Converting Scripts into Reproducible Workflow Research Objects

Lucas A. M. C. Carvalho
Institute of Computing
University of Campinas
Campinas, Brazil
Email: lucas.carvalho@ic.unicamp.br

Khalid Belhajjame
LAMSADE
Paris-Dauphine University
Paris, France
Email: Khalid.Belhajjame@dauphine.fr

Claudia Bauzer Medeiros
Institute of Computing
University of Campinas
Campinas, Brazil
Email: cmbm@ic.unicamp.br

Abstract—**Scientific discovery and analysis are increasingly computational and data-driven. While scripting languages, such as Python, R and Perl, are the means of choice of the majority of scientists to encode and run their data analysis, scripts are generally not amenable to reuse or reproducibility. Scripts do rarely get reused or even shared with third party scientists. We argue in this paper that the reproducibility of scripts can be promoted by converting them into *workflow research objects*. A workflow research object encodes a script into a production (executable) workflow that is accompanied by annotations, example datasets and provenance traces of their execution, thereby allowing third party users to understand the data analysis encoded by the original script, run the associated workflow using the same or different dataset, or even repurpose it for a different analysis. To this end, we present a methodology for converting scripts into workflow research objects in a principled manner, guided by requirements that we elicited for this purpose. The methodology exploits tools and standards that have been developed by the community, in particular YesWorkflow, Research Objects and the W3C PROV. It is showcased using a real world use case from the field of Molecular Dynamics.**

I. Introduction

Scripting languages have gained momentum among scientists as a means for enacting computational data analysis. Scientists in a number of disciplines use scripts written in general-purpose languages such as Python, R and Perl in their daily data analysis and experiments. We note, however, that scripts are difficult to understand by third parties who were not involved in their development (and sometimes even by the same scientists who developed them); they are, as such, not amenable to reuse and reproducibility. This is witnessed by a number of initiatives that were launched to bring some of the rich features that traditionally come with scientific workflow systems to manage scripts, see e.g., [1]–[4]. For example, McPhilips *et al.* [1] developed YesWorkflow, an environment for extracting a workflow-like graph that depicts the main components that compose a script and their data dependencies based on comments that annotate the script. Murta *et al.* [3] proposed noWorkflow, which also captures provenance traces of script executions.

While the above proposals bring new useful functionalities for understanding scripts and their execution traces they do not enable reuse or reproducibility of scripts. 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 [2].

To address the above issues and complement the landscape of solutions proposed by the community for promoting the reuse and reproducibility of scripts, we present in this paper a methodology for converting scripts into reproducible Workflow Research Objects [5] (WRO). WRO are Research Objects that encapsulate scientific workflows and additional information regarding the context and resources consumed or produced during the execution. In more detail, given a script, the methodology we propose drives the creation 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. These WROs also encapsulate provenance traces of the execution of those workflows, as well as auxiliary resources that help third party users to understand and reproduce such analyses. Examples of those resources include annotations that describe workflow steps, the hypotheses investigated by the data analyses that the scripts incarnate and the findings that the scientists may have made. We argue that such Workflow Research Objects provide potential users with the means to understand, replicate and reuse the data analyses carried by the scripts or part thereof – thanks to the executable workflows that embody such analyses and the accompanying auxiliary resources.

While developing the methodology, we strived to exploit tools and standards that were developed by the scientific community, in particular, YesWorkflow, Research Objects, the W3C PROV recommendations [1] as well as the Web Annotation Data Model [2]. To showcase our methodology, we use a real-world case study from the field of Molecular Dynamics.

The paper is organized as follows. Section II presents the case study that we use as a running example throughout the paper. Section III identifies the requirements that guided the development of our methodology, which is overviewed in

[1]https://www.w3.org/TR/prov-overview/
[2]https://www.w3.org/TR/annotation-model

section IV. Sections V through VIII show in detail each step of our methodology. Section IX briefly discusses related work. Finally, Section X concludes the paper underlining our main contributions and discussing our ongoing work.

Throughout this paper, we differentiate between at least two kinds of experts – $scientists$ and $curators$. Scientists are the domain experts who understand the experiment, and the script; this paper also calls them, sometimes, $users$. Curators may be scientists who are also familiar with workflow and script programming, or, alternatively, computer scientists who are familiar enough with the domain to be able to implement our methodology. Curators are moreover responsible for authoring, documenting and publishing workflows and associated resources.

II. Case Study - Molecular Dynamics

The motions of individual atoms in a multimolecular physical system can be determined if the forces acting on every atom are known; these forces may result from their mutual interactions or from the action of an external perturbation. Determining such motions is key to understanding the physical and chemical properties of a given system of interest.

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, a suite of scripts was designed by physiochemists [6]; its inputs are the protein structure (obtained from the RCSB PDB protein data bank[3]), 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 (biophysical chemistry) 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.

The main raw product of any MD simulation is a large set of interrelated molecular trajectories. Trajectory data is usually stored for subsequent analyses and consists of thousands of time-series of the Cartesian coordinates of every atom of the system.

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

In this paper, we will use a script that sets up a MD simulation. The script will be presented later on (see Listing 1), and used as a running example throughout the paper.

III. Requirements for Script Conversions

This section presents the requirements that guided the development of our solution, and section IV outlines the methodology we designed to meet them.

Since scripts are usually fine-grained, they are hard to understand - sometimes even the script author does not understand a script s/he developed in the past. To facilitate the task of understanding the script, its author may modularize the script by organizing it into functions. While modularity helps, the functions that compose the script are obtained through a refactoring process that primarily aims to promote code *reuse via the reuse script*, as opposed to *reuse via the main (logical) data processing units* that are relevant from the point of view of the computational analysis implemented by the script. This leads us to the first requirement.

Requirement 1: To help the scientist understand a script S, s/he needs a view of S that identifies 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 processing units.

The idea, here, is to provide curators with automatic means to obtain a workflow-like view of the script, i.e., an abstract workflow revealing the computational processes and data flows otherwise implicit in these scripts, displaying modules, ports, and data links. Though graphical visualizations may be useful to promote understandability of scripts, large scripts may result in very large (abstract) workflows. Thus, curators need to be able to create a multi-level view of scripts (e.g., through encapsulation of sub-workflows into more complex abstract tasks), or to pose queries against this workflow-like view.

An abstract workflow is a preliminary requirement to our end-goal, namely, to provide curators with the means to generate a (concrete) workflow that can be executed using a workflow management system. This, in turn, will bring to the scientists benefits that such systems provide, such as retrospective provenance recording.

Requirement 2: The user 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 the 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. In many cases, results will not be exactly the same, but similar. This also happens with script execution, in which two successive runs with identical inputs will produce non-identical results that are nevertheless valid and compatible. Thus, this requirement involves providing means of comparing the execution of script S and the workflow, and validating the workflow as a valid embodiment of the script.

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

Not only may a scientist want to 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 – e.g., which implement some activities in the workflow via alternative algorithms and/or different and potentially larger datasets. For example, s/he may want to modify a method call in a bioinformatics script that performs sequence alignment with a call to an EBI[4] web service that performs a sophisticated sequence alignment using larger and curated proteomic datasets. By the same token, in a Molecular Dynamics simulation, a protein data source may be modified.

The new (modified) workflow(s) correspond to *v*ersions 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 versions (e.g., if web services are invoked instead of a local code implementation).

Experiment reusability demands that the appropriate (pieces of) workflow be identified and selected for reuse. It is not enough to publish these pieces: potential users must be given enough information to understand how the workflow came to be, and how adequate it is to the intended uses. This leads us to Requirements 4 and 5, respectively involving the need for provenance information, and the elements that should be bundled together to ensure full reusability,

Requirement 4: Provenance information should be recorded.

This involves not only the provenance obtained by workflow execution. This requirement also implies recording the transformations carried out to transform the script into a workflow that embodies the script. Moreover, the transformations to workflows that modify the initial workflow using different resources also need to be recorded. As stressed by [7], provenance that is provided by the execution of a workflow corresponds to a workflow trace, which can be processed as an acyclic digraph, in which nodes are activities and/or data, and edges denote relationships among activities and data.

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

We follow the definition of [7]: ”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.” [7] also differentiates 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. The script S, the initial workflow, and all of its versions should be made available together with auxiliary resources that will allow understanding how these workflows came to be, and where they should be used. Such resources must include, at least, the provenance information documenting the transformation from the script to the workflows, datasets that are used as inputs, execution traces of the script and the workflows, as well as textual annotations provided by the curator.

[4]http://www.ebi.ac.uk

IV. Methodology to Assist in Script Conversions

To meet the requirements identified in Section III, we devised a methodology for converting a script into reproducible workflow research objects [5], [8]. As the use of workflow specifications on their own does not guarantee support to reusability, shareability, reproducibility, or better understanding of scientific methods, additional information may be needed. This includes annotations to describe the operations performed by the workflow; links to other resources, such as the provenance of the results obtained by executing the workflow, datasets used as input, etc. These richly annotation objects are called workflow-centric research objects [5]. A Research Object [9] provides the means to specify a kind of container that gathers resources of different types and provides a digital analogue of the ’Materials and Methods’ section of a research paper. Workflow Research Objects [5] (WRO) are a specific kind of Research Objects that can be viewed as an aggregation of resources that bundles a workflow specification and additional information to preserve the workflows and their context. Workflow research objects can be used by third parties to understand and run an experiment using the same data inputs used in the original script as well as different ones of her/his choosing.

The methodology is depicted in Figure 1, in which each step corresponds to one requirement. It is composed of five inter-related steps. Given a script S, the first step **Generate abstract workflow** is used to extract from the script an abstract workflow W_a identifying the main processing steps that constitute the data analysis implemented by the script, and their data dependencies. The workflow W_a obtained as a result in Step 1 is abstract in the sense that it cannot be executed.

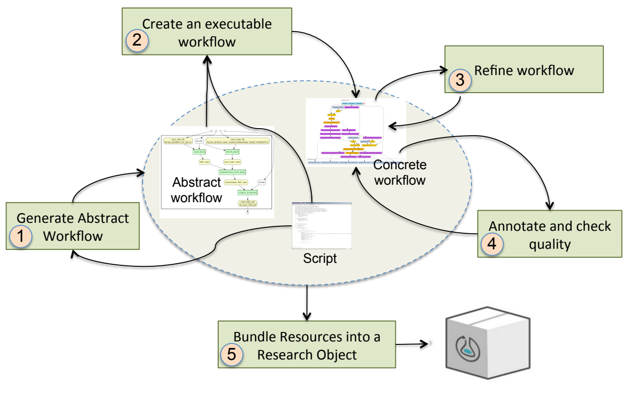

Create an executable workflow
2
Refine workflow
3
Abstract workflow
Concrete workflow
Generate Abstract Workflow
1
Annotate and check quality
4
Script
Bundle Resources into a Research Object
5

Fig. 1. Methodology for converting scripts into reproducible Workflow Research Objects.

Given this abstract workflow, the second step **Create an executable workflow** converts the abstract workflow into an executable one W_e by identifying, for each processing step in the abstract workflow, the realization that can be used for its implementation. The executable workflow obtained in step 2 is then **refined** in Step 3 by identifying appropriate third party datasets or processing steps that can, amongst other things, yield better results, generating workflow versions $W_{e1} \ldots W_{en}$. For example, the curator may prefer to use human annotated datasets than raw datasets with unknown lineage. In order to help potential users understand the workflow, the curator provides **annotations** describing the workflow, and potentially the resources it utilizes. The curator may also provide examples of provenance traces that have been obtained as a result of the workflow execution. As well as annotating the workflow, the curator should **run a series of checks** to verify the soundness of the workflow. Once tried and tested, the workflow and the auxiliary resources, i.e., annotations, provenance traces, examples of datasets that can be utilized as well as the original script, are **packaged into a Workflow Research Object** (for short, WRO) – see section VIII.

We next present the aforementioned steps in detail.

V. Generating an Abstract Workflow

The objective of this phase is to address Requirement 1 by generating an abstract workflow, W_a, given the script S. The generation of W_a entails the analysis of S to identify the main processing units, and their dependencies, that are relevant from the point of view of the scientists, as opposed to a programmer. To do so, we adopt the YesWorkflow tool [1], [4]. It enables scientists to annotate existing scripts with special comments that reveal the computational modules and data flows otherwise 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. To the best of our knowledge, YesWorkflow is the only tool that allow generating a graphical representation of a script as a workflow. It does so by processing curator-provided tags of the form @*tag value*, where @*tag* is a keyword that is recognized by YesWorkflow, and *value* is an optional value assigned to the tag.

We illustrate the semantics of the tags using Listing 1, which is an excerpt of a script of our use case (see section II) annotated with YesWorkflow tags. We point out that this excerpt shows only annotations, having eliminated most of the code. For the complete code, see [5], the final WRO. The tags @*begin* and @*end* are used to delimit the activities of the workflow, or the workflow itself. The @*begin* tag is followed by a *name* that identifies the activity in question within the workflow. For example, Listing 1 shows that the overall workflow, named *setup*, is composed of four activities: *split*, *psfgen*, *solvate*, and *ionize*. Those activities represent different parts of the script. For example, activity *split* corresponds to the code in the script delimited by lines 14 and 27.

The curator can annotate an activity with its description using the @*desc* tag. An activity may be characterized by a set of input and output ports, defined using the tags @*in* and @*out*, respectively. For example, activity *split* has one input port named *initial_structure* and three output ports named *protein_pdb*, *bglc_pdb* and *water_pdb*. Note that the names of script variables may not be self explanatory. The curator can associate the input and ouput ports with more meaningful names using the tag @*as*, which creates an alias names. For example, the output port *gh5_psf* of the *setup* activity (the whole workflow) is associated with the alias *final_structure_psf*. A script may retrieve or store the results used by an input port and generate an output port in a file during the execution. The @*uri* tag is used in such cases to specify the path of the file within the file system. For example, the output port *protein_pdb* is associated with the URI *protein.pdb*, representing the file where the *split* activity will store the content of the *protein_pdb* output port during the execution.

Just like activities, input and output ports can be annotated with text using the @*desc* tag. For example, the input port *initial_structure* has a description in line 5. The data dependencies connecting the activities in the workflow are inferred by matching the names of the input and output ports. A data link connecting an output port to an input port is constructed if those ports are associated with the same variable in the script.

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

```
1  #!/bin/bash
2
3  # @BEGIN setup @DESC setup of a MD simulation
4  # @PARAM directory_path @AS directory
5  # @IN initial_structure @DESC PDB: 8CEL
6  #   @URI file:{directory}/structure.pdb
7  # @IN topology_prot @URI file:top_all22_prot.rtf
8  # @IN topology_carb @URI file:top_all22_prot.rtf
9  # @OUT gh5_psf @AS final_structure_psf
10 #   @URI file:{directory}/gh5.psf
11 # @OUT gh5_pdb @AS final_structure_pdb
12 #   @URI file:{directory}/gh5.pdb
13
14 # @BEGIN split
15 # @IN initial_structure @URI file:structure.pdb
16 # @IN directory_path @AS directory
17 # @OUT protein_pdb  @URI file:{directory}/protein.pdb
18 # @OUT bglc_pdb  @URI file:{directory}/bglc.pdb
19 # @OUT water_pdb  @URI file:{directory}/water.pdb
20 structure = $directory_path"/structure.pdb"
21 protein = $directory_path"/protein.pdb"
22 water = $directory_path"/water.pdb"
23 bglc = $directory_path"/bglc.pdb"
24 egrep –v '(TIP3|BGLC)' $structure > $protein
25 grep TIP3 $structure > $water
26 grep BGLC $structure > $bglc
27 # @END split
28
29 # @BEGIN psfgen @DESC generate the PSF file
30 # @PARAM topology_prot @URI file:top_all22_prot.rtf
31 # @PARAM topology_carb @URI file:top_all36_carb.rtf
```

[5]http://w3id.org/w2share/s2rwro/

```
# @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
```

Once the script is annotated, YesWorkflow generates an abstract workflow representation. Figure 2 depicts the abstract workflow generated given the tags provided in Listing 1. It is merely a graphical representation of the script.

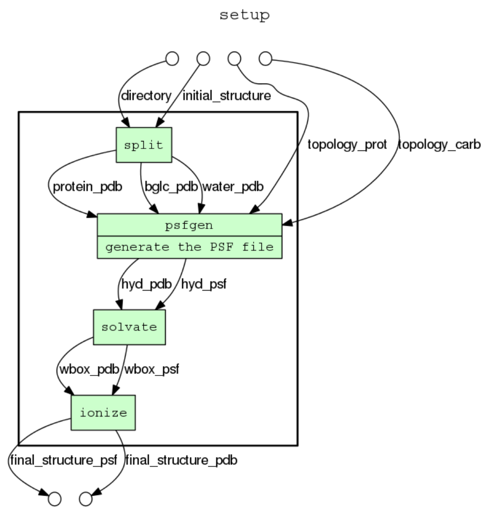


Fig. 2. Abstract workflow representation generated via YesWorkflow.

Tag recognition is script-language independent, therefore allowing a wide range of script-based experiments to be converted into workflows and consequently a wider adoption of our methodology. The abstract workflow representation is also platform independent. It will be transformed into a platform-specific executable representation in the next step of our methodology.

Furthermore, especially for large, hard-to-read, workflows, we can take advantage of some of the facilities offered by YesWorkflow to help scientists understand a workflow. In particular, YesWorkflow's implementation generates a Datalog file whose facts are constructed from the script tags and follow YesWorkflow's model (e.g., defining that a script is composed of program blocks, ports and channels; or that channels connect ports). We can thus pose Datalog queries against this file to reveal data flow and dependencies within a script. Such queries, as mentioned in [1], can, for instance, allow the user to list the activities defined in the script and their descriptions (when provided by the curator) or the activities that invoke a particular module or external program.

It is worth stressing that the curator needs to respect two constraints when using YesWorkflow in our context. The first constraint concerns **appropriate identification of all processing blocks**. Indeed, YesWorkflow extracts a workflow by processing curator-provided tags; but scientists may not always consider a given piece of script as relevant for tag processing - and thus YesWorkflow will not produce an ”appropriate” W_a. However, we are not merely trying to extract an abstract workflow, but to ultimately create an executable workflow that reflects the original script. Thus, the curator annotating the script needs to correctly identify the program blocks that cover the script in its entirety. In others words, taken together, the program blocks that are identified and annotated by the curator need to cover all of the original script.

The second constraint concerns **appropriately tagging all inputs and outputs of each processing block**. In other words, when using YesWorkflow in our context, for each program block identified by the curator, the input and output ports identified and annotated by the curator for that block need to cover all of the inputs needed by that block to be executed, as well as the outputs generated by that block as a result of the execution. Again, in the general case, YesWorkflow does not compel the curator to annotate all the inputs and outputs that are respectively needed and generated by a program block. However, since we are aiming for the creation of an executable workflow, this second constraint needs also to be met.

VI. Creating an Executable Workflow from the Abstract Workflow

Given the abstract workflow W_a generated previously, the curator needs to create an executable workflow W_e that embodies the data analysis and processes as depicted by W_a – and thus embodies the original script (Requirement 2). Subsequently, the curator may choose to use resources, i.e. datasets and operations, that are different from those used in the script as s/he sees fit (Requirement 3). Moreover, provenance information identifying how the executable workflow came to be and its relationship with the script need to be recorded (Requirement 4).

A. Step 2: Creating an Initial Executable Workflow

To create the executable workflow W_e, the curator 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 W_a, the corresponding code in W_e is generated by reusing the chunk (block) of the script that is associated with the abstract workflow activity.

For example, the $split$ abstract activity can be implemented by copying the code from the script between the corresponding $@begin$ and $@end$ tags (see lines 20 to 26 in Listing 1); the same would apply to the $solvate$ abstract activity (see lines 47 to 54 in Listing 1).

In the implementation of the activity, its input and output ports will be associated with the names of the input and output ports in the abstract workflow. However, they may be different from the corresponding variable names in the script. Therefore it is necessary to check consistency and, when required, change the implementation of the activity, so that the names of the variables are coherent with the port names. To do so, the curator replaces the variable names in the script code with the name used in the tag $@as$, when defined. This step can be performed in largely automatic fashion. Consider the implementation of the $split$ activity, where the $split$ program block have a $@in$ tag and an alias name defined via $@as$. In this implementation, the name of the variable $directory\_path$ is modified to be $directory$, the name defined via $@as$ (see Listing 2).

Listing 2. Script code - correcting variable name in the implementation of the $split$ abstract activity

```
structure = %%directory%%"/structure.pdb"
protein = %%directory%%"/protein.pdb"
water = %%directory%%"/water.pdb"
bglc = %%directory%%"/bglc.pdb"
egrep −v '(TIP3|BGLC)' $structure > $protein
grep TIP3 $structure > $water
grep BGLC $structure > $bglc
```

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, and (ii) the ability to check and debug the execution of W_e against the script execution, to correct eventual mistakes in script-to-workflow conversion.

Once the curator specifies the implementation of each activity in W_a, a concrete workflow specification W_e that is conform to a given scientific workflow system can be created. Without loss of generality, we used the Taverna system [10], 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 script language adopted in our case study.

The workflow curator must be aware of whether the language script is supported by the chosen SWfMS or s/he may assume the risk that the script will not be properly converted into an executable workflow. At this point, the curator will have an executable workflow designed to execute on a specific SWfMS; this workflow can be from now on edited taking advantage of the authoring capabilities of the chosen SWfMS. Figure 3 illustrates the result of this implementation for our case study; it shows a partial MD workflow that was created according to methodology steps 1 and 2, for Taverna.

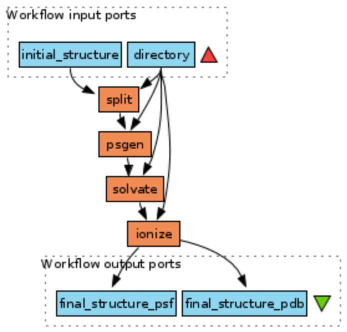


Fig. 3. Partial workflow for an MD script - initial implementation following the first two steps of the methodology.

Once scientists execute this workflow, provenance information regarding execution traces must be collected to serve as input to steps 4 and 5 of our methodology. By executing the workflow, s/he may verify, manually, its results, e.g., checking them against the script results. If this check is not satisfactory, the scientist should identify the problem with help of the execution traces and re-design or re-implement the faulty workflow elements – see more details at section VII.

B. Step 3: Refining the Executable Workflow

Requirement 3 states that the user should be able to modify the workflow to use different computational and data resources. To support this task, a list of available web services and data sets should be shown to the user. For instance, in our case study, scientists' scripts use local data files containing protein coordinates which they download from authoritative web sources. This forces them to download such files from the web, and 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 the use of web services to retrieve these files. An even more helpful refinement is, as we did, to reuse workflows that perform this task: we retrieved from the myExperiment repository[6] a small workflow that fetches a protein structure on Protein Data Bank (PDB) from the RCSB PDB archive[7]. This reused myExperiment workflow was inserted

[6]http://www.myexperiment.org
[7]http://www.rcsb.org/pdb/

in the beginning of our original workflow, replacing the local PDB file used in the original script (see figure 4).

Here, the $structure\_filepath$ input parameter of figure 3 was replaced by the sub-workflow within the light blue box, copied from myExperiment workflow repositories.

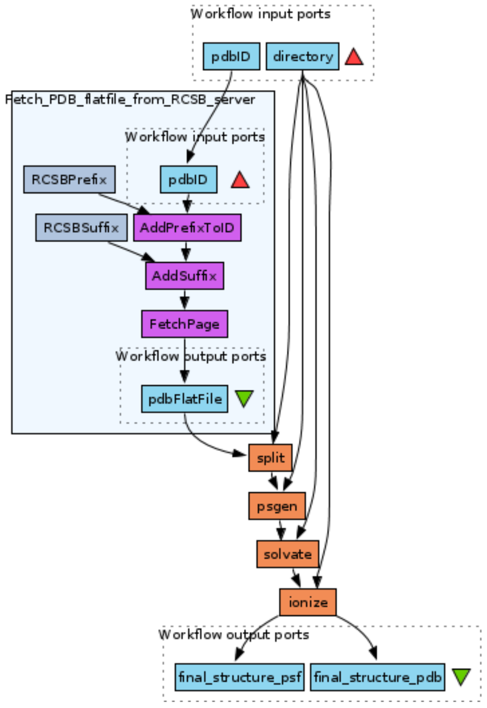


Fig. 4. Workflow refined to use a reusable component that fetches PDB files from the Web.

By the same token, in the life sciences, scientists can invoke web services or reuse data sets listed on portals such as Biocatalogue[8], which provides a curated catalogue of Web services, and Biotools[9], which is a tools and data services registry.

C. Recording Provenance Information of the Executable Workflow

Requirement 4 states that provenance information must be recorded to capture the steps performed in the transformation from script to workflow. This transformation is recorded using 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 versions 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.

Listing 3. PROV statements

```
1  @base <http://w3id.org/s2rwro/md-setup/>.
2  @prefix oa: <http://www.w3.org/ns/oa#>.
3  @prefix prov: <http://www.w3.org/ns/prov-o#>.
4  @prefix wfdesc: <http://purl.org/w4ever/wfdesc#>.
5  @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
6  @prefix wf4ever: <http://purl.org/wf4ever/wf4ever#>.
7
8  <resources/script.sh> a wf4ever:Script, prov:Entity.
9
10 <script/split> a wfdesc:ProcessImplementation;
11    prov:wasDerivedFrom <resources/script.sh>,
12    [
13          a oa:TextPositionSelector;
14          oa:start "674"^^xsd:Integer;
15          oa:end "933"^^xsd:Integer;
16          a prov:Entity
17    ].
18
19 <workflow/we> a wfdesc:Workflow, prov:Entity;
20    prov:wasDerivedFrom <resources/script.sh>;
21    wfdesc:hasSubProcess <workflow/we1/split>.
22
23 <workflow/we/split> a wfdesc:Process;
24    wfdesc:hasImplementation <script/split>.
25
26 <workflow/we1> a wfdesc:Workflow;
27    prov:wasDerivedFrom <workflow/we>;
28    wfdesc:hasSubProcess <workflow/we/split>.
```

[8]https://www.biocatalogue.org/

[9]https://bio.tools

Listing 3 shows RDF statements in Turtle syntax wrt the provenance of W_e, the first workflow derived directly from the script S, and the subsequent workflow W_{e1} derived from W_e. Line 8 describes the script resource as a $wfever : Script$. To identify the chunk of the script that corresponds to a given (executable) activity in W_e, we utilize the W3C Web Annotation Data Model[10]. For example, lines 10 to 17 show that a fragment of the script (delimited using the class $ao : TextPositionSelector$ and the position within the script source code) originated the implementation of a process $< script/split >$ (as a $wfdesc : ProcessImplementation$). This information was extracted from the program block defined using the YesWorkflow tags $@begin$ and $@end$. Lines 19 to 21 show the declaration of W_e; it was derived from the script and it has a subprocess which was declared in lines 23 and 24. This subprocess (defined as $wfdesc : Process$) is associated with the implementation $< script/split >$. Lines 26 to 28 declared W_{e1} as a derivation of W_e and with a subprocess which is the same one from W_e, identified as $< workflow/we/split >$.

VII. Annotating the Workflow and Checking its Quality

It is critical to have a quality check where the scientist explicitly assesses the workflow activities and data flow, comparing them to what was executed by the script.

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.

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

Part of these annotations can be migrated to the concrete workflow from the YesWorkflow tags - e.g., $@desc$ used in the script to describe its program blocks and ports. Most SWfMS, moreover, provide an annotation interface, which can be taken advantage of.

A. Quality dimensions

Quality, here, involves three different quality dimensions[11]: reproducibility, understandability for reuse, and performance. Reproducibility assesses whether W_e – the first workflow created from the script via conversion of the abstract workflow W_a – and its versions $W_{e1} \ldots W_{en}$ reproduce S within some scientist-defined tolerance thresholds. Understandability is promoted with Step 5 (section VIII), by creating Workflow Research Objects with associated annotations. This bundling makes sure that the Research Object is understandable (and thus reusable and reproducible by third parties). Performance concerns the versions $W_{e1} \ldots W_{en}$. Assuming that they satisfy the reproducibility criterion, performance provides measurements of the advantages of executing these versions (e.g., faster execution).

These three dimensions make up for a fourth global quality dimension - reliability. In this sense, we state that our methodology ensures reliable results in the transformation of script S into a bundled Workflow Research Object that supports experiment reproducibility, understandability for reuse, and meets performance requirements.

B. Assessing quality of the workflows

Perhaps the main challenge of assessing all the quality dimensions is to define how to compare script and workflow, and the metrics to use to perform this comparison – i.e., how to assess experiment reproducibility.

To check reproducibility, one may compare the script with the workflow code to check if they are equivalent. However, it is known that checking program equivalence is undecidable. Moreover, the refined workflow may use remote programs (e.g., via web services), for which the source code is not available.

A more pragmatic approach to checking reproducibility consists in shepherding the curator in assessing ”equivalence”, always highly dependent on human expertise. We stipulate that this should be performed in two stages: the first will compare S to W_e, and the second will compare W_e with each of its versions (obtained through refinement), to identify divergences.

a) Comparing S to W_e: In more detail, the analysis of differences between S and W_e should be performed in two successive steps: (i) Visual analysis of W_a by the scientist, to check for problems in, e.g., defining activities, or data flow; and (ii) Comparison of execution results, given that W_e uses exactly the same input files as S, and that the script code was copied from S to W_e. Step (i) was mentioned in section VI-A. Step (ii) can be automatic (e.g., for text files, use linux' $diff$), or semi-automatic, combining algorithms and visual checks. Nevertheless, at some point there may be the need to check data semantics; here, annotations (and semantic annotations of data) can help. Our case study basically involves text files (PDB and similar files as inputs, molecular trajectories as output), and thus textual comparison is enough.

[11]A dimension, in quality literature, is a specific quality property that needs to be taken into consideration in assessing quality.

In performing these steps, one must keep in mind that comparisons should be done with the help of the human curator. For instance, if the results are different in terms of values, then the curator can be solicited to see if they are scientifically similar or if they are completely different and signal a problem with the workflow.

Common mistakes when converting S to W_e typically include:

- the scientist did not clearly identify the main logical processing units in the script and inserted YesWorkflow tags in the wrong places - in this case, the visualization of the abstract workflow will help identify the problem;
- the scientist made a mistake when migrating script code into the corresponding activity - here, execution traces help since they will show that some data sources are used as inputs to incorrect activities;
- the scientist did not provide the correct input files and parameters - again, traces will help;
- the coding of the workflow itself contained errors - this may be checked with analysis of traces.

Last but not least, additional differences may be introduced by the computing environment itself – e.g., the programming environment used to execute the script wrt the SWfMS environment.

b) Comparing W_e to its workflow versions: The rest of the quality check is executed at the same time the scientist improves and/or modifies the workflow through versions (e.g., changing algorithms, or data sets). This comparison can take advantage of the PDIFF algorithm of [7]. PDIFF performs an ”equivalence check” of two workflows by comparing the traces of their executions. Trace comparison is based on 4 elements: the workflow graph obtained from the execution trace, the input data, third-party data and processes, and the SWfMS environment each workflow used. Traces become digraphs, and the authors perform comparison of these digraphs to obtain their differences. The specific point(s) of divergence are identified through graph analysis, assisting the workflow user to understand those differences. In our case, we assume that W_e and its versions run in the same SWfMS.

VIII. Bundle Resources into a Workflow Research Object

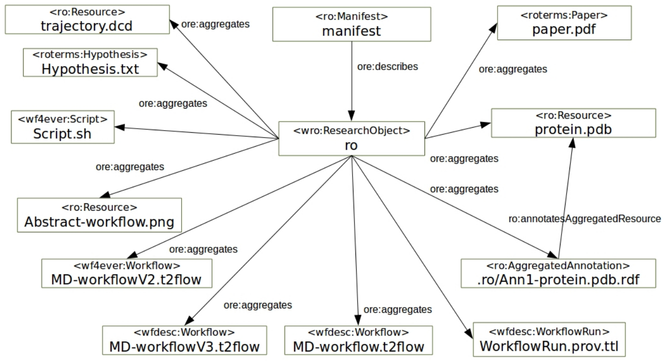
In this step, the curator creates a Workflow Research Object (WRO) that bundles the original script as well as other auxiliary resources obtained in the other steps of the methodology. The creation of the WRO is conducted in parallel with the rest of the methodology steps, in the sense that the curator adds resources to the Workflow Research Object while performing the other steps of the methodology.

The Workflow Research Object model allows curators to aggregate resources and explicitly specify the relationship between these resources and the workflow in a machine-readable format using a suite of ontologies [8].

The result is a WRO that bundles a number of resources that promote the understanding, reproducibility and ultimately the reuse of the workflows obtained through refinement. More specifically, the curator should bundle at least the following resources into the WRO:

- annotated script files (an experiment may involves multiple scripts);
- the workflow W_e (and its versions);
- workflow provenance (documenting the transformation from S to W_e and its versions);
- provenance traces of workflow executions (activities, inputs, outputs, intermediate results);
- research questions and hypotheses;
- output files;

By including these resources, it will be 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 workflow research object, e.g., technical reports and published papers.



<ro:Resource> trajectory.dcd
<roterms:Hypothesis> Hypothesis.txt
<wf4ever:Script> Script.sh
<ro:Resource> Abstract-workflow.png
<wf4ever:Workflow> MD-workflowV2.t2flow
<wfdesc:Workflow> MD-workflowV3.t2flow
<wfdesc:Workflow> MD-workflow.t2flow
<wfdesc:WorkflowRun> WorkflowRun.prov.ttl
<ro:AggregatedAnnotation> .ro/Ann1-protein.pdb.rdf
<ro:Resource> protein.pdb
<roterms:Paper> paper.pdf
<ro:Manifest> manifest
<wro:ResearchObject> ro
ore:describes
ore:aggregates
ro:annotatesAggregatedResource

Fig. 5. A graphical example of a WRO bundle derived from our case study.

Figure 5 shows an example of a WRO created for our use case. Arrows denote relationships and boxes denote instances of concepts defined in ontologies. We used the Research Object ontology[12] to define the RDF-based manifest file describing all the resources aggregated in the bundle and to define the relationships (as ore:aggregates) with the $wro : ResearchObject$ instance. The figure also shows an example of an annotation ($ro : AggregatedAnnotation$), defined in the $.ro/Ann1-protein.pdb.rdf$ file, describing $protein.pdb$. Every file defined in the manifest is a $ro : Resource$ and may be also specialized in specific types of resources such as $wfdesc : Workflow$, $wfdesc : WorkflowRun$ and $wf4ever : Script$. We used the RO Manager tool [13] to create the WRO bundle file at the end of our methodological steps. The bundle is available online at http://w3id.org/w2share/s2rwro/.

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 myExperiment and RO Hub[14] which have an interface to search and navigate between resources aggregated in a RO.

IX. Related Work

Part of related work was already discussed in the text, e.g., the work of [7] for workflow equivalence. Here, we present some brief comments on some relevant sources.

Our methodology guides the transformation from script to executable and verifiable workflow. We adopted YesWorkflow [1], [4] to generate the abstract workflow visualizations before creating the executable workflow (step 1 of our methodology). Our choice of YesWorkflow was primarily based on its simplicity of use, script language independence and platform independence, as well as its open code. Moreover, it allows generating a graphical representation of a script as a workflow.

Another (more system-specific) example of the construction of executable workflows from source code is pursued in [11]. It relies on analysis of Abstract Syntax Trees (ASTs) from the source code of Ruby scripts, to convert automatically such scripts into an executable workflow targeted to a specific SWfMS. Our approach differs from this in that we propose a language-independent methodology to assist scientists to convert scripts written in any language into an executable and reproducible workflow.

There are several other tools and systems to create executable workflows. Examples include HyperFlow [12], StarFlow [13] and Swift [14]. These focus on how a declarative language (defining the workflow model) in conjunction with a general-purpose programming language (defining the activities) can be combined to create executable workflows. Our approach differs in the sense that we do not change the way the scripts are developed, and neither is our approach limited to a specific language.

The work of [15] proposes an executable visual-based representation of a workflow. This was extended by [16] to allow scientists two alternative ways of working with workflows: script-based and visual-based representations. A two-way representation translator enables the conversion between representations; workflow execution uses a single enactor, independent from the users' preferred representation. [16] argues that, in some cases, scripts are preferable to specify workflows since scientists may want to look at code. We, instead, go the opposite way, given the need for reusability by third parties: we adopted a tool and a language and platform-independent approach to transform scripts into workflow research objects.

[12]http://purl.org/wf4ever/ro
[13]https://github.com/wf4ever/ro-manager
[14]http://www.rohub.org/

Another important aspect of our work concerns assessment of quality with respect to reproducibility, reuse and understandability. Our preliminary work towards this goal is based on [7], and their PDIFF algorithm that compares workflow traces, produced by their SWFMS environment, e-Science Central. Their framework uses as input the two provenance digraphs, and produces as output the difference graph, in which nodes may represent differences in data sources or outputs, or differences in activities. They also provide algorithms that compute the equivalence of three classes of data: text, XML and models. For the purposes of comparing XML documents, they use XOM[15]. To calculate the similarity of mathematical models, they use the Analysis of Covariance test that analyses the predictive performance of two models. Yet another possibility to compare workflows appears in [17]. Here, the technique used is based in detecting plagiarism in text. Though interesting, this is too generic for our goals.

X. Conclusions and Ongoing Work

This paper presented a methodology that guides curators in a principled manner to transform scripts into reproducible and reusable workflow research objects. This addresses an important issue in the area of script provenance – that of providing an executable and understandable provenance representation of domain script runs. The methodology was elaborated based on requirements that we elicited given our experience and collaborations with scientists who use scripts in their data analysis. The methodology was showcased via a real world use case from the field of Molecular Dynamics.

Our ongoing work includes the evaluation of our methodology using other use cases, from fields other than molecular dynamics. We are also considering the problem of synchronizing script changes to updates on the corresponding workflow research objects. Moreover, we are investigating extending YesWorkflow to support the semantic annotation of blocks and using concepts from ontologies and vocabularies, and to support workflow nesting, which is currently not supported by YesWorkflow. Last but not least, there is a need to evaluate the cost of the effectiveness of our proposal, in particular since in some cases it may require extensive involvement of scientists and curators.

Acknowledgments

Work partially financed by FAPESP (2014/23861-4), FAPESP/CEPID CCES (2013/08293-7), FAPESP-PRONEX (eScience project), INCT in Web Science, 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.

References

[1] 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*, vol. 10, no. 1, pp. 298–313, 2015.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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*, vol. 119, no. 29, pp. 9295–9303, 2014.

[7] P. Missier, S. Woodman, H. Hiden, and P. Watson, "Provenance and data differencing for workflow reproducibility analysis," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 4, pp. 995–1015, 2016.

[8] 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*, vol. 32, pp. 16–42, 2015.

[9] S. Bechhofer, D. De Roure, M. Gamble, C. Goble, and I. Buchan, "Research objects: Towards exchange and reuse of digital knowledge," 2010.

[10] 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*, vol. 41, no. W1, pp. W557–W561, 2013.

[11] M. Baranowski, A. Belloum, M. Bubak, and M. Malawski, "Constructing workflows from script applications," *Scientific Programming*, vol. 20, no. 4, pp. 359–377, 2012.

[12] B. Balis, "Increasing scientific workflow programming productivity with hyperflow," in *Proceedings of the 9th Workshop on Workflows in Support of Large-Scale Science*. IEEE Press, 2014, pp. 59–69.

[13] E. Angelino, D. Yamins, and M. Seltzer, "Starflow: A script-centric data analysis environment," in *Provenance and Annotation of Data and Processes*. Springer, 2010, pp. 236–250.

[14] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, no. 9, pp. 633–652, 2011.

[15] J. Montagnat, B. Isnard, T. Glatard, K. Maheshwari, and M. B. Fornarino, "A data-driven workflow language for grids based on array programming principles," in *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*. ACM, 2009, p. 7.

[16] K. Maheshwari and J. Montagnat, "Scientific workflow development using both visual and script-based representation," in *6th World Congress on Services (SERVICES-1)*. IEEE, 2010, pp. 328–335.

[17] D. d. O. Filipe Tadeu Santiago, "Verificação da Reprodução de Workflows Científicos por meio de Algoritmos de Detecção de Plágio (in Portuguese)," in *X Brazilian e-Science Workshop (BRESCI 2016)*. Sociedade Brasileira de Computação, 2016.

[15]http://xom.nu