules are sufficient for downstream users. We are currently designing a module extraction algorithm that not only traverses the subclass hierarchy (like MIREOT does) but also takes into account other kinds of relationships like mereological ones (part of, has part) which are, for biological use cases, equally important (often more important).

The NeOn Toolkit (32) is an open-source ontology-engineering environment, built on the code base of OntoStudio (33) (formally known as OntoEdit (34)), that aims to provide comprehensive support for the entire engineering lifecycle by providing an extensive set of plugins. It organizes projects in a workspace; each project can contain multiple ontologies that can be edited in a graphical user interface, which requires more knowledge than Protégé as much of the interface is pretty technical. A major advantage of the NeOn Toolkit is an open platform to which anybody can contribute, hence allowing for a wide variety of plugins developed by the community.

WebODE (35) is an extensible ontology-engineering suite developed over 16 years ago that aimed to be a common workbench for ontology development and management, middleware services (such as access and query services) and ontology-based application development. WebODE has many functionalities including a simple user interface, complete consistency checks, edition through a form-based user interface or a graphical editor and term import. However, WebODE was built to support ontology languages that are no longer in use. Support for the tool was discontinued in 2006, and while it remains open source, no support will be provided when problems are encountered.

Limitations

The dependency on Docker is one of the most significant limitations of the ODK architecture. While Docker itself is not needed to run a Docker container (Docker containers can be run with other container systems such as singularity, which are also widely supported), the non-availability of Docker for some of our users with (i) older Windows machines or (ii) no access to admin privileges on their work machines has been challenging. To mitigate this, we have worked on integrating many of the workflows as GitHub actions. For example, rather than running the command to build and deploy the documentation manually, a GitHub action can be launched to do this. The problem with doing the same for other key workflows like ontology import management or releases is that GitHub actions are limited in how much memory they can use (7 GB at the time of writing), which is often exceeded, especially for larger ontologies.

Discussion and conclusions

Creating and managing biomedical ontologies are complex tasks that require deep technical knowledge. Developing an ontology that is standardized to other ontologies and reuses them compounds this difficulty. This is often prohibitively difficult for many biologists and domain experts whose input is needed to make biomedical ontologies accurate and useful. The ODK provides tools and features that allow non-experts to build and edit ontologies with minimal training. The ODK helps ontologies conform to basic standards and sets users up with structures and documentation for good ontology-engineering practices. The import management system allows non-technical users to reuse existing ontologies, which would otherwise be incredibly complex. The ODK's built-in tools, such as templating systems, further enhance the users' tool belt, allowing highly powerful automation with minimal technical knowledge. Overall, the simplification achieved through the ODK allows users and developers to focus on the content while standardizing good practices and democratizing ontology development.

In a future release of the ODK, we plan to reconcile some of its workflows with other existing frameworks such as OntoAnimals, in particular OntoFox, and work with their developers towards a common solution for, at least, biomedical ontologies. Furthermore, there is a great need for better module extraction techniques for downstream usages as both SLME and MIREOT (the most prevalent approaches) fall short in various ways. Lastly, the ODK does not currently prevent bad ontology modelling—we hope to be able to make stronger use of design pattern-based validation and advanced semantic validation techniques such as (SHACL) Shapes Constraint Language; https://www.w3.org/TR/shacl/, (SHEX) Shape Expression; http://shex.io/ or (LinkML) Linked Open Data Modelling Language; https://linkml.io/ to further reduce the potential for human error.

We have already observed significantly lower error rates in many of the ontologies that use the ODK, thanks to the ability of the automated testing system provided by the ODK to catch errors early on. We hope to be able to roll out ever more useful tests to ever wider circles of ontologies to contribute to a community-wide increase in ontology quality. Our update system allows us to rapidly roll out new features, such as new quality tests and improved pipelines, to all our users by pulling the new Docker image from Docker Hub and running the 'update repo' workflow. Lastly, we seek to harmonize the representation of ontology release files through the use of standard release workflows, which result in standard release serializations (RDF/XML, OBO Flat File and OBO Graphs JSON) and metadata (version IRIs, licence information and more) to make ontologies more FAIR and interoperable.

Supplementary data

Supplementary data are available at *Database* Online. All code described in this manuscript can be found at https://github.com/INCATools/ontology-development-kit

Funding

Office of the Director, National Institutes of Health (R24-OD011883); National Human Genome Research Institute, 'Phenomics First' (RM1HG010860 to D.O-S., N.M., R.S. and A.R.C.); National Institutes of Mental Health (1RF1MH123220-01 to S.Z-K.T., H.K. and D.O-S.); National Heart, Lung, and Blood Institute 5U01HG009453-03; UK Biotechnology and Biological Sciences Research

Council/US National Science Foundation Directorate of Biological Sciences (BBSRC-NSF/BIO BB/T014008/1); The Wellcome Trust, 'Virtual Fly Brain' (105023MA); Director, Office of Science, Office of Basic Energy Sciences, of the US Department of Energy (DE-AC0205CH11231 to C.J.M.); European Molecular Biology Laboratory - European Bioinformatics Institute core funds (D.O-S., R.S., A.R.C., S.Z-K.T. and H.K., in part).

Conflict of interest

None declared.

References

1. Wilkinson,M.D., Dumontier,M., Aalbersberg,I.J.J. *et al.* (2016) The FAIR guiding principles for scientific data management and stewardship. *Sci. Data*, **3**, 160018.
2. Haendel,M., Su,A. and McMurry,J. (2016) FAIR-TLC: metrics to assess value of biomedical digital repositories: response to RFI NOT-OD-16-133; (2016).
3. Goodman,A., Pepe,A., Blocker,A.W. *et al.* (2014) Ten simple rules for the care and feeding of scientific data. *PLoS Comput. Biol.*, **10**, e1003542.
4. Tang,Y.A., Pichler,K., Füllgrabe,A. *et al.* (2019) Ten quick tips for biocuration. *PLoS Comput. Biol.*, **15**, e1006906.
5. McMurry,J.A., Juty,N., Blomberg,N. *et al.* (2017) Identifiers for the twenty-first century: how to design, provision, and reuse persistent identifiers to maximize utility and impact of life science data. *PLoS Biol.*, **15**, e2001414.
6. Hoehndorf,R., Schofield,P.N. and Gkoutos,G.V. (2015) The role of ontologies in biological and biomedical research: a functional perspective. *Brief. Bioinform.*, **16**, 1069–1080.
7. Smith,B., Ashburner,M., Rosse,C. *et al.* (2007) The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nat. Biotechnol.*, **25**, 1251–1255.
8. Jackson,R., Matentzoglu,N., Overton,J.A. *et al.* (2021) OBO Foundry in 2021: operationalizing open data principles to evaluate ontologies. *Database*, **2021**.
9. Köhler,S., Gargano,M., Matentzoglu,N. *et al.* (2021) The human phenotype ontology in 2021. *Nucleic Acids Res.*, **49**, D1207–D1217.
10. Bard,J., Rhee,S.Y. and Ashburner,M. (2005) An ontology for cell types. *Genome Biol.*, **6**, R21.
11. Diehl,A.D., Meehan,T.F., Bradford,Y.M. *et al.* (2016) The cell ontology 2016: enhanced content, modularization, and ontology interoperability. *J. Biomed. Semant.*, **7**, 44.
12. Mungall,C.J., Torniai,C., Gkoutos,G.V. *et al.* (2012) Uberon, an integrative multi-species anatomy ontology. *Genome Biol.*, **13**, R5.
13. Gkoutos,G.V., Green,E.C.J., Mallon,A.-M. *et al.* (2005) Using ontologies to describe mouse phenotypes. *Genome Biol.*, **6**, R8.
14. Tan,S.Z.K., Kir,H., Aevermann,B. *et al.* (2021) Brain Data Standards Ontology: a data-driven ontology of transcriptomically defined cell types in the primary motor cortex. *bioRxiv*. 10.1101/2021.10.10.463703.
15. Jackson,R.C., Balhoff,J.P., Douglass,E. *et al.* (2019) ROBOT: A tool for automating ontology workflows. *BMC Bioinform.*, **20**, 407.
16. Osumi-Sutherland,D., Courtot,M., Balhoff,J.P. *et al.* (2017) Dead simple OWL design patterns. *J. Biomed. Semant.*, **8**, 18.
17. Mungall,C., fbastian, kltm, Douglass,E. *et al.* (2020) owlcollab/owltools: 2020-04-06.
18. Steigmiller,A., Liebig,T. and Glimm,B. (2014) Konclude: system description. *J. Web Semant.*, **27–28**, 78–85.
19. Jordan,H., Scholz,B. and Subotić,P. (2016) Soufflé: on synthesis of program analyzers *Toronto, ON, Canada* . In: *Computer Aided Verification*. Springer International Publishing, pp. 422–430.
20. Mungall,C., Hegde,H., Kalita,P. *et al.* (2022) INCATools/ontology-access-kit: v0.1.22.
21. Matentzoglu,N., Balhoff,J.P., Bello,S.M. *et al.* (2022) A simple standard for sharing ontological mappings (SSSOM). *Database*, **2022**.
22. Merkel,D. (2014) Docker: lightweight Linux containers for consistent development and deployment. *Linux J.*
23. (2009) Credit where credit is due. *Nature*, **462**, 825.
24. Hastings J *et al* 2012 The ChEBI reference database and ontology for biologically relevant chemistry: enhancements for 2013 Nucleic Acids Research **41**, D456–D463. 10.1093/nar/gks1146.
25. Grau,H., Horrocks,I., Kazakov,Y. *et al.* 2008 Modular reuse of ontologies: theory and practice. *J. Artif. Intell.* **31**, 273–318.
26. Christie,T. (2014) MkDocs. Project documentation with MarkDown.
27. Franc,Y.L., Coen,G., Essen,J.P. *et al.* (2020) D2.2 FAIR semantics: first recommendations.
28. Musen,M.A. and Protégé Team. (2015) The Protégé project: a look back and a look forward. *AI Matters*, **1**, 4–12.
29. Horridge,M., Gonçalves,R.S., Nyulas,C.I. *et al.* (2019) WebProtégé: a cloud-based ontology editor. In: *Companion Proceedings of the 2019 World Wide Web Conference, WWW '19*. Association for Computing Machinery, New York, USA, pp. 686–689.
30. He,Y., Xiang,Z., Zheng,J. *et al.* (2018) The eXtensible ontology development (XOD) principles and tool implementation to support ontology interoperability. *J. Biomed. Semant.*, **9**, 3.
31. Courtot M, Courtot M, Gibson F, Lister A, Malone J, Schober D, Brinkman R and Ruttenberg A 2009 MIREOT: the Minimum Information to Reference an External Ontology Term Nature Precedings. 10.1038/npre.2009.3576.
32. Erdmann,M. and Waterfeld,W. (2012) Overview of the NeOn Toolkit. In: Suárez-Figueroa MC, Gómez-Pérez A, Motta E *et al.* (eds). *Ontology Engineering in a Networked World*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 281–301.
33. Weiten,M. (2009) OntoSTUDIO® as a ontology engineering environment. In: Davies J, Grobelnik M, Mladenić D (eds). *Semantic Knowledge Management: Integrating Ontology Management, Knowledge Discovery, and Human Language Technologies*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 51–60.
34. Sure,Y., Erdmann,M., Angele,J. *et al.* (2002) OntoEdit: collaborative ontology development for the semantic web *Sardinia, Italy*. In: *The Semantic Web—ISWC 2002*. Springer Berlin Heidelberg, pp. 221–235.
35. Arpírez,J.C., Corcho,O., Fernández-López,M. *et al.* (2001) WebODE: a scalable workbench for ontological engineering. In: *Proceedings of the 1st international conference on Knowledge capture, K-CAP '01*. Association for Computing Machinery, New York, USA, pp. 6–13.