# Wanted: standards for automatic reproducibility of computational experiments

SAMUEL GRAYSON, University of Illinois Urbana-Champaign, USA
JOSHUA TEVES, Sandia National Laboratories, USA
REED MILEWICZ, Sandia National Laboratories, USA
DANIEL S. KATZ, University of Illinois Urbana-Champaign Department of Computer Science, USA
DARKO MARINOV, University of Illinois Urbana-Champaign, USA

## 1 INTRODUCTION

A computational experiment is reproducible if another team using the same experimental infrastructure can make a measurement that concurs with the original. Investing in reproducibility can improve trustworthiness of scientific results, unlock productivity, and enable reusability and community support[4]. In practice, reproducers often need to manually work with the code to see how to build necessary libraries, configure parameters, find data, and invoke the experiment; it is not *automatic*.

In this work, we consider the use of specification languages that would provide machine-readable metadata on how to interact with a piece of software. It is not enough for the language to merely contain a run command in a heap of other commands; e.g., a Makefile which defines a rule for executing the experiment alongside rules for compiling intermediate pieces is not sufficient, because there is no machine-readable way to know which of the Make rules executes the experiment. Being able to automatically identifying the "main" command which executes the experiment, for instance, would be very useful for those seeking to reproduce results from past experiments or reusing experiments to address new use cases. Moreover, from a research perspective, having a standardized way to run many different codes at scale would open new avenues for data mining research on reproducibility (c.f., [1]).

Even with workflows, knowing how to invoke the experiment is still difficult. In practice more than 70% of workflows do not work out-of-the-box ; they might require the user to specify data or configure parameters for their use-case. While that flexibility is desirable, it is also desirable to provide a "default invocation" for testing purposes. Snakemake has a standard[1] for specifying the usage of its workflows, this standard does not have a place to put example data or an example invocation[2].

## 2 TOWARDS A STANDARD FOR AUTOMATIC REPRODUCIBILITY

At present, there are a diverse range of solutions for expressing how a code should be run, including bash scripts, environment management specifications (e.g., Spack, Nix, Python Virtualenv), continuous integration scripts, workflows, and container specifications. In our own research on reproducibility of scientific codes, as we scale up our studies to include many different codes, keeping track of how to execute each one becomes very complicated. Moreover, when a code fails to run or deliver reproducible results, it is difficult to assess whether there is a fault with the code or whether we failed to run the code in the correct way. While we

[1]https://snakemake.github.io/snakemake-workflow-catalog/?rules=true
[2]https://github.com/snakemake-workflows/dna-seq-varlociraptor/pull/204#issuecomment-1432876029

do not expect (or recommend) that the scientific software community converge on a single solution for executing codes, we do see value in having a standard way of documenting how each code ought to be run. Such a standard could be implemented as a linked-data ontology like the semantic web. This would enable us to build on a rich set of ontologies for describing digital and physical resources (RO-crate [7], Dublin Core metadata terms [8], Description of a Project [9], nanopublications [3], Citation Typing Ontology [5], Documnet Componnets Ontology [2]) and leverage the ecosytem of existing tools for ontology management (Protégé editor, SHACL, SHEX).

At a very basic level, one could specify available commands and the purpose that they serve (e.g., run make to compile underlying libraries and run main.py to generate figures). See node `#make` Appendix I, for example.

More than just that, a linked data specification standard would enable researchers to link inputs and outputs of codes directly to claims made in publications. See the node `#links-to-pub` in Appendix I for example. With such a specification, any person (or program) should be able to execute the experiments which generate figures or claims in an accompanying paper. For example, the CiTO vocabulary [5] can be used to how the result of a process is used as evidence in a specific publication. These references could be connected to other references of the same publication on the semantic web.

The description can be even more granular, such as by using the DoCO vocabulary [2] to point to specific elements (e.g., figures) within a document, or using Nanopublication [3] to reference specific scientific claims. See the node `#links-to-fig`, `#defines-nanopub`, and `#links-to-nanopub` in Appendix I for example.

RO-crate [6] has terms for describing dependencies between steps, which can be used to encode dependent steps or specify the computational environment. See nodes `#make-data`, and `#plot-figures` in Appendix I for example. The purpose of encoding dependencies is not to usurp the build-system or workflow engine, which both already handle task dependencies; if the experiment already uses a workflow, then the specification should simply invoke that. The purpose of task dependencies in the specification is for projects which do not use a workflow engine, or a task that installs the desired workflow engine.

Such a specification language could also be used to set bounds on the parameters of the experiment, such as the range of valid values or a list of toggleable parameters. See node `#example-of-parameters` in Appendix I for example. This parameter metadata would enable downstream automated experiments like parameter-space search studies, multi-fidelity uncertainty quantification, and outcome-preserving input minimization.

## 3 GETTING ADOPTION

Such a specification would need *some* human input to create; it is not just documenting algorithmic tasks, but what those steps *do* At the very least, the specification should contain the main command and publication DOI. However, the manual effort to write the specification can be greatly reduced.

Workflow engines could assist in generating this. Workflow engines already know all of the computational steps, inputs, outputs, and parameters. Then it could prompt the user with high-level questions (e.g., "what publication is this a part of"?) and generate the appropriate specification.

If the experiment does not use a workflow engine, but someone who can run the experiment is available, then much of the specification can captured from an interactive shell session. The user would invoke a shell that records every command, its exit status, its read-files, and its write-files (using syscall interposition); The user would run their code as usual, and after finishing, the shell would assemble the necessary computational steps and prompt the user for high-level questions.

As a last case resort, if one finds a publication linking to a certain repository, one can try to guess the main command. This is the current state-of-the-art for large-scale reproduction studies, except a standardized language would allow some large-scale reproduciton studies to inform future large-scale reproduction studies on what they did to execute this repository. Computational scientists at least had an opportunity to influence how their code is run in large-scale reproduction studies that is completely absent in the current state-of-the-art. The state-of-the-art's lack of opportunity for input was a frequent response of scientists to Collberg and Proebsting[3].

Computational scientists could benefit from creating these automated reproducibility specifications because large-scale reproduction studies like Collberg and Proebsting [1], Zhao et al. [10], and others serve as free testing and reproduction of their results.

Ideally the reproduction specification would be placed in the same location as the computational experiment (often a GitHub repository), so that it can be maintained alongside them. In cases where the authors of the GitHub repository are not cooperative, one can instead put reproduction specifications in a repository dedicated to holding reproduction specifications from anyone. Users seeking to reproduce a repository automatically would invoke a tool that looks for an automatic reproducibility specification in the source code repository, checks a list of these reproducibility library, and if none exists there, fall back on heuristic to guess how to reproduce the experiment. If it succeeds, the steps that the tool took could be automatically contributed into a reproducibility library.

Meanwhile, conferences and publishers could promote the use of such standard specifications as part of reproducibility requirements for publishing. To get an artifact evaluation badge, normally computational scientists would have to write a natural language description of what the software environment, what the commands are, how to run them, and where does the data end up; meanwhile, an artifact evaluator has to read, interpret, and execute their description by hand. An execution description could make this nearly automatic; if a execution description exists, the artifact evaluator uses an executor which understands the language and runs all of the commands that reference the manuscript in their `purpose` tag.

## 4 CONCLUSION

In this position paper, we argue that developing common standards for specifying how computational experiments should be run to get reproducible results would benefit the scientific community. It presents a compromise where different teams can implement their codes in whatever way they see fit while enabling others to easily run them. This would lead to greater productivity in the (re)use of scientific experiments, empower developers to build tools that leverage those common specifications, and enable software engineering researchers to study reproducibility at scale.

[3]See "Author Comments" in http://reproducibility.cs.arizona.edu/v2/index.html. The authors of publications whose labels are BarowyCBM12, BarthePB12, HolewinskiRRFPRS12, and others responded to Collberg and Proebsting (paraphrasing), "it would have worked; you just didn't invoke the right commands."

## 5 APPENDIX I: EXAMPLE DOCUMENT

This is not the final proposal for the complete vocabulary; the peer-review process is not well-suited to iterate on technical details. The point of this article is to argue that the community should spend effort developing this vocabulary.

```
<?xml version="1.0" encoding="utf-8"?>
<!--
RDF can be serialized as XML, JSON, or triples; backend RDF parsers don't care.
We chose XML because it might be more familiar to readers.
-->

<!--
The following tag imports several other vocabularies behind a namespace.
E.g., `rdf:type` refers to `type` in the `rdf` namespace, which resolves to:
http://www.w3.org/1999/02/22-rdf-syntax-ns#rdftype
Elements with no namespace are resolved within the default namespace,
which is our proposed execution-description vocabulary, http://example.org/execution-description/1.0.
-->

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:wikibase="http://wikiba.se/ontology#"
         xmlns:cito="http://purl.org/spar/cito"
         xmlns:doco="http://purl.org/spar/doco/2015-07-03"
         xmlns:prov="http://www.w3.org/TR/2013/PR-prov-o-20130312/"
         xmlns:wfdesc="http://purl.org/wf4ever/wfdesc#"
         xml:lang="en"
         >

  <!--
  Here, we list some relevant commands, and how they relate to the artifact.
  -->
  <process rdf:about="#make">
    <!-- The following would get run by the UNIX shell. -->
    <command>make libs</command>
    <!-- Here is a string representing the purpose. -->
    <purpose>compiles libraries</purpose>
  </process>

  <!--
  Here, we make a process that depends on a previous process using wfdesc.
  -->
  <process rdf:about="#make-data">
    <command>python3 make_data.py</command>
    <purpose>makes data</purpose>
  </process>
  <process rdf:about="#plot-figures">
    <command>python3 figures.py</command>
    <purpose>plot figures</purpose>
    <dependsOn rdf:resource="#make-data" />
    <!--
    The # is not a typo; the rdf:about becomes a URL fragment in the current document.
    This means one can access a computational step in another document here,
    like "https://example.com/software-experiment-23#make-data".
    -->
  </process>
  <!-- Users may choose the more complex wfdesc vocabulary if they wish. -->

  <!--
  Links to a publication.
  The publisher may or may not host a linked-data description of the documenta at this URL.
  The purpose of the URL is to unambiguously name the document.
  We need the rdf:Description to reference an external resource.
  -->
  <process rdf:about="links-to-pub">
    <command>make all</command>
    <purpose>
      <rdf:Description>
        <cito:isCitedAsEvidenceBy rdf:resource="https://doi.org/10.1234/123456789" />
      </rdf:Description>
    </purpose>

  <!-- Links to a specific figure within a publication -->
  <process rdf:about="links-to-fig">
    <command>make all</command>
    <purpose>
      <prov:generated>
        <doco:figure>
          <rdf:Description>
            <dc:title>Figure 2b</dc:title>
            <dc:isPartOf rdf:resource="https://doi.org/10.1234/123456789" />
          </rdf:Description>
        </doco:figure>
      </prov:generated>
    </purpose>

  <!--
  Describes an abstract nanopublication claim that this experiment supports.
  This one will say: "this experiment supports the claim that malaria is spread by mosquitoes"
  -->
  <process rdf:about="defines-nanopub">
    <command>make all</command>
    <purpose>
      <cito:supports>
        <!--
        We will use Wikidata here.
        They have catalogued many real-world objects and concepts as linked-data objects.
        -->
        <wikibase:Statement>
          <rdf:Description>
            <!-- Q12156 refers to malaria -->
            <subject rdf:resource="https://www.wikidata.org/entity/Q12156" />
            <!-- P1060 refers to disease transmission process (read: "is transmitted by") -->
            <predicate rdf:resource="http://www.wikidata.org/prop/P1060" />
            <!-- Q15304532 refers to mosquitoes -->
            <object rdf:resource="https://www.wikidata.org/entity/Q15304532" />
          </rdf:Description>
        </wikibase:Statement>
      </cito:supports>
    </purpose>

    <!--
    Alternatively, the nanopublication claim will live somewhere else.
    Linked data lets us seamlessly reference other documents.
    -->
    <purpose rdf:about="links-to-nanopub">
      <rdf:Description>
        <cito:supports rdf:resource="https://example.com/article24#claim31" />
      </rdf:Description>
    </purpose>
  </process>

  <!-- Here, we add parameters to the command -->
  <process rdf:label="example-of-parameters">
    <!-- These might be template filled like so: -->
    <command>./generate ${max_resolution} ${rounds}</command>
    <wfdesc:Parameter rdfs:label="max_resolution" />
  </process>

</rdf:RDF>
```

The above RDF/XML can be validated with Python and rdflib:

```
>>> import rdflib
>>> g = rdflib.Graph().parse("test.xml")
>>> # Now we can iterate over the triples contained in this RDF graph
>>> # Note that "anonymous nodes" will appear as rdflib.term.BNode('...')
>>> list(g)[:5]
[(rdflib.term.BNode('N979c272652c948f48598caa65eaf02da'),
  rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#type'),
  rdflib.term.URIRef('http://www.w3.org/TR/2013/PR-prov-o-20130312/generated')),
 (rdflib.term.URIRef('file:///home/sam/box/execution-description/se4rs/test.xml#plot-figures'),
  rdflib.term.URIRef('file:///.../purpose'),
  rdflib.term.Literal('plot figures', lang='en')),
 (rdflib.term.BNode('N979c272652c948f48598caa65eaf02da'),
  rdflib.term.URIRef('http://purl.org/spar/doco/2015-07-03figure'),
  rdflib.term.BNode('Ned5bd1d9a83b48bfa0798f2f1e296db7')),
 (rdflib.term.BNode('Nc4f1068252194a4d90b91a02f3860cf7'),
  rdflib.term.URIRef('http://wikiba.se/ontology#Statement'),
  rdflib.term.BNode('Nce17a7a5920846788169b713dd655c97')),
 (rdflib.term.BNode('N889f577571ab4c67bc063a0d032eb5cf'),
  rdflib.term.URIRef('file:///.../purpose'),
  rdflib.term.BNode('Nc4f1068252194a4d90b91a02f3860cf7'))]
```

## REFERENCES

[1] Christian Collberg and Todd A. Proebsting. 2016. Repeatability in computer systems research. *Commun. ACM* 59, 3 (Feb. 2016), 62–69. https://doi.org/10.1145/2812803

[2] Alexandru Constantin, Silvio Peroni, Steve Pettifer, David Shotton, and Fabio Vitali. 2016. The Document Components Ontology (DoCO). *Semantic Web* 7, 2 (Jan. 2016), 167–181. https://doi.org/10.3233/SW-150177 Publisher: IOS Press.

[3] Paul Groth, Andrew Gibson, and Jan Velterop. 2010. The anatomy of a nanopublication. *Information Services & Use* 30, 1-2 (Jan. 2010), 51–56. https://doi.org/10.3233/ISU-2010-0613 Publisher: IOS Press.

[4] Peter Ivie and Douglas Thain. 2018. Reproducibility in scientific computing. *ACM Computing Surveys (CSUR)* 51, 3 (2018), 1–36.

[5] David Shotton. 2010. CiTO, the Citation Typing Ontology. *Journal of Biomedical Semantics* 1, 1 (June 2010), S6. https://doi.org/10.1186/2041-1480-1-S1-S6

[6] Stian Soiland-Reyes, Sean Bechhofer, Khalid Belhajjame, Graham Klyne, Daniel Garijo, Oscar Corcho, Esteban García Cuesta, and Raul Palma. 2013. Wf4Ever Research Object Model. (Nov. 2013). https://doi.org/10.5281/ZENODO.12744 Publisher: Zenodo.

[7] Stian Soiland-Reyes, Peter Sefton, Mercè Crosas, Leyla Jael Castro, Frederik Coppens, José M. Fernández, Daniel Garijo, Björn Grüning, Marco La Rosa, Simone Leo, Eoghan Ó Carragáin, Marc Portier, Ana Trisovic, RO-Crate Community, Paul Groth, and Carole Goble. 2022. Packaging research artefacts with RO-Crate. *Data Science* 5, 2 (Jan. 2022), 97–138. https://doi.org/10.3233/DS-210053 Publisher: IOS Press.

[8] Stuart L. Weibel and Traugott Koch. 2000. The Dublin Core Metadata Initiative: Mission, Current Activities, and Future Directions. *D-Lib Magazine* 6, 12 (Dec. 2000). https://doi.org/10.1045/december2000-weibel

[9] Edd Wilder-James. 2017. Description of a Project wiki. https://github.com/ewilderj/doap/wiki

[10] Jun Zhao, Jose-Manuel Gomez-Perez, Khalid Belhajjame, Graham Klyne, Esteban Garcia-cuesta, Aleix Garrido, Kristina Hettne, Marco Roos, David De Roure, and Carole Goble. 2012. Why workflows break — understanding and combating decay in Taverna workflows. In *2012 IEEE 8th International Conference on E-Science (e-Science)*. IEEE, Chicago, IL, 9. https://doi.org/10.1109/eScience.2012.6404482