FAIR Data Publishing with Apache Maven

Claus Stadler[1,2][0000−0001−9948−6458], Simon Bin[1], and Lorenz Bühmann[1][0000−0002−1023−9993]

[1] Institute for Applied Informatics, Gördelerring 9, Leipzig, Germany, D-04109
[2] Leipzig University, Leipzig, Germany, D-04109
cstadler@informatik.uni-leipzig.de

Abstract. Design and management of a large number of data processing pipelines is a challenging task. Analogous to DevOps, the term DataOps was coined to capture all the practices, processes and technologies related to the management of the life cycle of data artifacts, including the tracking of provenance. The solution space has been constantly increasing with novel approaches and tools becoming available, however with – for instance – more than 100 workflow engines available it is by far no longer feasible to assess them all. Semantic Web technology features many aspects relevant to DataOps, such as interlinkability of resources, DCAT for building decentral data catalogs, PROV-O for provenance descriptions, VoID for describing statistics about the used classes and properties. Yet, there are only few approaches that establish a coherent and holistic connection between these elements. In this work, we perform an in-depth analysis of the Apache Maven build system and its surrounding ecosystem for how they can be leveraged for automated data processing, publishing and RDF metadata generation with provenance tracking. We present three novel Maven plugins for SPARQL and RML execution, the creation of an RDF database file, and uploading artifacts to a CKAN instance. Finally, we present a prototype architecture where a Maven deployment of a geographic RDF dataset results in the automated generation of DCAT, PROV-O and VoID metadata such that datasets can be browsed on a map and filtered e.g. by the used classes and properties. All our resources are freely available as Open Source.

Keywords: FAIR · DataOps · Data Management · Semantic Web · Apache Maven · Reproducible

1 Introduction

The vision of the Semantic Web is to establish a uniform machine readable infrastructure for data. Key technologies include: RDF[3] as a uniform data model to represent information about any domain, RDFS[4] and OWL[5] for schema definition

[3] `https://www.w3.org/TR/rdf11-concepts/`
[4] `https://www.w3.org/TR/rdf11-schema/`
[5] `https://www.w3.org/TR/owl2-overview/`

and reasoning, SHACL[6] and ShEx[7] for validation, and SPARQL[8] for querying. Foundational RDF vocabularies include: VoID[9] for statistical information about dataset content, DCAT[10] for decentral data (and service) catalogs, P-Plan[11] for describing execution plans, PROV-O[12] for provenance.

The FAIR principles [11] provide a conceptual framework for designing data publishing processes in a comprehensible way that makes the involved artifacts findable, accessible, interoperable and reusable. As an example, while mapping non-RDF data with RML[13] (and variants such as YARRML[14]) is common, the tracking of which input CSV artifact was converted by which RML mapping to which output RDF file is not. One possible approach to address this problem is the Common Workflow Language (CWL)[3]. One motivation for the creation of CWL is that by now there are dozens of workflow engines[15] and hundreds of data analysis pipeline systems[16] with hardly any interoperability, so there is a strong need for a unifying language. However, despite the availability of all these solutions for particular problems, approaches for publishing data in an automated way not only according to the FAIR principles but also in reproducible ways are still uncommon.

Apache Maven[17] excels at the following aspects w.r.t. the management of artifacts: (1) findability of artifacts through repository managers, especially Maven Central[18], (2) easy accessibility of artifacts via plain HTTP(S) downloads, (3) interoperability on the repository level with various tools (such as Gradle, Ivy, SBT) and (4) reusability of artifacts via dependency management. Further aspects are stable versioning of build outputs and the high extensibility of builds via plugins. We identify a gap between data catalog systems and repository managers and with this work we propose a prototype architecture to bridge it.

Our contributions are as follows: (1) We perform an in-depth analysis for whether and how the Apache Maven build system can be adopted for several DataOps aspects and assess the technical feasibility. (2) As a resource, we set up a website *Maven4Data*[19] which documents technical details and collects additional examples. (3) As software, we provide one Maven plugin for SPARQL and RML processing, and another for uploading artifacts to the Comprehensive

[6] https://www.w3.org/TR/shacl/
[7] http://shex.io/shex-semantics/
[8] https://www.w3.org/TR/sparql11-query/
[9] https://www.w3.org/TR/void/
[10] https://www.w3.org/TR/vocab-dcat-3/
[11] http://purl.org/net/p-plan
[12] https://www.w3.org/TR/prov-o/
[13] https://rml.io/specs/rml/
[14] https://rml.io/yarrrml/spec/
[15] https://github.com/meirwah/awesome-workflow-engines/blob/master/README.md
[16] https://s.apache.org/existing-workflow-systems
[17] https://maven.apache.org/
[18] https://central.sonatype.com
[19] https://scaseco.github.io/maven4data

Knowledge Archive Network (CKAN). (4) We provide *mvn-rdf-sync*[20] as an additional software prototype. It is a Docker Compose setup for reacting to changes in a Maven repository in order to auto-generate DCAT metadata as well as publishing that DCAT in a triple store. (5) A basic benchmark framework for assessing the performance impact of mvn-rdf-sync on a repository system is provided.

The remainder of this work is structured as follows. In Section 2 we present related work for processing, packaging and FAIR publishing of data artifacts based on Semantic Web technology. An introduction to the Maven build system is given in Section 3. Applying the Maven system to RDF data management is elaborated in Section 4. Synchronization of DCAT metadata from a Maven repository with a triple store is presented in Section 5. A brief performance evaluation for triggering metadata generation upon data deployment is presented in Section 6. Finally, we conclude this work in Section 7.

2 Related Work

OntoMaven [8] provides a set of Maven plugins for working specifically with ontologies (rather than RDF in general). Plugin goals exist to perform inferencing, testing, and the generation of HTML documentation with graph visualizations (somewhat similar to Javadoc). The focus of OntoMaven is to assist with agile ontology development process models such as RapidOWL[1] and the Corporate Ontology Lifecycle Methodology (COLM)[7].

DBpedia Databus is a data asset release management platform inspired by paradigms and techniques successfully applied in software release management[4]. The Databus project explicitly borrows concepts from Apache Maven and also features a set of maven plugins. However, the project features its own platform with its own APIs. The crucial difference between the Databus project and this work is, that here we investigate how data processing processes can be performed natively with Apache Maven and deployment idiomatically with the (WebDav) API support by conventional artifact repository management systems. Plugins such as those of the Databus can be used in addition.

Common Workflow Language (CWL) is an attempt to establish a common language for workflows. The reference implementation *cwltool* also aims to leverage Semantic Web technology as a means for data generation adhering to the FAIR principles. Whereas versioning of artifacts is an intrinsic concept in Maven, the architecture around CWL relies on external services such as WorkflowHub[21] [5]. Due to the relevance of CWL, we assembled a feature comparison with Maven in Table 1 in order to provide more insights into their similarities and differences.

[20] `https://github.com/Scaseco/mvn-rdf-sync`

[21] `https://workflowhub.eu/`

Research Object Crates (RO-Crate)[9][22] is a specification for annotating the content of a directory (or archive) with semantic descriptions. It seems feasible that such descriptions could be generated as part of a Maven build.

In [6], a system for data version control on the RDF graph level is discussed. This is an implementation that works only on the triple level of RDF data and has no regard to sources or transformations.

Data Version Control [23] is a system/suggestion to use a Git-like approach and decentralised repositories to control different versions of personal datasets. Using their registry requires a commercial subscription.

The Semantic Web components of our suggested approach are implemented using the Apache Jena[24] Semantic Web framework.

3 Introduction to Apache Maven

Although Maven[25] is a build tool mainly used for Java development, its underlying concepts are more general in nature.

Maven is based on the concept of a project object model (POM). The POM is the fundamental configuration file that contains information about the project, tasks, and dependencies. Maven uses this file to do everything that is needed to build a project. Applied to data management, this POM can be used to describe how to download, convert, and process data as well as the software and data dependencies required for a specific dataset. The POM also contains several metadata fields that can be used to further link the project to source code repositories, applicable license information, author information and so on, which can be easily mapped to RDF. Maven has built-in support for properties, together with substitution of placeholders in strings (aka interpolation) and files (aka resource filtering). This mechanism can be exploited for generating RDF with additional metadata. Furthermore, Maven is capable of publishing the processed and generated data as artifacts to repository systems, and to download existing artifacts from there. Maven build workflows are based on invoking Maven plugins. Maven plugins are written in the Java programming language and can be deployed as artifacts to such repositories. For a Java software project, Maven would typically call the `maven-compiler-plugin` to process the source code.

Maven introduces the notion of life cycles for carrying out tasks on a software project in a certain order. A life cycle has a name and defines a sequence of phases. The phases `generate-resources` and `process-resources` are specifically dedicated to generating and packaging resources – such as data. The POM follows a single inheritance model such that common configuration options can be placed into a “parent” POM. A typical use case for a parent POM is to configure

[22] `https://www.researchobject.org/`
[23] `https://dvc.org`
[24] `https://jena.apache.org`
[25] `https://maven.apache.org/`

Feature	CWL	Maven
Versioning build outputs	CWL itself does not manage versioning of build outputs.	Intrinsic support, using its coordinate system (groupId, artifactId, version).
Versioning workflow plans	No intrinsic versioning system. Relies on external version control systems like Git.	Versioning is intrinsic to the POM file, allowing versioning of the project definition itself alongside the codebase.
Dependency management	CWL specifies dependencies in terms of software containers and scripts required to run a workflow but relies on external tools for managing these dependencies.	Declaration and automatic downloading (and caching) of dependencies from central and custom repositories.
Deployment of build outputs	No instrinic support. Requires scripting or external tools.	Built-in support for deploying artifacts to repositories through its deployment lifecycle phases and plugins.
Containerization support (Docker)	Natively support for defining steps to be executed in Docker containers, making it straightforward to ensure environment consistency across executions.	Support via plugins but not as seamlessly integrated as CWL's native support.
Signing build output	CWL does not have built-in support for signing outputs. Any signing would need to be handled by external tools or steps defined within the workflow.	Maven supports signing artifacts via plugins (e.g., GPG Plugin) as part of its build process, ensuring the integrity and origin of build outputs.
Deployment to alternative repositories	Deployment to alternative repositories would require integration with other tools.	Highly extensible, with a wide range of plugins available for deploying to various types of repositories.
Execution model	DAG-based	Linear (Life-cycle based). Phases are executed in order - parallelism possible within phases.
Execution monitoring	In addition to logging, third-party tools or workflow engines that support CWL can provide insights into each step's execution status and resource usage. Live-graph visualization possible such as using Apache Airflow.	Due to the linear processing nature typically only console logging and reporting.

Table 1. Comparison between CWL and Maven

the repositories to use for dependency resolution and artifact deployment as well as which versions of certain plugins to use. Traditionally, POM files are written in XML and tool-chains typically operate on the XML model. However, nowadays is also possible to write POMs in other popular languages (such as YAML) using the `polyglot-translate-plugin`[26].

Each Maven project is described by a *groupId*, an *artifactId* and a *version* (GAV). The groupId is typically a reverse domain name such as `org.myorganization.mydepartment.myproject` and can thus be used to link an artifact to an organization in a Web-compatible way. By using domains that are under ones own control, unique global identifiers that are nevertheless human-readable are ensured. Artifact naming can also be leveraged for access control: For example, when publishing artifacts to Maven Central, permission is only granted to upload artifacts whose groupIds start with a specific prefix such as `org.myorganization`. Maven repositories can be configured to enforce that each version is only published exactly once, thereby ensuring that the exact state of a published dataset is not changed.

To each project GAV, any number of files can be attached. Files are differentiated by the type, such as jar, zip, ttl or nt.bz2, and a classifier, which can be any custom value. For example, in this work we use the classifier *dcat* to tag the RDF metadata datasets to load into the triple store. One way to see a GAV is as a URN for a folder (or archive) which contains files with different names and types. Publishing a Maven project usually also attaches the POM file itself. POMs can be designed in a self-contained way such that the creation of artifacts becomes reproducible from the POM itself.

As for repository systems, Maven by default resolves dependencies against the central repository. The plugins we developed as part of this work are published there and can thus be readily reused in Maven builds. In order to not misuse the central repository as a data dump, we set up our own organization-wide repository system instance.

4 Adapting Maven for Data Generation and Deployment

In this section, we present a concrete selection of relevant tasks related to dataset management, representative of similar ones. We then provide a brief overview of the Maven plugins used to solve them.

Figure 1 shows different setups with POM files that make use of different Maven plugins for the creation of artifacts by means of (a) downloading from the Climatetrace[27] website, (b) dockerized execution of a Python script that creates RDF based on data from the GDACS[28] and ReliefWeb[29] Web services, and (c) RML mapping execution based on data and mapping files present in the artifact repository. Each POM can be executed with `mvn deploy` which produces the

[26] `https://github.com/takari/polyglot-maven`
[27] `https://climatetrace.org/data`
[28] `https://www.gdacs.org/`
[29] `https://reliefweb.int/`

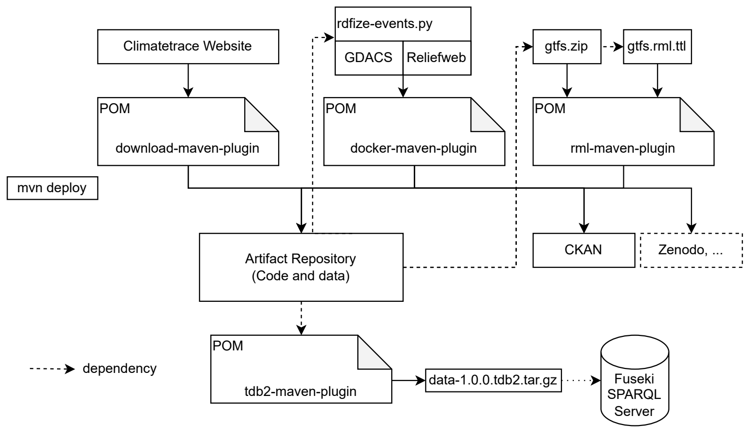


Fig. 1. Architecture with different POM files for unified data processing.

artifacts and uploads them both to a conventional Maven repository as well as to a CKAN instance. Additional plugins could be created for uploading artifacts also to e.g. Zenodo[30] or Huggingface[31]. Furthermore, the `tdb2-maven-plugin`[32] is capable of creating a TDB2 database file that can be served using a Fuseki SPARQL server.

Marking Files for Inclusion in a Project Distribution

Files need to be *attached* to a Maven project in order for them to be part of the file set that gets installed locally or deployed remotely. Many plugins that generate files, such as the Javadoc one, directly support controlling whether to attach the generated files. The `build-helper-maven-plugin` is the conventional way for attaching arbitrary files to a maven project using the `attach-artifact` goal. The GAV is predetermined by the POM, so attached files can only differ in their classifier and type values.

Docker-based Data Generation

The `docker-maven-plugin`[33] enables the use of Docker containers from within Maven builds. It supports copying files in and out of containers and requires a running Docker daemon. For a complete example that runs a Python script to fetch information about disaster events and output them as RDF we refer to our supplemental web page[34].

[30] `https://zenodo.org/`
[31] `https://huggingface.co/`
[32] `https://github.com/Scaseco/tdb2-maven-plugin`
[33] `https://dmp.fabric8.io/`
[34] `https://scaseco.github.io/maven4data/how-tos/build-anything-with-docker.html`

RML Conversion

In [2] we presented a system based on our RML Toolkit [10] (rmltk) that rewrites RML mappings to a sequence of extended SPARQL queries. Likewise, we wrapped this system as a Maven plugin. To use the system, two dependencies have been specified in the POM: one to an artifact containing the CSV source data, and one containing the RML mapping rules. Then, the `rml-maven-plugin` is specified in the plugin section, together with an rmltk-specific configuration (also detailed inside the POM). When Maven is called with process-resources, the dependencies and the plugins will be downloaded (unless they are already cached) and the mapping will execute. A complete example is available on GitHub[35].

RDF generation with SPARQL Queries

We also created the `sparql-maven-plugin` to enable running SPARQL statements against an embedded triple store. This enables loading of datasets as well as producing RDF with SPARQL CONSTRUCT queries as part of the build process. The configuration is similar to that of the `rml-maven-plugin` except that SPARQL queries can be provided as string and files. This plugin can be used to produce VoID, DCAT and PROV metadata. A crucial aspect is that Maven coordinate URNs of the POM can be used as the dataset identifier.

Deploying to CKAN

CKAN[36] is a popular open-source open data portal software for the storage and distribution of data. It has an API for uploading data and metadata. We created the `ckan-maven-plugin` which enables uploading files to CKAN. Several fields for the POM are directly mapped to fields in CKAN, such as: name, description and license. The plugin can be used in addition to any other plugins that carry out deployments. Typically, all deployments are executed in the `deploy` phase. In general, a Maven build can be designed such that properties and profiles are provided to control which (sub)sets of plugins to execute. These mechanisms can be leveraged to carry out only specific deployments. The plugin supports reading (possibly encrypted) CKAN API keys from Maven's `settings.xml` file, which is Maven's default place for user-level passwords. Note, that whether and how conventional dependencies can be (reasonably) resolved against a CKAN instance is future work. More details about the configuration of the CKAN plugin is available on GitHub[37].

Loading a Triple Store from Dependencies

We have created the `tdb2-maven-plugin`, that uses Apache Jena to convert RDF data into a TDB2 database suitable for use with Apache Jena. To use it,

[35] `https://github.com/Scaseco/resource-gtfs-bench-rml/`
[36] `https://ckan.org/`
[37] `https://github.com/Scaseco/ckan-maven-plugin`

the RDF datasets are specified as dependencies in the Maven POM, and the tdb2-maven-plugin is referenced in the build plugins section. A limitation is, that Maven does not natively support the automatic build of missing artifacts based on a reference to a POM that could build those. It may be possible to create further plugins that carry out such tasks by analyzing the dependency tree but but this is open to future work.

5 Synchronization of Metadata

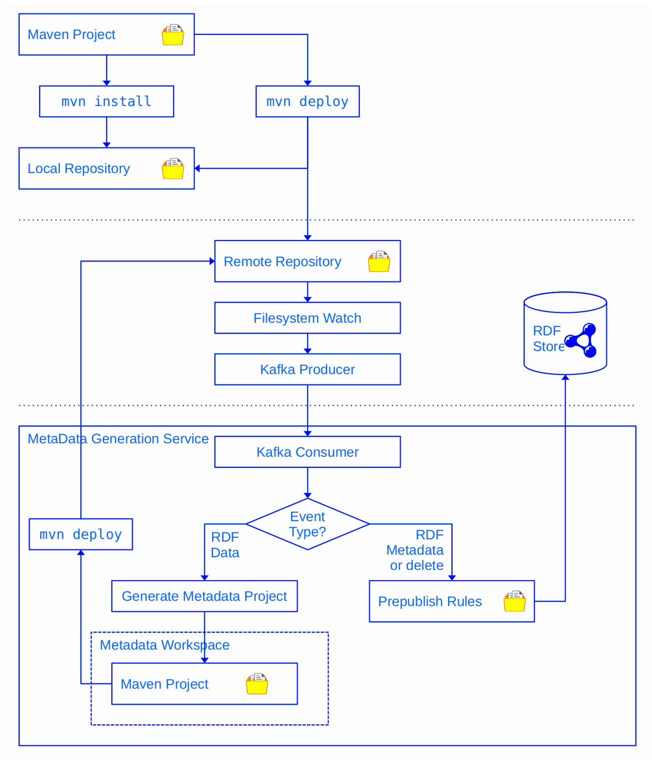


Fig. 2. Architecture of our Maven-RDF-Sync approach.

In the previous sections, we looked at how to build a Maven project that can deploy data. In this section, we present *mvn-rdf-sync*, an approach that after deployment of RDF data automatically generates and deploys metadata. The process is depicted in Figure 2. The steps are:

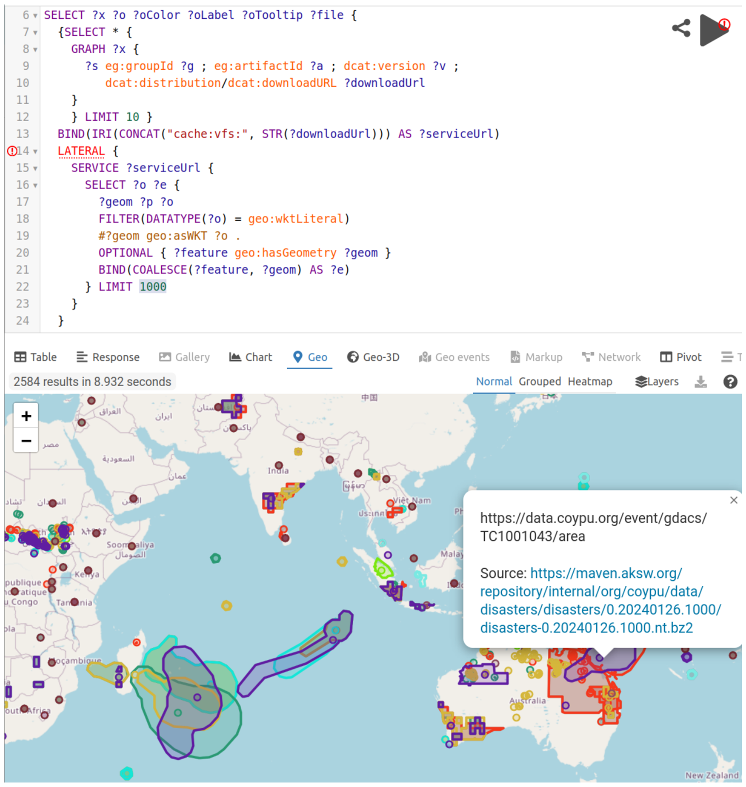
1. A trigger on a Maven repository's file system detects changes and publishes appropriate events.

2. An event consumer examines the event type. If it is an RDF data artifact, then one or more Maven projects for metadata generation are created. This step essentially creates an instance from a Maven template project[38], using parameters from the changed artifact's POM. By convention, we use the classifier `dcat` to indicate artifacts with DCAT metadata. In principle, a future version of the setup could support attached RO-Crate files.
3. The metadata project is then deployed as usual using `mvn deploy`.
4. The change to the file system is detected and another event is sent.
5. This time the event consumer sees the RDF file with the `dcat` classifier and can choose the flow for publishing metadata. In order to prevent arbitrary metadata to end up being published, metadata artifacts can be filtered by trusted groupIds.
6. A sequence of pre-publish rules, essentially SPARQL update queries, is used to transform the raw metadata into the final one. These rules are used to convert Maven URNs to resolvable download URLs and to run queries that for fixing any issues with previously published metadata.
7. Finally, the data ends up in the triple store for access. The metadata is loaded into a graph that matches the metadata artifact.
8. The data is now (publicly) accessible via SPARQL. Tools, such as Yasgui[39], can be used to query and visualize the content.

A screenshot of our online demo[40] is shown in Figure 3. It shows how the DCAT metadata in the triple store synchronized with the Maven repository can be queried with SPARQL and plotted on a map with Yasgui. As the mvn-rdf-sync process also generates VoID metadata, it is also possible to e.g. filter datasets by the used classes and properties. The datasets are identified by `urn:mvn:` URNs, such that a transfer of the repository to a different URL does not invalidate any dataset identifiers.

It is a design decision whether to use a single Maven template project to capture multiple metadata aspects, such as VoID, DCAT and PROV-O, or whether to maintain individual Maven templates for each of them. For the prototype we went with the former approach, but as the system grows it is likely that we will switch to the latter due to better modularity. The metadata project includes generation of the provenance triples that state that the original artifact was generated by an activity whose plan is the URN of the original artifact's POM. If that POM was designed to be self-contained then downloading the POM and running `mvn package` will re-run the build. If the POM was designed for reproducible builds then the output artifacts will match the deployed ones. For non-reproducible builds, such as those that consume live data from APIs, one should typically adjust the groupId and version appropriately before creating custom builds.

[38] `https://github.com/Scaseco/mvn-rdf-sync/blob/main/v3/dcat-generator/pom.xml`
[39] `https://yasgui.triply.cc/`
[40] `https://scaseco.github.io/maven4data/online-demo.html`



```
SELECT ?x ?o ?oColor ?oLabel ?oTooltip ?file {
  {SELECT * {
    GRAPH ?x {
      ?s eg:groupId ?g ; eg:artifactId ?a ; dcat:version ?v ;
         dcat:distribution/dcat:downloadURL ?downloadUrl
    }
    } LIMIT 10 }
  BIND(IRI(CONCAT("cache:vfs:", STR(?downloadUrl))) AS ?serviceUrl)
  LATERAL {
    SERVICE ?serviceUrl {
      SELECT ?o ?e {
        ?geom ?p ?o
        FILTER(DATATYPE(?o) = geo:wktLiteral)
        #?geom geo:asWKT ?o .
        OPTIONAL { ?feature geo:hasGeometry ?geom }
        BIND(COALESCE(?feature, ?geom) AS ?e)
      } LIMIT 1000
    }
  }
```

2584 results in 8.932 seconds

https://data.coypu.org/event/gdacs/TC1001043/area

Source: https://maven.aksw.org/repository/internal/org/coypu/data/disasters/disasters/0.20240126.1000/disasters-0.20240126.1000.nt.bz2

Fig. 3. Visualization of data catalog content on a map via SPARQL.

6 Resources and Evaluation

We have set up the *Maven4Data* website with guides and examples that we have shown in this paper, demonstrating the effectiveness of the approach.

A relevant question is to what extent our mvn-rdf-sync approach degrades the performance of a filesystem-based repository due to overhead of file system watches. For this purpose we devised the "repo-bench"[41] benchmark: We use a python script to generate a Maven project with $n = 500$ sub modules, where each sub module has a corresponding RDF dataset. The triple count for the $i-th$ project is $i \times 1000$ triples, with $1 \leq i \leq n$, amounting to $\frac{n}{2} \times (n+1) \times 1000 =$ 125.250.000 triples in total with a size of 14.2G of data.

We measure the time it takes to install the poms with and without the file system watch active. The experiment was carried out on a Dell XPS 9720 notebook with the following specs: CPU: Intel i9-12900HK CPU, SSD: NVMe PC801 NVMe SK hynix 2TB, RAM: 64GB. We use the following basic watch to check for changes to the file system:

[41] `https://github.com/Scaseco/mvn-rdf-sync/tree/main/benchmark`

```
inotifywait "$HOME/.m2/repository" --recursive --monitor --format '%e %w%f' \
  --event CLOSE_WRITE --event DELETE | xargs -n 1 -P 0 bash -c 'echo  "$@"; sleep 1' _ {}
```

The flag `-P 0` indicates to spawn a fresh process for each line emitted by inotifywait, such that maximum parallelism is utilized. The finding is that regardless whether the following watch is active or not, running `mvn install` takes ≈20 seconds, which indicates that the watch itself does not impact the performance significantly. However, more extensive analysis is needed for how real-world workloads affect overall system performance.

7 Conclusions and Future Work

This work is motivated by the multitude of solutions in the field of data management and the pursuit of a "minimal" one that is open source, can run locally, is extensible, and interoperable with Semantic Web technology. In pursuit of such a solution, we took a deep dive in the Apache Maven ecosystem. We presented a selection of use cases where the life cycle of data artifacts can be mapped to a Maven build process. A set of Maven plugins was devised for streamlining the generation of RDF and for the deployment of artifacts to CKAN instances. We showed that Maven's build specifications can be designed in a way that makes them self-contained w.r.t. versioning, data processing and deployment. The fact that upon deployment the POM is archived as well makes the process self-documenting and can be leveraged for reproducibility. Based on our findings, we devised the *mvn-rdf-sync* system, which listens to changes to Maven repository and triggers RDF metadata generation (via generated Maven projects) when RDF datasets are uploaded. We assembled at the Website *Maven4Data* where we provide additional information and examples. We also compared Maven to CWL and found that these systems are complementary: Maven's scope is more narrow than that of general workflow engines, yet it provides several features that are highly useful for versioning, packaging and deploying artifacts.

Future work is along the following lines: (1) Investigation of creating Research Object Crates from Maven. (2) Analysing the feasibility of Maven plugins that can deploy to further popular public archives. For (2), it needs to be investigated to what extent Maven's coordinate system can be bridged with the artifact naming system provided by those archives. For example, Zenodo provides APIs for custom tags which could be exploited for that purpose.

Acknowledgments

The authors acknowledge the financial support by the German Federal Ministry for Economic Affairs and Climate Action in the project Coypu (project number 01MK21007A) and by the German Federal Ministry for Digital and Transport in the Project Moby Dex (project number 19F2266A).

References

1. Auer, S.: The rapidowl methodology–towards agile knowledge engineering. In: 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'06). pp. 352–357. IEEE (2006)
2. Bin, S., Stadler, C., Bühmann, L.: KGCW2023 challenge report RDFProcessing-Toolkit / sansa. In: 4th International Workshop on Knowledge Graph Construction @ ESWC 2023. No. 3471 in CEUR workshop proceedings, Hersonissos, Greece (2023)
3. Crusoe, M.R., Abeln, S., Iosup, A., Amstutz, P., Chilton, J., Tijanić, N., Ménager, H., Soiland-Reyes, S., Gavrilović, B., Goble, C., et al.: Methods included: standardizing computational reuse and portability with the common workflow language. Communications of the ACM **65**(6), 54–63 (2022)
4. Frey, J., Götz, F., Hofer, M., Hellmann, S.: Managing and compiling data dependencies for semantic applications using databus client. In: Garoufallou, E., Ovalle-Perandones, M.A., Vlachidis, A. (eds.) Metadata and Semantic Research - 15th International Conference, MTSR 2021, Virtual Event, November 29 - December 3, 2021, Revised Selected Papers. Communications in Computer and Information Science, vol. 1537, pp. 114–125. Springer (2021)
5. Goble, C., Soiland-Reyes, S., Bacall, F., Owen, S., Williams, A., Eguinoa, I., Droesbeke, B., Leo, S., Pireddu, L., Rodríguez-Navas, L., et al.: Implementing fair digital objects in the eosc-life workflow collaboratory. Zenodo (2021)
6. Hauptmann, C., Brocco, M., Wörndl, W.: Scalable semantic version control for linked data management. In: LDQ@ ESWC. p. 27 (2015)
7. Luczak-Rösch, M., Heese, R.: Managing ontology lifecycles in corporate settings. In: Networked Knowledge-Networked Media: Integrating Knowledge Management, New Media Technologies and Semantic Systems, pp. 235–248. Springer (2009)
8. Paschke, A., Schäfermeier, R.: Ontomaven-maven-based ontology development and management of distributed ontology repositories. Synergies Between Knowledge Engineering and Software Engineering pp. 251–273 (2018)
9. Soiland-Reyes, S., Sefton, P., Crosas, M., Castro, L.J., Coppens, F., Fernández, J.M., Garijo, D., Grüning, B., La Rosa, M., Leo, S., et al.: Packaging research artefacts with ro-crate. Data Science **5**(2), 97–138 (2022)
10. Stadler, C., Bühmann, L., Meyer, L.P., Martin, M.: Scaling rml and sparql-based knowledge graph construction with apache spark. In: 4th International Workshop on Knowledge Graph Construction @ ESWC 2023. CEUR workshop proceedings, vol. 3471. Hersonissos, Greece (2023)
11. Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.W., da Silva Santos, L.B., Bourne, P.E., et al.: The fair guiding principles for scientific data management and stewardship. Scientific data **3**(1), 1–9 (2016)