

OntoFlow: A user-friendly Ontology Development Workflow

Gordian Dziwis¹, Lisa Wenige¹, Lars-Peter Meyer¹ and Michael Martin¹

¹Institute for Applied Informatics, Goerdelerring 9, 04109 Leipzig, Germany

Abstract

For many years, the development of widely applicable and high-quality ontologies has been an ongoing research topic. Among the various challenges, the lack of integrated development environments for non-technical domain experts has been one of the most pressing research issues. But while the participation of domain experts is vital for the applicability of ontologies, there are hardly any software tools available that facilitate their active engagement. We present a solution that addresses this research gap by automating the ontology development process with the help of a workflow engine. We define a pipeline that facilitates ontology implementation, serialization, documentation and testing within the scope of a seamless automatic routine that can be easily set up by the ontology engineer and triggered by a non-technical domain expert. Thus, the processing pipeline takes care of most of the operations that usually have to be carried out by an ontology or software engineer. We demonstrate the applicability of the approach by developing an ontology with OntoFlow and validating its functioning with a large-scale ontology dataset from Linked Open Vocabularies (LOV).

Keywords

Ontology, Workflow, Integrated Development Environment, Quality Assurance

1. Introduction

With the proliferation of knowledge graphs, ontology development has become a central part of data integration activities. Due to the increasing efforts around FAIR data and research data infrastructures worldwide [1], collaborative and decentralized processes around ontology development are on the rise.

However, available software tools fall short of the multi-layered requirements of ontology development which comprises labor-intensive tasks such as ontology modeling [2], serialization, validation and documentation [3]. Moreover, these tasks often have to be carried out collaboratively and in a decentralized manner as several (often geographically dispersed) groups of people are involved in the ontology development process, thus making it a multi-stakeholder endeavor [4]. In addition to that, many of the tools available are difficult to handle for domain experts without an IT or Semantic Web background [5]. While these experts have an extraordinarily high level of expertise in their field (e.g., in life sciences, physics or cultural heritage)


SemIIM'22: 1st International Workshop on Semantic Industrial Information Modelling, 30th May 2022, Hersonissos, Greece, co-located with 19th Extended Semantic Web Conference (ESWC 2022)

✉ dziwis@infai.org (G. Dziwis); wenige@infai.org (L. Wenige); lpmeyer@infai.org (L. Meyer); martin@infai.org (M. Martin)

🆔 0000-0002-9592-418X (G. Dziwis); 0000-0002-3707-3452 (L. Wenige); 0000-0001-5260-5181 (L. Meyer); 0000-0003-0762-8688 (M. Martin)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

they might not be as familiar with the methods and tools commonly applied in software and knowledge graph engineering.

This situation is exacerbated by the fact that there exists no standard tool stack for ontology development in the Semantic Web community. Although some tools, such as Protégé [6], are considerably widespread, they only cover partial aspects of the engineering process such as ontology modeling or editing and cannot be used in the sense of a fully-fledged integrated development environment (IDE). On top of that, many software tools for RDF file processing can either only be used as a command-line utility and/or provide limited usability in terms of a graphical user interface (GUI). However, virtualization and continuous integration (CI) techniques from software engineering provide novel opportunities for ontology development [7] and can also help to ease participation of domain experts.

With OntoFlow we propose a solution that bundles several necessary operations of ontology engineering in a single automated workflow by adopting best practices from CI pipelines [8]. Thus, we are able to integrate ontology modeling, serialization, testing and documentation in a unified process which makes time-consuming, repetitive and error-prone manual work mostly superfluous.

The main contributions of the paper are as follows:

- Survey of the state-of-the-art of software-aided ontology engineering (Sect. 2).
- Conceptualization and implementation of the **software tool OntoFlow that supports ontology development through automatic pipelines** to help users **develop ontologies faster and easier** (Sect. 3).
- Demonstration of the feasibility of OntoFlow by evaluating its performance for all ontologies listed in the Linked Open Vocabularies (LOV) repository (Sect. 4).
- Summarizing the most important findings and outlining future research directions for improving ontology development with OntoFlow. (Sect. 5).

2. Related Work

Although some existing software applications already support collaborative ontology development processes and also automate them in parts, there is a lack of approaches as to how these processes can be linked in the sense of a CI pipeline while at the same time ensuring that laymen can actively participate in ontology engineering through making changes and triggering updates [9]. Important impulses for designing fully-fledged pipelines for software-aided ontology development come from the following sub-disciplines.

2.1. DevOps for Data Collections

Continuous development and integration strategies have become an indispensable part of modern software engineering. They largely consist of clean-up operations, compilation of executables, application of automated tests, and the deployment of the finished application, including generation of appropriate documentation if necessary. Part of these processes is the

constant checking of any updates/new versions, as well as the triggering of troubleshooting activities if problems occur [7]. This ensures that software solutions are always up-to-date, that applications meet predefined quality standards and rely on stable software artifacts. Likewise, in the wake of ever-growing data volumes and increased relevance of data-driven applications, effective mechanisms to control the quality-assured publication of data products are required for IT operations. Processes that follow CI principles are equally needed for efficient production of data artifacts. The knowledge graph community has been one of the first to adopt DevOps best practices for datasets since it heavily relies on high-quality data schemas and reproducible workflows for data conversion, integration and fusion. In this line of research, several authors have proposed automated data pipelines that take care of data transformation, apply quality assurance operations and automatic procedures for data publication and effective description of data artifacts [10, 11, 12, 13, 14, 15, 16, 17]. CI mechanisms for data collection involve operations such as crawling, linking, or data transformation. These processes are usually executed automatically and are not interrupted by user interactions and manual intervention. If people work with software tools in this context, they are mostly data scientists or software engineers.

2.2. Ontology Editors

This modus operandi differs from the determining factors in ontology development. The creation of an ontology usually involves several experts with diverse backgrounds and different levels of technical expertise. Therefore, an effective ontology development environment/pipeline has to foster collaboration as well as provide a graphical editor to effectively support development processes. By this means, even domain experts with no IT expertise can actively take part in the creation of ontologies.

Ontology editors such as OntoEdit [18], OntoSeer [19], Protégé [6], Vocol [20] or WebVOWL [21] are already established software tools for ontology development and visualization. Additionally, the Semantic Application Design Language offers an English-like language for semantic modeling¹. SemML uses a template-based approach to help domain experts create ontologies [22]. Other applications in this area focus more on aspects of collaborative and version-based storage of ontologies so that changes can be managed decentrally and tracked over time. Software tools, such as Ontology [23] or the QuitStore [24] provide solutions for these kinds of requirements.

2.3. Ontology Documentation & Quality Assurance

Tools, such as Oops! focus more on the aspect of quality assurance [25] while general-purpose RDF data testing tools, such as RDFUnit [26] or pySHACL [27] can be equally applied for ontology testing. Just as important as quality assurance of ontologies is documentation for end-users. Software applications that automatically generate ontology documentations are WIDOCO [28], LODÉ [29] or pyLODE². They create an HTML representation from ontologies in standard RDF serialization formats. WIDOCO even goes beyond the purely automatic creation of ontology documentation by enabling metadata enrichment or ontology testing.

¹<https://github.com/SemanticApplicationDesignLanguage/sadl>

²<https://github.com/RDFLib/pyLODE>

Table 1
Requirements for OntoFlow

RQ1 ODP Automation	
RQ1.1 Ontology Serialization	Ontology artifacts can be automatically serialized in common RDF formats
RQ1.2 Ontology Validation	Automatic testing of ontology artifacts is integrated
RQ1.3 Ontology Post-processing	Integrating automatic post-processing operations, such as version control or diff detection
RQ1.4 Ontology Documentation	Automatic creation of a HTML ontology documentation
RQ1.5 Ontology Publication	Automatic deployment of ontology artifacts (serialization and HTML documentation) to a server
RQ2 High (Re-)Usability	
RQ2.1 GUI support	Support of ontology modelling through a graphical user interface
RQ2.2 Easy Execution	Domain experts (with little to no IT background) should be able to trigger ontology workflows
RQ2.3 Fast Execution	It should be possible to quickly generate an ontology, validate it and create its documentations
RQ2.4 Easy Modification	Ontology developers should be able to modify them easily to suit their individual requirements

However, it does not provide an easy-to-use interface for domain experts. Moreover, it is a Java-based monolithic software, which makes modifications and extensions with tools from other technology stacks (e.g., Python-based software applications) difficult. The ROBOT framework offers many operations needed for ontology manipulation in the development process but lacks workflow capabilities. The framework's documentation refers to the rather technical oriented *Make*³ tool for more complex needs.

3. Ontology Development Workflow

3.1. Requirements

The goal of OntoFlow is the best possible optimization and automation of ontology development processes which typically currently involve a great number of labor-intensive tasks, such as modeling, serialization, updating, testing and documentation. Due to the fact that such development processes are carried out by several stakeholders, the workflow environment should foster a collaborative and decentralized way of working.

Since the development of ontologies often involves domain experts who have only limited expertise in the field of software and data engineering, OntoFlow should provide a GUI for

³<https://www.gnu.org/software/make/>

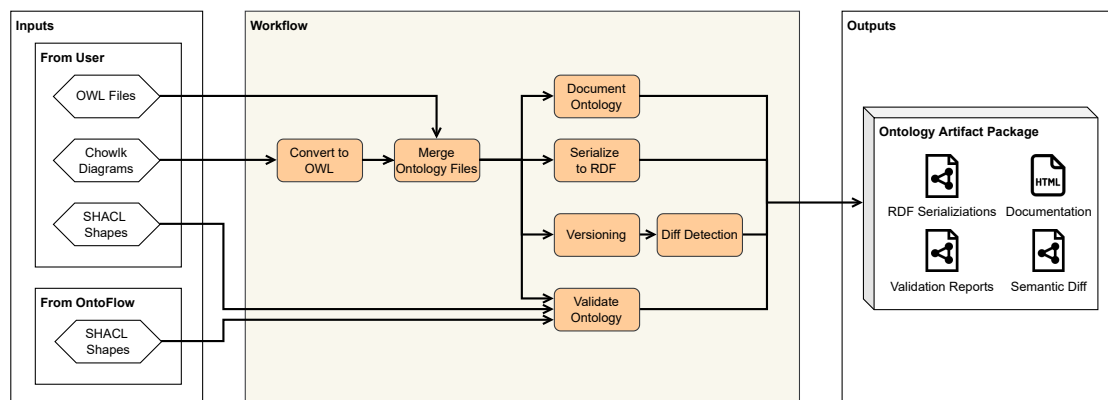


Figure 1: OntoFlow Workflow

editing the ontology while automating the most common tasks (e.g., bash scripting and *Git* interaction) in the background. Due to the limited IT experience of some involved stakeholders, it is also vital that workflows can be triggered without in-depth technical understanding of software development or semantic technologies. Reducing the amount of necessary skills for the domain expert is critical, because generally ontology development is a one-time job for them. A reusable ontology workflow setup directed at domain experts helps to avoid costs for learning the involved technologies and setting up a development and hosting infrastructure (e.g., running a web server for publishing the ontology and documentation) which - apart from working time - can incur further costs for licenses and hardware. Because there is no standard established methodology for ontology development, and it happens in diverse environments, OntoFlow must be flexible and facilitate easy modifications to accommodate different needs. Table 1 gives an overview of the requirements detailed in the above sections.

3.2. Workflow Structure

Figure 1 gives an overview of the workflow. Data inputs for the workflow are generated by the ontology developer. He/she provides ontology diagrams, OWL files and validation shapes. An ontology diagram is a visual representation of an ontology. The OWL files define ontologies with the Web Ontology Language (OWL) and are serialized as an RDF file. The schema is tested with SHACL shapes to ensure that it satisfies certain conditions. A set of validation shapes, which test the compliance to best practices for ontology development, are supplied by OntoFlow.

The first process is the transformation of the ontology diagrams to OWL files and merging those with (potentially existent) other OWL files provided by the user. The resulting ontology is the input for the following processing steps carried out in parallel: A HTML documentation is generated, describing the ontology’s metadata, classes and properties. The ontology is serialized into multiple RDF formats. During validation, the ontology is tested against some previously defined SHACL shapes. If needed, OntoFlow can be adapted so that the publication of the ontology fails whenever certain SHACL constraints are violated. OntoFlow picks up

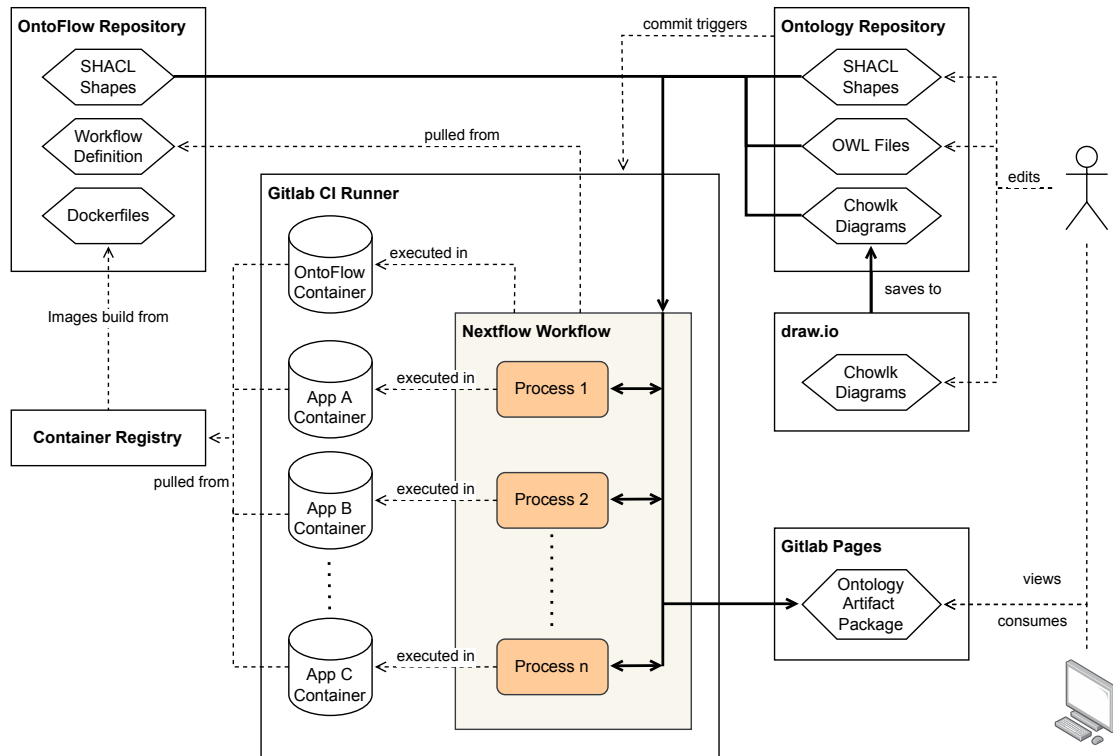


Figure 2: Component Diagram of the OntoFlow Architecture

the 'owl:priorVersion' property and detects semantic differences between versions. This gives an overview of which classes changed in the current version compared to the previous one. The serializations, documentation and validation reports as well as the semantic differences compose the Ontology Artifact Package.

3.3. Implementation and Architecture

The architectural model of OntoFlow is a pipeline consisting of processes where each operation takes inputs, transforms them and outputs the results. Process inputs and outputs are interconnected. In the component diagram 2 this is represented with the beige box labeled "Nextflow Diagram". Operations are carried out by different application programs with a command-line interface. All applications are containerized, and the container images are defined by a configuration file. A workflow engine manages container life cycles, executes the applications and pipes the inputs and outputs between the processes. The pipeline operations with their executing containers and commands as well as inputs and outputs are defined in a workflow file. Each application container is based on an image for which a configuration file exists. There is a container for OntoFlow itself, which has all the necessary dependencies for running the workflow engine. The images are hosted in a container registry. A continuous integration script builds the images and pushes them to the registry. Image, as well as ontology files, are managed inside *Git* repositories to enable version control.

OntoFlow provides a continuous integration script, which is triggered whenever a change in the ontology repository occurs. When executed, the OntoFlow container is started with the mounted ontology repository. Inside the container, the engine initializes the workflow with the files from the ontology repository and OntoFlow's validation shapes as inputs. Each process step is then executed in its defined application container. The final outputs are deployed to their target environment.

The pipeline was implemented as a Nextflow [30] workflow and Docker was chosen as the container engine. For each tool used in a process step, a Docker image was chosen or created. The following list describes the software tools used in OntoFlow:

- **Chowlk Converter** and **draw.io**⁴: Converts the ontology diagrams to OWL files. Chowlk[31] provides a library of OWL diagram elements for draw.io⁵ which is an open source diagram software.
- **pySHACL**: Validates the ontology against SHACL shapes.
- **Bubastis**⁶: Creates a semantic *diff* in comparison to an older ontology version.
- **Jena Riot**⁷: Serializes and merges RDF files.
- **sparql-integrate**⁸: Extracts and manipulates data in RDF files.
- **pyLODE**: Creates the ontology documentation in HTML.

OntoFlow can be run on any Linux system with Java and Docker installed, which facilitates easy modifications and extensions. OntoFlow is integrated into the GitLab ecosystem, thus boosting the options for automation. GitLab's container registry hosts the tool images, and its continuous integration infrastructure is utilized for running OntoFlow. The Ontology Artifact Package is hosted as a GitLab page. With the schema files managed in a GitLab repository, it is possible to edit the ontology collaboratively, because draw.io can use a GitLab repository as its storage backend. This also works when multiple persons are editing the same diagram. This provides the option for collaborative editing without the user needing to explicitly interact with *Git*. Chowlk diagrams are draw.io diagrams of an ontology or parts of an ontology. They are defined with elements from the Chowlk Ontology Visual Notation library, exported to an XML File. A line-wise diff comes for free through the usage of a *Git* repository. For documentation purposes, the semantic *diff* provided by Bubastis[32] gives a more useful overview over the changes between ontology versions. Bubastis analyzes and reports on the five major types of ontology changes.

Before an ontology *diff* can be created, the serialized file and location of the previous ontology version has to be determined. To achieve this, we run a custom SPARQL query against the input

⁴<https://github.com/oeg-upm/Chowlk>

⁵<https://app.diagrams.net/>

⁶<https://github.com/EBISPOT/bubastis>

⁷<https://jena.apache.org/documentation/io/>

⁸<https://github.com/SmartDataAnalytics/RdfProcessingToolkit>

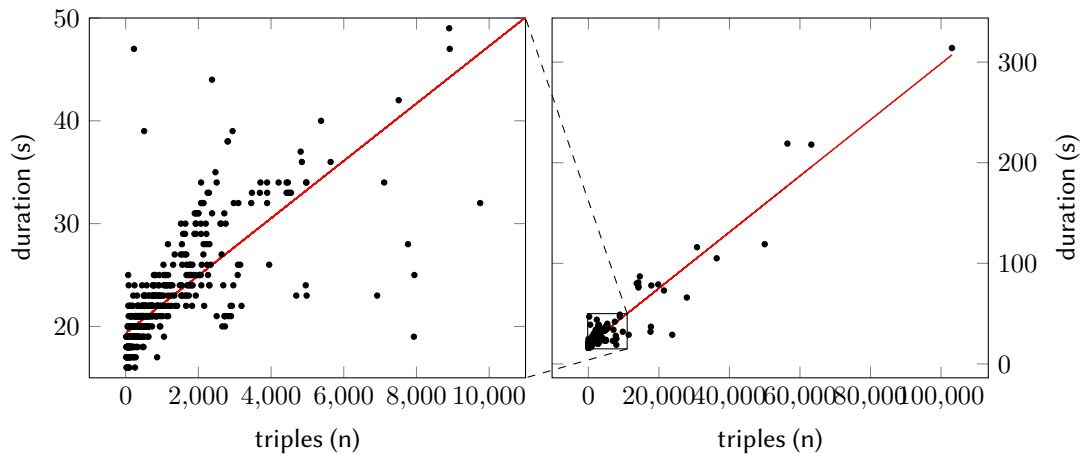


Figure 3: OntoFlow Execution Durations, for normal and all triple counts

RDF file with SPARQL-integrate and output the URI locating the prior ontology version. We curated a set of tools that fulfill our functional requirements, packaged each in a Docker container, wired them together with Nextflow and integrated the resulting workflow with the GitLab ecosystem. The OntoFlow code and documentation are available on GitLab, so that anyone interested can use the software and see how the tool works.⁹

4. Evaluation

We verified the proper execution of the workflow in Section 4.1, by testing it on a large-scale benchmark dataset of publicly available ontologies, to be able to make some valid statements about scalability and applicability of our approach for a wide range of domains. In order to validate our approach, we evaluated in Section 4.2 to what extent the requirements formulated in Section 3 are fulfilled, in regard to the implementation and the real world use of OntoFlow in the KupferDigital project.

4.1. Benchmark with LOV Ontologies

We obtained the benchmark dataset from the ontology repository *Linked Open Vocabularies* and downloaded the corresponding ontology files.¹⁰ We decided to use LOV as a reference point for benchmark generation as it is one of the most comprehensive collection of ontologies and vocabularies throughout the Semantic Web [33].

In total, we extracted 799 files from the LOV registry¹¹. We counted the number of triples of the ontologies and processed each ontology with OntoFlow. Since LOV ontologies are already available in a common RDF format and no ontology diagrams are provided for them, we skipped

⁹<https://gitlab.com/infai/ontoflow>

¹⁰<https://lov.linkeddata.es/dataset/lov/sparql>

¹¹<https://gitlab.com/infai/ontoflow-benchmark/-/tree/main/ontologies>

the processing step of ontology serialization from the workflow for these vocabularies. We logged the processing time and additional information on whether the workflow completed successfully. From the 799 ontologies, OntoFlow was able to process all of them and did produce their output files (i.e., Ontology Artifact Packages). We also randomly selected 25 ontologies and checked their output files for plausibility and consistency after the workflow had been executed. OntoFlow produced syntactically correct and plausible ontology files, error reports and documentations for each one of these ontologies.

Because the architectural model is a pipeline with no conditional branching and the inputs and outputs can go only one way through the workflow steps, we are confident that OntoFlow also processed the other ontologies successfully and thus provides a stable environment for the generation, serialization, validation, post-processing and publication of ontologies.

The successful execution of the workflow for numerous ontologies demonstrates that OntoFlow is a valuable software tool for automating ontology development processes for a wide range of application domains.

4.2. Requirements Evaluation

OntoFlow's implementation was tested during the materials science project KupferDigital, which intends to develop a platform for the digitalization of materials research and the metal processing industry. Several experts and institutes are currently involved in the development of a set of ontologies, which semantically represent the work involved in the development of copper materials, from ore extraction and alloy development to production and recycling of the material. Each sub-ontology can consist of several components that can each be modeled with Chowlk, subsequently processed, merged and documented with OntoFlow. The advantage of the approach is that domain experts can work independently as well as collaboratively on the separate ontology components as the corresponding files are stored in a GitLab repository which serves as a storage backend for draw.io. At the moment, numerous domain experts are working simultaneously on 14 different sub-ontologies from the copper domain. For each change made to a sub-ontology, OntoFlow is triggered and a new version of the ontology artifact package is created. Beta versions of those ontologies developed with OntoFlow are publicly available.¹²

The following list summarizes the OntoFlow functionalities with regard to the previously identified requirements in light of the insights that were gained during the use case, our implementation and the benchmark.

RQ1.1 Ontology Serialization: OntoFlow can output any RDF serialization supported by Apache Riot.

RQ1.2 Ontology Validation: The ontology is validated against SHACL shapes and the result is added to the output.

RQ1.3 Ontology Post-processing: OntoFlow picks up a prior version from the ontology and outputs a semantic *diff*. Currently, OntoFlow is not able to add the metadata about versioning during a release automatically.

¹²<https://gitlab.com/kupferdigital/wiki/-/wikis/Ontologies#subontologies>

RQ1.4 Ontology Documentation: OntoFlow does generate HTML documentation with pyLODE and is also able to add diagrams to illustrate the modeling of ontology parts. The semantic *diff*, serializations and validation results are not yet integrated into the documentation. We are waiting for a rewrite of pyLODE, which will make it easier to modify the HTML templates.

RQ1.5 Ontology Publication: When set up as a continuous integration job, OntoFlow completely automates the publishing process. There are two limitations. One is, when published on GitLab pages, the URI has always the schema `http(s)://groupname.example.io/projectname`. For a custom URI a DNS record, which redirects to the GitLab page URI, has to be set up. Secondly, GitLab pages do not support content negotiation¹³. With content negotiation, an HTTP client can specify which type (or types) of content it would prefer to receive when it attempts to dereference a URI. For example, while a browser accessing the ontology's URI would like to receive the ontology's documentation, a script would prefer to receive a machine-readable serialization.

RQ2.1 GUI support: The ontology can be built in draw.io with diagram elements from the Chowlk shape library, which is extensively documented.¹⁴ Because the Chowlk diagram depicts OWL and not a general RDF graph, it is not possible to model all aspects of the ontology with a diagram. Metadata like provenance, examples for classes or comments have to be defined in an additional RDF file. The domain experts liked the visual definition approach, due to the similarity to object-oriented modeling approaches. But while this reduces the barrier of entry for domain experts, this familiarity is also a fallacy, and we see a risk for badly designed diagrams. We got the feedback from the domain experts, that OntoFlow's output from the process steps is quite technical and difficult to use. At the moment, it is only accessible via scanning the CI pipeline's job log.

RQ2.2 Easy execution: When run as a GitLab CI job, OntoFlow starts automatically upon saving the draw.io XML. Setting up this automated workflow is done by copying a CI configuration file to the ontology repository. Running OntoFlow locally only requires Java, Docker and Nextflow installations to be present on the host machine. OntoFlow can then be triggered by a CLI command. With the help of the documentation, the domain experts were able to set up the CI job on their own. They did not try to run OntoFlow locally, because they are forced by their IT to use Windows as their OS. This was also an issue for an industry partner, who also had policies against using external repositories and could not execute OntoFlow at all.

RQ2.3 Fast execution: Benchmark evaluations on the LOV dataset revealed that OntoFlow's execution time scales linearly with the number of triples. Fig. 3 shows the relationship between the triple count in an ontology and the OntoFlow's processing time. It took OntoFlow at least 17 seconds to process an ontology, and the maximum workflow duration was 314 seconds for an ontology with 103,098 triples. A linear regression model with a coefficient determinant of 0.91 is a good fit to model the relationship between triple count and execution time. Workflows were run on a 4-core laptop with a solid-state disk. The zoomed graph from Fig. 3 shows that for most ontologies, durations are clustered around 25s. Execution time could be further reduced by optimizing the architectural setup, but the need did not arise during the development of the

¹³<https://lov.linkeddata.es/dataset/lov/sparql>

¹⁴<https://chowlk.linkeddata.es/notation.html>

copper ontology.

RQ2.4 Easy modification: The main component of OntoFlow is the Nextflow script that describes the workflow. With 215 lines of code, it is very short, and it follows a simple logic of shell commands which exchange files as their inputs and outputs. Everyone familiar with basic command-line skills can modify this script (e.g., by modifying or adding workflow steps) to suit their needs. OntoFlow’s only dependencies are Java, Nextflow and Docker, so starting development only encompasses installing those and cloning the OntoFlow repository.

5. Conclusion & Future Work

We have presented the OntoFlow approach that automates ontology development processes. By transferring CI techniques from software engineering to the domain of ontology and data management, we are able to perform previously complex and error-prone ontology engineering tasks faster and without the excessive labor that was previously required. This is made possible by the use of virtualized containers that take care of different steps of ontology processing. The containers are intelligently linked to reduce the need for manual interventions. In this way, the development process consisting of ontology modeling, editing, serialization, validation, and documentation can be realized faster and easier for the end-user. In addition, the use of GUI-based open-source modeling software and its integration with a tool for automated shape-to-RDF conversion enables participation of domain experts who have little to no Semantic Web expertise. Simultaneous collaborative engagement of different stakeholders in multiple geographically dispersed locations is enabled through Git integration.

OntoFlow workflows are executable for any type of ontology and can be set up by configuring a GitLab repository with ontology files and a CI configuration. The containerized bundling of different components implements a modular structure that ensures easy maintainability and reusability. We have tested the correct operation of the approach for the use case of the copper ontology. The artifacts automatically generated by the workflow can be viewed publicly (see Sect. 4).

In addition, we have carried out a large-scale evaluation on an ontology benchmark dataset from *Linked Open Vocabularies* to test the scalability, feasibility and applicability of the approach for a wide range of ontologies. The evaluation has shown that OntoFlow is easily executable for the several hundred ontologies listed in LOV, as the workflow completed successfully for each example in the dataset. In addition, performance analysis has shown that OntoFlow’s execution time scales linearly with the number of triples. It is also positive to note that processing times for small to medium-sized ontologies (< 10,000 triples) are in the range of seconds. This could probably be reduced even further when one would build a single container for all tools as in a public GitLab pipeline each image get pulled for each execution.

To make the pipeline output better accessible for the domain experts, we started with rendering SHACL validation results with the help of a Jekyll RDF¹⁵ template and include more artifacts to the rendered documentation.

We did not expect, that being able to draw arbitrary diagrams with draw.io could also be useful. The domain experts can start by using draw.io for expressing the shared conceptualization with

¹⁵<https://github.com/AKSW/jekyll-rdf>

free form diagrams in an informal, intuitive way and then refine it to a “formal specification of a shared conceptualization” [34]. We will research if it is possible to integrate more aspects of the informal conceptualization phase of the ontology development process, like a glossary or competency questions, into OntoFlow. For supporting the formal specification, we will extend OntoFlow with a reasoner and evaluate if the output from the reasoner can be integrated in a meaningful way. We also aim to extend the *git diff*-based version tracking for ontologies so that major changes (e.g., in the class hierarchy) are automatically recorded in the documentation. Another line for future research is the conceptualization and implementation of effective and modularized workflows for automatic ontology-based data transformation, post-processing and linking tasks.

References

- [1] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, et al., The fair guiding principles for scientific data management and stewardship, *Scientific data* 3 (2016) 1–9.
- [2] N. F. Noy, D. L. McGuinness, et al., *Ontology development 101: A guide to creating your first ontology*, 2001.
- [3] R. Subhashini, J. Akilandeswari, A survey on ontology construction methodologies, *International Journal of Enterprise Computing and Business Systems* 1 (2011) 60–72.
- [4] Y. Sure, S. Staab, R. Studer, *Ontology engineering methodology*, in: *Handbook on ontologies*, Springer, 2009, pp. 135–152.
- [5] T. Tudorache, *Ontology engineering: Current state, challenges, and future directions*, *Semantic Web* 11 (2020) 125–138. doi:10.3233/SW-190382.
- [6] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, S. W. Tu, The evolution of protégé: an environment for knowledge-based systems development, *International Journal of Human-computer studies* 58 (2003) 89–123.
- [7] M. Fowler, M. Foemmel, *Continuous integration (2006)*. URL: <https://www.martinfowler.com/articles/continuousIntegration.html>.
- [8] J. Humble, D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Addison-Wesley Signature Series (Fowler), Pearson ITP, 2010.
- [9] C. Mungall, H. Dietze, S. Carbon, A. Ireland, S. Bauer, S. Lewis, *Continuous integration of open biological ontology libraries*, in: *The 15th Annual Bio-Ontologies Meeting 2012*, 2012.
- [10] S. Cirulli, *Continuous integration for xml and rdf data*, *XML LONDON* (2015) 52–60.
- [11] J. Klímek, P. Skoda, M. Necaský, *Requirements on linked data consumption platform.*, in: *LDOW@ WWW*, 2016.
- [12] J. Kucera, D. Chlapek, J. Klímek, M. Necaský, *Methodologies and best practices for open data publication.*, in: *DATESO*, 2015, pp. 52–64.
- [13] R. Meissner, K. Junghanns, *Using devops principles to continuously monitor rdf data quality*, in: *Proceedings of the 12th International Conference on Semantic Systems*, 2016, pp. 189–192.
- [14] J. A. Rojas Meléndez, B. Van de Vyvere, A. Gevaert, R. Taelman, P. Colpaert, R. Verborgh, *A preliminary open data publishing strategy for live data in flanders*, in: *Companion Proceedings of the The Web Conference 2018*, 2018, pp. 1847–1853.
- [15] D. Roman, M. Dimitrov, N. Nikolov, A. Putlier, D. Sukhobok, B. Elvesæter, A. Berre, X. Ye, A. Simov, Y. Petkov, *Datagraft: Simplifying open data publishing*, in: *European Semantic Web Conference*, Springer, 2016, pp. 101–106.
- [16] C. Stadler, L. Wenige, M. Martin, S. Tramp, K. Junghanns, *Rdf-based deployment pipelining for efficient dataset release management.*, in: *SEMANTICS Posters&Demos*, 2019.
- [17] M. Brümmer, C. Baron, I. Ermilov, M. Freudenberg, D. Kontokostas, S. Hellmann, *Dataid: Towards semantically rich metadata for complex datasets*, in: *Proceedings of the 10th International Conference on Semantic Systems*, 2014, pp. 84–91.
- [18] Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, D. Wenke, *Ontoedit: Collaborative*

- ontology development for the semantic web, in: International semantic web conference, Springer, 2002, pp. 221–235.
- [19] P. Bhattacharyya, R. Mutharaju, Ontoseer: A tool to ease the ontology development process, in: 8th ACM IKDD CODS and 26th COMAD, 2021, pp. 428–428.
- [20] L. Halilaj, N. Petersen, I. Grangel-González, C. Lange, S. Auer, G. Coskun, S. Lohmann, Vocol: An integrated environment to support version-controlled vocabulary development, in: European Knowledge Acquisition Workshop, Springer, 2016, pp. 303–319.
- [21] S. Lohmann, V. Link, E. Marbach, S. Negru, Webvowl: Web-based visualization of ontologies, in: International Conference on Knowledge Engineering and Knowledge Management, Springer, 2014, pp. 154–158.
- [22] B. Zhou, Y. Svetashova, A. Gusmao, A. Soyulu, G. Cheng, R. Mikut, A. Waaler, E. Kharlamov, Semml: Facilitating development of ml models for condition monitoring with semantics, *Journal of Web Semantics* 71 (2021) 100664.
- [23] A. Alobaid, D. Garijo, M. Poveda-Villalón, I. Santana-Perez, O. Corcho, Ontology, a tool for collaborative development of ontologies., in: ICBO, 2015.
- [24] N. Arndt, P. Naumann, N. Radtke, M. Martin, E. Marx, Decentralized collaborative knowledge management using git, *Journal of Web Semantics* 54 (2019) 29–47.
- [25] M. Poveda-Villalón, A. Gómez-Pérez, M. C. Suárez-Figueroa, Oops!(ontology pitfall scanner!): An on-line tool for ontology evaluation, *International Journal on Semantic Web and Information Systems (IJSWIS)* 10 (2014) 7–34.
- [26] D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann, R. Cornelissen, A. Zaveri, Test-driven evaluation of linked data quality, in: Proceedings of the 23rd international conference on World Wide Web, 2014, pp. 747–758.
- [27] A. Sommer, N. Car, pyshacl, 2021. doi:10.5281/zenodo.5503838.
- [28] D. Garijo, Widoco: a wizard for documenting ontologies, in: International Semantic Web Conference, Springer, 2017, pp. 94–102.
- [29] S. Peroni, D. Shotton, F. Vitali, The live owl documentation environment: a tool for the automatic generation of ontology documentation, in: International Conference on Knowledge Engineering and Knowledge Management, Springer, 2012, pp. 398–412.
- [30] P. D. Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, C. Notredame, Nextflow enables reproducible computational workflows, *Nature Biotechnology* 35 (2017) 316–319. doi:10.1038/nbt.3820.
- [31] S. Chávez-Feria, R. García-Castro, M. Poveda-Villalón, Converting UML-based ontology conceptualizations to OWL with chowlk, in: The Semantic Web: ESWC 2021 Satellite Events, Springer International Publishing, 2021, pp. 44–48. doi:10.1007/97830308041838.
- [32] J. Malone, E. Holloway, T. Adamusiak, M. Kapushesky, J. Zheng, N. Kolesnikov, A. Zhukova, A. Brazma, H. Parkinson, Modeling sample variables with an Experimental Factor Ontology, *Bioinformatics* 26 (2010) 1112–1118. doi:10.1093/bioinformatics/btq099.
- [33] P.-Y. Vandenbussche, G. A. Atemezing, M. Poveda-Villalón, B. Vatan, Linked open vocabularies (lov): a gateway to reusable semantic vocabularies on the web, *Semantic Web* 8 (2017) 437–452.
- [34] W. N. Borst, Construction of engineering ontologies for knowledge sharing and reuse, 1997.

Acknowledgments

This work has been funded by the German Federal Ministry of Education and Research under grant numbers 13XP5119F and 13XP5116B and by the German Federal Ministry for Digital and Transport under grant number 19FS2001A.