

Making Canonical Workflow Building Blocks interoperable across workflow languages

[Stian Soiland-Reyes](#), [Genis Bayarri](#), [Pau Andrio](#), [Robin Long](#), [Douglas Lowe](#), [Ania Niewielska](#), [Adam Hospital](#), [Paul Groth](#)

Revised manuscript, submitted 2021-11-25 to Data Intelligence for special issue on Canonical Workflow Frameworks for Research.

This preprint: <https://doi.org/10.5281/zenodo.5727730>

Abstract

We introduce the concept of *Canonical Workflow Building Blocks* (CWBB), a methodology of describing and wrapping computational tools, in order for them to be utilized in a reproducible manner from multiple workflow languages and execution platforms. The concept is implemented and demonstrated with the BioExcel Building Blocks library (BioBB), a collection of tool wrappers in the field of computational biomolecular simulation. Interoperability across different workflow languages is showcased through a protein Molecular Dynamics setup transversal workflow, built using this library and run with 5 different Workflow Manager Systems (WfMS). We argue such practice is a necessary requirement for FAIR Computational Workflows and an element of Canonical Workflow Frameworks for Research (CWFR) in order to improve widespread adoption and reuse of computational methods across workflow language barriers.

1. Introduction

The need for *reproducibility* of research software usage is well established [[Stodden 2016](#), [Leipzig 2021](#), [Katz 2021](#)], and adaptation of *workflow management systems* (WfMS) together with *software packaging and containers* [[Möller 2017](#)] have been proposed as key ingredients for making research software usage FAIR and reproducible [[Cohen-Boulakia 2017](#), [Grüning 2018a](#), [Lamprecht 2020](#)]. Recently it is also argued that computational workflows should also be treated as FAIR Digital Objects [[De Smedt 2020](#)] in their own right, with identifier, metadata and interoperability requirements [[Goble 2020](#)].

[BioExcel](#), a European Centre of Excellence for Computational Biomolecular Research, has a particular focus on the research domains molecular dynamics simulations and bioinformatics with use of *High Performance Computing* (HPC) to approach Exascale performance, while also improving usability. The *BioExcel Building Blocks* (BioBB) [[Andrio 2019](#)] have been created as portable wrappers of open-source computational tools identified as useful for BioExcel workflows, forming several *families* of documented and interoperable operations that can be called from multiple workflow systems. This interoperability is shown with the BioBB demonstrator workflows, along with multiple tutorials and notebooks.

We propose that these building blocks and their families can themselves be considered *composite Digital Objects*: collections of software packages and their source code, guides and tutorials, as well as workflow management system integrations and workflow examples. In addition, the building blocks, as wrappers of upstream open source tools, benefit from and refer to the tools' existing documentation, support forums, academic publications and wider development context.

Given BioBB as a starting point, we define a generalized methodology of *Canonical Workflow Building Blocks* (CWBB), through the definition of a set of requirements and recommendations for how to formalize and develop a family of compatible computational tools as Digital Objects. These building blocks let researchers instantiate a Canonical Workflow in multiple workflow management systems, while also benefiting from the FAIR aspects of the CWBB Digital Objects.



2. Methods

The [BioExcel Building Blocks](#) library [[Andrio 2019](#)], created and implemented within the BioExcel CoE, is a collection of portable wrappers of common biomolecular simulation tools. The BioBB library is designed to i) increase the *interoperability* between the tools wrapped; ii) *ease the implementation* of biomolecular simulation workflows; and iii) increase the *reusability and reproducibility* of the generated workflows. To achieve these main goals, the library was designed following the FAIR principles for research software development best practices [[Lamprecht 2020](#)].



The result is a collection of building block modules, divided in sets of tool wrappers focused on similar functionalities (e.g. Molecular Dynamics, Virtual Screening). Each of the modules is built from a combination of (i) software packaging ([Pip](#), [BioConda](#), BioContainers), (ii) documentation ([ReadTheDocs](#)), (iii) interactive tutorials ([Jupyter Notebooks](#), [myBinder](#)), (iv) registry & findability ([bio.tools](#), [BioSchemas](#), [WorkflowHub](#)), (v) WfMS integration stubs ([CWL](#), [Galaxy](#), [PyCOMPSs](#)), (vi) source Code ([GitHub](#)) and (vii) REST APIs ([OpenAPI](#), [Swagger](#)). Notably all building blocks follow the same pattern of installation, configuration and interaction.

Since the publication of the library, several [new building block modules](#) (Chemistry, Machine Learning, AMBER MD, Virtual Screening, etc.) have been added, and the set of operations for the existing BioBB families have been expanded. While we previously provided curated adapters (Figure 1) for running BioBB in workflow systems using Common Workflow Language (CWL) and PyCOMPSs, along with Galaxy Toolshed bindings, we have now started auto-generating these bindings, along with command line wrappers and REST web service APIs, using annotations within BioBB's Python docstrings as source. These annotations include sufficient information for a WfMS to launch a particular building block: *input* and *output* parameters (including *mandatory/optional* flags), compatible *formats* (including EDAM ontology formats [[Ison 2013](#)]), *example* files (essential for testing purposes), default values and *dependencies*. This ensures human-readable documentation, FAIR metadata and programmatic accessibility can be generated consistently and comparably.

```
1 #!/usr/bin/env cwl-runner
2 cwlVersion: v1.0
3
4 class: CommandLineTool
5
6 label: Wrapper class for the GROMACS editconf module.
7
8 doc: |-
9   The GROMACS solvate module generates a box around the selected structure.
10
11 baseCommand: editconf
12
13 hints:
14   dockerRequirement:
15     dockerPull: https://quay.io/biocontainers/biobb_md:3.6.0--pyhdf78af_0
16
17 inputs:
18   input_gro_path:
19     label: Path to the input GRO file
20     doc: |-
21       Path to the input GRO file
22     type: string
23     File type: input
24     Accepted formats: gro
25     Example file: https://github.com/bioexcel/biobb_md/raw/master/biobb_md/test/data/gromacs/editconf.gro
26     type: File
27     format:
28       - edam:format_2033
29   inputBinding:
30     position: 1
31     prefix: --input_gro_path
```



```
1 # Python
2 import os
3 import sys
4 import traceback
5 # Pycomps
6 from pycomps.api.task import task
7 from pycomps.api.parameter import FILE_IN, FILE_OUT
8 # Adapters commons pycomps
9 from biobb_adapters.pycomps.biobb_commons import task_config
10 # wrapped biobb
11 from biobb_md.gromacs.editconf import Editconf # Importing class instead of module to avoid name collision
12
13 task_time_out = int(os.environ.get("TASK_TIME_OUT", 0))
14
15
16 @task(input_gro_path=FILE_IN, output_gro_path=FILE_OUT,
17       on_failure="IGNORE", time_out=task_time_out)
18 def _editconf(input_gro_path, output_gro_path, properties, **kwargs):
19
20     task_config.put(os.environ)
21
22     try:
23         editconf(input_gro_path=input_gro_path, output_gro_path=output_gro_path, properties=properties, **kwargs).launch()
24     except Exception as e:
25         traceback.print_exc()
26         raise e
27     finally:
28         sys.stdout.flush()
29         sys.stderr.flush()
30
31
32 def editconf(input_gro_path, output_gro_path, properties=None, **kwargs):
33
34     if (output_gro_path is None or os.path.exists(output_gro_path)) and \
35         not print("WARN: Task Editconf already executed.")
36     else:
37         _editconf(input_gro_path, output_gro_path, properties, **kwargs)
```



```
1 <tool id="biobb_md_editconf_ext" name="Editconf" version="3.0.0" >
2   <description> Wrapper class for the GROMACS editconf module.</description>
3   <requirements>
4     <requirement type="package">biobb_md</requirement>
5   </requirements>
6   <command detect_errors="aggressive"><[[CDATA[
7
8     ln -s -f $(input_gro_path) $(input_gro_path).$(input_gro_path.ext);
9
10    #if $config.sele == "option1":
11      ln -s -f $(config.properties) $(config.properties).$(config.properties.ext);
12    #end if
13
14    editconf
15
16    #if $config.sele == "option1":
17      --config $(config.properties).$(config.properties.ext)
18    #else if $config.sele == "option2":
19      --config $(config.jsonstr)
20    #else if $config.sele == "option3":
21      --config "_oc__dq_distance_to_molecule_dq_":$(config.distance_to_molecule),
22              "_dq_box_type_dq_":dq_$(config.box_type)_dq_:"dq_center_molecule_dq_":$(c
23              _dq_box_lib_dq_":dq_$(config.gmx_lib)_dq_:"dq_gmx_path_dq_":dq_$(confi
24    #end if
25
26    #if str(input_gro_path) != "None":
27      --input_gro_path $(input_gro_path).$(input_gro_path.ext)
28    #end if
29
30    --output_gro_path $_new_file_path_/$outname_output_gro_path
```

```
40 /**
41  * This is the model implementation of MD_editconf.
42  * GROMACS editconf. Adding a box to the structure, which is needed to solvate it afterwards.
43  */
44 # Author BioExcel
45 */
46 public class MD_editconfNodeModel extends NodeModel {
47
48   /** Constant for the input index. */
49   public static final int IN_PORT = 0;
50
51   /** String id for the input gro. */
52   public static final String GRO_IN = "input-gro";
53
54   // the logger instance
55   private static final NodeLogger logger = NodeLogger
56     .getLogger(MD_editconfNodeModel.class);
57
58   /** the settings key which is used to retrieve and
59     store the settings (from the dialog or from a settings file)
60     (package visibility to be usable from the dialog). */
61   static final String CFGKEY_BOXTYPE= "BoxType";
62   static final String CFGKEY_DISTANCE= "Distance";
63   static final String CFGKEY_CENTER= "Center";
64
65   /** initial default values. */
66   static final String DEFAULT_BOXTYPE = "cubic";
67   static final Double DEFAULT_DISTANCE = 1.0;
68   static final Boolean DEFAULT_CENTER= true;
69
70   private final SettingsModelString boxtype =
71     new SettingsModelString(MD_editconfNodeModel.CFGKEY_BOXTYPE, MD_editconfNodeModel.DEFAULT_BOXTYPE);
72
73   private final SettingsModelDoubleBounded distance =
74     new SettingsModelDoubleBounded(MD_editconfNodeModel.CFGKEY_DISTANCE, MD_editconfNodeModel.DEFAULT_DISTANCE, 0.1, 20.0);
75
76   private final SettingsModelBoolean center =
77     new SettingsModelBoolean(MD_editconfNodeModel.CFGKEY_CENTER, MD_editconfNodeModel.DEFAULT_CENTER);
78 }
```

Figure 1: Code snippets for the BioBB WfMS bindings: CWL, PyCOMPSs, Galaxy and KNIME

The library is showcased through a collection of [demonstration workflows](#) [Hospital 2020]. Here, each workflow introduces individual building blocks as needed to explain a particular scientific computational method. We primarily expose the workflows as Jupyter Notebooks [Kluyver 2016], which has been highlighted as a valuable tool for reproducible scientific workflows [Beg 2021]. This offers a graphical interactive interface, including documentation (integrated markdown) related to the workflow and the building blocks used, but also to the biomolecular simulation methods used in the pipeline. Moreover, as we have demonstrated with our own [Binder](#) [Jupyter 2018] hosting, these workflows are reproducible across platforms, assisted by BioConda [Grüning 2018b] packaging of the building blocks and their software dependencies.

This assembly of available demonstration workflows have been successfully used in the BioExcel CoE for dissemination with a range of [training events](#) (e.g. BioExcel Summer & Winter School, webinars and virtual training). In training we particularly utilised the Binder infrastructure of the BioExcel Cloud portal [Niewielska 2020] to give users a web-based first experience of the building blocks before they try them in other workflow systems.

We can observe that workflow building blocks such as BioBB are necessarily composed of a comprehensive list of digital objects, encompassing source code, packaging, containerization, documentation, attributions, citations, registry entries, WfMS integrations and REST APIs.

We propose to consider building blocks as *composite digital objects* in their own right: gathering the above software components along with their metadata, identifiers and operations then forms a *Canonical Workflow Building Block (CWBB)*. We suggest this concept as a fundamental element of FAIR Digital Objects for Computational Workflows: researchers use the building blocks computationally as functional operations across WfMSs, while the FAIR aspect of CWBB propagates information and resources that are essential for reproducibility, reuse and understanding by anyone discovering the workflow.

2.1 Interoperability across different workflow languages

The concept of Canonical Workflow Building Blocks is here showcased with the BioBB library, by using a transversal workflow present in many different computational biomolecular projects: a [Molecular Dynamics \(MD\) protein setup](#). This workflow prepares a protein structure to be used as input for an MD simulation, going through a series of steps where the protein is completed (adding hydrogen and missing atoms), optionally introducing a residue mutation, then submerging the protein in a virtual box of water molecules with a particular ionic concentration, and finally energetically equilibrating the system (so that solvent and ions are well accommodated around the protein at the desired temperature).

This simulation process involves a non-negligible number of steps, using a variety of biomolecular tools. The BioBB library was used to assemble this workflow, interconnecting building blocks using Python functions (Jupyter Notebook, Command Line Interface), auto-generated bindings (Galaxy [Afgan 2018], CWL [Crusoe 2021], PyCOMPSs [Tejedor 2017]) or manually generated bindings (KNIME [Fillbrunn 2017]). Corresponding workflows for the different WfMS can be found in [WorkflowHub](#) and graphical extracts can be seen in Figure 2.

This example demonstrates how the same canonical building blocks can be used in different WfMS. Wrappers and tools executed behind the workflows are exactly the same, but the workflows are built using different WfMS, some of them in a graphical way (drag & drop, Galaxy, KNIME), some in a command line way (Jupyter Notebook, PyCOMPSs, CWL); workflows can be focused on short/interactive executions (Jupyter Notebook), or on High Throughput/High Performance Computing (HT-HPC) executions (PyCOMPSs); some of them prepared for a particular WfMS installation (Galaxy), others completely system-agnostic (CWL).

The current number of available WfMS bindings include Jupyter Notebook, PyCOMPSs, CWL, Galaxy and KNIME WfMS, in addition to a [command line](#) mechanism. Thanks to the extensive documentation added in the source code as Python docstrings, new bindings for available WfMS can be generated. We are also experimenting with generating a REST API exposing the building services as Web services. However, it should be noted that such automatic generation of bindings is not always practically feasible. As an example, KNIME nodes require a complete Java skeleton code, as well as a definition of new data types for all inputs/outputs required, which makes their automatic generation a heavy and potentially error-prone task. Bindings for

workflow languages with a *domain-specific language* (DSL) for tool definitions (e.g. Galaxy, CWL) can on the other hand be generated in a more straightforward fashion.

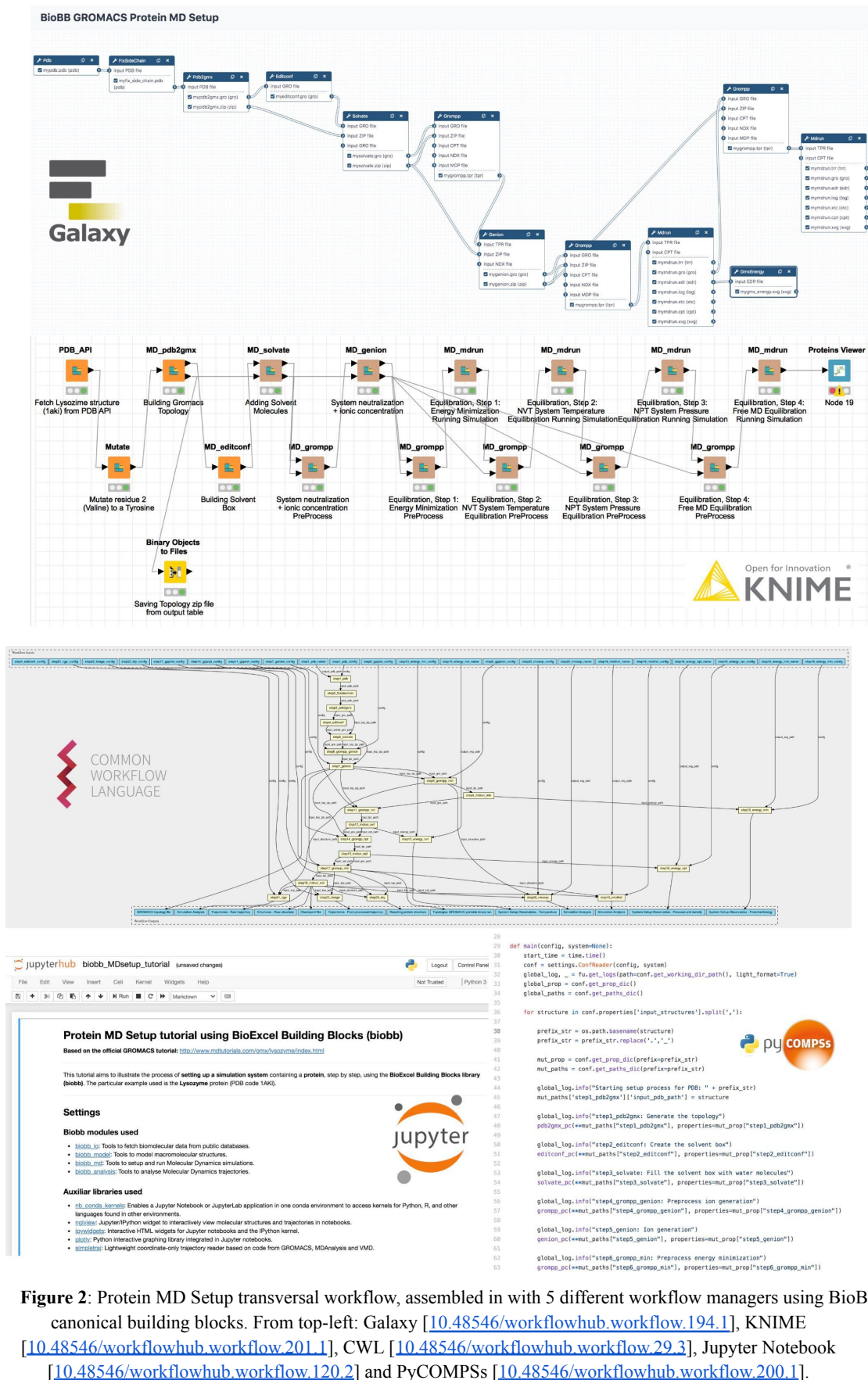


Figure 2: Protein MD Setup transversal workflow, assembled in with 5 different workflow managers using BioBB canonical building blocks. From top-left: Galaxy [10.48546/workflowhub.workflow.194.1], KNIME [10.48546/workflowhub.workflow.201.1], CWL [10.48546/workflowhub.workflow.29.3], Jupyter Notebook [10.48546/workflowhub.workflow.120.2] and PyCOMPS [10.48546/workflowhub.workflow.200.1].

The transversal [protein MD setup workflow](#) was chosen as a real example that is readily understandable by domain experts. More [complex pipelines](#) involving a broader set of wrapped biomolecular tools have been developed using the BioBB library, primarily as Jupyter Notebooks. A selection of these will similarly be assembled for different WfMS using the auto-generated bindings and uploaded to the [WorkflowHub repository](#).

3. Discussion

Early work on libraries of workflows fragments include Web Service-based approaches where tools are wrapped and exposed using common, interoperable data types in [BioMoby](#) [Biomoby 2008] for bioinformatics and similarly [caBIG](#) [Saltz 2006] for cancer genomics. While these efforts were interoperable across WfMSs they required a large up-front investment in agreeing to and adapting native data to common RDF or XML representations.

The notion of *abstract workflows* [Garijo 2011], structural workflow descriptions separated from their concrete execution realizations and augmented with Linked Data annotations, have been emphasized as essential for reuse and consistency across workflow systems. Identifying *common motifs* for workflow operations [Garijo 2014] (e.g. Data preparation, Format transformation, Filter, Combine) are important to simplify and understand otherwise fine-grained workflow provenance traces.

Most other efforts to standardize a set of disparate analytical tools have been done within the scope of a single WfMS, allowing customized user interaction, data visualization, configuration and findability, for instance [Taverna components](#) had prototypical building blocks [Giovanni 2016] which were instantiated at runtime by reference from a registry. [KNIME components and metanodes](#), shared on the [KNIME Hub](#) are frequently designed to be interoperable, but with a perhaps weaker notion of component families. The [Galaxy toolshed](#) [Blankenberg 2014] is likewise populated with different sets of tool wrappers that are largely made to be interoperable within a category.

The *Common Workflow Language (CWL)* [Crusoe 2021] has a strong emphasis on interoperable command line tool descriptions, with [support for containers](#) and Conda packaging, as well as [support for FAIR metadata](#) like contributors, license and EDAM ontology type annotations. With multiple leading workflow engines now supporting CWL, and experimental Galaxy support, this seems perhaps the most promising candidate for both making and describing canonical workflow building blocks, however we've identified a few stumbling blocks.

One obvious challenge is that the implementing WfMS needs to have CWL support, along with support for either containers or Conda packaging to find the described executables. While it is possible to run a CWL tool directly using a `#!/usr/bin/env cwl-runner shebang` on POSIX systems, this still requires pre-installation and possibly configuration of a CWL engine like [cwltool](#) or [Toil](#) [Vivian 2017]. However workflow engines have multiple dependencies and often cannot easily be run from a container themselves¹.

Within the CWL community it was originally envisioned that a wider set of workflow systems would adopt CWL for tool description/execution, with a subset implementing full CWL workflow support. This would allow shared community effort for describing tools, say in the [Common Workflow Library](#), rather than each WfMS needing to duplicate this tool wrapping in separate repositories and languages. However, with the exception of experimental tool support in Galaxy, in practice all CWL implementers have gone for full workflow support.

Another challenge is that making a set of building blocks frequently requires the use of *shims*, for instance file conversion, small search/replace operations or file renames. In a CWL approach these can either be performed with an [Expression](#) using JavaScript snippets which only has limited access to file content, or as an additional workflow step added before or after the main tool step. This combination could then be nested as a subworkflow, similar to KNIME's metanodes, and would also be flexible by allowing different containers or packages for any pre- or post-steps. Such a CWL building block however becomes harder to access from a non-CWL WfMS, because of lack of control over configuration/execution options for the now nested CWL

¹ To execute the wrapped tool, a containerized workflow engine would need *nested containers* which are not generally recommended for security reasons. It is possible to work around this limitation using [Singularity](#) or [Conda](#).

tools. In practice², executing a nested CWL workflow from a native WfMS language would require the engine to implement full CWL Workflow support (or delegate to a CWL engine).

For the main BioBB building blocks we implemented [demonstrator workflows](#) that highlight how the tools should be used in different workflow management systems; each having a primary exemplar using Jupyter Notebook, which can be explored interactively using the [BioExcel myBinder](#). If we consider the abstract demonstrator workflows as *canonical workflows* they are therefore very much active objects, but can also be seen as *workflow templates*, as any real use case will need to specialize the workflow to tweak parameters, data selection etc.

We therefore also provide such workflow templates for multiple WfMS, including CWL, PyCOMPSs and Galaxy. These are fairly disparate workflow languages, yet by the use of the same canonical workflow building blocks (which again invoke the same software binaries), such WfMS-specific workflows effectively are instantiations of the same canonical workflow.

One challenge found is how to publish such canonical workflows in registries like the [WorkflowHub](#). The hub supports the registration of Digital Objects in the form of RO-Crate [[Soiland-Reyes 2021](#)], with the option of abstract CWL for describing the canonical workflow template, along with direct references to the workflow's GitHub repository.

For instance in the RO-Crate for <https://doi.org/10.48546/workflowhub.workflow.200.1>, which can also be [rendered](#) from GitHub, we have an entry for the [main workflow](#) according to the [Workflow RO-Crate profile](#), detailing each canonical workflow building block used (e.g. [biobb-md metadata](#)). Here the FAIR aspect of the building blocks to help software citation is exercised, as the building block wrapper has one set of authors, documentation and license (Apache-2.0), while the wrapped software (e.g. [GROMACS metadata](#)) has different authors, license (GPL-2.1+) and documentation.

However the deposit of such RO-Crates in WorkflowHub results in one registration entry per workflow language, which are not otherwise related and may not even share the same source code repository. Thus we've identified the need for adding an overall *canonical workflow entry*, which can bring in workflow documentation and references shared across WfMS implementations, including a set of links to the more granular canonical workflow building blocks used by the workflow, but also to the individual WfMS implementations as separate digital objects.

A similar question of granularity applies at the workflow tool level [[Möller 2017](#)], particularly for Findability and Accessibility, as we can consider at lowest granularity the *scientific method* in general (e.g. any algorithm for sequence alignment), followed by an *application suite* (bio.tools entry, homepage, documentation), instantiated as a particular *software installation* (Debian package, Docker container) with its dependencies at same level. The installation includes one or more *software executables* (a particular binary, a running service service), providing at the highest detailed granularity level the specific types of *software functionality* (a particular mode of operation, choice of analysis), for instance using certain command line flags.

For canonical workflow building blocks, with a focus on pluggable composability, this is mainly defined at this high granularity level of specific software functionality: explicit operations from an installed tool, which are then combined in a workflow. This is indeed the level WfMS tool definitions are typically done, e.g. a CWL Command Line Tool specifies a particular way to run a particular software binary. However to be an actionable CWBB, the building block needs to additionally convey the lower granularity levels; particularly to support multiple options for interoperable installation and execution, as well as metadata at the most general level, such as documentation and scholarly citations.

While workflow management systems typically only operate at the highest granularity levels for execution details, and are frequently unaware of (or not exposing metadata at) the more general levels, we argue that in order for a Canonical Workflow [[Wittenburg 2021](#)] to follow and support FAIR principles for itself and its data, the workflow management system need to *propagate structured metadata* about the tools used by the workflow. We propose that in order to support the workflow's applicability to multiple WfMS, the tools themselves must also have a consistent packaging and formal description that enables consistent computational invocation.

² It is worth mentioning that it would also be possible to generate WfMS-specific bindings from CWL descriptions (e.g. as demonstrated with [cwl2script](#) for Bash, [exargparse](#) for Galaxy, [cwl2wdl](#) for WDL), although this necessitates constraining the tool and workflow definitions to a limited mappable subset of CWL.

At the most general level, a canonical workflow built using such CWBBs is even conceptually reproducible because the FAIR documentation of the workflow, through its canonical workflow building blocks, identifies how individual tools and software applications are composed, which in worst case can be rebuilt using different installation methods in a different WfMS, or in best case inspected to detect and cross-link the same canonical workflow appearing in different WfMS instantiations. This view of software as composition of other software typically also applies at individual tool level, which themselves depend on programming language runtimes, libraries, services and reference data.

4. Requirements for Canonical Workflow Building Blocks

Building on the experiences with BioBB, we here propose requirements and recommendations for establishing Canonical Workflow Building Blocks (CWBB) as implementations of *canonical steps* introduced for Canonical Workflow Frameworks for Research [[Wittenburg 2021](#)].

The core purpose of a CWBB is to wrap a command line tool or other software that can perform an operation as part of a computational workflow. As such, the general advice for making software workflow-ready applies [[Brack 2021](#)] (e.g. easy to install, documented, parallelizable, reproducible output), however a CWBB is also permitted to make use of additional scripts or *shims* to further adapt a third-party tool for workflow use and for data interoperability across blocks.

The way tools are installed or invoked varies slightly across WfMS and operating systems, therefore a CWBB should provide multiple methods for distributing software; currently containers (Docker, Singularity) and distribution-independent packaging (e.g. Conda, Homebrew) are promising by having reproducible install recipes and a wide range of open source dependencies (e.g. Java, Python). Additionally building blocks should allow overriding execution paths, e.g. for use with HPC module system and hardware-optimized binaries.

The CWBBs should have sufficient annotations to be able to generate bindings for different WfMSs and REST APIs, e.g. parameter names and descriptions, types and default values; enumerators for options, file formats for inputs/outputs.

Building blocks should be grouped into families that are interoperable through common data structures and file formats, as well as having joint naming conventions for configuration options. A CWBB family should be released as a single version following [semantic versioning](#) rules, which should have a corresponding persistent identifier (PID).

Metadata for CWBBs should be captured following FAIR guidelines, and distributed as part of the block family and resolvable from the PID as a FAIR Digital Object. Metadata should include references to the CWBB software distributions (e.g. quay.io container URL) as well as attributions, citations and documentation for the wrapped tool.

Example workflows showing CWBB usage should be included in a WfMS-neutral language such as Jupyter Notebooks, which may have equivalent variants for each workflow binding. These workflows should be registered in a workflow registry like WorkflowHub or Dockstore, and assigned their own PIDs.

5. Conclusions

The proposed concept of Canonical Workflow Building Blocks can bridge the gap between FAIR Computational Workflows, interoperable reproducibility and for building canonical workflow descriptions to be used and described FAIRly across WfMSs.

The realization of CWBBs can be achieved in many ways, not necessarily using the Python programming language together with RO-Crate as explored here. In particular if the envisioned Canonical Workflow Frameworks for Research become established in multiple WfMSs with the use of FAIR Digital Objects, the different implementations will need to agree on object types, software packaging and metadata formats in order to reuse tools and provide interoperable reproducibility for canonical workflows.

Likewise, to build a meaningful collection of building blocks for a given research domain, a directed collaborative effort is needed to consistently wrap tools for a related set of WfMSs, chosen to target particular use cases (a family of canonical workflows).

For individual users, a library of Canonical Workflow Building Blocks simplifies many aspects of building pipelines, beyond the FAIR aspects and data compatibility across blocks. For instance they can benefit from training of a CWBB family using Jupyter Notebooks, and then use this knowledge to utilize the same building blocks in a scalable HPC workflow with a CWL engine like Toil, knowing they will perform consistently thanks to the use of containers.

While we have demonstrated CWBB in the biomedical domain, this approach is generally applicable to a wide range of sciences that execute pipelines of multiple file-based command line tools, however it may be harder to achieve with more algebraic “in memory” types of computational workflows, where steps could be challenging to containerize and distinguish as separate block.

We admit that biomolecular research is quite a homogenous field with respect to computational analyses and now becoming relatively mature in terms of tool composability in workflows, building on the experiences of the “FAIR pioneers” in the field of bioinformatics. Other fields, such as social sciences or ecology, can have a wider variety of methods and computational tools, often with human interactions, and may have to adapt the software to be workflow-ready [Brack 2021] before using them as Canonical Workflow Building Blocks. Domains adapting CWBB approach (or workflow systems in general) should take note of the great benefits of hosting collaborative events where developers meet each other and their potential users, demonstrated in our field with events such WorkflowsRI [Ferreira da Silva 2021] and Biohackathons [Garcia 2020].

The Common Workflow Language shows promise as a general canonical workflow building blocks mechanism: gathering execution details of tools along with their metadata and references, augmented with [abstract workflows](#) to represent canonical workflows. However this would need further work to implement our CWBB recommendations in full. Future work for the Canonical Workflow Building Blocks concept includes formalizing and automating publication practices, to make individual blocks available as FAIR Digital Objects on their own or as part of an aggregate collection like RO-Crate.

Acknowledgements

This work has been done as part of the BioExcel CoE (<https://www.bioexcel.eu/>), a project funded by the European Union contracts [H2020-INFRAEDI-02-2018 823830](#), [H2020-EINFRA-2015-1 675728](#). Additional work is funded through EOSC-Life (<https://www.eosc-life.eu/>) contract [H2020-INFRAEOSC-2018-2 824087](#), and ELIXIR-CONVERGE (<https://elixir-europe.org/>) contract [H2020-INFRADEV-2019-2 871075](#).

Author affiliations

Stian Soiland-Reyes <https://orcid.org/0000-0001-9842-9718>

Department of Computer Science, The University of Manchester, Manchester, UK;
Informatics Institute, University of Amsterdam, NL

Genís Bayarri <https://orcid.org/0000-0003-0513-0288>

Institute for Research in Biomedicine (IRB Barcelona), The Barcelona Institute of Science and Technology (BIST),
Barcelona, Spain

Pau Andrio <https://orcid.org/0000-0003-2116-3880>

Barcelona Supercomputing Center (BSC), Barcelona, Spain

Robin Long <https://orcid.org/0000-0003-2249-645X>

Lancaster University, Lancaster, UK;
Research IT, The University of Manchester, Manchester, UK

Douglas Lowe <https://orcid.org/0000-0002-1248-3594>

Research IT, The University of Manchester, Manchester, UK

Ania Niewielska <https://orcid.org/0000-0003-0989-3389>

European Bioinformatics Institute (EMBL-EBI), Cambridge, UK

Adam Hospital <https://orcid.org/0000-0002-8291-8071>
Institute for Research in Biomedicine (IRB Barcelona), The Barcelona Institute of Science and Technology (BIST),
Barcelona, Spain
Paul Groth <https://orcid.org/0000-0003-0183-6910>
Informatics Institute, University of Amsterdam, NL

References

- [Afgan 2018] Enis Afgan, Dannon Baker, B erence Batut, Marius van den Beek, Dave Bouvier, Martin  ech, John Chilton, Dave Clements, Nate Coraor, Bj orn Gr uning, Aysam Guerler, Jennifer Hillman-Jackson, Vahid Jalili, Helena Rasche, Nicola Soranzo, Jeremy Goecks, James Taylor, Anton Nekrutenko, and Daniel Blankenberg (2018): **The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update**. *Nucleic Acids Research* **46**(W1) pp W537–W544. <https://doi.org/10.1093/nar/gky379>
- [Andrio 2019] Pau Andrio, Adam Hospital, Javier Conejero, Luis Jord a, Marc Del Pino, Laia Codo, Stian Soiland-Reyes, Carole Goble, Daniele Lezzi, Rosa M. Badia, Modesto Orozco, Josep Ll. Gelpi (2019): **BioExcel Building Blocks, a software library for interoperable biomolecular simulation workflows**. *Scientific Data* **6**:169 <https://doi.org/10.1038/s41597-019-0177-4>
- [Beg 2021] Marijan Beg, Juliette Taka, Thomas Kluyver, Alexander Kononov, Min Ragan-Kelley, Nicolas M. Thiery, Hans Fangohr (2021): **Using Jupyter for Reproducible Scientific Workflows**. *Computing in Science & Engineering* **23**(2), pp 36–46. <https://doi.org/10.1109/mcse.2021.3052101>
- [Biomoby 2008] The BioMoby Consortium (2008): **Interoperability with Moby 1.0—It's better than sharing your toothbrush!** *Briefings in Bioinformatics* **9**(3), pp 220–231. <https://doi.org/10.1093/bib/bbn003>
- [Blankenberg 2014] Daniel Blankenberg, Gregory Von Kuster, Emil Bouvier, Dannon Baker, Enis Afgan, Nicholas Stoler, James Taylor, Anton Nekrutenko, the Galaxy Team (2014): **Dissemination of scientific software with Galaxy ToolShed**. *Genome Biology* **15**:403 <https://doi.org/10.1186/gb4161>
- [Brack 2021] Paul Brack, Peter Crowther, Stian Soiland-Reyes, Stuart Owen, Douglas Lowe, Alan R Williams, Quentin Groom, Mathias Dillen, Frederik Coppens, Bj orn Gr uning, Ignacio Eguinoa, Phil Ewels, Carole Goble (2021): **10 Simple Rules for making a software tool workflow-ready**. (*submitted*) <https://doi.org/10.5281/zenodo.5636487>
- [Crusoe 2021] Michael R. Crusoe, Sanne Abeln, Alexandru Iosup, Peter Amstutz, John Chilton, Nebojsa Tijani , Herv  M nager, Stian Soiland-Reyes, Carole Goble, The CWL Community (2021): **Methods Included: Standardizing Computational Reuse and Portability with the Common Workflow Language**. *Communication of the ACM* (accepted). *arXiv*:2105.07028 <https://doi.org/10.1145/3486897>
- [Cohen-Boulakia 2017] Sarah Cohen-Boulakia, Khalid Belhajjame, Olivier Collin, J r me Chopard, Christine Froidevaux, Alban Gaignard, Konrad Hinszen, Pierre Larmande, Yvan Le Bras, Fr d ric Lemoine, Fabien Mareuil, Herv  M nager, Christophe Pradal, Christophe Blanchet (2017): **Scientific Workflows for Computational Reproducibility in the Life Sciences: Status, Challenges and Opportunities**. *Future Generation Computer Systems* **75**, pp 284–298. <https://doi.org/10.1016/j.future.2017.01.012>
- [De Smedt 2020] Koenraad De Smedt, Dimitris Koureas, Peter Wittenburg (2020): **FAIR Digital Objects for Science: From Data Pieces to Actionable Knowledge Units**. *Publications* **8**(2):21 <https://doi.org/10.3390/publications8020021>
- [Ferreira da Silva 2021] Rafael Ferreira da Silva, Henri Casanova, Kyle Chard, Ilkay Altintas, Rosa M Badia, Bartosz Balis, Tain a Coleman, Frederik Coppens, Frank Di Natale, Bjoern Enders, Thomas Fahringer, Rosa Filgueira, Grigori Fursin, Daniel Garijo, Carole Goble, Dorran Howell, Shantenu Jha, Daniel S. Katz, Daniel Laney, Ulf Leser, Maciej Malawski, Kshitij Mehta, Lo c Pottier, Jonathan Ozik, J. Luc Peterson, Lavanya Ramakrishnan, Stian Soiland-Reyes, Douglas Thain, Matthew Wolf (2021): **A Community Roadmap for Scientific Workflows Research and Development**. [arXiv:2110.02168](https://arxiv.org/abs/2110.02168) [cs.DC]
- [Fillbrunn 2017] Alexander Fillbrunn, Christian Dietz, Julianus Pfeuffer, Ren  Rahn, Gregory A. Landrum, Michael R. Berthold (2017): KNIME for reproducible cross-domain analysis of life science data. *Journal of Biotechnology* **261** pp 149–156. <https://doi.org/10.1016/j.jbiotec.2017.07.028>
- [Garijo 2011] Daniel Garijo, Yolanda Gil (2011): **A New Approach for Publishing Workflows**. *Proceedings of the 6th Workshop on Workflows in Support of Large-Scale Science - WORKS '11*. <https://doi.org/10.1145/2110497.2110504>
- [Garijo 2014] Daniel Garijo, Pinar Alper, Khalid Belhajjame, Oscar Corcho, Yolanda Gil, Carole Goble (2014): **Common Motifs in Scientific Workflows: An Empirical Analysis**. *Future Generation Computer Systems* **36** pp 338–51. <https://doi.org/10.1016/j.future.2013.09.018>
- [Garcia 2020] Leyla Garcia, Erick Antezana, Alexander Garcia, Evan Bolton, Rafael Jimenez, Pjotr Prins, Juan M. Banda, Toshiaki Katayama (2020): **Ten simple rules to run a successful BioHackathon**. *PLOS Computational Biology* **16**(5):e1007808. <https://doi.org/10.1371/journal.pcbi.1007808>

- [Giovanni 2016] Renato De Giovanni, Alan R. Williams, Vera Hernández Ernst, Robert Kulawik, Francisco Quevedo Fernandez, Alex R. Hardisty (2016): **ENM Components: a new set of web service-based workflow components for ecological niche modelling**. *Ecography* **39**, pp 376–383. <https://doi.org/10.1111/ecog.01552>
- [Gray 2017] Alasdair Gray, Carole Goble, Rafael Jimenez, Bioschemas Community (2017): **Bioschemas: From Potato Salad to Protein Annotation**. Poster, *International Semantic Web Conference (ISWC)*, Vienna Austria, 2017-10-23. <https://iswc2017.semanticweb.org/paper-579/>
- [Goble 2020] Carole Goble, Sarah Cohen-Boulakia, Stian Soiland-Reyes, Daniel Garijo, Yolanda Gil, Michael R. Crusoe, Kristian Peters, Daniel Schober (2020): **FAIR Computational Workflows**. *Data Intelligence* **2**(1), pp 108–121. https://doi.org/10.1162/dint_a_00033
- [Grüning 2018a] Björn Grüning, John Chilton, Johannes Köster, Ryan Dale, Nicola Soranzo, Marius van den Beek, Jeremy Goecks, Rolf Backofen, Anton Nekrutenko, James Taylor (2018): Practical Computational Reproducibility in the Life Sciences. *Cell Systems* **6**(6) pp 631–635. <https://doi.org/10.1016/j.cels.2018.03.014>
- [Grüning 2018b] Björn Grüning, Ryan Dale, Andreas Sjödin, Brad A. Chapman, Jillian Rowe, Christopher H. Tomkins-Tinch, Renan Valieris, the Bioconda Team, Johannes Köster (2018): **Bioconda: Sustainable and Comprehensive Software Distribution for the Life Sciences**. *Nature Methods* **15**, pp 475–476. <https://doi.org/10.1038/s41592-018-0046-7>
- [Hardisty 2021] Alex Hardisty, Paul Brack, Carole Goble, Laurence Livermore, Ben Scott, Quentin Groom, Stian Soiland-Reyes (2021): **The Specimen Data Refinery: A canonical workflow framework and FAIR Digital Object approach to speeding up digital mobilisation of natural science collections**. Submitted, *Data Intelligence, this issue*.
- [Hospital 2020] Adam Hospital, Genís Bayarri, Stian Soiland-Reyes, Jose Lluís Gelpi, Pau Andrió, Daniele Lezzi, Sarah Butcher, Ania Niewielska, Yvonne Westermaier, Rosa Maria Badia, Rodrigo Vargas, Alexandre Bonvin (2020): **BioExcel-2 Deliverable 2.3 – First release of demonstration workflows**. Project deliverable, *Zenodo*. <https://doi.org/10.5281/zenodo.4540432>
- [Ison 2013] Jon Ison, Matúš Kalaš, Inge Jonassen, Dan Bolser, Mahmut Uludag, Hamish McWilliam, James Malone, Rodrigo Lopez, Steve Pettifer, Peter Rice (2013): **EDAM: an ontology of bioinformatics operations, types of data and identifiers, topics and formats**. *Bioinformatics* **29**(10), pp 1325–1332. <https://doi.org/10.1093/bioinformatics/btt113>
- [Ison 2021] Jon Ison, Hans Ienasescu, Emil Rydzka, Piotr Chmura, Kristoffer Rapacki, Alban Gaignard, Veit Schwämmle, Jacques van Helden, Matúš Kalaš, Hervé Ménager (2021): **biotoolsSchema: a formalized schema for bioinformatics software description**. *GigaScience*, **10**(1):giaa157 <https://doi.org/10.1093/gigascience/giaa157>
- [Jupyter 2018] Jupyter Project, Matthias Bussonnier, Jessica Forde, Jeremy Freeman, Brian Granger, Tim Head, Chris Holdgraf, et al. (2018): **Binder 2.0 - Reproducible, Interactive, Sharable Environments for Science at Scale**. *Proceedings of the 17th Python in Science Conference*. SciPy, 2018. <https://doi.org/10.25080/majora-4af1f417-011>
- [Katz 2021] Daniel S. Katz, Morane Gruenpeter, Tom Honeyman, Lorraine Hwang, Mark D. Wilkinson, Vanessa Sochat, Hartwig Anzt, Carole Goble, FAIR4RS Subgroup 1 (2021): **A Fresh Look at FAIR for Research Software**. [arXiv:2101.10883](https://arxiv.org/abs/2101.10883)
- [Kluyver 2016] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, Jupyter Development Team (2016): **Jupyter Notebooks – a publishing format for reproducible computational workflows**. *Positioning and Power in Academic Publishing: Players, Agents and Agendas* pp 87 - 90. Fernando Loizides, Birgit Schmidt (eds), Proceedings of the 20th International Conference on Electronic Publishing, IOS Press. <https://doi.org/10.3233/978-1-61499-649-1-87>
- [Lamprecht 2020] Anna-Lena Lamprecht, Leyla Garcia, Mateusz Kuzak, Carlos Martinez, Ricardo Arcila, Eva Martin Del Pico, Victoria Dominguez Del Angel, et al. **Towards FAIR Principles for Research Software**. *Data Science* **3**(1), pp 37–59. <https://doi.org/10.3233/ds-190026>
- [Leipzig 2021] Jeremy Leipzig, Daniel Nüst, Charles Tapley Hoyt, Karthik Ram, Jane Greenberg (2021): **The role of metadata in reproducible computational research**. *Patterns* **2**(9):100322. <https://doi.org/10.1016/j.patter.2021.100322>
- [McMurry 2017] Julie A McMurry, Nick Juty, Niklas Blomberg, Tony Burdett, Tom Conlin, Nathalie Conte, Mélanie Courtot, John Deck, Michel Dumontier, Donal K Fellows, Alejandra Gonzalez-Beltran, Philipp Gormanns, Jeffrey Grethe, Janna Hastings, Jean-Karim Hériché, Henning Hermjakob, Jon C Ison, Rafael C Jimenez, Simon Jupp, John Kunze, Camille Laibe, Nicolas Le Novère, James Malone, Maria Jesus Martin, Johanna R McEntyre, Chris Morris, Juha Muilu, Wolfgang Müller, Philippe Rocca-Serra, Susanna-Assunta Sansone, Murat Sariyar, Jacky L Snoep, Stian Soiland-Reyes, Natalie J Stanford, Neil Swainston, Nicole Washington, Alan R Williams, Sarala M Wimalaratne, Lilly M Winfree, Katherine Wolstencroft, Carole Goble, Christopher J Mungall, Melissa A Haendel, Helen Parkinson (2017): **Identifiers for the 21st century: How to design, provision, and reuse identifiers to maximize utility and impact of life science data**. *PLOS Biology* **15**(6):e2001414 <https://doi.org/10.1371/journal.pbio.2001414>
- [Möller 2017] Steffen Möller, Stuart W. Prescott, Lars Wirzenius, Petter Reinholdtsen, Brad Chapman, Pjotr Prins, Stian Soiland-Reyes, Fabian Klötzl, Andrea Bagnacani, Matúš Kalaš, Andreas Tille, Michael R. Crusoe (2017): **Robust**

- cross-platform workflows: How technical and scientific communities collaborate to develop, test and share best practices for data analysis.** *Data Science and Engineering* **2**, pp 232–244. <https://doi.org/10.1007/s41019-017-0050-4>
- [Niewielska 2020] Ania Niewielska, Sarah Butcher, Yvonne Westermaier (2020): **BioExcel-2 Deliverable 2.5 - Provision of a Workflow Environment at BioExcel portal.** *Zenodo* <https://doi.org/10.5281/zenodo.4916060>
- [Saltz 2006] Joel Saltz, Scott Oster, Shannon Hastings, Stephen Langella, Tahsin Kurc, William Sanchez, Manav Kher, Arumani Manisundaram, Krishnakant Shanbhag, Peter Covitz (2006): **caGrid: design and implementation of the core architecture of the cancer biomedical informatics grid.** *Bioinformatics* **22**(15), pp 1910–1916, <https://doi.org/10.1093/bioinformatics/btl272>
- [Soiland-Reyes 2021] 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, Carole Goble (2021): **Packaging research artefacts with RO-Crate.** *Data Science* (accepted). [arXiv:2108.06503v1](https://arxiv.org/abs/2108.06503v1)
- [Stodden 2016] Victoria Stodden, Marcia McNutt, David H. Bailey, Ewa Deelman, Yolanda Gil, Brooks Hanson, Michael A. Heroux, John P.A. Ioannidis, Michela Taufer (2016): **Enhancing reproducibility for computational methods.** *Science* **354**(6317), pp 1240–1241 <https://doi.org/10.1126/science.aah6168>
- [Tejedor 2017] Enric Tejedor, Yolanda Becerra, Guillem Alomar, Anna Queralt, Rosa M. Badia, Jordi Torres, Toni Cortes, Jesús Labarta (2017): **PyCOMPSs: Parallel computational workflows in Python.** *LJHPCA* 31(1): 66-82. <https://doi.org/10.1177/1094342015594678>
- [Vivian 2017] John Vivian, Arjun Arkal Rao, Frank Austin Nothhaft, Christopher Ketchum, Joel Armstrong, Adam Novak, Jacob Pfeil, Jake Narkizian, Alden D Deran, Audrey Musselman-Brown, Hannes Schmidt, Peter Amstutz, Brian Craft, Mary Goldman, Kate Rosenbloom, Melissa Cline, Brian O'Connor, Megan Hanna, Chet Birger, W James Kent, David A Patterson, Anthony D Joseph, Jingchun Zhu, Sasha Zaranek, Gad Getz, David Haussler & Benedict Paten (2017): **Toil enables reproducible, open source, big biomedical data analyses.** *Nature Biotechnology* **35**, pp 314–316. <https://doi.org/10.1038/nbt.3772>
- [Wittenburg 2021] Peter Wittenburg et al. (2021): **CWFR Position Paper.** *OSF*, January 6, 2021. <https://osf.io/3rekv/>