Describing configurations of software experiments as Linked Data

Joachim Van Herwegen[1], Ruben Taelman[1], Sarven Capadisli[2], Ruben Verborgh[1]

[1] IDLab, Department of Electronics and Information Systems, Ghent University – imec
[2] Enterprise Information Systems Department, University of Bonn

Abstract. Within computer science engineering, research articles often rely on software experiments in order to evaluate contributions. Reproducing such experiments involves setting up software, benchmarks, and test data. Unfortunately, many articles ambiguously refer to software by name only, leaving out crucial details for reproducibility, such as module and dependency version numbers or the configuration of individual components in different setups. To address this, we created the *Object-Oriented Components* ontology for the semantic description of software components and their configuration. This article discusses the ontology and its application, and demonstrates with a use case how to publish experiments and their software configurations on the Web. In order to enable semantic interlinking between configurations and modules, we published the metadata of all 500,000+ JavaScript libraries on npm as 200,000,000+ RDF triples. Through our work, research articles can refer by URL to fine-grained descriptions of experimental setups. This brings us faster to accurate reproductions of experiments, and facilitates the evaluation of new research contributions with different software configurations. In the future, software could be instantiated automatically based on these descriptions and configurations, reasoning and querying can be applied to software configurations for meta-research purposes.

Canonical HTML view of this article:
https://linkedsoftwaredependencies.org/articles/describing-experiments/

1. Introduction

A large number of computer science articles describe experimental software evaluations, but many of them refer to that software only by name or version number. This information is insufficient for readers to understand which *exact* version of the software, which versions of its *dependencies*, and which detailed *configuration* of the software's components has obtained the reported results. Therefore, potential users cannot always mirror the correct software installation that will behave according to the article's conclusions. Other researchers might fail to reproduce the results because of differences in these aspects.

As Claerbout's Principle [1] explains, "an article about computational science in a scientific publication is not the scholarship itself, it is merely *advertising* of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures." This stresses the importance of reproducibility, and essentially mandates a detailed description of the executed experiment, all of the involved artefacts and actors, and the processing of the retrieved data.

Using Linked Data [2] to publish experiment descriptions provides two immediate benefits: the experimental setup and parts thereof can be *identified by IRIs*, and their details can be retrieved by *dereferencing those IRIs*. Therefore, if research articles complement their textual explanation of an experiment with the IRI of the full setup, reproducibility is strongly facilitated. Moreover, the IRIs of the entire experiment or its parts can be reused in other articles or experiments to unambiguously refer to the same conditions.

In this article, we focus on the description of *software configurations* and *software modules*, such that an evaluated software setup can be referred to unambiguously by an IRI.

This article is structured as follows. In Section 2, we discuss related work. Section 3 introduces the semantic description of software modules and discusses a semantic description of software components and configurations. Section 4, describes a use case where we apply software descriptions to an experimental evaluation. Finally, we discuss our conclusions and future work in Section 5.

2. Related Work

In this section, we discuss related work on the reproducibility of scientific experiments in scholarly articles and ontologies for describing these experiments.

2.1. Reproducibility of software experiments

In order to better keep track of experiments and minimize information loss at CERN, Information Management: A Proposal [3], recommends a system (WWW) to address questions like "Where is this module used? Who wrote this code? Which systems depend on this device?". We contend that the vision to link information systems in the domain of scientific experiments and scholarly articles is not fully realized on the Web. Identifiable parts of experiments, workflows, as well as the articles which refer to them, still predominantly require human intervention and interpretation, thereby leaving deterministic reproducibility an open problem on the Web. Our work focuses on improving the state of "black box" science, in particular to experiments using software written for the Web's programming language JavaScript.

In four-level provenance [4], the authors show that infrastructural, environmental, workflow and data provenance, are needed to achieve reproducibility of scientific workflows. The information captured at different levels of quality enables different levels of reproducibility or repeatability. While our work is conceptually grounded on the same levels, we describe our concrete work on globally identifiable and semantic descriptions of software modules and configurations.

With the goal of improving the way dataset-based software evaluations are performed in the Semantic Web, LOD Lab [5] was introduced. It offers a service to simplify software evaluation against a large amount of Linked Datasets. This was done because (Semantic Web) experiments are typically done using only a few datasets, since handling them requires significant manual labor. This service not only makes it easier for researchers to *develop* experiments, it also makes it easier for others to *reproduce* these experiments, because the manual phase of dataset setup is simplified or even removed. While reusability of datasets is one aspect of experiment reproducibility, our work focuses on reusability of software within experiments, and replication of the environment.

2.2. Ontologies and vocabularies for describing experiments

The PROV Ontology [6] is a domain-independent ontology to capture provenance information about entities, activities, and agents involved in producing data. The OPMW-PROV Ontology [7] is an ontology for describing abstract and executable workflows. It extends PROV-O and the P-PLAN Ontology [8] which is designed to represent scientific processes. The RDF Data Cube Vocabulary [9] enables defining and publishing multi-dimensional data structures and observations. DDI-RDF Discovery Vocabulary [10] is a vocabulary for publishing metadata about research and survey data.

Workflow-Centric Research Objects [11] realizes a suite of ontologies with the *Wf4Ever Research Object Model* based on empirical analysis of workflow decay and repair in order to improve scientific workflow preservation requirements. It has the means to aggregate or bundle resources like workflows, provenance of executions, publications and datasets. Ontologies for Describing the Context of Scientific Experiment Processes [12] compliments the Research Objects model with the *TIMBUS Context Model* by process preservation.

LODFlow [13] proposes the *Linked Data Workflow Project Ontology* to describe and plan workflows, tool configurations, and reporting. Tool specifications and their configurations in LODFlow workflows are described declaratively by a human user without a prescribed schema. Such descriptions are however interpretive in that any given tool is subject to having multiple descriptions by different users. In contrast to the human-driven descriptions, our work both enables and accelerates the generation of machine-driven Linked Data descriptions of software modules, their components, as well as their configurations to be uniformly created. Consequently, this makes it possible to accurately describe software experiments that can be reused and compared with unambiguously.

3. Ontology

To fully describe experiments in RDF, we need a way to link to the specific software components that were used when running the experiment. As a use case, we considered the largest ecosystem of the popular JavaScript language: the registry of the *Node Package Manager (npm)*. We converted the entire npm registry, consisting of over 500,000 JavaScript packages, to RDF. After conversion, we published the resulting 200,000,000+ triples through multiple interfaces. We opted for the npm registry due to it being one of the largest package repositories available and JavaScript's close ties to Web technology.

We also created an ontology to describe how software components can be configured. That way, we can not only describe all the packages used by the software, but also how the software itself is configured.

3.1. Software modules

There are several levels of granularity on which software can be described, going from a high-level package overview to a low-level description of the actual code. In descriptions, we can use several of these layers, depending on the context and the requirements. Drilling down from the top to the bottom, we have the following layers:

- a **bundle** is a container with metadata about the software and its functionality across different points in time. An example is the *N3.js* library *(https://linkedsoftwaredependencies.org/bundles/npm/n3)*.
- a **module** or *version* is a concrete software package and an implementation of a bundle. *N3.js 0.10.0* is a module.
- a **component** is a specific part of a module that can be called in a certain way with a certain set of parameters. The *N3.js 0.10.0 Parser* is a component.

Bundles and modules are described in the npm dataset. For describing components, we will use our newly created ontology.

3.2. Node Package Manager (npm)

All npm data is stored in a CouchDB instance with one entry per bundle. This corresponds to the metadata, manually added by the package developer in a `package.json` file, with additional metadata automatically added by the npm publishing process. To uniquely identify and interlink software components, we developed a server *(https://github.com/LinkedSoftwareDependencies/npm-extraction-server)* that converts the JSON metadata provided by the npm registry to RDF. An important focus of the conversion process was URI re-use, maximizing the available links between different resources.

3.3. Describing components and their configuration

The *Object-Oriented Components ontology (https://linkedsoftwaredependencies.org/vocabularies/object-oriented)* is an ontology for describing software components and their instantiation in a certain configuration. Within this ontology, we reuse Fowler's definition of a software component [14] as a "glob" of software. The purpose of a component is to encapsulate functionality that can be reused by other components. The instantiation of a component can require certain parameters, similar to how object-oriented programming (OOP) languages allow constructors to have certain arguments. We assume OOP in the broad sense of the word, which only requires *classes*, *objects* and *constructor parameters*. Fig. 1 shows an overview of the ontology.

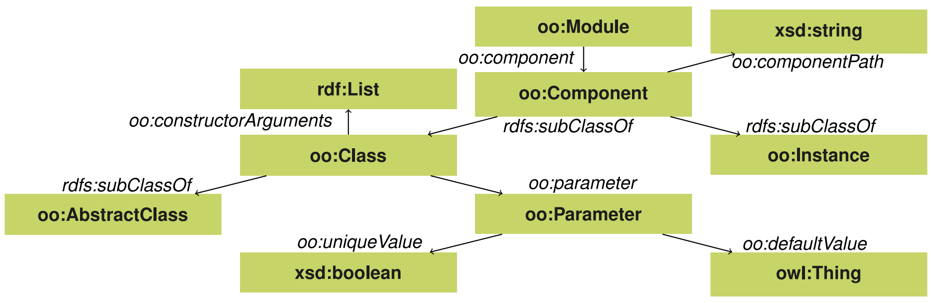

Fig. 1: Classes and properties in the *Object-Oriented Components* ontology *(https://linkedsoftwaredependencies.org/vocabularies/object-oriented#)*, with as prefix `oo`.

We define `oo:Component` as a *subclass* of `rdfs:Class`. The parameters to construct a component can therefore be defined as an `rdfs:Property` on a component. This class structure enables convenient semantic descriptions of components instantiations through the regular `rdf:type` predicate. For instance, a software module representing a certain datasource can be described as `ldfs:Datasource:Hdt rdf:type oo:Class.`, and a concrete instance is `:myHdtDatasource rdf:type ldfs:Datasource:Hdt`. To actually link components to their modules there is the `oo:component` predicate, combined with the `oo:Module` class.

Several `oo:Component` subclasses are defined. An `oo:Component` can be an `oo:Class`, which means that it can be instantiated based on parameters. Each component can refer to its path within a module using the `oo:componentPath` predicate, which can for instance be the package name in Java. All instantiations of `oo:Class` instances are an `oo:Instance`. An `oo:Class` can also be an `oo:AbstractClass`, which does not allow directly instantiating this component type. Abstract components can be used to define a set of shared parameters in a common ancestor. Conforming to the RDF semantics, components can have multiple ancestors, and are indicated using the `rdfs:subClassOf` predicate.

The parameters that are used to instantiate an `oo:Class` to an `oo:Instance` are of type `oo:Parameter`. An `oo:Parameter` is a *subclass* of `rdfs:Property`, which simplifies its usage as an RDF property. `oo:defaultValue` allows parameters to have a default value when no other values have been provided. The `oo:uniqueValue` predicate is a flag that can be set to indicate whether the parameter can only have a single value.

4. Use case: describing a Linked Data Fragments experiment

In this section, we provide a semantic description of the experiment performed in a previous research article, as a guiding example on how to create such descriptions for other evaluations. The intention is that future research articles directly describe their experimental setup this way, either through HTML with embedded RDFa or as a reference to an IRI of an external RDF document, using, for example, the ontologies described in Subsection 2.2.

This experiment we will describe originates from an article on query interfaces [15] and involves specific software configurations of a Linked Data Fragments (LDF) client and server. We have semantically described the LDF server module and its 32 components. Instead of the former domain-specific JSON configuration file, the semantic configuration *(https://github.com/LinkedDataFragments/Server.js/blob/feature-lsd/config/config-example.json)* is Linked Data. Furthermore, we provide an automatically generated semantic description of all concrete installed dependency versions for both the LDF client and server. This is necessary because, modules indicate a compatibility range instead of a concrete version.

The LDF experiment can be described using the following workflow (RDFa-annotated):

1. Create 1 virtual machine for the server.
2. Create 1 virtual machine for a cache.
3. Create 60 virtual machines for clients.
4. Copy a generated Berlin SPARQL benchmark \[16\] dataset to the server.
5. Install the server software configuration *(https://linkedsoftwaredependencies.org/raw/ldf-availability-experiment-config.jsonld)*, implementing the TPF specification *(https://www.hydra-cg.com/spec/latest/triple-pattern-fragments/)*, with its dependencies on the server.
6. Install the client software configuration *(https://github.com/LinkedDataFragments/Client.js)*, implementing the SPARQL 1.1 protocol, with its dependencies *(https://linkedsoftwaredependencies.org/raw/ldf-availability-experiment-client.ttl)* on each client.
7. Execute four processes of the Berlin SPARQL benchmark \[16\] with the client software for each client machine.
8. Record CPU time, RAM usage of each client, the CPU time and RAM usage of the server, and measure the ingoing and outgoing bandwidth of the cache.
9. Publish the results *(http://data.linkeddatafragments.org/benchmark)* online.

An *executed* workflow corresponding to the *abstract* experiment workflow above generates entities based on each activity as performed by various agents. The resulting observations of the experiment are among other valuable immutable provenance level data, which plays a vital role in verifying and reproducing the steps that led to the outcome. Concretely, the conclusions in the article have the resulting data as provenance, which in turn was generated by applying the steps above.

Crucially, in the description above, we refer to the exact software configurations by their IRI, their specific dependency versions, and the specifications they implement. These serve as further documentation of the provenance. The Object-Oriented

Components ontology captures the low-level wiring between components, enabling researchers to swap individual algorithms or component settings.

For example, based on the above description, the exact same experiment can be performed with different client-side algorithms [17] or different server-side interfaces [18]. A common practice to achieve this currently, as done in the aforementioned works [17][18], is to implement modifications in separate code repository branches. Unfortunately, such branches typically diverge from the main code tree, and hence cannot easily be evaluated afterwards with later versions of other components. By implementing them as independent modules instead, they can be plugged in automatically by minimally altering the declarative experiment description. This simultaneously records their provenance, and ensures their sustainability.

5. Conclusion

The core idea of the scientific process is *standing on the shoulders of giants*. This not only means we should build upon the work of others, but also that we should enable others to build upon our work. Reproducibility is an essential aspect of this process. This concept obviously applies to *Web* research as well—moreover, the Web is an ideal platform to *improve* the scientific process as a whole.

In this article, we introduced an ontology for semantically describing software components and their configuration. Publishing this information alongside experimental results is beneficial for the reproduction of experiments, and completes the provenance of experimental results.

Through this work, vague textual references to software configurations in works of research can be replaced with unambiguous URLs. These URLs point to Linked Data that exactly captures a single software configuration, which in turn also uses URLs to identify existing software modules. We already made such module URLs and corresponding descriptions available for all npm modules, which we interconnected using RDF triples. The same approach can be applied to other software ecosystems, such as for instance the Maven Central repository of the Java language, the RubyGems repository for the Ruby language, the Comprehensive Perl Archive Network (CPAN) repository for the Perl language, or the Python Package Index (PyPI) for Python. The usage of Linked Data to describe the software and experiments has the added advantage of linking different descriptions together, which allows the answering of such questions as “Which experiments made use of this benchmark?”

Semantic description can serve many more roles in addition to identifying and describing software configurations. Since the descriptions are exact and machine-readable, an automated dependency injection [14] framework could automatically *instantiate* the software based on a configuration URI and wire its dependent components together. Swapping components in and out, and trying different configurations, comes down to simply editing the configuration, which can be shared again as Linked Data. This leverages the power of the Web to simplify the reproduction of existing experiments and the creation of new ones.

References

1. Buckheit, J.B., Donoho, D.L.: WaveLab and Reproducible Research. Stanford University, http://www-stat.stanford.edu/~donoho/Reports/1995/wavelab.pdf (1995).
2. Berners-Lee, T.: Linked Data. https://www.w3.org/DesignIssues/LinkedData.html (2009).
3. Berners-Lee, T.: Information Management: A Proposal. https://www.w3.org/History/1989/proposal.html (1989).
4. Banati, A., Kacsuk, P., Kozlovszky, M.: Four level provenance support to achieve portable reproducibility of scientific workflows. In: 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE (2015).
5. Rietveld, L., Beek, W., Schlobach, S.: LOD Lab: Experiments at LOD Scale. In: Proceedings of the 14th International Semantic Web Conference. pp. 339–355. Springer (2015).
6. Lebo, T., Sahoo, S., McGuinness, D.: Prov-O: The PROV Ontology. W3C, https://www.w3.org/TR/prov-o/ (2013).
7. Garijo, D., Gil, Y.: OPMW-PROV. http://www.opmw.org/model/OPMW/ (2014).
8. Garijo, D., Gil, Y.: The P-PLAN Ontology. http://vocab.linkeddata.es/p-plan/ (2014).
9. Lebo, T., Sahoo, S., McGuinness, D.: The RDF Data Cube Vocabulary. W3C, https://www.w3.org/TR/vocab-data-cube/ (2014).
10. Bosch, T., Cyganiak, R., Wackerow, J., Zapilko, B.: DDI-RDF Discovery Vocabulary. http://rdf-vocabulary.ddialliance.org/discovery.html (2015).
11. Belhajjame, K., Zhao, J., Garijo, D., Gamble, M., Hettne, K., Palma, R., Mina, E., Corcho, O., Gómez-Pérez, J.M., Bechhofer, S., Klyne, G., Goble, C.: Using a suite of ontologies for preserving workflow-centric research objects. Web Semantics: Science, Services and Agents on the World Wide Web. 32, 16–42 (2015).
12. Mayer, R., Miksa, T., Rauber, A.: Ontologies for Describing the Context of Scientific Experiment Processes. In: 2014 IEEE 10th International Conference on e-Science. IEEE (2014).
13. Rautenberg, S., Ermilov, I., Marx, E., Auer, S., Ngonga Ngomo, A.-C.: LODFlow: A Workflow Management System for Linked Data Processing. In: Proceedings of the 11th International Conference on Semantic Systems. pp. 137–144. ACM, New York, NY, USA (2015).
14. Fowler, M.: Inversion of Control Containers and the Dependency Injection pattern. https://martinfowler.com/articles/injection.html (2004).
15. Verborgh, R., Hartig, O., De Meester, B., Haesendonck, G., De Vocht, L., Vander Sande, M., Cyganiak, R., Colpaert, P., Mannens, E., Van de Walle, R.: Querying Datasets on the Web with High Availability. In: Proceedings of the 13th International Semantic Web Conference. pp. 180–196. Springer (2014).
16. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. International journal on Semantic Web and information systems. 5, 1–24 (2009).
17. Van Herwegen, J., Verborgh, R., Mannens, E., Van de Walle, R.: Query Execution Optimization for Clients of Triple Pattern Fragments. In: The Semantic Web. Latest Advances and New Domains. pp. 302–318 (2015).
18. Hartig, O., Buil-Aranda, C.: Bindings-Restricted Triple Pattern Fragments. In: Proceedings of the 15th International Conference on Ontologies, DataBases, and Applications of Semantics. pp. 762–779. Springer (2016).