Provenance Oriented Reproducibility of Scripts using the REPRODUCE-ME Ontology

Sheeba Samuel, Birgitta König-Ries

Heinz-Nixdorf Chair for Distributed Information Systems
Friedrich-Schiller University, Jena, Germany
`sheeba.samuel@uni-jena.de, birgitta.koenig-ries@uni-jena.de`

Abstract. Scripts are an important part of many scientific experiments. They are used for data analysis, for integration of data from different sub experiments and computation of results. Reproducibility of these scripts is thus an important step towards reproducibility of the experiments as a whole. In this paper, we propose an approach to ensure this reproducibility by collecting the provenance data of the script execution and using REPRODUCE-ME Ontology extended from the existing vocabularies PROV-O and P-Plan to describe the steps and sequence of steps in the execution of a script. We also present a proof of concept which explores example queries to investigate the reproducibility of scripts.

Keywords: Reproducibility, Provenance, Scripts, Ontology, OBDA

1 Introduction

Reproducibility of experiments allows scientists not only to put trust in their data but also to understand how the data was derived, check the validity of the results and bring new insights in the research by modifying previously conducted experiments [14]. Writing scripts has become a vital part of the research lifecycle of experiments for scientists for automation, measurement and analysis of data. Scripts can be executed in several trials with different parameters for the analysis of data with less effort. In order to reproduce results or to detect which anomalies occurred in the output, it is required to know which input data was responsible for the output, the steps involved in generating them, the functions, the input parameters involved, the dependencies to other modules, time taken for the execution of each function, the side effects etc.
The aim of our work is to semantically describe the provenance of the complete execution of a script in a structured form using linked data without worrying about any underlying technologies or programming languages. For the current prototype, we use provenance data collected from the noWorkflow [11] tool and map the data using the REPRODUCE-ME ontology extended from PROV-O [7] and P-Plan [6] to describe the functions, arguments, output, plan and executions. We provide an additional semantic layer on top of the captured provenance with ontology-based data access and query system to answer the following competency questions.

1. What is the complete derivation of a script output?
2. What is the sequence of steps in the execution of a script with input parameters and intermediate results in each step required to generate the final output?

3. Which are the steps that invoke a particular module?
4. Which are the environmental attributes in the execution of a script?
5. List the user, the operating system, the processor, programming language version, the working directory associated with the execution of a script.

Conventional (non-semantic) systems for provenance management cannot answer these questions easily in a structured format. In this paper, we discuss the various works done in this area in Section 2. We present the ontology, its development process and how to enable provenance-based reproducibility of scripts by capturing and presenting the steps of a script execution and attempting to answer the competency questions in Section 3. The evaluation results of this work are presented in Section 4. The paper is concluded with an insight on future work in Section 5.

2 State of the art

Our idea is to use the existing tools to capture provenance information of a script execution and semantically describe this data. Work related to our approach can be divided into two categories: (i) Tools to capture provenance from scripts. (ii) Semantic Web technologies to represent provenance data from scripts.

There are several tools developed to capture provenance information at different levels of granularity. The tool presented by Frew et al. [5] captures provenance at the operating system level which tracks process and system calls while Tariq et al. [15] describe a method to collect intraprocess provenance automatically. Several version management tools like Git allow developers to track the provenance of files by providing mechanisms to look at the history of versions and the ability to revert to previous versions.

There are several tools which collect provenance information from scripts at function or system level. The Sumatra [4] tool collects input, output, module and data dependencies from Python scripts with the version-control system. It also provides a Web-based interface to view, annotate and search provenance records. The noWorkflow tool [11] captures provenance at the function level from experiment scripts written in Python. It captures three type of provenance - definition (function definitions, calls, and parameters), deployment (execution environment like operating system, modules the script depends on) and execution (function execution, parameter values and return values). It also allows users to analyze the captured provenance using graph, query and diff based analysis methods. The query-based analysis is possible by exporting the provenance data through Prolog. Yesworkflow tool [8] is another tool which collects provenance from scripts and provides many benefits of scientific workflow management systems. Scientists can annotate the scripts with YesWorkflow annotations which are extracted, analyzed and presented as graphical rendering. It is a language-independent user-oriented tool which reveals workflow structure and dependencies from scripts based on user annotations.

However, these tools provide provenance data collected from the script execution in different levels of granularity but do not semantically describe this data. Currently, we provide an additional semantical layer to describe the provenance of a script execution at the function level. As a proof of concept, we use noWorkflow tool to collect provenance data since it provides definition, deployment and execution provenance which is

required to describe a script execution.
There are several ontologies developed to describe provenance. PROV-O [7], an OWL ontology developed by the W3C Provenance Working group, provides a set of classes and properties to represent provenance information generated in various systems. It is general-purpose, domain-independent and has high flexibility to align between different ontologies. There are works which extend PROV-O for modeling provenance information for different domains [[3],[2]]. PROV-O uses plans to describe the set of actions or steps to achieve some goal. But PROV-O does not specify how the plans can be described or the steps they consist of. P-Plan, an OWL2 Ontology, extends PROV-O to describe the provenance of an execution and their relationship with the parts of the plan. It is used to describe the provenance of scientific workflows as plans. It describes the plan which consists of smaller executable steps and dependencies between them.
Meester et al. [9] presents Function Ontology to semantically declare and describe functions. The Ontology consists of Function, Problem, Algorithm, Parameter, Output and Execution as classes. However, the ontology does not capture the dependencies between execution, modules and files. We extend REPRODUCE-ME Ontology from PROV-O and P-Plan to describe functions in the scripts, its execution trace and dependencies in terms of plans, steps and variables.
We are looking to explore the definitions of reproducibility mentioned in [10] and [1]. Moreau [10] defines a provenance graph to be reproducible if it is combined with a primitive environment and contains sufficient information to be interpreted as a program, so that its execution can produce an isomorphic provenance graph. He defines the reproducibility semantics for Open Provenance Model graphs. According to [1], if and only if every job of scientific workflow is reproducible, then the scientific workflow is reproducible.
Based on these definitions of reproducibility, we focus our work on whether two executions of a script generates the same execution graph under the same environment attributes like input parameters, machine configuration, the output of each step, the sequence of steps etc. We focus on the provenance storage of the script information in the relational database and use Ontology-based data access to query the data using REPRODUCE-ME Ontology [12]. We capture the steps and its details using the ontology which is extended from PROV-O and P-Plan. This allows the user to make SPARQL queries on top of this and get answers for the complete execution trace. This logic can be used not only for the scripts but also for the experiments.

3 Capturing Steps from Scripts

We developed the proposed approach using the provenance data collected through the noWorkflow tool. The noWorkflow tool captures provenance of a script by running the command “now run <script>”. The provenance data is stored in SQLite relational database in the same directory where the script is executed. The noWorkflow captures information of each run of a script, the function definitions, start and finish time of each trial and activation of the function. We make use of this information and store this information in a PostgreSQL database. We map the collected provenance data to the REPRODUCE-ME ontology [13].
The workflow for the execution of a script is described using the REPRODUCE-ME

ontology by extending PROV-O and P-Plan. The prefixes `prov`, `p-plan` and `repr` are used to indicate the terms of PROV, P-Plan and REPRODUCE-ME respectively. We provide the details of the important concepts and properties defined by PROV-O, P-PLAN and REPRODUCE-ME ontology needed to describe the execution of script.

- `p-plan:Plan`: subclass of `prov:Plan` consists of smaller steps (`p-plan:Step`) which consume and produce variables (`p-plan:Variable`).
- `p-plan:Step`: represents a planned execution activity and can be different in different executions of the `p-plan:Plan`.
- `p-plan:Activity`: subclass of `prov:Activity` describes the execution of the process planned in a `p-plan:Step`.
- A `p-plan:Variable` represents a description of the input or output of the `p-plan:Step`.
- `p-plan:isPrecededBy`: represents a relationship to describe the dependencies between P-Plan Steps.
- `p-plan:correspondsToStep`: links `prov:Activity` to its planned `p-plan:Step`.
- `repr:Script`: subclass of `p-plan:Plan` represents any program written in any programming language.
- `repr:Trial`: subclass of `p-plan:Activity` represents each execution of a script
- `repr:Argument`: subclass of `p-plan:Variable` represents the arguments of a function.
- `repr:Module`: subclass of `p-plan:Plan` represents the module the script imported during the execution.
- `repr:FunctionActivation`: subclass of `p-plan:Step` represents the function which was activated in a particular `repr:Trial`.
- `repr:Experimenter`: subclass of `prov:Person` represents the person who is associated with the trial.
- `repr:EnvironmentAttribute`: subclass of `repr:Setting` represents the enviroment variables during the trial.
- `repr:correspondsToActivity`: links `p-plan:Activity` to `p-plan:Step`.

Figure. 1 shows REPRODUCE-ME Ontology and how it is extended from PROV-O and P-Plan to capture the provenance information from scripts.
We take a simple example of a Python script to illustrate our work. Program 1.1

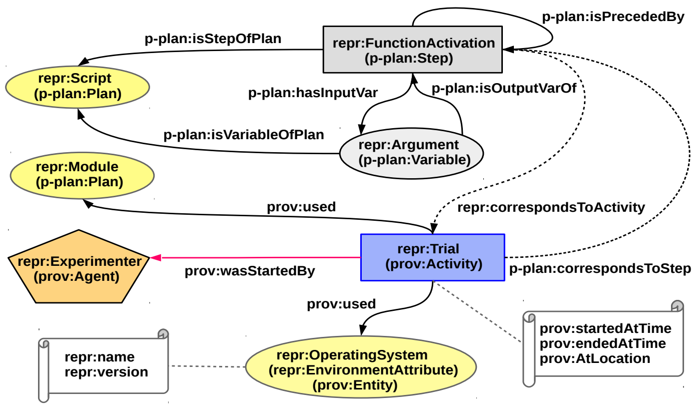
repr:FunctionActivation (p-plan:Step)
p-plan:isPrecededBy
p-plan:isStepOfPlan
repr:Script (p-plan:Plan)
p-plan:hasInputVar
p-plan:isOutputVarOf
p-plan:isVariableOfPlan
repr:Argument (p-plan:Variable)
repr:Module (p-plan:Plan)
prov:used
repr:correspondsToActivity
repr:Experimenter (prov:Agent)
prov:wasStartedBy
repr:Trial (prov:Activity)
p-plan:correspondsToStep
prov:used
prov:startedAtTime
prov:endedAtTime
prov:AtLocation
repr:name
repr:version
repr:OperatingSystem (repr:EnvironmentAttribute) (prov:Entity)

Fig. 1: REPRODUCE-ME Ontology extended from PROV-O and P-Plan

calculates the factorial of a number with recursion and imports the module 1.2 which prints the message.

```python
import display
#Python program to find the factorial of a number
def main():
   num = int(input("Enter a number to find the factorial: "))
   if num < 0:
      message="Factorial does not exist for negative numbers"
   elif num == 0:
      message = "Factorial of 0 is 1"
   else:
      factorial_value = factorial(num)
      message = "Factorial of ",num,"is ", factorial_value
   display.print_message(message)
#Recursive function to find the factorial of a number
def factorial(num):
   if num == 1:
      return num
   else:
      return num*factorial(num-1)
main()
```

Listing 1.1: factorial.py

```python
def print_message(msg):
  print msg
```

Listing 1.2: display.py

The provenance data captured from the execution of a script using noWorkflow tool are populated in the database tables and mapped to the ontology. Listing 1.3 represents the mapping for a `repr:trial`, `repr:FunctionActivation` and the sequence of `p-plan:Step` in a `repr:trial`.

```
mappingId Trial
target   :trial/{id} a :Trial ; prov:value {id} ; prov:
    startedAtTime {start} ; prov:endedAtTime {finish} .
source   select id, start, finish from trial

mappingId Function Activation
target   :activation/{trial_id}/{id} a :FunctionActivation; :
    name {name}; prov:startedAtTime {start} ; prov:
    endedAtTime {finish} .
source   select trial_id, id, name, start, finish from
    function_activation

mappingId Step preceded by another Step
target   :activation/{trial_id}/{id} p-plan:isPrecededBy
:activation/{trial_id}/{caller_id} .
source   select trial_id, id, caller_id from
    function_activation
```

Listing 1.3: Mappings for Script Execution

4 Evaluation

We evaluate our approach by using REPRODUCE-ME ontology to answer our competency questions related to provenance and reproducibility of scripts. We answer these evaluation queries using SPARQL. Here we present two example SPARQL queries related to the information of a script execution.
Query 1: The complete derivation of a script output. The SPARQL Query for Query 1 is listed in Listing 1.4.

```
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX : <http://fusion.cs.uni-jena.de/fusion/repr/#>
PREFIX p-plan: <http://purl.org/net/p-plan#>
SELECT DISTINCT ?function_2_name ?function_1_name ?started_at
     ?script_name ?output_val WHERE {?function_1 a :
    FunctionActivation . ?function_2 a :FunctionActivation .
    ?function_1 p-plan:isPrecededBy ?function_2 .
?function_1 :name  ?function_1_name . ?function_2 :name ?
    function_2_name .
?output p-plan:isOutputVarOf ?function_1 . ?output prov:value
     ?output_val .
?function_1 prov:startedAtTime ?started_at . ?function_1 :
    correspondsToActivity ?trial . ?trial a :Trial . ?trial
    prov:used ?script . ?script :name ?script_name . ?trial
    prov:value ?trial_id FILTER(?trial_id="2"^^xsd:integer)}
ORDER by ?started_at
```

Listing 1.4: List all the steps of a Program involved in deriving a particular final output

Query 2: List the user, the operating system, the processor, programming language version, the working directory associated with the execution of the script.

```
SELECT DISTINCT ?trial ?experimenter_name ?os_name ?
    os_version ?programming_language_name ?
    programming_language_version ?execution_directory WHERE {
  ?os a :OperatingSystem . ?os :name ?os_name . ?os :version
    ?os_version .  ?trial prov:used ?os . ?trial a :Trial . ?
    experimenter a :Experimenter . ?trial prov:wasStartedBy ?
    experimenter . ?experimenter :name ?experimenter_name . ?
    pl a :ProgrammingLanguage . ?trial prov:used ?pl . ?
    programming_language :name ?programming_language_name . ?
    programming_language :version ?
    programming_language_version . ?trial prov:atLocation ?
    execution_directory
}
```

Listing 1.5: List the environment attributes of the execution of the script like operating system

Table 1 shows the result for the SPARQL Query 1.4. The script “factorial.py” is executed twice with the same input 5 and same environment attributes like operating system, programming language version, processor etc. We see that the two trials of a script under the same execution environment and same input parameters in each step follows the same path to generate the same final output which is 120. The environment settings are also required for the reproducibility of experimental data. Here we do not take the randomness factor of input parameters into consideration. Also, we do not consider line by line execution of script rather focus on functions as steps.

function_2_name	function_1_name	output
factorial.py	main	None
main	factorial	120
factorial	factorial	24
factorial	factorial	6
factorial	factorial	2
factorial	factorial	1
main	print_message	None

Table 1: Result for SPARQL Query 1.4

The REPRODUCE-ME Ontology and the results expressed in SPARQL are publicly available [1].

5 Conclusion and Future Work

This paper presents a way to semantically describe the execution of a script using the REPRODUCE-ME ontology, an extension to PROV-O and P-Plan. We also evaluate the ontology using SPARQL queries which are used to answer competency questions related to reproducibility of scripts.
Currently, the provenance is described at function level of granularity but in future, we consider granularity at the level of lines so that we can introduce sub-steps of steps in the ontology. Our future goal is to capture the steps of a scientific experiment using the same logic with REPRODUCE-ME Ontology.

Acknowledgements

This research is supported by the Deutsche Forschungsgemeinschaft (DFG) in Project Z2 of the CRC/TRR 166 High-end light microscopy elucidates membrane receptor function - ReceptorLight. We thank Christoph Biskup and Kathrin Groeneveld from University Hospital Jena, Germany, for providing the requirements to develop the proposed approach and validating the system and our colleagues Martin Bücker, Frank Taubert und Daniel Walther for their feedback.

[1] http://fusion.cs.uni-jena.de/fusion/repr/

References

1. Bánáti, A., Kacsuk, P., Kozlovszky, M.: Investigation of the descriptors to make the scientific workflows reproducible. In: 2016 IEEE 17th International Symposium on Computational Intelligence and Informatics (CINTI). pp. 000129–000134 (Nov 2016)
2. Ciccarese, P., Soiland-Reyes, S., Belhajjame, K., Gray, A.J., Goble, C., Clark, T.: PAV ontology: provenance, authoring and versioning. Journal of Biomedical Semantics 4(1), 37 (2013), http://dx.doi.org/10.1186/2041-1480-4-37
3. Compton, M., Corsar, D., Taylor, K.: Sensor Data Provenance: SSNO and PROV-O together at last. Terra Cognita and Semantic Sensor, Networks pp. 67–82 (2014)
4. Davison, A.: Automated capture of experiment context for easier reproducibility in computational research. Computing in Science Engineering 14(4), 48–56 (July 2012)
5. Frew, J., Metzger, D., Slaughter, P.: Automatic capture and reconstruction of computational provenance. Concurr. Comput. : Pract. Exper. 20(5), 485–496 (Apr 2008), http://dx.doi.org/10.1002/cpe.v20:5
6. Garijo, D., Gil, Y.: Augmenting prov with plans in p-plan: scientific processes as linked data. CEUR Workshop Proceedings (2012)
7. Lebo, T., Sahoo, S., McGuinness, D., Belhajjame, K., Cheney, J., Corsar, D., Garijo, D., Soiland-Reyes, S., Zednik, S., Zhao, J.: PROV-O: The PROV Ontology. W3C Recommendation 30 (2013)
8. McPhillips, T., Song, T., Kolisnik, T., Aulenbach, S., Belhajjame, K., Bocinsky, K., Cao, Y., Chirigati, F., Dey, S., Freire, J., et al.: YesWorkflow: a user-oriented, language-independent tool for recovering workflow information from scripts. arXiv preprint arXiv:1502.02403 (2015)
9. Meester, D., Dimou, Verborgh, Mannens, Walle: An ontology to semantically declare and describe functions. In: Sack, H., Rizzo, G., Steinmetx, N., Mladenić, D., Auer, S., Lange, C. (eds.) The Semantic Web; ESWC 2016 Satellite Events. Lecture Notes in Computer Science, vol. 9989, pp. 46–49. Springer International Publishing (oct 2016), http://link.springer.com/chapter/10.1007/978-3-319-47602-5_10
10. Moreau, L.: Provenance-based reproducibility in the semantic web. Web Semantics: Science, Services and Agents on the World Wide Web 9(2), 202–221 (2011)
11. Murta, L., Braganholo, V., Chirigati, F., Koop, D., Freire, J.: noWorkflow: capturing and analyzing provenance of scripts. In: International Provenance and Annotation Workshop. pp. 71–83. Springer (2014)
12. Samuel, S.: Integrative data management for reproducibility of microscopy experiments. In: The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 - June 1, 2017, Proceedings, Part II. pp. 246–255 (2017), https://doi.org/10.1007/978-3-319-58451-5_19
13. Samuel, S., König-Ries, B.: REPRODUCE-ME: Ontology-based data access for reproducibility of microscopy experiments. In: European Semantic Web Conference Poster. Springer (2017), to appear
14. Samuel, S., Taubert, F., Walther, D., König-Ries, B., Bücker, H.M.: Towards reproducibility of microscopy experiments. D-Lib Magazine 23(1/2) (2017)
15. Tariq, D., Ali, M., Gehani, A.: Towards automated collection of application-level data provenance. In: 4th Workshop on the Theory and Practice of Provenance, TaPP'12, Boston, MA, USA, June 14-15, 2012 (2012), https://www.usenix.org/conference/tapp12/workshop-program/presentation/tariq