A PROV-Compliant Approach for the Script-to-Workflow Process

Lucas A. M. C. Carvalho [a,*],
Khalid Belhajjame [b]
and Claudia Bauzer Medeiros [a]
[a] *Institute of Computing, University of Campinas, Campinas, Brazil*
E-mails: lucas.carvalho@ic.unicamp.br, cmbm@ic.unicamp.br
[b] *LAMSADE, Paris-Dauphine University, Paris, France*
E-mail: khalid.belhajjame@dauphine.fr

Abstract. Scientific discovery and analysis are increasingly computational and data-driven. Scripting languages, such as Shell, Python and R, are the means of choice of the majority of scientists to encode and run their simulations and data analyses. Although widely used, scripts are hard to understand, adapt, reuse, and reproduce. To tackle the problems faced by scripts, several approaches have been proposed such as YesWorkflow and noWorkflow. However, they neither allow to fully document the experiment nor do they help when third parties want to reuse just part of the code. Scientific Workflow Management Systems (SWfMSs) are being increasingly recognized to mitigate these problems. They help to document and reuse experiments by supporting scientists in the design and execution of their experiments, which are specified and run as interconnected (reusable) workflow components (a.k.a. building blocks). Taking this into account, we designed W2Share, a novel approach for the management, reuse, and reproducibility of script-based experiments. W2Share transforms a script into an executable workflow that is accompanied by annotations, example datasets and provenance traces of their execution, all of which encapsulated into a workflow research object. This allows third party users to understand the data analysis encoded by the original script, run the associated workflow using the same or different datasets, or even repurpose it for a different analysis. W2Share also enables traceability of the script-to-workflow process, thereby establishing trust in this process. All processes in W2Share follow a methodology that is based on requirements that we elicited for this purpose. The methodology exploits tools and standards that have been developed by the scientific community, in particular, YesWorkflow, Research Objects and the W3C PROV. This paper highlights the main components of W2Share, which is showcased through a real world use case from Molecular Dynamics. We furthermore validate our approach by testing the ability to answer competency questions that address the script-to-workflow process.

Keywords: Scientific Workflows, Script-to-Reproducible Research, Workflow Research Objects, Provenance

1. Introduction

Scripts and Scientific Workflow Management Systems (SWfMSs) [1, 2] are common approaches that have been used to automate the execution flow of processes and data analysis in scientific (computational) experiments[1]. Scripts are widely adopted in many disciplines to create pipelines for experiment execution, e.g., to clean and analyze a large amount of data. However, they are hard to understand, adapt, and reuse, often containing hundreds of lines of domain-specific code. This, in turn, forces scientists to repeatedly (re)code scripts that perform the same functions, since the effort to reuse is not worthwhile, and reproducibility is restricted to repeating the execution of exactly the same script. For this reason, several solutions have been proposed to aid experiment reproducibility for script-based environments such as Jupyter Notebooks [3], ReproZip [4], YesWorkflow [5], and noWorkflow [6].

*Lucas A. M. C. Carvalho. E-mail: lucas.carvalho@ic.unicamp.br
[1] In this paper, the term *experiment* refers to scientific experiments that are executed *in silico* – e.g., simulations.

Though those solutions help scientists capture experimental details, they neither allow to fully document the experiment, nor do they help when third parties want to reuse just part of the code. For example, the workflow-like graph obtained using YesWorkflow is abstract (in the sense that it cannot be executed by the scientists). On the other hand, the provenance traces captured using noWorkflow are fine-grained, and therefore cumbersome for the user who would like to understand the lineage of the script results [7]. SWfMSs [8], on the other hand, help documentation and reuse by supporting scientists in the design and execution of their experiments, which are specified and run as interconnected (reusable) workflow components (a.k.a. building blocks).

While workflows are better than scripts for understandability and reuse, they still require additional documentation to support reproducibility. To this end, we designed and implemented W2Share, a computational framework that supports a (script-to-reproducible research) methodology. The methodology, implemented in W2Share via a suite of tools, guides scientists in a principled manner to transform scripts into reproducible and reusable workflow research objects (WRO) [9]; it drives the development of research objects that contain the scripts that the scientist authored together with executable workflows that embody and refine the computational analyses carried out by these scripts and all associated data and documentations. Our methodology thus leverages the concept of Workflow Research Objects as a means to ensure reproducibility.

W2Share's WRO encompasses information such as the workflow itself, and datasets and provenance traces related to its execution. The WRO model [9] allows the aggregation of resources, explicitly specifying the relationship between these resources and a workflow, using a suite of ontologies. W2Share's WROs allow scientists to understand the relationships between an initial script and the resulting workflow, and to document workflows runs – e.g., annotations to describe the operations performed by the workflow, or links to other resources, such as the provenance of the results obtained by executing the workflow. Using W2Share, scientists can share and reuse scripts through the corresponding WROs.

Our approach differs in several ways from similar work to convert scripts into workflows such as [10–12]. In particular, our steps to convert scripts into executable workflows are more generic, in the sense that they are independent from the script language and the workflow system, while these other solutions are mainly designed for specific environments. Moreover, we are not only concerned about the workflow specification derived from the script code, but also to preserve the script in the WRO, allowing scientists to check experiment provenance and reproducibility.

For that, we support two forms of provenance [13]: (1) prospective and (2) retrospective. Prospective provenance captures the specification of steps and their data dependencies for a given computational task (whether it is a script or a workflow). Retrospective provenance captures the steps executed and the order of execution, along with the data consumed and produced by each step as well as different kind of metadata that help understanding and reproducing the execution.

The main contributions of this paper therefore include:

1. A methodology to guide scientists in a principled manner to transform scripts into reproducible and reusable workflow research objects[2];
2. A data model that identifies the main elements of the methodology and their relationships, which helps automate the steps of the methodology, and their documentation;
3. A computational framework that provides scientists with the tooling necessary for (semi)-automatically performing some of the steps of the methodology. Here, we emphasize that we make use of semantic web technologies, web standards and tools developed by the scientific community;
4. A case study to showcase our solution; and
5. An evaluation of the proposed model and conversion mechanism via the identification and execution of competency questions.

This paper extends previous work [14] of ours, where we defined and presented the methodology, and showcased its use via a case study (through manual implementation of the conversion). This paper describes how we now enable (semi) automatic script-to-workflow conversion, and its validation using competency queries that address requirements used to design our methodology. The automation of the conversion process is based on our data model that identifies the resources that are used and generated by our methodology as well as the agents responsible

[2]This methodology was described in [14], and is included here for completeness.

for performing the methodology steps. Last but not least, besides providing traceability for experiment execution (via workflow mechanisms), we innovate by providing traceability for the script-to-workflow process itself. We describe the design and implementation of this extended conversion process, which takes advantage of ontologies adopted by the scientific community, namely W3C PROV-O [15], Web Annotation Data Model[3], and Research Object ontology. These ontologies support the semantics of the traceability of the *script-to-workflow* process.

This paper is structured as follows. Section 2 introduces our methodology specification. Section 3 presents the model that describes its main elements. Section 4 describes the case study and the implementation of the methodology steps. Our conversion steps are described in Sections 5 and 6. Section 5 describes the first step, showing how we map scripts to abstract workflows using ontologies. Section 6 shows how an executable workflow is generated from abstract workflows. Section 7 presents the evaluation of our proposed approach. Section 8 discusses related work. Section 9 summarizes our results and identifies future work.

2. Methodology for Script Conversion into WRO

Parts of sections 2.1 and 4 appeared in our paper [14] in which we defined our methodology and exemplified its application. This has been included in this paper for clarity sake, and to make it self-contained.

2.1. Overview

Our methodology guides scientists throughout the conversion of script-based experiments into executable workflows, and then packaging all the resources and annotations into a Workflow Research Object (WRO) [16]. WROs are the means through which experiments can be reused, audited and documented. As mentioned in Section 1, our WRO encapsulates the scripts and the corresponding executable workflows together with other resources, such as datasets and provenance traces of their execution.

The methodology was designed to meet five major requirements that were derived during our long-time collaboration with scientists that run scripts for their computational experiments (e.g., in bioinformatics, chemistry and agriculture). These requirements are the following:

Requirement 1 Let S be a script that embodies a computational experiment. The scientist needs a view of S that identifies the main processing units and dependencies between such processing units.

This helps the scientist understand S, and the main processing units that are relevant from the point of view of the *in silico* analysis implemented by the script, as well as the dependencies between such units.

We call this view an *abstract workflow*. In more detail, an abstract workflow, for the purposes of this paper, is a process, in which the steps designate script blocks, and the dependencies designate data dependencies between these blocks. The workflow is abstract in that it is not executable *per se*, but rather provides a process view of a script at a higher level of granularity (logical steps as opposed to script instructions).

Requirement 2 The scientist should be able to execute the workflow that embodies the script S.

Though seemingly obvious, this is far from being a trivial requirement. It is not enough to "be able to execute". This execution should reflect what is done in script S. In other words, not only should the workflow generated be executable; the scientist must be given the means to compare its results to those of script execution, and validating the workflow as a valid embodiment of the script.

Requirement 3 The scientist should be able to modify the workflow that embodies script S, to use different computational and data resources.

Not only may a scientist be able to replicate the computational experiment encoded by S; s/he may want to repeat the analysis implemented in the script using third party resources.

The new (modified) workflow(s) correspond to variants of the initial workflow. They will help the user, for example, to inspect if the results obtained by script S can be reproduced using different resources (algorithms and datasets). Scientists will also be able to compare the execution of S with that of the variants (e.g., if web services are invoked instead of a local code implementation).

Requirement 4 Provenance information should be recorded.

Provenance information is key to traceability and quality assessment. This involves not only the provenance obtained by workflow execution. This requirement also implies recording the transformations car-

[3] https://w3.org/TR/annotation-model

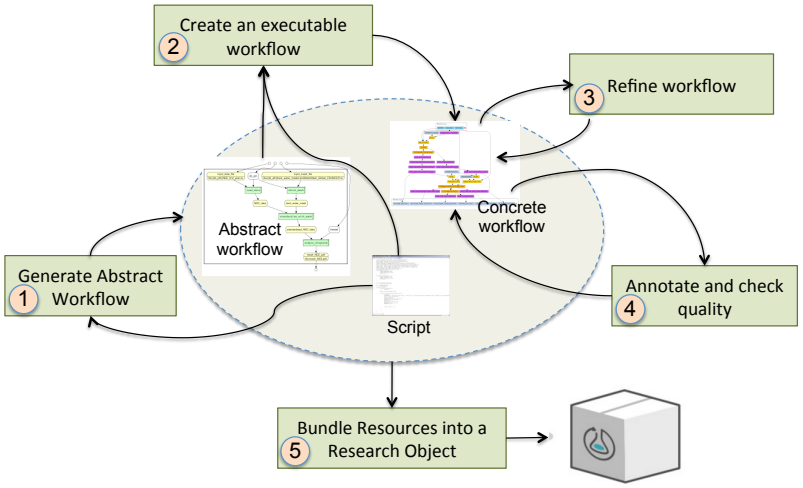


Fig. 1. Methodology for converting scripts into reproducible Workflow Research Objects, extracted from [14].

ried out to transform the script into a workflow that embodies the script. Moreover, the workflow variants also need to be recorded. As stressed by [17], provenance that is provided by the execution of a workflow corresponds to a workflow trace, and can be used for several purposes, such as to support dynamic steering of workflows [18, 19].

Requirement 5 All elements necessary to reproduce the experiment need to be captured together to promote reproducibility.

We follow the definition of [17]: "reproducibility denotes the ability for a third party who has access to the description of the original experiment and its results to reproduce those results using a possibly different setting, with the goal of confirming or disputing the original experimenter's claims." Missier et al. [17] also differentiate reproducibility from repeatability, for which results must be the same, and no changes are made anywhere.

Full reproducibility and reusability require ensuring that all elements of an experiment are recorded. S, the initial workflow, and all of its variants should be made available together with auxiliary resources that will allow understanding how these workflows came to be, and where they should be used.

Given the five requirements, we proposed our methodology composed by five inter-related steps (see figure 1). Step 1, **Generate abstract workflow**, is used to produce an abstract workflow Wa based on a script S provided by a user. This stage elicits the main processing units that constitute the analysis implemented by the script, and their data dependencies. This requires user intervention, to identify these units and dependencies within the script. Workflow Wa obtained from Step 1 is abstract in the sense that it cannot be executed – it is only a workflow-like high level specification of the script. Already at this stage, even though unable to execute the workflow, this is already a step towards promoting understandability – the abstract workflow is a high-level specification of the script, and can be visualized as a graph linking computational units. The objective of this phase is to address Requirement 1.

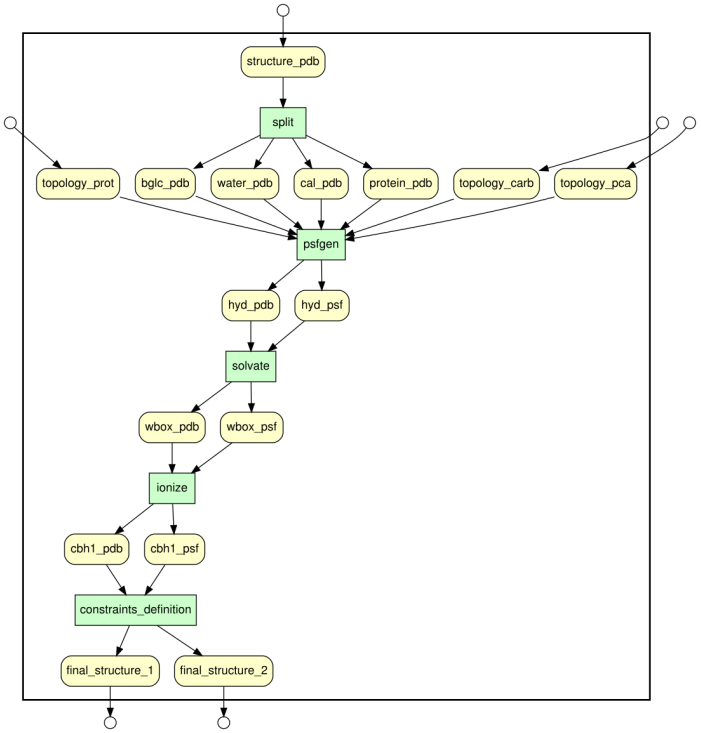
Figure 2 illustrates this step. The left side shows an excerpt of the script (from hundreds of lines of script code) and the right side the corresponding abstract workflow. This example will be discussed at length in subsequent sections; the figure is introduced here to give a high level view of this first step of the conversion process.

Step 2, **Create an executable workflow**, converts the abstract workflow Wa into an executable one We. The objective of this phase is to address Requirement 2. This is achieved by actually replacing each processing unit in the abstract workflow by its implementation (e.g., encapsulating the corresponding script code), and adding code to allow the required I/O operations across these units.

Scientists frequently try different variants of a computational experiment, e.g., to improve results, or to check alternatives. Script-based experiments are not easily modified, and it is hard to keep track of these variants. Tools such as version control systems allow to track the versions/changes of scripts and programs in general. Our methodology contemplates this activity. Step 3, **Refine workflow**, addresses Requirement 3

```
# @BEGIN solvate        @DESC Using the program solvate to imerse the system in a water box.
# @IN hyd_pdb           @DESC crystal structure with hydrogen atoms added
#                       @URI file:hyd.pdb
# @IN hyd_psf           @DESC structure file for the crystal structure with hydrogen atoms added
#                       @URI file:hyd.psf
# @OUT wbox_pdb         @DESC coordinates of the solvated system
#                       @URI file:wbox.pdb
# @OUT wbox_psf         @DESC structure file of the solvated system
#                       @URI file:wbox.psf
echo "
package require solvate
solvate hyd.psf hyd.pdb -rotate -t 16 -o wbox
exit
" > solv.tcl

vmd -dispdev text -e solv.tcl
rm solv.tcl
# @END solvate

# @BEGIN ionize         @DESC  With the program autoionize, introduce ions to neutralize the syst
# @IN wbox_pdb          @DESC coordinates of the solvated system
#                       @URI file:wbox.pdb
# @IN wbox_psf          @DESC structure file of the solvated system
#                       @URI file:wbox.psf
# @OUT cbh1_pdb         @DESC initial coordinates for the simulations, containing protein, water,
#                       @URI file:cbh1.pdb
# @OUT cbh1_psf         @DESC structure file for the coordinate file cbh1.pdb
#                       @URI file:cbh1.psf
# ionic strength = 0.10
echo "
package require autoionize
autoionize -psf wbox.psf -pdb wbox.pdb -nna 21 -ncl 10 -o cbh1
exit
" > ionize.tcl
vmd -dispdev text -e ionize.tcl
rm ionize.tcl wbox* hyd*

# @END ionize

# @BEGIN constraints_definition @DESC We define the atoms that should be kept fixed in the equili
# @IN cbh1_pdb                  @DESC initial coordinates for the simulations, containing protein
#                               @URI file:cbh1.pdb
# @IN cbh1_psf                  @DESC structure file for the coordinate file cbh1.pdb
#                               @URI file:cbh1.psf
# @OUT fixed_1                  @DESC coordinates for the whole system (cbh1.pdb), indicating whi
#                               @URI file:fixed.1.pdb
#                               @AS final_structure_1
# @OUT fixed_2                  @DESC coordinates for the whole system (cbh1.pdb), indicating whi
#                               @URI file:fixed.2.pdb
#                               @AS final_structure_2
echo '
mol new cbh1.psf
mol addfile cbh1.pdb
set all [atomselect top all]
$all set beta 0
set fixed [atomselect top "(protein or segname BGLC) and noh"]
$fixed set beta 500
$all writepdb fixed.1.pdb
```

(a)



(b)

Fig. 2. The first step of our methodology concerns the generation of an abstract workflow from the script, illustrated here by (a) excerpt of the script and (b) the corresponding abstract workflow. This enhances understandability, and thus collaborative work.

and supports full reusability. It allows the creation of variants of the executable workflow, e.g., by adding new processing units, or changing data sources.

At the end of Step 3, the scientist will have one or more workflow variants $We_1 \ldots We_n$. The idea, here, is that there is a difference between the concepts of repeatability and reproducibility. The first consists in exact reproduction of the experiment – running the same code, with the same data sets. Reproducibility, on the other hand, means that the results of an experiment should be reproducible, but not necessarily by invoking the same processes – e.g., code optimization can improve execution time.

Moreover, versioning allows scientists to try out variants of an experiment [20, 21], comparing and testing alternative outcomes. Overall, there are several kinds of refinements that can be performed at this step, all of which facilitated by the use of workflows and their components as reusable units.

During steps 2 and 3, provenance data both from the workflow executions and the process of conversion are collected to be used in Steps 4 and 5, and address requirement 4.

Step 4, **Annotate and check quality**, is in charge of evaluating whether the workflow reproduces the script results within some scientist-defined tolerance thresholds. It takes advantage of workflow execution mechanisms, that keep track of execution traces. Step 4 uses the workflow information generated in the previous steps, including provenance traces.

Finally, in Step 5, **Bundle Resources into a Research Object**, the workflow and the auxiliary resources, i.e., annotations, provenance traces, datasets, among others, are packaged into a WRO. WROs are then stored and made available to third parties for experiment validation, reproducibility checks, and reuse of workflow components. The objective of this phase is to address Requirement 5.

3. W2Share's Data Model: Supporting the Methodology

The full implementation of the methodology requires an appropriate data model, described here. It helps dynamic documentation of the conversion pro-

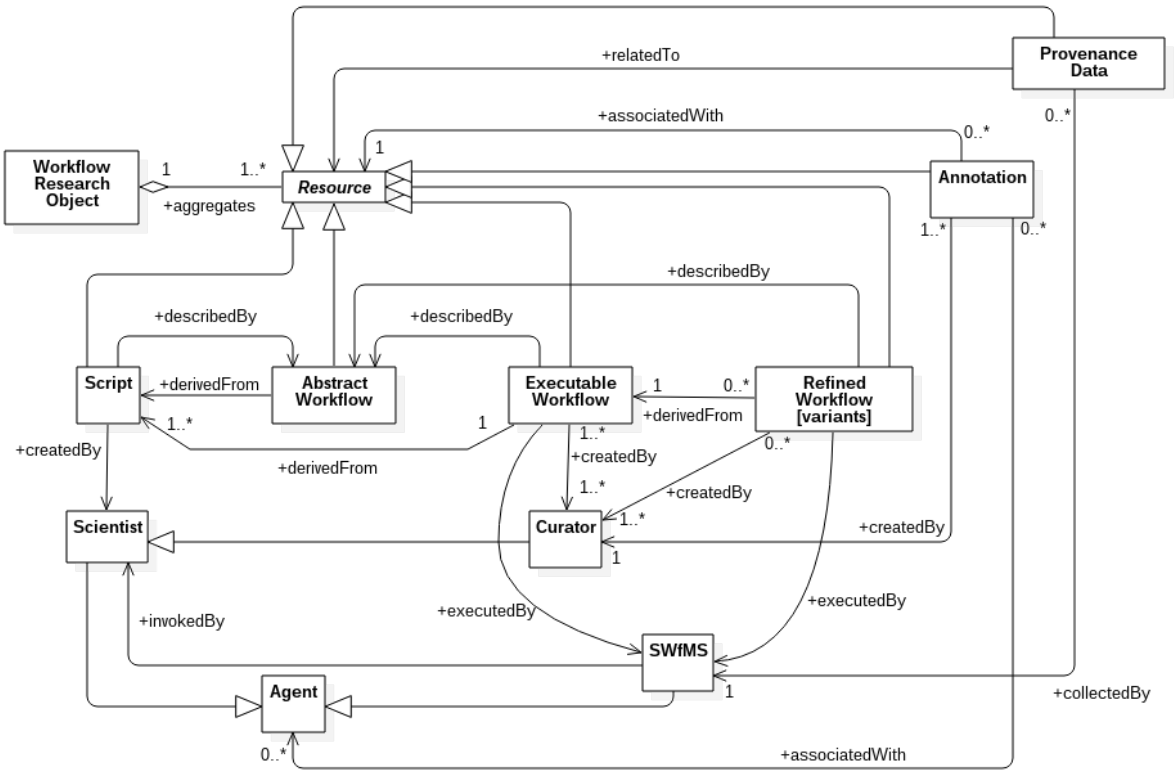


Fig. 3. Model describing main elements and relationships used in our methodology.

cess, thereby ensuring traceability of that process. We point out this adds a new dimension to traceability, which is usually restricted to the execution of experiments, but not to their evolution.

Figure 3 shows the UML class diagram of W2Share's data model, which reproduces the main entities and relationships involved in our methodology. The model describes the Resources that compose a WRO and the Agents that are responsible for creating these Resources.

Resources include, for instance, `Script`, `Abstract Workflow`, `Annotation`, `Provenance Data`, among others. These resources are not independent of one another – the model accounts for the relationships created in the transformation process – from Script to Abstract Workflow to Executable workflow, which can then give origin to several Refined workflows (variants).

Agents perform the activities in the methodology. As example of an Agent, a SWfMS (Scientific workflow Management System), which is invoked by a Curator (another Agent), is responsible for executing workflows and collecting Provenance Data. The model differentiates between generic Scientists and Curators, scientists who are knowledgeable about documentation and resource management.

As mentioned before, we support two kinds of traceability – of experiment execution (based on SWfMS "logs") and of the conversion process itself. Traceability of the conversion process is enabled via relationships that are based on PROV. Examples include `annotatedBy`, `generatedBy`, `createdBy`, `derivedFrom`, and `collectedBy`. The adoption of PROV allows to navigate the derivation between the Executable Workflow and its variants in Refined Workflow. Executable workflows We are not derived from a Wa but directly from S. On the other hand, Refined Workflows We_1, We_2 ... We_n are derived from We.

Figure 4 shows a UML class diagram that refines part of figure 3, and is used to record the provenance of resources generated under our methodology. Here, we can see that Scripts are composed of Code Blocks; workflows (`Abstract Workflow`, `Executable Workflow` and `Refined Workflow` classes) are

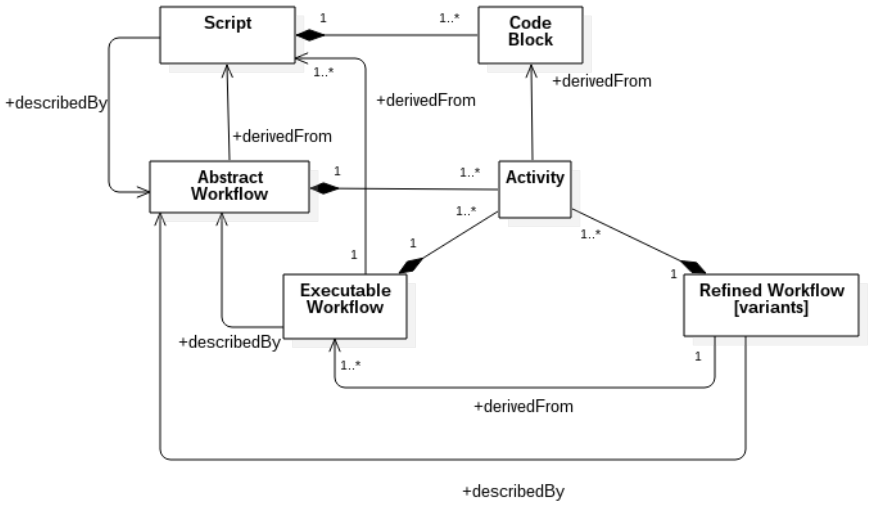


Fig. 4. Relationships between main entities regarding the tracking of the script-to-workflow conversion in our model.

composed of Activities, and the latter may be derived from script Code Blocks. Abstract Workflows describe Scripts and Workflows, thus allowing scientists to pose queries to explore prospective provenance. Eventual one-to-one relationship cardinalities were omitted from both figures, for readability.

Next, we show how to take advantage of the methodology to convert a script into a WRO in our case study from Molecular Dynamics.

4. Case Study – Molecular Dynamics

4.1. Overview

Molecular dynamics (MD) simulations consist of a series of algorithms developed to iteratively solve the set of coupled differential equations that determine the trajectories of individual atoms that constitute the particular physical system. This involves a long sequence of scripts and codes.

MD simulations are used in many branches of material sciences, computational engineering, physics and chemistry. A typical MD simulation experiment receives as input the structure, topology and force fields of the molecular system and produces molecular trajectories as output. Simulations are subject to a suite of parameters, including thermodynamic variables.

Many groups have implemented their specific MD simulations using special purpose scripts. In our case study, a suite of scripts was designed by physiochemists [22]; its inputs are the protein structure (obtained from the RCSB PDB protein data bank[4]), the simulation parameters and force field files.

There are many kinds of input files and variables, and their configuration varies with simulation processes. For instance, the input multimolecular structure contains the initial set of Cartesian coordinates for every atom/particle in the system, which will evolve in time in the MD simulation. This initial structure varies according to the system to be simulated and research area. Our case study requires immersing proteins in a solvent. Protein Cartesian atomic coordinates are made available in specialized data repositories, most notably the Protein Data Bank (PDB). Typical systems contain from several thousands to millions of covalently bound atoms.

Parts of the text in this section are based on [14].

4.2. Implementation of Methodology Steps

W2Share was conceived to take advantage of tools and standards that have been developed by the scientific community to support reproducibility and reuse, in particular YesWorkflow [5] and Research Objects. Its implementation includes elements of the PROV ontology, thereby facilitating provenance annotation. This is presented in Section 5 of this paper, and is one of the paper's contributions.

[4] http://www.rcsb.org/pdb/

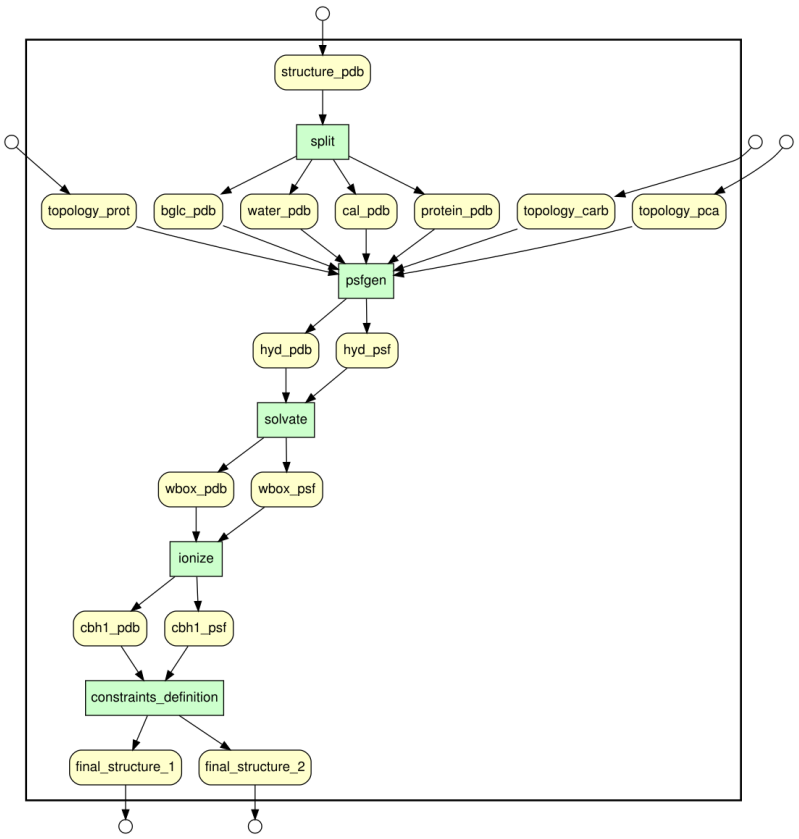


Fig. 5. Visualization of the abstract workflow of our MD case study, extracted from [14].

Consider the script that sets up an MD simulation. First, a scientist identifies the main processing units, and their dependencies. To do so, W2Share adopts the YesWorkflow tool. It enables scientists to annotate existing scripts with special comments that reveal the computational modules and data flows implicit in these scripts. YesWorkflow extracts and analyzes these comments, represents the scripts in terms of entities based on the typical scientific workflow model, and provides graphical renderings of this workflow. It does so by processing scientist-provided tags of the form @*tag value*, where @*tag* is a keyword that is recognized by YesWorkflow, and *value* is a value assigned to the tag. Tag recognition is script-language independent, therefore allowing a wide range of script-based experiments to be converted into a workflow representation and consequently a wider adoption of our methodology. W2Share creates the corresponding abstract workflow *Wa* (see figure 5 for the corresponding visualization) from the annotated script *S*, available in Listing A.1 in Appendix A.

This abstract workflow is a first approximation of what is needed for full reproducibility. Section 5 will detail how W2Share supports the creation of PROV-compliant machine-readable abstract workflows. The rest of this section will ignore these details, since they are not necessary to describe the full implementation of the methodology steps.

Given the abstract workflow *Wa* generated previously, the scientist needs to create an executable workflow *We* that embodies the data analysis and processes as depicted by *Wa* – and thus embodies the original script. For this, the scientist needs to specify, for each activity in the abstract workflow, the corresponding concrete activity that implements it. A simple, yet effective approach to do so consists in exploiting a readily available resource, namely the script code itself. Given an activity in *Wa*, the corresponding code in *We* can be generated by reusing the chunk (block) of the script that is associated with that abstract workflow activity. This approach for conversion comes with two advantages: (i) ease of conversion, since we are using a readily available resource, i.e. the script code,

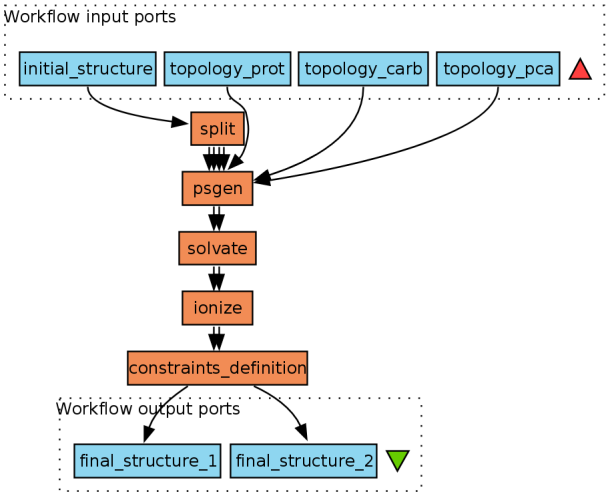


Fig. 6. Executable workflow of our MD case study, extracted from [14].

and (ii) the ability to check and debug the execution of *We* against the script execution, to correct eventual mistakes in script-to-workflow conversion. Once the scientist specifies the implementation of each activity in *Wa*, a concrete workflow specification *We* that is conform to a given scientific workflow system can be created. This manual conversion, detailed at length in [14], is now supported by W2Share – the semi-automatic implementation of Step 2 is detailed in Section 6.

Without loss of generality, we used the Taverna system [23], although our solution can be adapted to other scientific workflow systems. We chose Taverna as our implementation platform due to its widespread adoption in several eScience domains and because it supports the execution of shell scripts, the script language adopted in our case study. Figure 6 shows the executable workflow *We*, which is derived from *S* and described by *Wa*.

W2Share also helps scientists in creating workflow variants. For instance, in our case study, scripts use local data files containing protein coordinates which scientists download from authoritative web sources. This forces them to download such files from the web, and to update them locally whenever they are modified, moreover making them keep track of many file directories, sometimes with redundant information. An example of refinement would be to use web services to retrieve these files. We exemplify an even more helpful refinement – rather than reuse code, to reuse workflows that perform this task: we retrieved from the my-

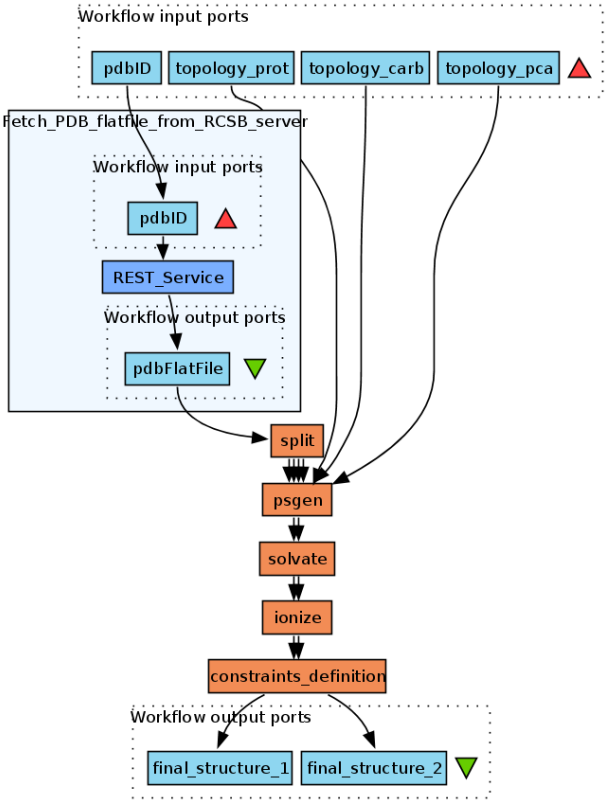


Fig. 7. Refined workflow of our MD case study, extracted from [14].

Experiment repository[5] a small workflow that fetches a protein structure on Protein Data Bank (PDB) from the RCSB PDB archive[6]. The reused myExperiment workflow was inserted in the beginning of our original workflow (see figure 7 for the workflow variant We_1).

Here, the *initial_structure* input parameter of figure 6 (the local PDB file) was replaced by the sub-workflow within the light blue box, copied from the myExperiment workflow repositories. This new (sub)workflow downloads the protein file from the web using a web service (whereas the original code used a local protein file). Similarly, in the life sciences, scientists can invoke web services or reuse data sets listed on portals such as Biocatalogue[7], which provides a curated catalogue of Web services, and Biotools[8], which is a tools and data services registry.

During the conversion process, additional activities must be performed. First, it is critical to have a quality

[5]http://www.myexperiment.org
[6]http://www.rcsb.org/pdb/
[7]https://www.biocatalogue.org/
[8]https://bio.tools

check where the scientist explicitly assesses the workflow activities and data flow, comparing them to what was executed by the script. Hence, throughout the process of workflow creation and modification, the scientist should provide annotations describing it (i.e. activities and ports), and potentially the resources it utilizes. Part of these annotations can be migrated to the executable workflow taking advantage of the YesWorkflow tags - e.g., @*desc* used in the script to describe its program blocks and ports. Most SWfMSs, moreover, provide an annotation interface, which can be taken advantage of. Our work [24] partially describes some of those annotation tasks.

Second, provenance information is used by W2Share for several purposes. Besides experiment reproducibility, it is recorded to capture the steps performed in the transformation from script to workflow. This uses a provenance model, which allows identifying the correspondence between workflow activiti(es) and script code, and reusable components/web services and script excerpts. The lineage of variants of the workflow should be stored, as well. It is important to inform to future users that the workflow was curated, and how this curation process occurred. Provenance capture is presented in section 5 of this paper.

Finally, W2Share assists the scientist to create a Workflow Research Object (WRO) that bundles the original script as well as other auxiliary resources obtained in the other steps of the methodology. The Workflow Research Object model [9, 16] allows scientists to aggregate resources and explicitly specify the relationship between these resources and the workflow in a machine-readable format using a suite of ontologies.

The resulting WRO bundles a number of resources that promote the understanding, reproducibility and ultimately the reuse of the workflows obtained through refinement. By including these resources, it is possible for scientists not only to understand how the experiment was conducted, but also its context. Moreover, curators can also bundle additional documents that may help scientists understand the WRO, e.g., technical reports and published papers.

We use the RO Manager tool[9] to create the WRO bundle file. The bundle for this case study is available in [25].

However, it is not enough to create such research objects; they must be made available to the scientific community in a user-friendly manner, so that not only machines, but also scientists can select the most appropriate ones. A possible solution is to make them available by depositing them in a Research Object Portal such as W2Share[10], myExperiment and RO Hub[11] which have an interface to search and navigate between resources aggregated in a RO.

5. Revisiting the Implementation of Step 1: Mapping Scripts into PROV-Compliant Machine-Readable Abstract Workflows

The first step of our methodology is the conversion of scripts into abstract workflows. One of our innovations is the creation of a new kind of abstract workflow for scripts – one that is ontology-based and, moreover, machine-readable. We call this a "machine-readable abstract workflow" (as opposed to the abstract workflows described in the literature, which are usually structures devoid of any semantics).

This section explains how W2Share enables the transformation of a script S into the machine-readable abstract workflow Wa. The latter, in turn, is used to create and describe the corresponding executable workflow We, and its subsequent variants We_1, We_2, etc.

Section 4.2 shows how a scientist can easily transform a script into an abstract workflow with help of the YesWorkflow suite of tools [5, 26]. However, these abstract workflows are not machine-readable. Indeed, YesWorkflow has two outputs – an image of a workflow, and Datalog code that encodes the corresponding structure. This limits its interoperability with approaches that use semantic technologies. Moreover, YesWorkflow's workflow representation, if considered apart from the originating script, does not allow obtaining provenance information on how it was derived from the script.

Our solution is to transform script S into machine-readable abstract workflows Wa in a three-stage process. First, we use YesWorkflow to extract the workflow topology from S. Next, we transform this structure into an ontology-based structure using a workflow specification ontology. Finally, we add provenance information to link Wa back to S (thereby also supporting traceability of the conversion process).

[9] https://github.com/wf4ever/ro-manager

[10] https://w3id.org/w2share

[11] http://www.rohub.org/

Table 1
Mapping of YesWorkflow tags to workflow ontologies

Tag	Class	Property
@begin	wfdesc:Workflow	wfdesc:hasSubProcess
	wfdesc:Process	rdfs:label
	wf4ever:Script	
@desc	–	dct:description
@in	wfdesc:Input	wfdesc:hasIntput; rdfs:label
@param		
@out	wfdesc:Output	wfdesc:hasOutput; rdfs:label
	wfdesc:DataLink	wfdesc:hasDataLink
		wfdesc:hasSource
		wfdesc:hasSink
@as	–	rdfs:label

In more detail, the first stage creates the YesWorkflow abstract representation. In the second stage, we use ontologies to transform this representation into a semantic one. This is achieved by mapping YesWorkflow tags to workflow entities that are semantically defined via wfdesc [9], a workflow specification ontology from the Research Object suite of ontologies [16], and other additional ontologies, e.g., Wf4ever [12], RDF Schema and Dublin Core [13]. Finally, in the third stage, we process tags to insert provenance information, i.e., we create an additional layer of provenance over the abstract semantic workflow.

Table 1 summarizes the second stage, showing how YesWorkflow tags are mapped to classes and properties of ontologies. Here, we use the following name spaces: `ro` for Research Object, `wfdesc` for the wfdesc Ontology, `wf4ever` for the Wf4ever Schema, `dct` for the Dublin Core terms, `rdfs` for RDF schema and `prov` for the PROV ontology.

Figure 8 shows an excerpt of the second stage. On the left side of this figure we have a script using YesWorkflow tags. The right side shows the RDF triples that correspond to these tags. Numbers (0), (1), (2), and (3) connect both sides of the figure. For instance, on the left side, (1) has annotation `@begin md_setup`, which is mapped to the class `wfdesc:Workflow` and the property `rdfs:label` with value `md_setup`. While `@desc setup of a MD simulation` is mapped to property `dc:description` with value `setup of a MD simulation`.

[12] https://w3id.org/ro/wf4ever
[13] http://dublincore.org/

On the left side, (2) originates the input and output RDF triples on the right side. The input and output ports of the workflow use classes `wfdesc:Input` and `wfdesc:Output`, and properties `rdfs:label` and `dc: description`. Nested `@begin` tag ((3) – left side) are mapped to the `wfdesc:hasSubProcess` property and the `wfdesc:Process` class, specifying an activity of the workflow. The mapping also uses the `wfdesc:DataLink` class, as well as the properties `wfdesc:hasSource` and `wfdesc:hasSink`, identifying a link between two ports in the workflow.

At the third stage of the transformation of S to Wa, provenance information is added to the triples code. Provenance semantics are provided by using the PROV ontology [15]. Each abstract workflow element is defined as a `prov:Entity`. Again, in Figure 8, in (0), the script filename is mapped to triples defining the script resource `<resources/script.sh>` as a `wf4ever:Script`. This specific mapping is independent of the use of any YesWorkflow tag. The property `prov:wasDerivedFrom` is created with value `<resources/script.sh>`, identifying from which script that workflow was derived, since an experiment may have more than one script file. The script code committed within a block in (4) (left side), originates the identification of the text position in the script code using properties and classes such as `prov:wasDerivedFrom` from PROV and `oa:TextPositionSelector`, `oa:start`, and `oa:end` from the Web Annotation Ontology, to delimit this code. Another useful provenance information added at this stage is by whom and when the transformation was performed.

Summing up, this section described W2Share's process to generate a machine-readable (semantic) abstract representation of a workflow in which workflow blocks are linked back to the original script block, and script resources are duly semantically annotated. Annotations also indicate the agents responsible for the script-to-workflow conversion. This helps reproducibility by documenting the conversion process. This also helps reuse of workflow specifications.

Listing A.2 in A shows an excerpt of Wa and Listing A.3 in A shows the provenance information generated by the transformation. These listings use the RDF Turtle format[14] and are results of the three transformation stages from S to Wa using our MD case study.

[14] http://www.w3.org/TR/turtle/

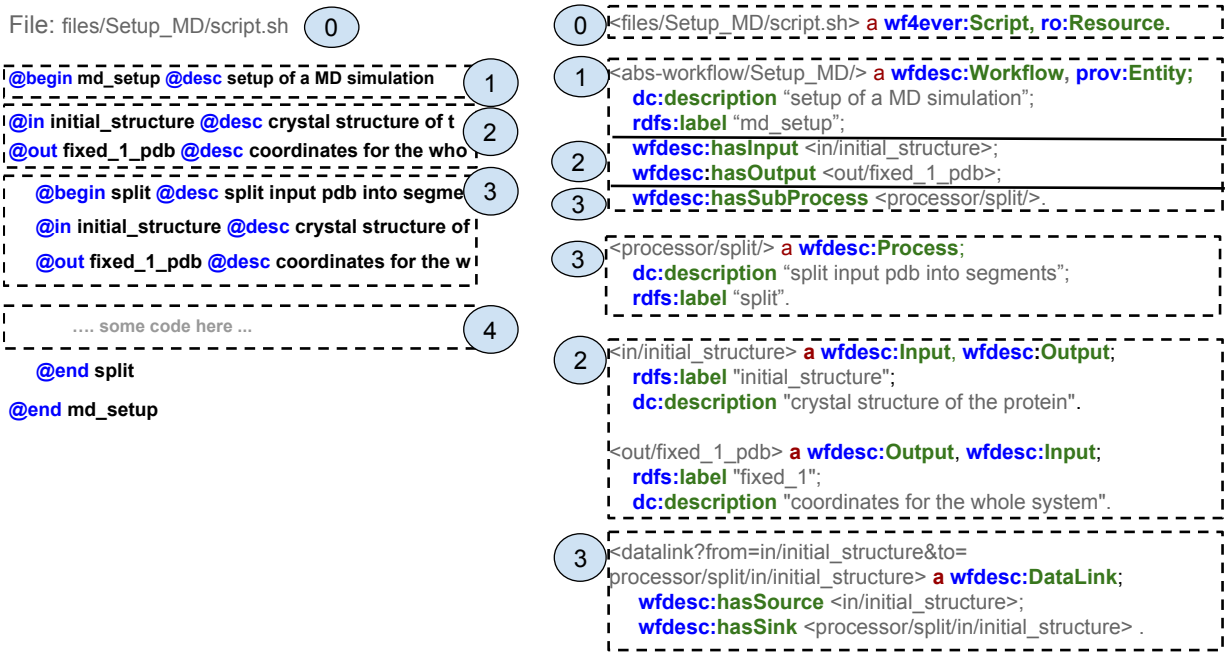


Fig. 8. Mapping between YesWorkflow tags (left side) and classes and properties of ontologies (right side).

6. Revisiting Step 2: (Semi-)Automatically Transforming Abstract Workflows into Executable Workflows

After creating *Wa*, a machine-readable abstract workflow, the next step is to create an executable workflow *We*, which corresponds to Step 2 of our methodology. We here show how this can be done automatically, in the best case and semi-automatically in the other cases, mapping *Wa* elements to elements that can be executed in a SWfMS, e.g., Taverna. This step was outlined in a previous work [24]; here we provide a detailed description.

As presented in Section 5, our machine-readable abstract workflow *Wa* describes the activities encoded by the script code. In our provenance layer, entities point back to the corresponding script code block. The *Wa* to *We* conversion process will now take advantage of this provenance information. At the end, *We* and *S* share the same abstract workflow *Wa* to describe their activities in a higher level.

During the creation of *We*, the original script *S* code may be manually changed (e.g., to allow appropriate workflow execution in the chosen SWfMS). So, the scientist must be aware of potential issues caused by these modifications. Some changes in the code can be performed automatically, e.g., library imports. Others might need manual intervention, such as changing a reference to an absolute path to a file to obtain the file from a workflow port. By identifying these manual changes in the workflow implementation, experts can describe the reason behind the changes, which helps documenting the conversion process.

W2Share's machine-readable abstract workflows allow linking *We* to *S*, thus enabling questions related to the sequence of transformation steps that led to the production of an executable workflow. Examples include *"which script block originated a specific activity in this workflow?"*, or *"which workflow activities do not have exactly the same code as the script code that originated it?"*. The latter question would use the current implementation of the workflow activity and a simple comparison with the original script block code. In addition, scientists can use annotations regarding the reason behind the changes to foster the understanding of the process.

Listing A.3 in A shows an excerpt of PROV statements in RDF Turtle format to allow tracing back elements of the executable workflow *We* to the elements of script *S* through *Wa*.

7. Evaluation

7.1. Overview

To validate our proposal, we adopt the notion of *competency queries*, e.g., as defined by [27] "A competency query is a query that is useful for the community at hand, e.g. for a human member (e.g. a scientist), or for building applications for that domain. Therefore, a list of such queries can sketch the desired scope and the desired structuring of the information."

Our competency queries show W2Share's ability to answer questions about a workflow's lineage thanks to prospective provenance generated during the script-to-workflow process. Questions about workflow executions are answered thanks to retrospective provenance obtained from the SWfMS during workflow execution. All these resources are bundled in WROs.

The queries proposed in this section should help scientists to understand and explore the conversion process and consequently assess the quality and establish trust in this process. To achieve this goal, the queries return prospective and retrospective provenance information. Examples of the prospective view include, e.g, how the workflow was created from the script, and who created the workflow. Retrospective views include associating workflow results to the script which originated the workflow.

Given as input S, Wa, We, We_1, ... We_n, we consider the following kinds of queries for prospective provenance:

1. tracking elements: activities, data, data flows in We back to S;
2. metadata: information describing elements of script, workflows, and agents;
3. provenance of a given data source (before execution).

We consider the following kinds of queries for retrospective provenance:

1. establishing trust: comparison of workflow and script results, and comparison of workflow variant results;
2. tracking elements: link elements derived from S to traces.

Here, we consider that these queries are important to help the scientists to establish trust and assess the quality of the conversion by comparing the workflow results to the script results.

7.2. Executing Queries

Here, we specify competency questions associated to the requirements of Section 2. To each question, we provide a specific SPARQL query, which we evaluate against the contents of the WRO generated by our case study.

We point out that these queries do not inspect the scripts or executable workflows. Rather, queries are processed against machine-readable abstract workflows. All these prospective provenance representations use *wfdesc*, PROV and Web Annotation ontologies. The retrospective provenance representation uses *wfprov* ontology which is part of the Research Object suite of ontologies [16]. The competency questions, the SPARQL queries and results, and the RDF statements representing our case study can be found online in [28].

We are interested in the following competency queries to address the requirements:

1. Retrieving information about the abstract workflow Wa derived from S (i.e., processing units and their dependencies) – addressing requirement R1.
2. Retrieving information about workflow We derived from S – addressing requirement R2.
3. Retrieving information about workflow variants derived from We (i.e., We_1, We_2, ... We_n) – addressing requirement R3.
4. Retrieving lineage information associating We and script elements (i.e., input, outputs, activities and data links) – addressing requirement R4.
5. Retrieving metadata about the conversion process (e.g, the person who created a given executable workflow) – addressing requirement R4.
6. Retrieving information tracking workflow execution traces to script blocks – addressing requirement R4.
7. Retrieving the resources available in the WRO associated to an experiment – addressing requirement R5.

We now present a description of each query, why it is relevant, the kind of situations for which it would be needed, their translation into SPARQL[15], and the results obtained by evaluating them.

These queries are run against the RDF statements in Listings A.2, A.3, A.4, and A.5 in Appendix A.

[15] https://www.w3.org/TR/rdf-sparql-query/

Query 1 **Retrieving information about abstract workflow *Wa* derived from *S*.** This query is responsible for identifying, given *S*, the processing units and their dependencies which compose *Wa*. It is useful, for example, for scientists to understand the data analysis carried out by the experiment.

Listing 1 shows the SPARQL query that can be used for answering this query. We use the WRO URI as base for the queries: https://w3id.org/w2share/wro/md-setup/.

Listing 1: Query 1 translated into a SPARQL

```
select ?abs ?winput ?woutput ?process ?pinput ?poutput
where {
   ?abs prov:wasDerivedFrom <files/Setup_MD/script.sh> ;
     wfdesc:hasInput ?winput;
     wfdesc:hasOutput ?woutput ;
     wfdesc:hasSubProcess ?process .
   ?process wfdesc:hasInput ?pinput;
            wfdesc:hasOutput ?poutput .
}
```

Table 2 shows the results obtained by evaluating the query. The results point out that *Wa* <abs-workflow/Setup_MD/> is the abstract workflow derived from <files/Setup_MD/script.sh>. *Wa* has input <abs-workflow/Setup_MD/in/structure_pdb> and output <abs-workflow/Setup_MD/out/fixed_1_pdb>. Also, it has <abs-workflow/Setup_MD/processor/split> as one of its activities, which has input <abs-workflow/Setup_MD/ processor/split/in/structure_pdb> and output <abs-workflow/Setup_MD/processor/split/out/fixed_1_pdb>.

Table 2
Result of evaluating query 1

Variable	Value
abs	<abs-workflow/Setup_MD>
winput	<abs-workflow/Setup_MD/in/structure_pdb>
woutput	<abs-workflow/Setup_MD/out/fixed_1_pdb>
processor	<abs-workflow/Setup_MD/processor/split>
pinput	<abs-workflow/Setup_MD/processor/split/in/initial_structure>
poutput	<abs-workflow/Setup_MD/processor/split/out/cbh1_pdb>

Query 2: **Retrieving information about the executable Workflow derived from *S*.** This query is responsible for identifying which executable workflow is derived from *S*. This is useful for scientists interested in executing or reusing pieces of this workflow.

Listing 2: Query 2 translated into a SPARQL

```
select ?workflow
where {
   ?workflow a wfdesc:Workflow;
         prov:wasDerivedFrom <files/Setup_MD/script.sh> .
}
```

Listings 2 shows the SPARQL code for this query. Table 3 shows the results obtained by evaluating the query. The results point out that <workflow/Setup_MD> is derived from <files/Setup_MD/script.sh>.

Table 3
Result of evaluating query 2

Variable	Value
Workflow	<workflow/Setup_MD>

To have a deep understanding of the differences between the implementation of the workflow *We* and the script *S*, it would be necessary to compare the specification of both implementations, or to have annotations describing the rationale of these changes. However, this is outside the scope of this paper.

Query 3: **Retrieving information about workflow variants derived from *We*.** This query is responsible for identifying, given an executable workflow *We*, which workflows are derived from it. It is useful, for example, for scientists to find workflow variants to run, reuse or compare their results. This query is also useful when a scientist updates an executable workflow, and needs to propagate this update to the workflow variants.

Listing 3 shows the SPARQL code that can be used for answering this query.

Listing 3: Query 3 translated into a SPARQL

```
select ?variant
where {
   ?variant prov:wasDerivedFrom <workflow/Setup_MD/> .
}
```

Table 4 shows the results obtained by evaluating the query. The results point out <workflow/Setup_MD> originated the variant <workflow/Setup_MD/variant>.

Table 4
Result of evaluating query 3

Variable	Value
variant	<workflow/Setup_MD/variant/>

Again, to have a deep understanding of the differences between the implementations of the workflow We and its variants We_1, We_2, ... We_n, it would be necessary to compare the specification of both implementations, or to have annotations describing the rationale of these changes. Again, this is outside the scope of this paper.

Query 4: **Retrieving lineage information associating We and script elements.**

This query is responsible for identifying, given an executable workflow We, which script blocks originated each workflow activity. It is useful, for example, for scientists to compare the executable workflow and script implementations.

We use the Web Annotation Ontology element *oa:TextPositionSelector* and its properties *oa:start* and *oa:end* to delimit the textual position of blocks in the script code. Listing 4 shows the SPARQL code that can be used for answering this query.

Listing 4: Query 4 translated into a SPARQL

```
select ?abs ?process ?start ?end
where {
   ?abs prov:wasDerivedFrom
    ↪  <files/Setup_MD/script.sh> ;
      wfdesc:hasSubProcess ?process .
   ?process prov:wasDerivedFrom ?code .
   ?code oa:start ?start ;
      oa:end ?end .
}
```

Table 5 shows the results obtained by evaluating the query. The results point out that <workflow/Setup_MD/split> was derived from <abs-workflow/Setup_MD/split>, which is defined in the text position from 1644 to 1786 in S.

Query 5: **Retrieving metadata about the conversion process.** This query is responsible for retrieving metadata regarding each step of the script-to-workflow conversion process (e.g., who was the person in charge of annotating a script S to create the abstract workflow).

Table 5
Result of evaluating query 4

Variable	Value
workflow_process	<workflow/Setup_MD/processor/split>
script_process	<abs-workflow/Setup_MD/processor/split>
block_start	1644
block_end	1786

This query is useful helping to establish trust in the conversion process. Listings 5 shows the corresponding SPARQL code. Table 6 shows the result of the query, which points out Lucas Carvalho is the curator associated with the creation of <abs-workflow/Setup_MD/> for S.

Listing 5: Query 5 translated into a SPARQL

```
select distinct ?curator
where {
  <abs-workflow/Setup_MD/>
    ↪  prov:wasAttributedTo ?agent .
  ?agent foaf:name ?curator .
}
```

Table 6
Result of evaluating query 5

Variable	Value
curator	Lucas Carvalho

Query 6: **Retrieving information tracking workflow execution traces of We to script blocks.** This query is responsible for identifying, given a workflow execution trace, the original script blocks associated with it. This query is useful when a scientist wants to retrieve workflow executions and compare them with script executions.

Listing 6 shows the SPARQL code for this query. Table 7 shows the result of the query, which points out that <workflow/processor/split/> was derived from the script block <abs-workflow/processor/split>, used as input <> and produced as output <data/4e0a1f-fc0f/output/bglc.pdb>.

Listing 6: Query 6 translated into a SPARQL

```
SELECT DISTINCT ?workflow ?workflowRun
  ↪  ?process ?output ?input
WHERE {
```

```
    ?workflow prov:wasDerivedFrom
     ↪ <files/Setup_MD/script.sh> .
    ?workflow wfdesc:hasSubProcess ?process .
    ?processRun wfprov:describedByProcess
     ↪ ?process ;
        prov:used ?input ;
        wfprov:wasPartOfWorkflowRun
         ↪ ?workflowRun .
    ?output prov:wasGeneratedBy ?processRun .
}
```

Table 7
Result of evaluating query 6

Variable	Value
workflow	<workflow/Setup_MD/>
workflowRun	<run/4e0a1f-fc0f/>
process	<workflow/Setup_MD/processor/split/>
input	<data/4e0a1f-fc0f/input/structure.pdb>
output	<data/4e0a1f-fc0f/output/blgc.pdb>

Query 7: **Retrieving the resources available in a WRO.** This query is responsible for identifying, given a WRO, which resources are aggregated by it. This query is useful to identify which resources to reuse, for example.

Table 7 shows the result of the query, which points out that the WRO aggregates the resources `script.sh`, `executable-workflow.t2flow`, `refined-workflow.t2flow`, and their inputs and output files (resources) aggregated in the WRO.

Listing 7: Query 7 translated into a SPARQL

```
select distinct ?resource ?type
where {
  <> ore:aggregates ?resource .
  ?resource a ?type .
  FILTER(ro:Resource != ?type)
}
```

Table 8
Result of evaluating query 7

resource	type
<workflow/executable-workflow.t2flow>	wf4ever:Workflow
<workflow/refined-workflow.t2flow>	wf4ever:Workflow
<files/script.sh>	wf4ever:Script
<workflowrun.prov.ttl>	wfdesc:WorkflowRun
<data/4e0a1f-fc0f/input/structure.pdb>	wf4ever:Dataset
<data/4e0a1f-fc0f/output/blgc.pdb>	wf4ever:Dataset

8. Related Work

Here, we provide a comparison of our work with research on script-to-workflow conversion and traceability. Additional related work was already commented on throughout the text.

Scripts are usually difficult to understand, reuse, or reproduce by people other than the original implementers. In previous work [14], we described a preliminary manual transformation from script to executable and verifiable workflow. We adopted YesWorkflow to generate the abstract workflow visualizations before creating the executable workflow. YesWorkflow was developed by McPhilips *et al.* [5, 26]; it is a script language-independent environment for extracting a workflow-like view that depicts the main components that compose a script and their data dependencies based on comments that annotate the script.

At the time of writing this paper, the YesWorkflow group is working towards supporting the exportation, in RDF, of an abstract workflow representation that is PROV-compliant, using the ProvONE model [29]. We, instead, use *wfdesc* and *wfprov* ontologies as PROV extensions because we are targeting Research Objects, and also the Taverna SWfMS. Moreover, our provenance information is used to link the abstract workflow back to the script.

There are a few other approaches to construct executable workflows from scripts. For instance, [10] uses the abstract syntax tree (AST) created from source code to map the script elements into workflow structures. Our approach differs from this in that we reuse parts of the script code to create the workflow activities.

The work of [11] migrates script-based experiments from a local High Performance Computer (HPC) cluster to workflows on a cloud computing infrastructure. Their requirements include traceability of the workflow results to meet reproducibility, one of their reasons to migrate to a SWfMS. One of the differences to our approach is that our methodology is more generic, in the sense that we do not focus on HPC scripts nor on cloud computing environments, which require specific kinds of scripts. Also, we do not consider any specific approach to meet the challenge of converting scripts with control-flow constructs into data-flow patterns, which is addressed in both [11] and [10].

In [12], the authors present an approach to convert electronic notebooks into workflows. Their approach goes directly from notebook code to executable workflow. It is based on a set of guidelines that recom-

mend changes to the notebook structure to facilitate the capture of the dataflow encoded in the notebook and enable its conversion into an executable workflow. We, instead, go from script-to-abstract and abstract-to-executable steps, thereby clearly separating abstract specification from code. This helps documentation, understandability and reuse. At the end, their conversion is performed by NiW, a tool that semi-automatically creates the workflow structure based on the notebook's code. Similar to our approach, after running their tool, a scientist may need to improve the corresponding workflow implementation due to differences in the environments. Our annotation and abstract workflows guide the scientists to identify the main processing units and dataflow in the script.

In [24], we presented the first implementation of a web prototype for W2Share that integrates the tools used to convert scripts into WRO. In that paper, we also address quality checking of the conversion process. However, we do not address tracking issues during the conversion process. This is a recent result that is being reported here.

The work of [30] presents an approach to track changes in workflows to capture the evolution of workflows and allow the comparison of results and structure between different versions. We, instead, focus on issues related to the traceability of the script-to-workflow conversion process, which enables relating workflow elements back to the original script blocks, comparing differences between script and workflow implementation, and comparing differences between script and workflow results.

Another difference between ours and other script-to-workflow approaches is that ours use of WROs. These objects bundle the executable workflow, but also the workflow specification and auxiliary resources, as well as workflow runs and data used in these runs. Thus, one single object (the WRO) is needed to support experiment reproducibility, reuse, and checking of experiments in a transparent way.

9. Conclusions and Ongoing Work

This paper is a step towards fully reproducible research. It presented W2Share, a computation framework that supports a (script-to-reproducible research) methodology. The methodology, implemented in W2Share via a suite of tools, guides scientists in a principled manner to transform scripts into reproducible and reusable research objects. W2Share addresses an important issue in the area of provenance of scientific experiments modeled as scripts – that of providing an executable and understandable provenance representation of domain script runs. We point out that provenance is not just metadata for others: "provenance-for-self" queries can be used by researchers to better understand experiments, and to speed up the conversion process. Thus, there is a need for support to hybrid provenance queries for scripts (i.e., involving both prospective and retrospective queries).

Our ontology-based approach to generate machine-readable abstract workflows is also useful for querying purposes (e.g., traceability). It also allows associating the executable workflow, represented using the wfdesc ontology, and provenance information, represented using the wfprov ontology, in ontology-based queries.

W2Share was elaborated based on requirements that we elicited given our experience and collaborations with scientists who use scripts in their simulations. Moreover, it enables traceability of the script-to-workflow process, thereby establishing trust in this process. The approach was showcased via a real world use case from Molecular Dynamics. We showed through competency questions that W2Share successfully meets those requirements. The competency questions and the case studies are additional contributions of our work. An initial implementation of our methodology is described in [24] and available at https://w3id.org/w2share.

Our ongoing and future work include promoting the use of the conversion process in an e-Science infrastructure, investigating further real use cases with the objective of extending it to fit (new) user requirements and other script environments. Also, we plan to evaluate the cost effectiveness of our proposal, in particular since in some cases it may require extensive involvement of scientists. Last but not least, we do not consider versioning. Thus, yet another future direction would be to provide support to such version control when refining executable workflows, for instance, by considering an Ontology of Research Object Evolution.

Acknowledgments

Work partially financed by Sao Paulo Science Foundation (FAPESP) under grants #2014/23861-4, #2017/03570-3, FAPESP/CEPID CCES under grant #2013/08293-7, and individual grants from CNPq. We thank Prof. Munir Skaf and his group from the Institute of

Chemistry at Unicamp for making their scripts and data available and for their valuable feedback in the molecular dynamics case study.

References

[1] E. Deelman, T. Peterka, I. Altintas, C.D. Carothers, K.K. van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer and J. Vetter, The future of scientific workflows, *The International Journal of High Performance Computing Applications (IJHPCA)* **32**(1) (2018), 159–175.

[2] S. Cohen-Boulakia, K. Belhajjame, O. Collin, J. Chopard, C. Froidevaux, A. Gaignard, K. Hinsen, P. Larmande, Y. Le Bras, F. Lemoine et al., Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities, *Future Generation Computer Systems* **75** (2017), 284–298.

[3] T. Kluyver, B. Ragan-Kelley, F. Pérez, B.E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J.B. Hamrick, J. Grout, S. Corlay et al., Jupyter Notebooks-a publishing format for reproducible computational workflows., in: *ELPUB*, 2016, pp. 87–90.

[4] F. Chirigati, R. Rampin, D.E. Shasha and J. Freire, ReproZip: Computational Reproducibility With Ease., in: *SIGMOD Conference*, ACM, 2016, pp. 2085–2088. ISBN 978-1-4503-3531-7.

[5] T. McPhillips, T. Song, T. Kolisnik, S. Aulenbach, K. Belhajjame, R.K. Bocinsky, Y. Cao, J. Cheney, F. Chirigati, S. Dey et al., YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts, *International Journal of Digital Curation* **10**(1) (2015), 298–313.

[6] L. Murta, V. Braganholo, F. Chirigati, D. Koop and J. Freire, noworkflow: Capturing and analyzing provenance of scripts, in: *Provenance and Annotation of Data and Processes*, Springer, 2014, pp. 71–83.

[7] S. Dey, K. Belhajjame, D. Koop, M. Raul and B. Ludäscher, Linking prospective and retrospective provenance in scripts, in: *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)*, 2015.

[8] J. Liu, E. Pacitti, P. Valduriez and M. Mattoso, A Survey of Data-Intensive Scientific Workflow Management, *Journal of Grid Computing* **13**(4) (2015), 457–493, ISSN 1572-9184.

[9] K. Belhajjame, O. Corcho, D. Garijo, J. Zhao, P. Missier, D. Newman, R. Palma, S. Bechhofer, E. García Cuesta, J.M. Gómez-Pérez et al., Workflow-Centric Research Objects: First Class Citizens in Scholarly Discourse, in: *Proceedings of Workshop on the Semantic Publishing, (SePublica 2012)*, 2012.

[10] M. Baranowski, A. Belloum, M. Bubak and M. Malawski, Constructing workflows from script applications, *Scientific Programming* **20**(4) (2012), 359–377.

[11] J. Cala, Y. Xu, E.A. Wijaya and P. Missier, From scripted HPC-based NGS pipelines to workflows on the cloud, in: *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, IEEE, 2014, pp. 694–700.

[12] L.A.M.C. Carvalho, R. Wang, D. Garijo and Y. Gil, NiW: Converting Notebooks into Workflows to Capture Dataflow and Provenance, in: *2017 Workshop on Capturing Scientific Knowledge (SciKnow), held in conjunction with the ACM International Conference on Knowledge Capture (K-CAP), December 4-6*, Austin, TX, USA, 2017, pp. 1–8.

[13] J. Freire, D. Koop, E. Santos and C.T. Silva, Provenance for computational tasks: A survey, *Computing in Science & Engineering* **10**(3) (2008), 11–21.

[14] L.A.M.C. Carvalho, K. Belhajjame and C.B. Medeiros, Converting Scripts into Reproducible Workflow Research Objects, in: *Proc. of the IEEE 12th Int. Conf. on eScience, October 23-26*, IEEE, Baltimore, MD, USA, 2016, pp. 71–80.

[15] T. Lebo, S. Sahoo, D. McGuinness, K. Belhajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik and J. Zhao, PROV-O: the PROV ontology. W3C Recommendation, *World Wide Web Consortium* (2013). https://www.w3.org/TR/prov-o/.

[16] K. Belhajjame, J. Zhao, D. Garijo, M. Gamble, K. Hettne, R. Palma, E. Mina, O. Corcho, J.M. Gómez-Pérez, S. Bechhofer et al., Using a suite of ontologies for preserving workflow-centric research objects, *Web Semantics: Science, Services and Agents on the World Wide Web* **32** (2015), 16–42.

[17] P. Missier, S. Woodman, H. Hiden and P. Watson, Provenance and data differencing for workflow reproducibility analysis, *Concurrency and Computation: Practice and Experience* **28**(4) (2016), 995–1015, ISSN 1532-0634.

[18] R. Souza, V. Silva, A.L. Coutinho, P. Valduriez and M. Mattoso, Data reduction in scientific workflows using provenance monitoring and user steering, *Future Generation Computer Systems* (2017). doi:10.1016/j.future.2017.11.028.

[19] M. Mattoso, J. Dias, K.A. Ocaña, E. Ogasawara, F. Costa, F. Horta, V. Silva and D. de Oliveira, Dynamic steering of HPC scientific workflows: A survey, *Future Generation Computer Systems* **46** (2015), 100–113.

[20] L.A.M.C. Carvalho, B.T. Essawy, D. Garijo, C.B. Medeiros and Y. Gil, Requirements for Supporting the Iterative Exploration of Scientific Workflow Variants, in: *2017 Workshop on Capturing Scientific Knowledge (SciKnow), held in conjunction with the ACM International Conference on Knowledge Capture (K-CAP)*, 2017, pp. 1–8.

[21] L.A.M.C. Carvalho, D. Garijo, C.B. Medeiros and Y. Gil, Semantic Software Metadata for Workflow Exploration and Evolution, in: *Proc. of the IEEE 14th Int. Conf. on eScience, Oct 29-Nov 1*, IEEE, Amsterdam, Netherlands, 2018.

[22] R.L. Silveira and M.S. Skaf, Molecular dynamics simulations of family 7 cellobiohydrolase mutants aimed at reducing product inhibition, *The Journal of Physical Chemistry B* **119**(29) (2014), 9295–9303.

[23] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M.P. Balcazar Vargas, S. Sufi and C. Goble, The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud, *Nucleic Acids Research* **41**(W1) (2013), 557–561.

[24] L.A.M.C. Carvalho, J.E.G. Malaverri and C.B. Medeiros, Implementing W2Share: Supporting Reproducibility and Quality Assessment in eScience, in: *Proc. of the 11th Brazilian e-Science Workshop (BreSci), July 5-6, 2017*, Brazilian Computer Society, Sao Paulo, Brazil, 2017.

[25] L.A.M.C. Carvalho and C.B. Medeiros, W2Share Case Study: Workflow Research Object (WRO), 2018. doi:10.5281/zenodo.1465897.

[26] T. McPhillips, S. Bowers, K. Belhajjame and B. Ludäscher, Retrospective provenance without a runtime provenance recorder, in: *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)*, 2015.

[27] Y. Tzitzikas, N. Minadakis, Y. Marketakis, P. Fafalios, C. Allocca, M. Mountantonakis and I. Zidianaki, MatWare: Constructing and Exploiting Domain Specific Warehouses by Aggregating Semantic Data, in: *The Semantic Web: Trends and Challenges: 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings*, V. Presutti, C. d'Amato, F. Gandon, M. d'Aquin, S. Staab and A. Tordai, eds, Springer International Publishing, 2014, pp. 721–736.

[28] L.A.M.C. Carvalho and C.B. Medeiros, W2Share Evalution: Competency Questions, 2018. doi:10.5281/zenodo.1465893.

[29] V. Cuevas-Vicenttín, B. Ludäscher, P. Missier, K. Belhajjame, F. Chirigati, Y. Wei and B. Leinfelder, Provone: A prov extension data model for scientific workflow provenance, 2015. http://purl.org/provone.

[30] J. Freire, C.T. Silva, S.P. Callahan, E. Santos, C.E. Scheidegger and H.T. Vo, Managing Rapidly-Evolving Scientific Workflows, in: *Provenance and Annotation of Data*, L. Moreau and I. Foster, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 10–18. ISBN 978-3-540-46303-0.

Appendix A. Molecular Dynamics Case Study

Resources of the case study from Molecular Dynamics.

Listing A.1: Excerpt of an annotated MD script using YesWorkflow tags.

```
#!/bin/bash

# @BEGIN setup @DESC setup of a MD simulation
# @PARAM directory_path @AS directory
# @IN initial_structure @DESC PDB: 8CEL
#    @URI file:{directory}/structure.pdb
# @IN topology_prot
#    @URI file:top_all22_prot.rtf
# @IN topology_carb
#    @URI file:top_all22_prot.rtf
# @OUT gh5_psf @AS final_structure_psf
#    @URI file:{directory}/gh5.psf
# @OUT gh5_pdb @AS final_structure_pdb
#    @URI file:{directory}/gh5.pdb

# @BEGIN split
# @IN initial_structure
#     @URI file:structure.pdb
# @IN directory_path @AS directory
# @OUT protein_pdb
#     @URI file:{directory}/protein.pdb
# @OUT bglc_pdb
#     @URI file:{directory}/bglc.pdb
# @OUT water_pdb
#     @URI file:{directory}/water.pdb
structure = $directory_path"/structure.pdb"
protein = $directory_path"/protein.pdb"
water = $directory_path"/water.pdb"
bglc = $directory_path"/bglc.pdb"
egrep -v '(TIP3|BGLC)' $structure > $protein
grep TIP3  $structure > $water
grep BGLC  $structure > $bglc
# @END split

# @BEGIN psfgen @DESC generate the PSF file
# @PARAM topology_prot
#     @URI file:top_all22_prot.rtf
# @PARAM topology_carb
#     @URI file:top_all36_carb.rtf
# @IN protein_pdb
#     @URI file:protein.pdb
# @IN bglc_pdb
#     @URI file:bglc.pdb
# @IN water_pdb
#     @URI file:water.pdb
# @OUT hyd_pdb
#     @URI file:hyd.pdb
# @OUT hyd_psf
#     @URI file:hyd.psf

... commands ...

# @END psfgen

# @BEGIN solvate
# @IN hyd_pdb
#     @URI file:hyd.pdb
# @IN hyd_psf
#     @URI file:hyd.psf
# @OUT wbox_pdb
#     @URI file:wbox.pdb
# @OUT wbox_psf
#     @URI file:wbox.psf
echo "
package require solvate
solvate hyd.psf hyd.pdb -rotate -t 16 -o wbox
exit
" > solv.tcl

vmd -dispdev text -e solv.tcl
rm solv.tcl
# @END solvate

# @BEGIN ionize
# @IN wbox_pdb
#     @URI file:wbox.pdb
# @IN wbox_psf
#     @URI file:wbox.psf
# @OUT gh5_pdb @AS final_structure_pdb
#     @URI file:gh5.pdb
# @OUT gh5_psf @AS final_structure_psf
#     @URI file:gh5.psf

... commands ...

# @END ionize
```

```
# @END setup
```

Listing A.2: Excerpt of specification of *Wa*, the result of transforming script *S* into an equivalent machine-readable abstract workflow.

```
@base <https://w3id.org/w2share/wro/md-setup/abs-workflow/Setup_MD/>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix wf4ever: <http://purl.org/wf4ever/wf4ever#>.
@prefix oa:      <http://www.w3.org/ns/oa#>.
@prefix wfdesc: <http://purl.org/w4ever/wfdesc#>.
@prefix prov: <http://www.w3.org/ns/prov-o#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

<>
  a wfdesc:Workflow, prov:Entity;
  rdfs:label "setup"^^xsd:string;
  wfdesc:hasSubProcess <processor/split/>;
  wfdesc:hasInput       <in/structure_pdb>;
  wfdesc:hasOutput      <out/fixed_1_pdb> .

<in/initial_structure>
  a wfdesc:Input, wfdesc:Output;
  rdfs:label    "structure_pdb"^^xsd:string;
  dcterms:title "crystal structure of the protein"^^xsd:string .

<out/fixed_1_pdb>
  a wfdesc:Output, wfdesc:Input;
  rdfs:label    "fixed_1"^^xsd:string;
  dcterms:title "coordinates for the whole system (cbh1.pdb), indicating which atoms should be kept fixed along the simulation"^^xsd:string .

<datalink?from=in/initial_structure&to=processor/split/in/initial_structure>
  a wfdesc:DataLink;
  wfdesc:hasSource  <in/initial_structure>;
  wfdesc:hasSink    <processor/split/in/initial_structure> .

<processor/split/>
  a wfdesc:Process;
  rdfs:label "split"^^xsd:string;
  wfdesc:hasInput  <processor/split/in/initial_structure>;
  wfdesc:hasOutput <processor/split/out/cbh1_pdb> .

<processor/split/in/initial_structure>
  a wfdesc:Input;
  rdfs:label "structure_pdb"^^xsd:string;
  dcterms:description "crystal structure of the protein"^^xsd:string .

<processor/split/out/cbh1_pdb>
  a wfdesc:Output;
  rdfs:label          "cbh1_pdb"^^xsd:string;
  dcterms:description "coordinates of the protein atoms"^^xsd:string .
```

Listing A.3: Excerpt of PROV-statements describing the derivation of *S* to *Wa* to *We* to We_1.

```
@base <https://w3id.org/w2share/wro/md-setup/>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix wf4ever: <http://purl.org/wf4ever/wf4ever#>.
@prefix oa:      <http://www.w3.org/ns/oa#>.
@prefix wfdesc: <http://purl.org/w4ever/wfdesc#>.
@prefix prov: <http://www.w3.org/ns/prov-o#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.


<files/Setup_MD/script.sh> a wf4ever:Script.

<abs-workflow/Setup_MD/>
   prov:wasDerivedFrom <files/Setup_MD/script.sh> ;
   prov:wasAttributedTo [
       a prov:Agent ;
       foaf:name "Lucas Carvalho" ] .

<abs-workflow/Setup_MD/processor/split/>
  prov:wasDerivedFrom [
    a prov:Entity, oa:TextPositionSelector;
    oa:start "1644"^^xsd:Integer;
    oa:end "1786"^^xsd:Integer;
  ] .

<workflow/Setup_MD/> a wfdesc:Workflow,
   prov:Entity ;
   prov:wasDerivedFrom <files/Setup_MD/script.sh> ;
   prov:wasDerivedFrom <abs-workflow/Setup_MD/> ;
   wfdesc:hasSubProcess <workflow/Setup_MD/processor/split> .
```

```
<workflow/Setup_MD/processor/split/>
    prov:wasDerivedFrom
    <abs-workflow/Setup_MD/processor/split/>
    .

<workflow/Setup_MD/variant> a
    wfdesc:Workflow, prov:Entity ;
    prov:wasDerivedFrom <workflow/Setup_MD/>
    .
```

Listing A.4: Excerpt of workflow execution traces of *We*.

```
@base
    <https://w3id.org/w2share/wro/md-setup/>
    .
@prefix prov:  <http://www.w3.org/ns/prov#> .
@prefix wfprov:
    <http://purl.org/wf4ever/wfprov#> .
@prefix rdfs:
    <http://www.w3.org/2000/01/rdf-schema#>
    .
@prefix wfdesc:
    <http://purl.org/wf4ever/wfdesc#> .
@prefix tavernaprov:
    <http://ns.taverna.org.uk/2012/
    tavernaprov/> .
@prefix owl:
    <http://www.w3.org/2002/07/owl#> .
@prefix xsd:
    <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:
    <http://www.w3.org/1999/02/22-rdf-
    syntax-ns#> .
@prefix dct:   <http://purl.org/dc/terms/> .

<run/e0fa2f25-0755/>
    rdf:type wfprov:WorkflowRun ;
    dct:hasPart
    <run/e0fa2f25-0755/process/f0a0bd65-78d3/>
    ;
    wfprov:describedByWorkflow
    <workflow/Setup_MD/> ;
    prov:used
    <data/5c65c151-0333/ref/61f8795e-e650> ;
    dct:hasPart
    <run/e0fa2f25-0755/process/c06ff05e-eceb/>
    ;
    prov:endedAtTime
    "2016-06-16T11:25:24.549-03:00"
    ^^xsd:dateTime ;
    prov:startedAtTime
    "2016-06-16T11:25:12.838-03:00"
    ^^xsd:dateTime ;
    wfprov:usedInput
    <data/5c65c151-0333/ref/61f8795e-e650>;

<data/5c65c151-0333/ref/61f8795e-e650>
    tavernaprov:content
    <data/4e0baa1f-fc0f/input/
    structure.pdb> ;
    wfprov:describedByParameter
    <workflow/Setup_MD/processor/split/in/
    initial_structure> ;
    wfprov:describedByParameter
    <workflow/Setup_MD/processor/in/
    initial_structure> ;
    prov:wasGeneratedBy
    <run/4e0baa1f-fc0f/process/c3a0e8c0-dcb0/>
    ;
    rdf:type wfprov:Artifact ;
    rdf:type prov:Entity .

<data/e0fa2f25-0755/ref/55269975-380f>
    tavernaprov:content
    <data/4e0baa1f-fc0f/output/bglc.pdb> ;
    wfprov:describedByParameter
    <workflow/Setup_MD/processor/psgen/in/
    bglc_pdb> ;
    wfprov:describedByParameter
    <workflow/Setup_MD/processor/split/out/
    bglc_pdb> ;
    wfprov:wasOutputFrom
    <run/4e0baa1f-fc0f/process/c3a0e8c0-dcb0/>
    ;
    prov:wasGeneratedBy
    <run/4e0baa1f-fc0f/process/c3a0e8c0-dcb0/>
    ;
    rdf:type wfprov:Artifact ;
    rdf:type prov:Entity .

<run/4e0baa1f-fc0f/process/c3a0e8c0-dcb0/>
    wfprov:describedByProcess
    <workflow/Setup_MD/processor/split/> ;
    wfprov:usedInput
    <data/5c65c151-0333/ref/61f8795e-e650> ;
    prov:wasAssociatedWith <#taverna-engine>
    ;
    rdf:type wfprov:ProcessRun ;
    prov:endedAtTime
    "2017-03-10T08:19:32.405-03:00"
    ^^xsd:dateTime ;
    prov:startedAtTime
    "2017-03-10T08:19:31.075-03:00"
    ^^xsd:dateTime ;
    prov:used
    <data/5c65c151-0333/ref/61f8795e-e650> ;
    wfprov:wasPartOfWorkflowRun
    <run/e0fa2f25-0755/> .
```

Listing A.5: Excerpt of the WRO manifest.

```
@base
    <https://w3id.org/w2share/wro/md-setup/>.
@prefix ro: <http://purl.org/wf4ever/ro#> .
```

```
@prefix ore:
  ↪ <http://www.openarchives.org/ore/terms/>
  ↪ .
@prefix wf4ever:
  ↪ <http://purl.org/wf4ever/wf4ever#> .

<files/Setup_MD/script.sh> a ro:Resource,
  ↪ wf4ever:Script .
<workflow/executable-workflow.t2flow> a
  ↪ ro:Resource, wf4ever:Workflow .
<workflow/refined-workflow.t2flow> a
  ↪ ro:Resource, wf4ever:Workflow .
<data/4e0a1f-fc0f/input/structure.pdb> a
  ↪ ro:Resource, wf4ever:Dataset .
<data/4e0a1f-fc0f/output/bglc.pdb> a
  ↪ ro:Resource, wf4ever:Dataset .

<> a ro:ResearchObject ;
   ore:aggregates
     <files/Setup_MD/script.sh>,
     <workflow/executable-workflow.t2flow>,
     <workflow/refined-workflow.t2flow>,
     <data/4e0a1f-fc0f/input/structure.pdb>,
     <data/4e0a1f-fc0f/output/bglc.pdb> .
```