Universidade Estadual de Campinas
Instituto de Computação

Lucas Augusto Montalvão Costa Carvalho

Reproducibility and Reuse of Experiments in eScience: Workflows, Ontologies and Scripts

Reprodutibilidade e Reuso de Experimentos em eScience: Workflows, Ontologias e Scripts

CAMPINAS
2018

Lucas Augusto Montalvão Costa Carvalho

Reproducibility and Reuse of Experiments in eScience: Workflows, Ontologies and Scripts

Reprodutibilidade e Reuso de Experimentos em eScience: Workflows, Ontologias e Scripts

Tese apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Thesis presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Supervisor/Orientadora: Profa. Dra. Claudia Maria Bauzer Medeiros
Co-supervisor/Coorientadora: Profa. Dra. Yolanda Gil

Este exemplar corresponde à versão final da Tese defendida por Lucas Augusto Montalvão Costa Carvalho e orientada pela Profa. Dra. Claudia Maria Bauzer Medeiros.

CAMPINAS
2018

Agência(s) de fomento e nº(s) de processo(s): FAPESP, 2013/08293-7; FAPESP, 2014/23861-4; FAPESP, 2017/03570-3
ORCID: http://orcid.org/0000-0002-2412-7183

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

C253r

Carvalho, Lucas Augusto Montalvão Costa, 1985-
Reproducibility and reuse of experiments in eScience : workflows, ontologies and scripts / Lucas Augusto Montalvão Costa Carvalho. – Campinas, SP : [s.n.], 2018.

Orientador: Claudia Maria Bauzer Medeiros.
Coorientador: Yolanda Gil.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Sistemas de gestão de fluxo de trabalho. 2. Fluxo de trabalho - Processamento de dados. 3. Software - Reutilização. 4. Reprodutibilidade dos testes. 5. Ontologias (Recuperação da informação). I. Medeiros, Claudia Maria Bauzer, 1954-. II. Gil, Yolanda. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Reprodutibilidade e reuso de experimentos em eScience : workflows, ontologias e scripts
Palavras-chave em inglês:
Workflow management systems
Workflow - Data processing
Computer software - Reusability
Reproducibility of results
Ontologies (Information retrieval)
Área de concentração: Ciência da Computação
Titulação: Doutor em Ciência da Computação
Banca examinadora:
Claudia Maria Bauzer Medeiros [Orientador]
Flávio Eduardo Aoki Horita
Eduardo Soares Ogasawara
Julio Cesar dos Reis
Leonardo Montecchi
Data de defesa: 14-12-2018
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação

Lucas Augusto Montalvão Costa Carvalho

Reproducibility and Reuse of Experiments in eScience: Workflows, Ontologies and Scripts

Reprodutibilidade e Reuso de Experimentos em eScience: Workflows, Ontologias e Scripts

Banca Examinadora:

- Profa. Dra. Claudia Maria Bauzer Medeiros
  IC/Unicamp
- Prof. Dr. Flávio Eduardo Aoki Horita
  CMCC/UFABC
- Prof. Dr. Eduardo Soares Ogasawara
  EIC/CEFET-RJ
- Prof. Dr. Julio Cesar dos Reis
  IC/Unicamp
- Prof. Dr. Leonardo Montecchi
  IC/Unicamp

A ata da defesa com as respectivas assinaturas dos membros da banca encontra-se no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 14 de dezembro de 2018

Dedicatory

To my loving wife, Flávia, who was with me along all this journey.

So the problem is not so much to see what nobody has yet seen, as to think what nobody has yet thought concerning that which everybody sees.

(Arthur Schopenhauer)

Acknowledgments

First of all, I would like to thank God for my life and all opportunities given to me.

I would like to thank my supervisors Claudia Bauzer Medeiros at Unicamp and Yolanda Gil at the University of Southern California, for their guidance and encouragement during all this time. I will take their advices with me for the rest of my life.

I owe a special thank you to my wife Flávia, who was by my side and supported me along the way, even when we were in different countries.

I am grateful to my parents, who always supported my studies. My family and relatives made every moment very enjoyable in each visit or each time I went home.

I also would like to thank my first supervisor, Hendrik Teixeira Macedo, who introduced me to Science during my undergraduate studies, and again as my supervisor in my Master's degree.

I would like to thank all my collaborators who co-authored some parts of this work: Daniel Garijo, Khalid Belhajjame, Joana Malaverri, Bakinam Essawy, Regina Wang, Prof. Munir Skaf, Caroline Simões and Rodrigo Silveira. I also would like to thank Prof. Benilton Carvalho and Welliton Souza for their valuable support to construct and validate the bioinformatics case study. Many other collaborators who influenced my work and I would like to acknowledge are: Murilo Borges, Varun Ratnakar, Suzanne Pierce, Daniel Hardesty, Deborah Khider, Margaret Knoblock, Scott Peckham, and Christopher Duffy.

I would like to thank my lab mates from the Laboratory of Information Systems (LIS) at Unicamp, specially Márcio, Jacqueline, Ivelize and Francisco, as well as the friends I met when I was in Los Angeles, specially Yibo, TG, and Micah.

I owe a special thank to Daniel Garijo, who helped me a lot before, during and after my internship at ISI. Without his help, this work would not be the same.

All other professors, staff, and colleagues at UNICAMP and the University of Southern California's Information Sciences Institute (ISI) deserve my gratitude in this journey.

Finally, I would like to thank the Sao Paulo Science Foundation (FAPESP) for funding my scholarships: grant 2014/23861-4, FAPESP/BEPE grant 2017/03570-3, and FAPESP/CEPID CCES grant 2013/08293-7. These scholarships have funded the research internship and the research described on this document. I show my appreciation for the US National Science Foundation (NSF) and the US Defense Advanced Research Projects Agency (DARPA) which funded part of my expenses in conferences and technical visits during my internship, as well as CAPES and CNPq which partially funded this research. The opinions expressed in this work do not necessarily reflect those of the funding agencies.

Resumo

Scripts e Sistemas Gerenciadores de Workflows Científicos (SGWfC) são abordagens comumente utilizadas para automatizar o fluxo de processos e análise de dados em experimentos científicos computacionais. Apesar de amplamente usados em diversas disciplinas, scripts são difíceis de entender, adaptar, reusar e reproduzir. Por esta razão, diversas soluções têm sido propostas para auxiliar na reprodutibilidade de experimentos que utilizam ambientes baseados em scripts. Porém, estas soluções não permitem a documentação completa do experimento, nem ajudam quando outros cientistas querem reusar apenas parte do código do script. SGWfCs, por outro lado, ajudam na documentação e reuso através do suporte aos cientistas durante a modelagem e execução dos seus experimentos, que são especificados e executados como componentes interconectados (reutilizáveis) de workflows. Enquanto workflows são melhores que scripts para entendimento e reuso dos experimentos, eles também exigem documentação adicional. Durante a modelagem de um experimento, cientistas frequentemente criam variantes de workflows, e.g., mudando componentes do workflow. Reuso e reprodutibilidade exigem o entendimento e rastreamento da proveniência das variantes, uma tarefa que consome muito tempo. Esta tese tem como objetivo auxiliar na reprodutibilidade e reuso de experimentos computacionais. Para superar estes desafios, nós lidamos com dois problemas de pesquisas: (1) entendimento de um experimento computacional, e (2) extensão de um experimento computacional. Nosso trabalho para resolver estes problemas nos direcionou na escolha de workflows e ontologias como respostas para ambos os problemas. As principais contribuições desta tese são: (i) apresentar os requisitos para a conversão de experimentos baseados em scripts em experimentos reprodutíveis; (ii) propor uma metodologia que guia o cientista durante o processo de conversão de experimentos baseados em scripts em *workflow research objects* reprodutíveis. (iii) projetar e implementar funcionalidades para avaliação da qualidade de experimentos computacionais; (iv) projetar e implementar o W2Share, um arcabouço para auxiliar a metodologia de conversão, que explora ferramentas e padrões que foram desenvolvidos pela comunidade científica para promover o reuso e reprodutibilidade; (v) projetar e implementar o OntoSoft-VFF, um arcabouço para captura de informação sobre software e componentes de workflow para auxiliar cientistas a gerenciarem a exploração e evolução de workflows. Nosso trabalho é apresentado via casos de uso em Dinâmica Molecular, Bioinformática e Previsão do Tempo.

Abstract

Scripts and Scientific Workflow Management Systems (SWfMSs) are common approaches that have been used to automate the execution flow of processes and data analysis in scientific (computational) experiments. Although widely used in many disciplines, scripts are hard to understand, adapt, reuse, and reproduce. For this reason, several solutions have been proposed to aid experiment reproducibility for script-based environments. However, they neither allow to fully document the experiment nor do they help when third parties want to reuse just part of the code. SWfMSs, 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. During experiment design, scientists frequently create workflow variants, e.g., by changing workflow components. Reuse and reproducibility require understanding and tracking variant provenance, a time-consuming task. This thesis aims to support reproducibility and reuse of computational experiments. To meet these challenges, we address two research problems: (1) understanding a computational experiment, and (2) extending a computational experiment. Our work towards solving these problems led us to choose workflows and ontologies to answer both problems. The main contributions of this thesis are thus: (i) to present the requirements for the conversion of script to reproducible research; (ii) to propose a methodology that guides the scientists through the process of conversion of script-based experiments into reproducible workflow research objects; (iii) to design and implement features for quality assessment of computational experiments; (iv) to design and implement W2Share, a framework to support the conversion methodology, which exploits tools and standards that have been developed by the scientific community to promote reuse and reproducibility; (v) to design and implement OntoSoft-VFF, a framework for capturing information about software and workflow components to support scientists manage workflow exploration and evolution. Our work is showcased via use cases in Molecular Dynamics, Bioinformatics and Weather Forecasting.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Motivation

Data driven experiments increasingly depend on research on various aspects of information technology. This phenomenon, known as eScience, is characterized by conducting joint research in computer science and other fields to support the entire research cycle, from collection and mining of data to visual representation and data sharing. It encompasses techniques and technologies for data-intensive science, the new paradigm for scientific exploration [45].

However, the existing tools and solutions do not help reuse and reproducibility of those experiments. Currently, there are many kinds of computational environments to support the execution of experiments, from scripts to workflows. Script 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 or Shell in their daily data analysis and experiments. However, 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), and as such are 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 Management Systems (SWfMS) to manage scripts, see e.g., [31, 56, 57, 61].

Although these proposals bring new useful functionalities for understanding scripts and their execution traces, they do not enable reuse and reproducibility of scripts. For example, YesWorkflow [31, 61] allows scientists to annotate scripts so that they can be subsequently visualized as if they were workflows that cannot be executed. The provenance traces captured from scripts using noWorkflow [61] are fine-grained, and therefore cumbersome, for the user who would like to understand the lineage of the script results, as shown in [31].

Reproducibility requires understanding a computational experiment (in the sense of transparency). Here, we follow the definition of [58]: "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."

Extending a computational experiment is one of the facets of reuse. While scripts

are hard to reuse and understand, scientific workflows [29] are presented as a solution to such problems. They play an important role in data-centric scientific experiments. As such, they have been often pointed out as a means to speed up the construction of new experiments, and foster collaboration through reuse of workflow fragments, and/or adaptation and repurposing of entire workflows [24].

In spite of these advantages, there are several challenges to be met when using workflows – not only in modeling, but in storing and understanding workflows. Understanding is especially complicated when scientists work in distinct domains, due to heterogeneity in vocabularies, methodologies, perspectives of solving a problem and granularity of objects of interest. For instance, biodiversity projects require cooperation across many time and space scales, varying from the global (climate) level to the micro (species physiology) level.

The graphical representation of workflows is a step towards understanding. However, modeling an experiment through a SWfMS is a time-consuming task. Therefore, reuse of workflow fragments from previous modeled experiments is essential [24, 41]. In [41], it is argued that designing new workflows by reusing and re-purposing previous workflows or workflow patterns has the advantages of reducing workflow authoring time, improving quality through shared workflow development, improving experimental provenance through reuse of established and validated workflows and avoiding workflow redundancy.

Our work is concerned with meeting the needs of a heterogeneous research environment, and is based on our ongoing experience with the Center for Computational Engineering and Science (CCES)[1], established at University of Campinas (Unicamp), Brazil. CCES congregates experts from 6 different domains – Computer Science, Chemistry, Physics, Biology, Applied Mathematics and Mechanical Engineering. CCES scientists use scripts and are not familiar with workflows. We, moreover, face the intrinsic heterogeneity of the groups we are working with.

1.2 Basic Terminology

1.2.1 Scientific Workflow Management Systems

Scientific workflow Management Systems (SWfMS) [1, 50] provide domain scientists with an easy-to-use system for capturing the execution of simulations and data analysis pipelines as scientific workflows. According to Altintas et al. [1], "scientific workflows are a formalization of the ad-hoc process that a computational scientist may go through to get from raw data to publishable results". Scientists use SWfMSs to design, execute, monitor, re-run, and document experiments. SWfMSs allow the design of components that can be reused across different workflows.

SWfMSs record provenance information [27], which must be associated and stored with the new data products and contain enough details to enable reproducibility. An important piece of information present in workflow provenance is information about causality [33]. Causality captures the sequence of steps that, together with input data and parameters,

[1] `http://www.escience.org.br`

caused the creation of a data product.

Although scientific workflows have some characteristics similar to business process workflows, there are several challenges not present in the business workflow scenario [78, 50]. For example, scientific workflows often perform large-scale simulation and data analysis pipelines, which are very computationally intensive and produce complex data products that may be archived for reuse in further experiments. Also, scientific workflows operate on large, complex and heterogeneous data. SWfMSs often manage data movement and task execution on distributed computational environments [30]. Moreover, unlike business workflows that are control-oriented, scientific workflows are often dataflow-oriented.

1.2.2 Ontology and Semantic Annotations

An ontology can be defined as "an explicit specification of a conceptualization" [42]. From a computer science perspective, an ontology can be viewed as a controlled vocabulary of concepts, data model or metadata schema that formally represents a set of concepts within a domain and the relationships between these concepts. Each of these concepts and relationships has explicitly defined and machine-processable semantics. By defining shared and common domain concepts, ontologies help people and machines to communicate unambiguously.

Ontologies may be represented using formats such as RDF (Resource Description Framework) [54] and OWL (Web Ontology Language) [55]. Experts construct ontologies often importing and reusing existing structures.

The Semantic Web proposes annotating digital documents using well-defined knowledge representation languages, represented as ontologies [74]. Semantic Web annotations (also known as semantic tagging or semantic enrichment) formally identify concepts (e.g., people, things, places, organizations) and relationships between concepts in documents, attaching additional semantic information to a given text file or another digital content. Semantic Web annotations are intended primarily for use by machines, for example, to improve and automate search capabilities and discover resources.

1.2.3 Research Objects

From a computer science perspective, a Research Object is "a semantically rich aggregation of resources, that possess some scientific intent or support some research objective" [5]. The primary goal of the Research Object approach is to improve the reproducibility of scientific investigations by providing a mechanism to associate together related resources about a scientific investigation so that they can be shared together using a kind of self-contained unit of knowledge. A Research Object provides a digital analogue of the 'Materials and Methods' section of a research paper.

Workflow Research Objects (WRO) [6] is an approach to the preservation of scientific workflows by aggregating data and metadata that enrich the workflow specifications. It is realised as a suite of ontologies to explicitly specify the relationship between the valuable resources of an experiment and the workflow [7]. For instance, WROs allow a third-party

scientist to understand the experiment and run the original workflow using the same data inputs as the original experiment or run the workflow using different data inputs.

1.3 Problem Statement and Challenges

The main problem we are tackling in this thesis is to support reproducibility and reuse of computational experiments. This thesis aims to answer the following research question: "how can we support scientists facing challenges in understanding and extending computational experiments designed by themselves or others?". This, in turn, will help reproducibility and reuse.

There are two issues involved: (1) how to represent the computational steps of an experiment and their data dependencies; (ii) how to represent the computational steps implementation and their evolution; and (iii) how to provide tools to help scientists create workflow variants. Due to the complexity involving these subjects, we performed our research exploring two research problems: (1) understanding a computational experiment; (2) extending a computational experiment.

1.3.1 Problem 1: Understanding a computational experiment

As mentioned, scripts and Scientific Workflow Management Systems (SWfMSs) [29, 23] are common approaches that have been used to automate the execution flow of processes and data analysis in scientific (computational) experiments.

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 Jupyter Notebooks [46], ReproZip [22], YesWorkflow [57], and noWorkflow [61]. Although 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.

While workflows are better than scripts for understandability and reuse, they still require additional documentation to support reproducibility.

1.3.2 Problem 2: Extending a computational experiment

Scientific Workflow Management Systems play a major role in supporting scientists to design, document and execute their computational experiments. During workflow design, scientists use third-party software or their own code to implement workflow components. The challenge arises when such software evolves – and thus a scientist's workflow is affected. There are many reasons for scientists to modify a workflow that they created, either by changing specific steps of the workflow or changing the workflow structure.

Changes in software used to implement components are common and could happen for different reasons, e.g., a newer version is available, or older software is not maintained. Also, data sources change, e.g. when datasets are updated, which may require adjustments in existing components and adding new ones. Thus, due to changes in software and data, workflows must be updated accordingly to avoid workflow decay [79] and reproducibility issues [36].

Another important reason to update workflows is when scientists are exploring alternative ways of performing a computational experiment. During these exploratory tasks, scientists often want to compare methods or try different approaches to implement a workflow component. In current workflow systems, scientists manage these updates manually. However, updating a workflow is a complex and time-consuming task, as it requires tracking down information about the different versions of software and functions used in the components of the workflow and understanding the impact in other workflow steps.

1.4 Contributions

This PhD research resulted in 5 main contributions, summarized as follows:

- To present the requirements for the conversion of scripts to reproducible research. These requirements aims to promote the understandability and reuse of script-based experiments. This contribution is a result of research problem 1 and is presented in Chapter 2.

- To propose a methodology that guides the scientists to convert script-based experiments into reproducible workflow research objects. This is motivated by research problem 1 and was proposed to fill the gap of the absense of a methodology for script to reproducible research conversion. Results are presented in Chapter 2.

- To design and implement W2Share, a computational framework that supports the script-to-reproducible-research methodology. It exploits tools and standards that have been developed by the community, in particular YesWorkflow, Research Objects and the W3C PROV. It is generic in the sense it is script-language-independent, domain-agnostic and environment independent. This contribution is a result of the research problems 1 and it is presented in Chapters 3 and 4. In particular, Chapter 3 is concerned with supporting annotations that allow scientists to assess the quality of a workflow, thereby helping reuse decisions.

- To design and implement OntoSoft-VFF, a framework for capturing information about software and workflow components that is important for managing workflow exploration and evolution. It is based on a novel ontology designed to describe the functionality and evolution through time of any software used to create workflow components. The framework uses a software metadata catalog to support comparison and semantic search of software metadata. This contribution is a result of research problem 2 and is presented in Chapter 5.

- To analyze real world examples, showing how they can benefit from our proposal. We present how three case studies from Molecular Dynamics, Bioinformatics and Machine Learning can take advantage of the script to reproducible research conversion performed by scientists, pointing out the advantages of this approach. This contribution are presented in Chapters 2, 3, 4, and 5.

In more detail, table 1.1 shows the main contributions, their association with the chapters, and which challenge they are trying to meet. For instance, the table shows that Chapter 1 is geared towards answering the first research question "how to understand a computational experiment?" and this is mainly answered via the requirements and the methodology.

Figures 1.1 and 1.2 respectively illustrate how we deal with the two problems. Figure 1.1 shows the aspects within W2Share that support the script-to-workflow conversion (thereby helping scientists to understand each other's computing experiment - Problem 1). Figure 1.2 shows how OntoSoft-VFF (Chapter 5) supports scientists in the task of creating workflow variants.

In more detail, Figure 1.1 shows that scientists start from separating process units within a script (left), through introducing YesWorkflow tags. The resulting script is automatically transformed into an executable Taverna workflow, which may receive quality annotations from the scientist. At the end, script, workflow and additional files are bundled into a workflow research object. Tags and annotations are manual tasks, since they depend on human interpretation.

Figure 1.2 shows that scientists start from a executable workflow. The scientists retrieve information from a software metadata catalog to understand differences between software that implement workflow components. New workflow components may be automatically generated by the software metadata catalog. At the end, a workflow variant is generated by scientists.

Table 1.1: Mapping of the contributions and chapters.

	Q1-Contributions	Q2-Contributions
Chapter 2	Methodology and requirements.	
Chapter 3	Design and implementation of W2Share – Conversion and Quality annotations.	
Chapter 4	Design and implementation of W2Share – traceability of conversion.	
Chapter 5		Catalog of software metadata to support users in workflow variants.

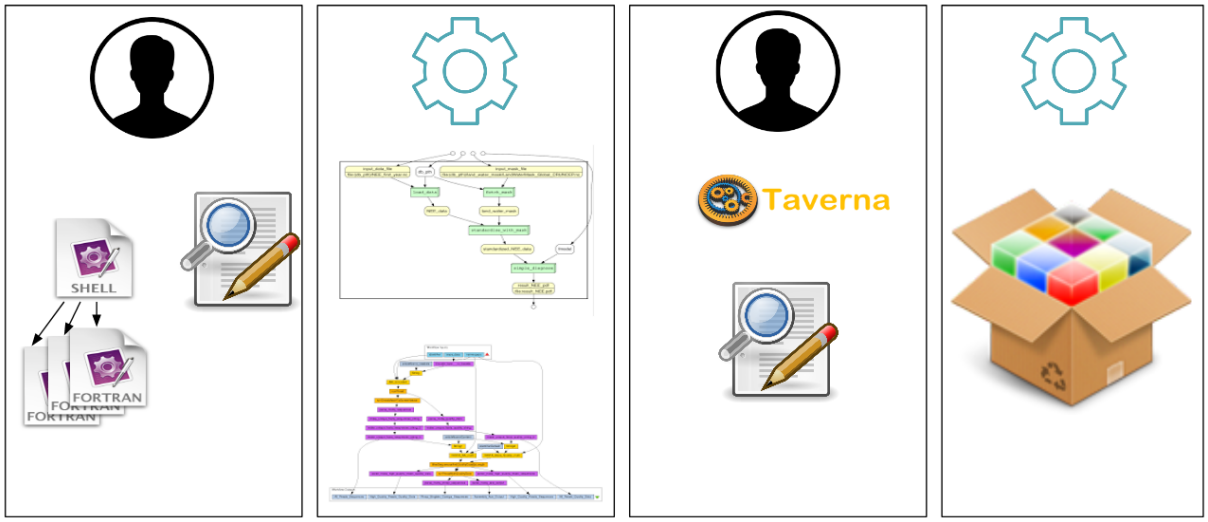


Figure 1.1: Overview of solution for Research Problem 1.

1.5 Thesis Organization

This chapter presented the motivation, goal, research problems and main contributions of this PhD research. The organization in chapters indirectly followed our research methodology. The entire research was conducted by starting with questions raised by scientists concerning reusability issues. We then tried to find approaches to solve the problem, and from then evaluate the results, strongly based on case studies. The remainder of this text is organized as a collection of papers, as follows.

Chapter 2 corresponds to the paper "Converting Scripts into Reproducible Workflow Research Objects", published in the Proceedings of the 12th International Conference on eScience [12]. This chapter discusses the methodology that guides scientists to convert script into workflow research objects in a principled manner. Also, it presents the requirements elicited to develop the methodology and a case study from Molecular Dynamics.

Chapter 3 corresponds to the paper "Implementing W2Share: Supporting Reproducibility and Quality Assessment in eScience, published in the Proceedings of the 11th Brazilian e-Science Workshop [17]. This chapter introduces our first implementation of W2Share and discusses how W2Share incorporates features that allow annotating experiments with quality information. The chapter showcases W2Share by using a real-world scenario in Bioinformatics.

Chapter 4 corresponds to the paper "A PROV-Compliant Approach for the Script-to-Workflow Process", submitted to the Semantic Web journal [13]. This chapter, currently under review, describes how W2Share enables traceability of the script-to-workflow process, thereby establishing trust in this process. This chapter presents the implementation of methodology steps, and proposes a machine-readable abstract workflow for scripts. It explains how W2Share enables the transformation of a script into the machine-readable abstract workflow and how provenance information links elements from abstract workflows back to scripts. It also shows how the conversion process takes advantage of this provenance information to create semi-automatically an executable workflow. This chapter presents a validation of the approach through answers to competency queries, which

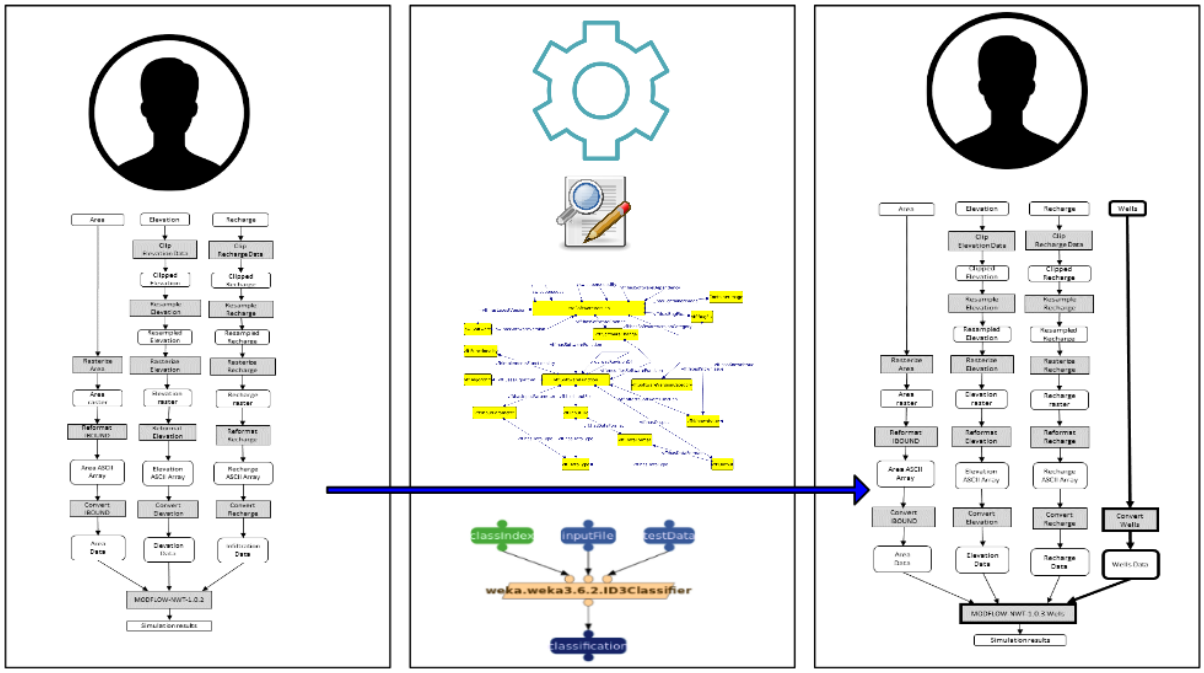


Figure 1.2: Overview of solution for Research Problem 2.

address the requirements presented in Chapter 2.

Chapter 5 corresponds to the paper "Semantic Software Metadata for Workflow Exploration and Evolution", published in the Proceedings of the 14th International Conference on eScience [16]. This chapter presents OntoSoft-VFF, a framework to describe the functionality and evolution through time of any software used to create workflow components. OntoSoft-VFF is composed by a software metadata repository and a novel ontology. This chapter also shows how the catalog supports comparison and semantic search of software metadata. OntoSoft-VFF is showcased by using machine learning workflow examples. This chapter shows the validation of the approach by testing how a workflow system could benefit of the approach by comparing differences in software metadata, explaining software updates and describing the general functionality of workflow steps to scientists.

Chapter 6 contains conclusions and some directions for future work.

Besides the papers in Chapters 2, 3, 4 and 5, others were also published in the course of this thesis, directly related to this research. There follows a list of publications, including the ones that compose the thesis.

- L. A. M. C. Carvalho, R. L. Silveira, C. S., Skaf, M. S.Pereira, and C. B. Medeiros. Provenance-based retrieval: Fostering reuse and reproducibility across scientific disciplines. In *Proceedings of the 6th International Provenance and Annotation Workshop (IPAW)*, June 7-8, 2016, pages 183–186, McLean, VA, USA. Springer.

- L. A. M. C. Carvalho, C. B. Medeiros. Provenance-Based Infrastructure to Support Reuse of Computational Experiments. *Proceedings of the Satellite Events of the 31st Brazilian Symposium on Databases (Thesis and Dissertations Workshop)*, Sociedade Brasileira de Computação Salvador, Bahia, Brazil, October 4-7, 2016.

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

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

- L. A. M. C. Carvalho, R. Wang, D. Garijo, 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.

- L. A. M. C. Carvalho, B. T. Essawy, D. Garijo, C. B. Medeiros, 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.

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

- L. A. M. C. Carvalho, K. Belhajjame, C. B. Medeiros, A PROV-Compliant Approach to Script-to-Workflow Process. In *The Semantic Web Journal*. IOS Press, 2018 (under review).

Chapter 2

Converting Scripts into Reproducible Workflow Research Objects

2.1 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., [57, 31, 61, 56]. For example, McPhilips *et al.* [57] 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.* [61] 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 [31].

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 [6] (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 2.2 presents the case study that we use as a running example throughout the paper. Section 2.3 identifies the requirements that guided the development of our methodology, which is overviewed in section 2.4. Sections 2.5 through 2.8 show in detail each step of our methodology. Section 2.9 briefly discusses related work. Finally, Section 2.10 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.

2.2 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.

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

Many groups have implemented their specific MD simulations using special purpose scripts. In our case, a suite of scripts was designed by physiochemists [68]; 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.

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.

2.3 Requirements for Script Conversions

This section presents the requirements that guided the development of our solution, and section 2.4 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.

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

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

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

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 [58], 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 [58]: "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." [58] 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.

2.4 Methodology to Assist in Script Conversions

To meet the requirements identified in Section 2.3, we devised a methodology for converting a script into reproducible workflow research objects [6, 7]. 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 [6]. A Research Object [5] 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 [6] (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 2.1, 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 Wa identifying the main processing steps that constitute the data analysis implemented by the script, and their data dependencies. The workflow Wa obtained as a result in

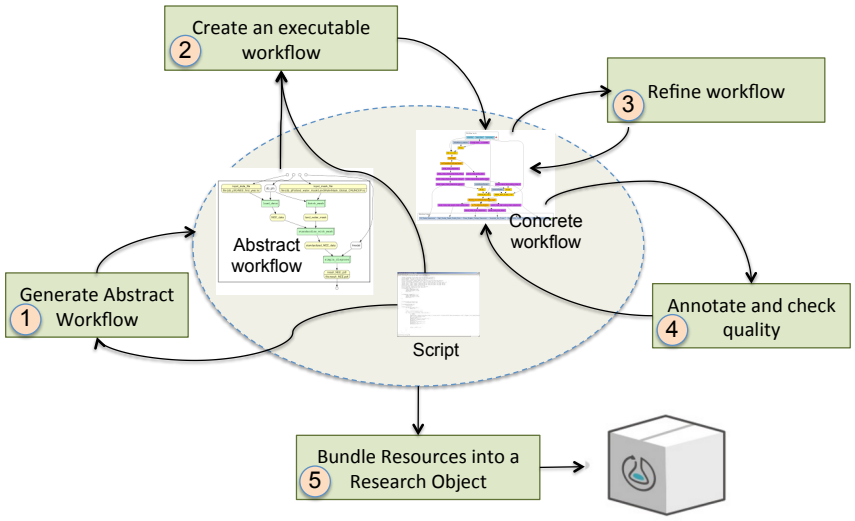


Figure 2.1: Methodology for converting scripts into reproducible Workflow Research Objects.

Step 1 is abstract in the sense that it cannot be executed.

Given this abstract workflow, the second step **Create an executable workflow** converts the abstract workflow into an executable one We 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 $We_1 \ldots We_n$. 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 2.8.

We next present the aforementioned steps in detail.

2.5 Generating an Abstract Workflow

The objective of this phase is to address Requirement 1 by generating an abstract workflow, Wa, given the script S. The generation of Wa 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 [57, 56]. 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 2.2) 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
```

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

```
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
32 # @IN protein_pdb          @URI file:protein.pdb
33 # @IN bglc_pdb           @URI file:bglc.pdb
34 # @IN water_pdb           @URI file:water.pdb
35 # @OUT hyd_pdb                 @URI file:hyd.pdb
36 # @OUT hyd_psf                @URI file:hyd.psf
37
38 ... commands ...
39
40 # @END psfgen
41
42 # @BEGIN solvate
43 # @IN hyd_pdb @URI file:hyd.pdb
44 # @IN hyd_psf @URI file:hyd.psf
45 # @OUT wbox_pdb @URI file:wbox.pdb
46 # @OUT wbox_psf @URI file:wbox.psf
47 echo "
48 package require solvate
49 solvate hyd.psf hyd.pdb -rotate -t 16 -o wbox
50 exit
51 " > solv.tcl
52
53 vmd -dispdev text -e solv.tcl
54 rm solv.tcl
55 # @END solvate
56
57 # @BEGIN ionize
58 # @IN wbox_pdb @URI file:wbox.pdb
59 # @IN wbox_psf @URI file:wbox.psf
60 # @OUT gh5_pdb @AS final_structure_pdb
61 #                        @URI file:gh5.pdb
62 # @OUT gh5_psf @AS final_structure_psf
63 #                        @URI file:gh5.psf
64
```

```
65 ... commands ...
66
67 # @END ionize
68
69 # @END setup
```

Once the script is annotated, YesWorkflow generates an abstract workflow representation. Figure 2.2 depicts the abstract workflow generated given the tags provided in Listing 1. It is merely a graphical representation of the script.

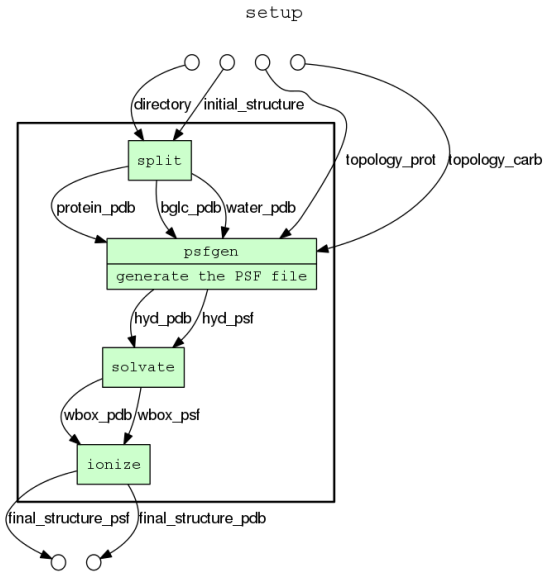


Figure 2.2: Abstract workflow representation generated via YesWorkflow.

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 [57], 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" Wa. 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.

2.6 Creating an Executable Workflow from the Abstract Workflow

Given the abstract workflow Wa generated previously, the curator needs to create an executable workflow We that embodies the data analysis and processes as depicted by Wa – 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).

2.6.1 Step 2: Creating an Initial Executable Workflow

To create the executable workflow We, 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 Wa, the corresponding code in We 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 We against the script execution, to correct eventual mistakes in script-to-workflow conversion.

Once the curator 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. Without loss of generality, we used the Taverna system [77], 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 2.3 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.

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 2.7.

2.6.2 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

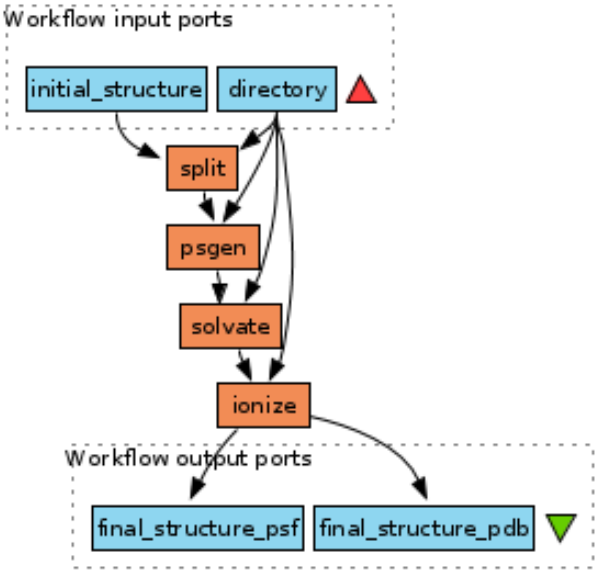


Figure 2.3: Partial workflow for an MD script - initial implementation following the first two steps of the methodology.

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 in the beginning of our original workflow, replacing the local PDB file used in the original script (see figure 2.4).

Here, the *structure_filepath* input parameter of figure 2.3 was replaced by the sub-workflow within the light blue box, copied from myExperiment workflow repositories.

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.

2.6.3 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

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

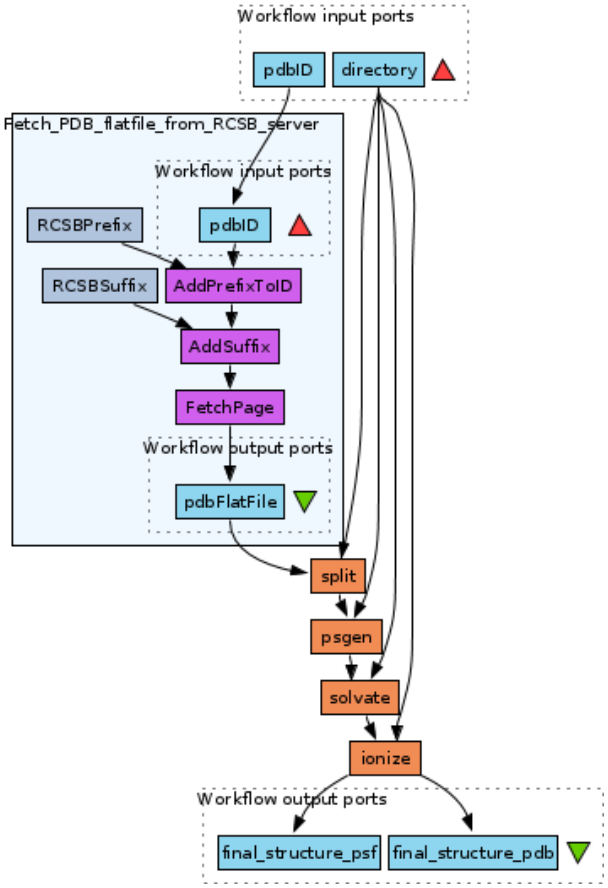


Figure 2.4: Workflow refined to use a reusable component that fetches PDB files from the Web.

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

```
@base <http://w3id.org/s2rwro/md-setup/>.
@prefix oa: <http://www.w3.org/ns/oa#>.
@prefix prov: <http://www.w3.org/ns/prov-o#>.
@prefix wfdesc: <http://purl.org/w4ever/wfdesc#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix wf4ever: <http://purl.org/wf4ever/wf4ever#>.

<resources/script.sh> a wf4ever:Script, prov:Entity.

<script/split> a wfdesc:ProcessImplementation;
    prov:wasDerivedFrom <resources/script.sh>,
    [
            a oa:TextPositionSelector;
            oa:start "674"^^xsd:Integer;
            oa:end "933"^^xsd:Integer;
            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>.
```

Listing 3 shows RDF statements in Turtle syntax wrt the provenance of We, the first workflow derived directly from the script S, and the subsequent workflow We_1 derived from We. Line 8 describes the script resource as a `wf4ever:Script`. To identify the chunk of the script that corresponds to a given (executable) activity in We, 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 We; 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 We_1 as a derivation of We and with a subprocess which is the same one from We, identified as `<workflow/we/split>`.

2.7 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. 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.

2.7.1 Quality dimensions

Quality, here, involves three different quality dimensions[11]: reproducibility, understandability for reuse, and performance. Reproducibility assesses whether We – the first workflow created from the script via conversion of the abstract workflow Wa – and its versions

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

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

$We_1 \ldots We_n$ reproduce S within some scientist-defined tolerance thresholds. Understandability is promoted with Step 5 (section 2.8), 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 $We_1 \ldots We_n$. 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.

2.7.2 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 We, and the second will compare We with each of its versions (obtained through refinement), to identify divergences.

Comparing S to We In more detail, the analysis of differences between S and We should be performed in two successive steps: (i) Visual analysis of Wa by the scientist, to check for problems in, e.g., defining activities, or data flow; and (ii) Comparison of execution results, given that We uses exactly the same input files as S, and that the script code was copied from S to We. Step (i) was mentioned in section 2.6.1. 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.

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 We 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.

Comparing We 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 [58]. 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 We and its versions run in the same SWfMS.

2.8 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 [7].

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 We (and its versions);

- workflow provenance (documenting the transformation from S to We 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.

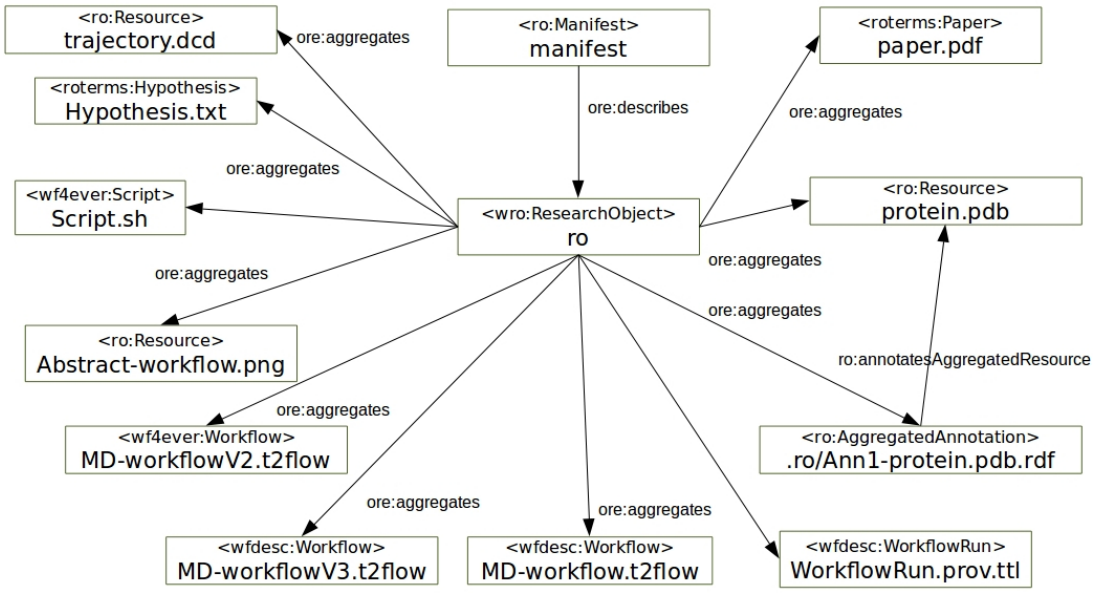


Figure 2.5: A graphical example of a WRO bundle derived from our case study.

Figure 2.5 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 `https://w3id.org/w2share/s2rwro/`.

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

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.

2.9 Related Work

Part of related work was already discussed in the text, e.g., the work of [58] 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 [57, 56] 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 [4]. 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 [3], StarFlow [2] and Swift [75]. 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 [59] proposes an executable visual-based representation of a workflow. This was extended by [51] 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. [51] 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.

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 [58], and their PDIFF algorithm that compares workflow traces, produced by their SWFMS environment, e-Science Central. Their framework uses as input the two

[14] `http://www.rohub.org/`

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 [32]. Here, the technique used is based in detecting plagiarism in text. Though interesting, this is too generic for our goals.

2.10 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.

[15] `http://xom.nu`

Chapter 3

Implementing W2Share: Supporting Reproducibility and Quality Assessment in eScience

3.1 Introduction

Reproducibility denotes the ability for a third party to reproduce results of an experiment using a possibly different setting, aiming at confirming or disputing the original experimenter's claims [58]. As we know, scientific transparency and integrity rely on the ability to reproduce experiments, e.g., for independent validation, adoption of procedures or building new solutions to move forward in a particular research domain.

Scripts and Scientific Workflow Management Systems (SWfMSs) are common approaches that have been used to allow the automation of processes and data analysis in experiments. Scripts are widely adopted in many disciplines to create pipelines in experiments, e.g., to clean and analyze a large amount of data. However, they are hard to understand, adapt and reuse. For this reason, several solutions have been proposed to help experiment reproducibility for script-based environments such as ReproZip [22], YesWorkflow [57] and noWorkflow [61]. Though those solutions help scientists to capture experimental details, they neither allow to fully document the experiment and nor add new additional information such as support quality assessment of the experiments. SWfMSs [49], on the other hand, help reproducibility by supporting scientists in the design and execution of their experiments, which are specified and run as interconnected (reusable) components. However, there is a gap between the script and the workflow communities. Moreover, workflows alone are not enough to ensure reproducibility.

Taking this overall scenario into account, we designed W2Share, a framework for retrieval and conversion of script-based experiments into executable workflows [20]. The script-to-workflow conversion is based on a methodology proposed by us [12]. Reproducibility is enabled, via this methodology, by the adoption of Workflow Research Objects (WRO) [7]. The WRO model allows the aggregation of resources, explicitly specifying the relationship between these resources and workflow using a suite of ontologies. A WRO encompasses information such as datasets and provenance traces related to the execution

of workflows. In W2Share, a WRO also encapsulates the scripts that were transformed into workflows and quality annotations. Via W2Share, third-party users are thus able to understand the data analysis encoded by the original script and obtain the resources required to run or reuse the associated workflow and data.

This paper presents the first implementation of W2Share that implements script conversion and quality assessment. The prototype is available at `https://w3id.org/w2share`. It incorporates our work on Quality Flow, [69, 70] – a workflow-based computational framework for data quality assessment of scientific experiments. Our solution can be used in different situations such as: (i) publishing procedures and datasets related to an experiment; (ii) training members in a research group to gain skills in computational scientific procedures; and (iii) dynamically assessing the quality of experiments. This prototype was validated in a bioinformatics experiment. As discussed in the paper, through W2Share, we semi-automatically transformed a suite of R scripts into an executable workflow, which was annotated with quality information. Then, still under W2Share, several runs of this workflow were executed, each of which with potentially distinct quality annotations. The entire set (script, workflow, provenance traces, quality information) is encapsulated in WROs, that are stored in W2Share repository for WRO.

3.2 W2Share Instantiation

W2Share is an abstract generic framework to support executing and documenting experiments to enable their reuse and reproduction. As such, it can be instantiated in many different ways[1].

3.2.1 Overview

This section presents a specific instantiation, which moreover supports our methodology to guide scientists in the process of transforming scripts into workflows and their executions, with subsequent encapsulation into WROs. Figure 3.1 gives a high level overview of this instantiation, which is composed of three main modules: (i) **Script Converter** – responsible for guiding the scientist through the conversion of scripts into workflows; (ii) **WRO Manager** – responsible for creating, updating and exporting WRO bundles; and (iii) **Quality Flow** – responsible for annotating the workflow and provenance data with quality information, and creating quality according to users' needs.

These modules store and retrieve objects from the Knowledge Base (KB) using Semantic Web technology (in particular SPARQL queries). The KB encapsulates the WRO repository which includes scripts, workflows, provenance data, annotations, and input and output data; and the Quality Flow repository, which is responsible for storing quality dimensions, its metrics and creators. The KB is implemented using Virtuoso Open Source Edition[2].

The implementation uses a Model-View-Controller (MVC) architecture implemented

[1] For brevity sake, we do not present the overall framework. For details see [20].

[2] `https://github.com/openlink/virtuoso-opensource`

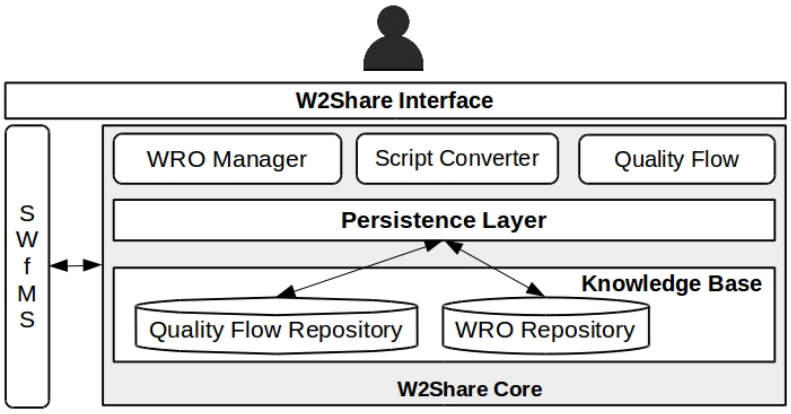


Figure 3.1: W2Share Software Architecture - workflow management is performed by the SWfMS.

using the PHP language and the Symfony Framework[3], a well established framework for web development. Among some of the tools and ontologies reused and/or extended in our software architecture are: YesWorkflow [57], Quality Flow [69, 70] and the WRO model ontologies [7]. We also created an ontology for annotating data quality information (see section 3.2.3). These modules take advantage of the SWfMS to manage, design and execute workflows.

3.2.2 Script Converter

Script conversion works as follows. First, the scientist annotates the script with YesWorkflow tags (e.g., @*begin* and @*end* to delimiter a given task). This annotated script is fed into the YesWorkflow tool suite, integrated by us into W2Share. This suite produces a set of Datalog facts, and a visual rendering of the script as a workflow. Next, we developed code to process these facts to produce the executable workflow that corresponds to that visual rendering. From then on, this workflow can be executed, updated and tested using the underlying SWfMS. Our present code only allows creating workflows that can be executed in the Taverna system [77]. We chose Taverna due to its popularity and availability of tools to ease the process of creating an executable workflow, exporting provenance data and integrating with the WRO.

3.2.3 Quality Flow

In [12], we state that there is a need to check the quality of the actual script-to-workflow conversion, and to assess the quality of the resulting workflow. In our solution, we adopt Quality Flow [70] to allow scientists to annotate the elements that make up the workflow with quality information. By doing this, scientists are able to compute quality metrics related to data, processes, and overall execution, and request evaluation of specific quality metrics based on combining quality dimensions with provenance information generated at each run of the workflow.

[3] `https://symfony.com`

Quality Flow is a software tool that allows domain experts to define their own quality dimensions and metrics for workflows and their components. Thus, a given workflow (and even a given execution) may have multiple quality assessments, depending on each expert's point of view. Data quality metrics may be computed as simple equations on numeric values or more complex as a set of inference rules. On demand, these metrics are used to compute a dimension of quality, without need to modify the workflow structure. Quality dimensions and metrics can be defined at workflow creation or, little by little, as distinct scientists interact with the workflow. Such metrics define how to calculate quantitative quality dimensions like accuracy and efficiency or how to relate and summarize different qualitative quality dimensions like reliability, utility and so on. W2Share embeds these features from Quality Flow[4] responsible for managing, extracting and processing quality information. W2Share links the quality-annotated workflow with its runs and (provenance) traces. Distinct scientists can assess quality differently, defining distinct quality metrics for a quality dimension for a given piece of data or process. As a consequence, the result of a single experimental run can be assessed multiple ways. These distinct assessments are embedded into a WRO (see Section 3.2.4) to be published.

We created an ontology[5] to represent the quality information generated, thus allowing the adoption of semantic web technology integrated with the WRO ontologies and supporting the construction of inference rules to calculate the data quality metrics. Basically, the main entities of the ontology are: Quality Dimension, Quality Metrics and Quality Annotation. Quality Dimension describes quality properties, such as freshness or understandability. Quality Metrics defines functions to compute a specific quality dimension. Quality Annotation associates a quality dimension and quality metrics with a workflow and/or elements of the workflow on request.

3.2.4 WRO Manager

This module implements the last step in the methodology, aggregating all resources used or produced in an experiment and their quality annotations into a reproducible and reusable WRO bundle. The WRO management capabilities supported by W2Share include: (a) creating a WRO bundle; (b) exploring a WRO bundle created or uploaded into the system; (c) annotating resources; and (d) exporting a WRO bundle for publishing on other repositories or for sharing it directly with other scientists. We adopted the RO Manager tool[6] to create the WRO bundle files.

3.3 Case Study: DNA Methylation Microarray Analysis

Overview of the experiment Epigenome-Wide Association Studies (EWAS) examine the epigenetic status of many *loci* (a set of positions on a chromosome) for a set of indi-

[4]Though, we used only some Quality Flow's features, we maintained the same name in the text.
[5]`https://w3id.org/w2share/ontologies/quality-flow.owl`
[6]`https://github.com/wf4ever/ro-manager`

viduals and assess whether any of these *loci* is associated with the phenotype of interest. To evaluate the feasibility of our instantiation, we implemented an EWAS experiment in W2Share. This experiment [72] was developed by the Biostatistics and Computational Biology Laboratory (BCBLab)[7] at Unicamp. It aimed at investigating how epigenetic marks change between two different tissues, prefrontal cortex and white blood cells, by assessing the DNA methylation profiles of control patients from a publicly available data set. The results were obtained comparing the profiles of these two tissues. The steps involved in a typical differential DNA methylation analysis pipeline include: quality control, filtering, data exploration, normalization and statistical testing for identifying differentially methylated regions (DMR).

This experiment was implemented via a script (GSE37579_analysis) in the R language. The script uses as input data: (i) the Gene Expression Omnibus (GEO) accession number GSE37579[8]; (ii) symbol names of genes of interest; and (iii) names of the sample groups (tissue names) used to filter the public dataset. Data outputs at each run are files containing high-resolution graphic charts for manual inspection and files containing tables of the DMRs identified when compared the sample groups.

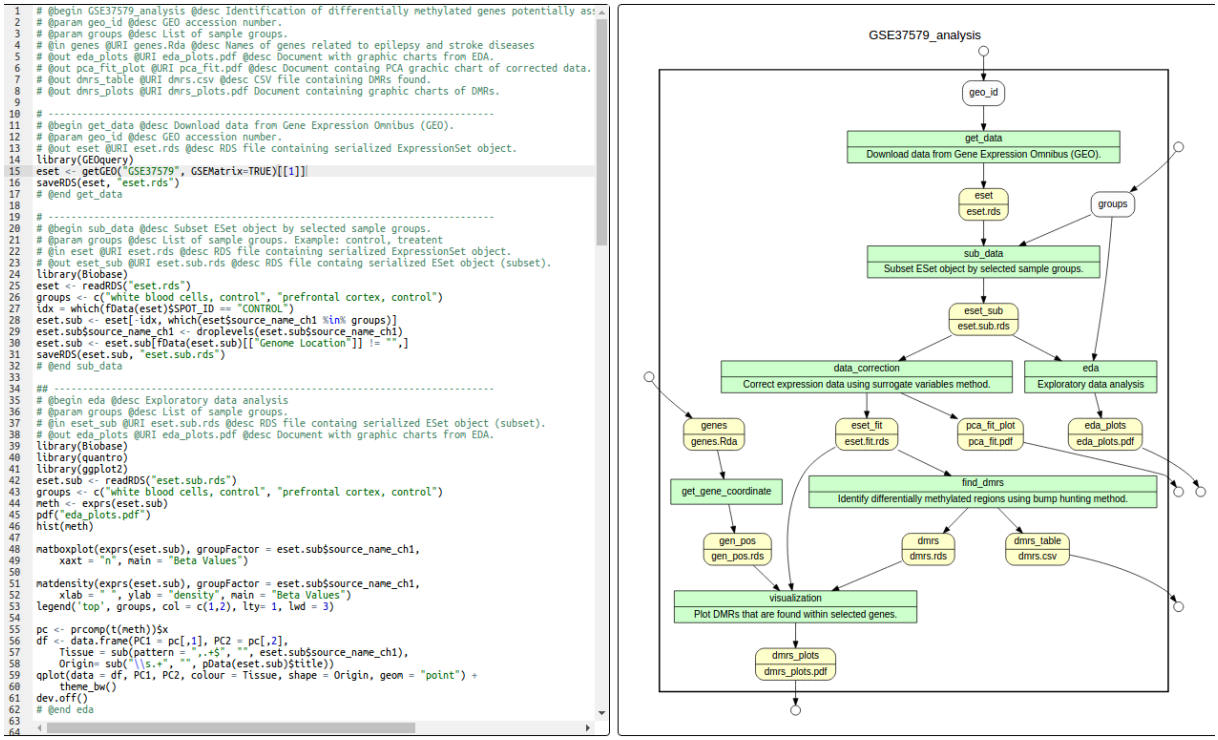


Figure 3.2: W2Share Editor for converting script into workflow.

Generating the executable workflow We first annotated script GSE37579_analysis using W2Share (see section 3.2.2). Figure 3.2 illustrates the annotated script (on the left side) and the abstract workflow generated (on the right side). Lines 11 to 17 from the script show tag @*begin* identifying *get_data* activity, @*desc* describing the activity as "Download data from Gene Expression Omnibus (GEO)", @*param* and @*desc* describing

[7] `http://bcblab.org/`

[8] http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE37579

the parameter *geo_id* as "GEO accession number." and @*out*, @*uri* and @*desc* specifying the file *eset.rds* to the output port *eset* and describing this port as "the eset RDS file containing serialized ExpressionSet object". This first activity of the script is represented in the abstract workflow as the first green box, where the first row shows *get_data* and the second one, the description of the activity obtained from the @*desc* tag.

W2Share next automatically creates the corresponding executable workflow[9]. In the current version of W2Share, we run the workflow using the Taverna workflow system – and thus, we capture all provenance information provided by Taverna.

Annotating with Quality information At any point, different scientists can annotate the workflow and components with quality information. Figure 3.3 shows a form to annotate the process *find_dmrs* with quality information.

As an example of how this form works, consider that at some point, a scientist *Lucas* defined the *accuracy* quality dimension. Scientist *Joana* retrieves from W2Share the information about GSE37579_analysis. *Joana*, then, annotates the process *find_dmrs* with the *accuracy* dimension value 0.85. The same scientist defines that the metrics to compute *accuracy* are (*correctly_identified_methylate_regions* / *total_selecting_samples*). At present, quality dimensions and metrics are stored as textual descriptions. This allows any kind of dimensions and metrics to be defined using W2Share. However, our approach still requires of manual specification of quality information. To overcome such a limitation, as future work, we intend to design a semi-automatic approach to define and assess the quality metrics associated with a dimension.

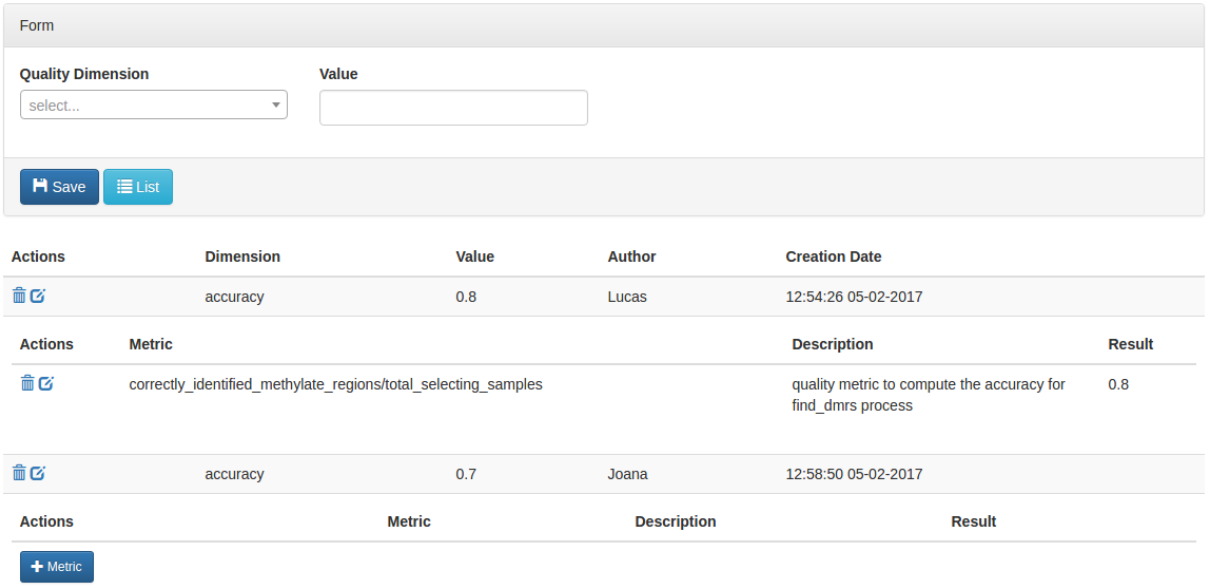


Figure 3.3: Quality Annotation form on W2Share.

Generating the WRO Finally, the scientist generates the WRO, which is stored in the corresponding W2Share repository. This output is available at `https://w3id.org/w2share/usecase-bresci2017`.

[9]Our implementation generates Taverna workflows

3.4 Related Work

There is a variety of work addressing reproducibility of script-based experiments. For instance, ReproZip [22] helps users to capture all the necessary components, environment variables and library dependencies used to execute data analysis, bundling them in a single, distributable package. However, unlike W2Share, ReproZip does not have a web interface to allow the exploration and annotation of the resources included in the package. Our solution provides a friendly web user interface to explore WRO bundles. noWorkflow [61] captures the execution provenance of Python scripts to support reproducibility. Our solution is not limited to a specific script language and we use the SWfMS to capture the provenance data related to the experiment execution. Furthermore, ReproZip and noWorkflow lack features to allow scientists to understand main script components to learn about the experiment. YesWorkflow [57] is a tool that allow scientists to annotate their scripts to generate a workflow-like graphic view of the data analysis carried out by the script. In our work, we extend YesWorkflow features to allow transforming scripts components into executable workflow elements, storing the scripts and the abstract workflows into a WRO for reproducibility.

We use WRO as a component to package and publish the information of the experiment on W2Share. myExperiment [28] and ROHub [63] are public web repositories that store Research Objects. ROHub is a repository for general-purpose Research Objects, whereas myExperiment stores WROs. Both repositories lack features such as annotation of quality information, available in W2Share, and none of them have a focus on exploring the provenance data of executions. Our solution is not limited to providing a repository of WROs, but also a system able to manage and enrich the WROs with quality information, activity descriptions, among others. The quality assessment facilities in W2Share are based on Quality Flow [70]. We adapted Quality Flow to semantically represent its data and to work in an integrated manner with the modules of W2Share.

3.5 Conclusion and Future work

This paper presented our implementation efforts to instantiate the W2Share framework, which supports scientists in transforming their scripts into executable workflows, and assessing the quality of workflow runs. Our instantiation is enhanced with Quality Flow's features to annotate the workflows, the processes and the output data with quality information. This allows users to assess the quality of an experiment and create WROs extended with quality information to be consumed by other scientists. To the best of our knowledge, this is the first attempt to fully integrate use-tailored dynamic quality assessment to a reproducibility environment. Our implementation strived to reuse software tools, standards and ontologies developed by the scientific community. As such, the incorporation of quality assessment features is one of our major contributions of this work. Nevertheless, much need to be done to meet full support to reproducibility and quality management. One possible direction is to ensure support to data citation standards. Full reproducibility may moreover encompass preserving the original execution environment (e.g., variables and software configuration).

Automatic comparison of the quality of the experiment results based on the original script and the workflow is also left as future work. Finally, we desire to explore Common Workflow Language (CWL)[10] to create executable workflows to use a standard that works across multiple SWfMS.

[10] http://www.commonwl.org/

Chapter 4

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

4.1 Introduction

Scripts and Scientific Workflow Management Systems (SWfMSs) [29, 23] 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 [46], ReproZip [22], YesWorkflow [57], and noWorkflow [61].

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 [31]. SWfMSs [49], 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

[1] In this paper, the term *experiment* refers to scientific experiments that are executed *in silico* – e.g., simulations.

workflow research objects (WRO) [6]; 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 [6] 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 [4, 8, 21]. 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 [33]: (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.

[2]This methodology was described in [12], and is included here for completeness.

This paper extends previous work [12] 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 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 [48], 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 4.2 introduces our methodology specification. Section 4.3 presents the model that describes its main elements. Section 4.4 describes the case study and the implementation of the methodology steps. Our conversion steps are described in Sections 4.5 and 4.6. Section 4.5 describes the first step, showing how we map scripts to abstract workflows using ontologies. Section 4.6 shows how an executable workflow is generated from abstract workflows. Section 4.7 presents the evaluation of our proposed approach. Section 4.8 discusses related work. Section 4.9 summarizes our results and identifies future work.

4.2 Methodology for Script Conversion into WRO

Parts of sections 4.2.1 and 4.4 appeared in our paper [12] 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.

4.2.1 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) [7]. WROs are the means through which experiments can be reused, audited and documented. As mentioned in Section 4.1, 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:

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

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 carried out to transform the script into a workflow that embodies the script. Moreover, the workflow variants also need to be recorded. As stressed by [58], 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 [71, 53].

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

We follow the definition of [58]: "reproducibility denotes the ability for a third party who has access to the description of the original experiment and its results to reproduce

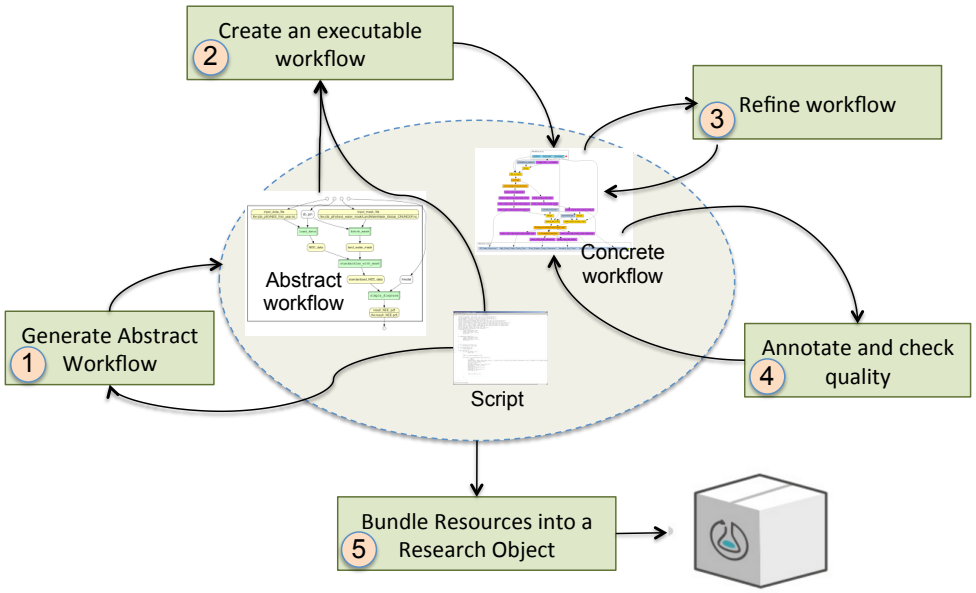


Figure 4.1: Methodology for converting scripts into reproducible Workflow Research Objects, extracted from [12].

those results using a possibly different setting, with the goal of confirming or disputing the original experimenter's claims." Missier et al. [58] 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.

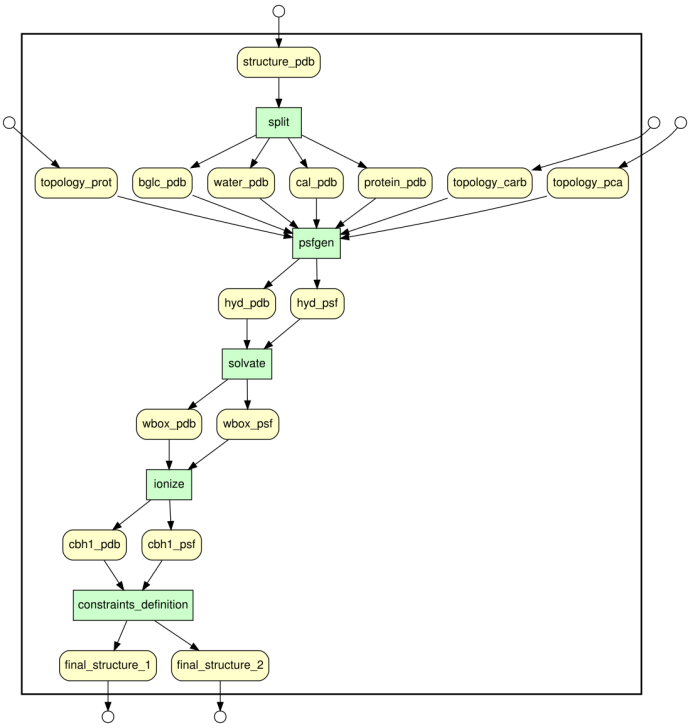
Given the five requirements, we proposed our methodology composed by five interrelated steps (see figure 4.1). Step 1, **Generate abstract workflow**, is used to produce an abstract workflow Wa based on a script S provided by a user. This stage elicits the main processing units that constitute the analysis 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 4.2 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

```
# @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 proteir
#                               @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)

Figure 4.2: 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.

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 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 [14, 16], 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 work-

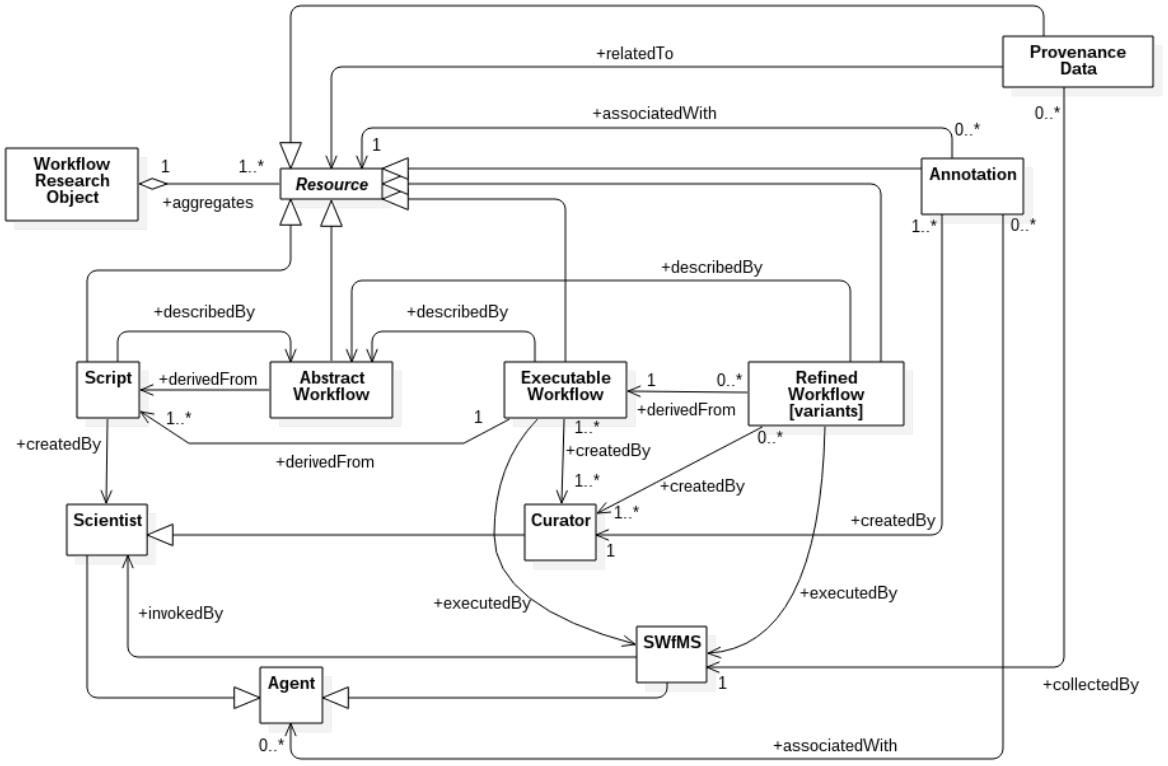


Figure 4.3: Model describing main elements and relationships used in our methodology.

flow 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.

4.3 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 process, 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 4.3 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.

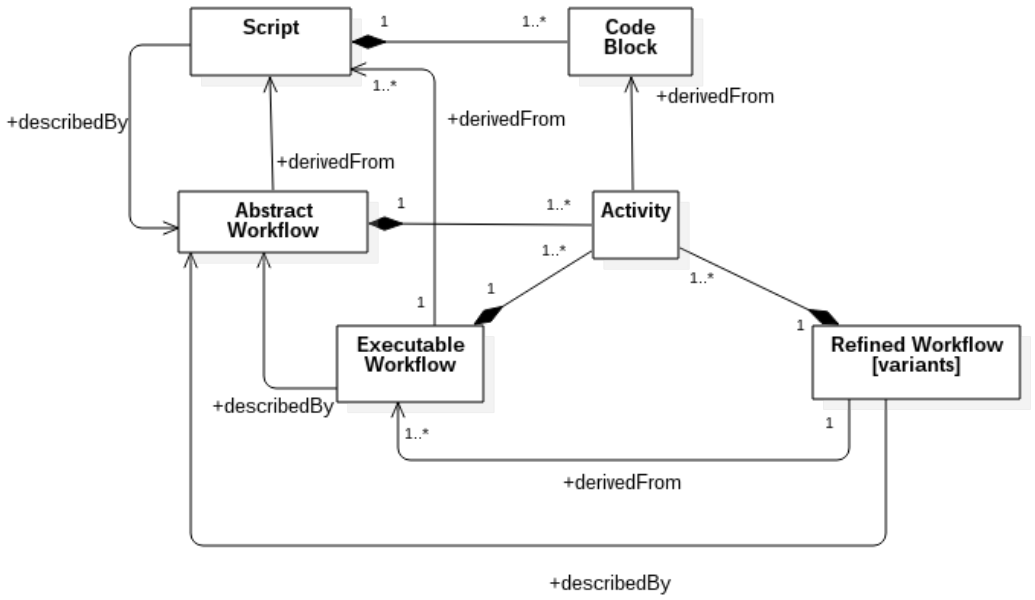


Figure 4.4: Relationships between main entities regarding the tracking of the script-to-workflow conversion in our model.

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 4.4 shows a UML class diagram that refines part of figure 4.3, 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 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.

4.4 Case Study – Molecular Dynamics

4.4.1 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 [68]; 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 [12].

4.4.2 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 [57] and Research Objects. Its implementation includes elements of the PROV ontology, thereby facilitating provenance annotation. This is presented in Section 4.5 of this paper, and is one of the paper's contributions.

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

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

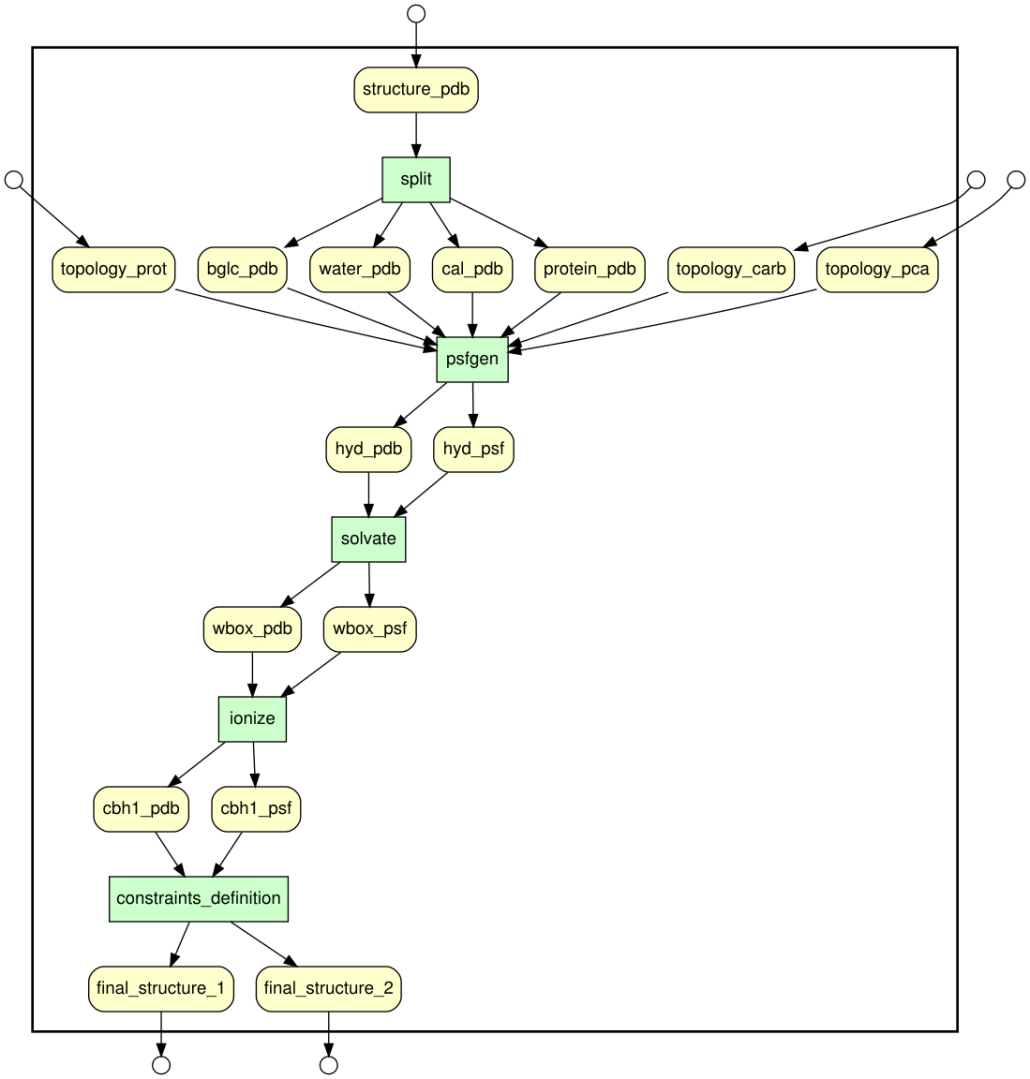


Figure 4.5: Visualization of the abstract workflow of our MD case study, extracted from [12].

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 4.5 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.

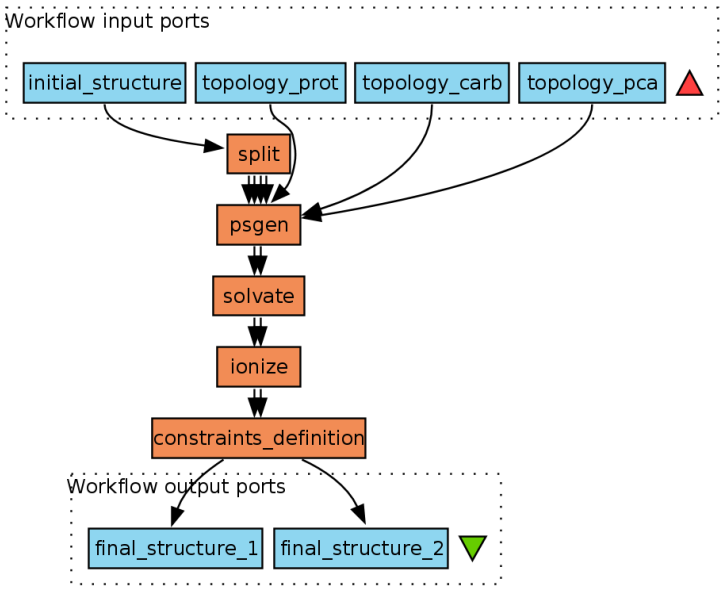


Figure 4.6: Executable workflow of our MD case study, extracted from [12].

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, 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 [12], is now supported by W2Share – the semi-automatic implementation of Step 2 is detailed in Section 4.6.

Without loss of generality, we used the Taverna system [77], 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 4.6 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 myExperiment repository[5] a small workflow that fetches a

[5] `http://www.myexperiment.org`

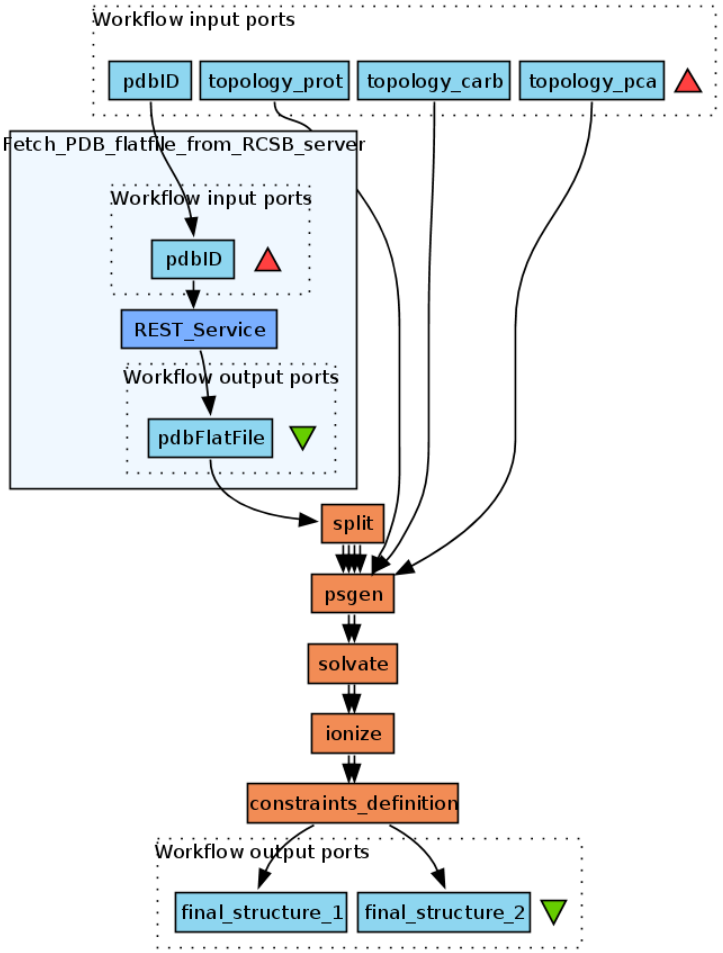


Figure 4.7: Refined workflow of our MD case study, extracted from [12].

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 4.7 for the workflow variant We_1).

Here, the *initial_structure* input parameter of figure 4.6 (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 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

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

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 [17] 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 4.5 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 [6, 7] 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 [18].

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.

4.5 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).

[9] `https://github.com/wf4ever/ro-manager`
[10] `https://w3id.org/w2share`
[11] `http://www.rohub.org/`

Table 4.1: Mapping of YesWorkflow tags to workflow ontologies

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

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 4.4.2 shows how a scientist can easily transform a script into an abstract workflow with help of the YesWorkflow suite of tools [57, 56]. 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).

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 [6], a workflow specification ontology from the Research Object suite of ontologies [7], 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 4.1 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 4.8 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

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

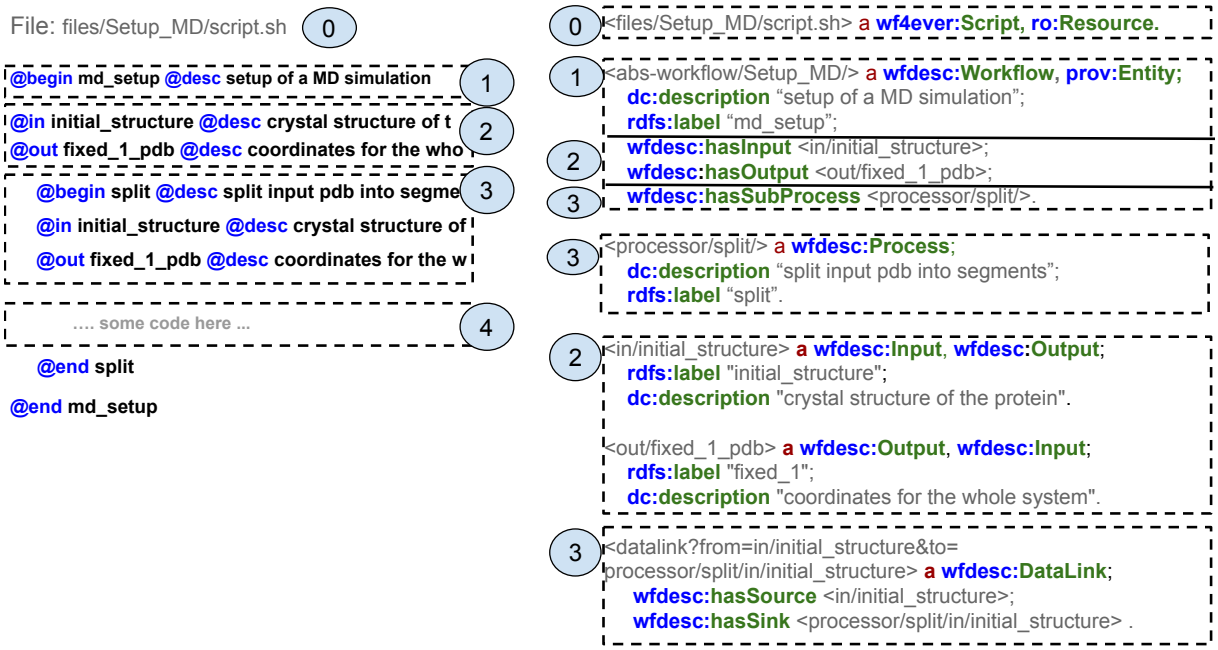


Figure 4.8: Mapping between YesWorkflow tags (left side) and classes and properties of ontologies (right side).

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 test to go` is mapped to property `dc:description` with value `test to go`.

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 [48]. Each abstract workflow element is defined as a `prov:Entity`. Again, in Figure 4.8, 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 Appendix A shows an excerpt of Wa and Listing A.3 in Appendix 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.

4.6 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 [17]; here we provide a detailed description.

As presented in Section 4.5, 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 Appendix 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.

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

4.7 Evaluation

4.7.1 Overview

To validate our proposal, we adopt the notion of *competency queries*, e.g., as defined by [73] "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.

4.7.2 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 [7]. The competency questions, the SPARQL queries and results, and the RDF statements representing our case study can be found online in [19].

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.

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 4 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 4: Query 1 translated into a SPARQL

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

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

```
4        wfdesc:hasInput ?winput;
5        wfdesc:hasOutput ?woutput ;
6        wfdesc:hasSubProcess ?process .
7     ?process wfdesc:hasInput ?pinput;
8              wfdesc:hasOutput ?poutput .
9  }
```

Table 4.2 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 4.2: 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 5: Query 2 translated into a SPARQL

```
1  select ?workflow
2  where {
3     ?workflow a wfdesc:Workflow;
4           prov:wasDerivedFrom <files/Setup_MD/script.sh> .
5  }
```

Listings 5 shows the SPARQL code for this query. Table 4.3 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 4.3: Result of evaluating query 2

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

Listing 6: Query 3 translated into a SPARQL

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

Table 4.4 shows the results obtained by evaluating the query. The results point out `<workflow/Setup_MD>` originated the variant `<workflow/Setup_MD/variant>`.

Table 4.4: 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 7 shows the SPARQL code that can be used for answering this query.

Listing 7: Query 4 translated into a SPARQL

```
select ?script ?abs ?process ?start ?end
where {
   ?abs prov:wasDerivedFrom <> ;
      wfdesc:hasSubProcess ?process .
   ?process prov:wasDerivedFrom ?code .
   ?code oa:start ?start ;
```

```
7       oa:end ?end .
8  }
```

Table 4.5 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.

Table 4.5: 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

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).

This query is useful helping to establish trust in the conversion process. Listings 8 shows the corresponding SPARQL code. Table 4.6 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 8: Query 5 translated into a SPARQL

```
1  select distinct ?curator
2  where {
3    <abs-workflow/Setup_MD/> prov:wasAttributedTo ?agent .
4    ?agent foaf:name ?curator .
5  }
```

Table 4.6: Result of evaluating query 5

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 9 shows the SPARQL code for this query. Table 4.7 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 9: Query 6 translated into a SPARQL

```
SELECT DISTINCT ?workflow ?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 4.7: Result of evaluating query 6

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 4.7 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 10: Query 7 translated into a SPARQL

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

4.8 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 [12], we described a preliminary manual transformation from script to executable and verifiable workflow. We adopted YesWorkflow

Table 4.8: 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

to generate the abstract workflow visualizations before creating the executable workflow. YesWorkflow was developed by McPhilips *et al.* [57, 56]; 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 [26]. 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, [4] 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 [8] 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 [8] and [4].

In [21], 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 recommend 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 [17], 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 [34] 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.

4.9 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 [17] 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.

Chapter 5

Semantic Software Metadata for Workflow Exploration and Evolution

5.1 Introduction

Workflow management systems [39] play a major role in supporting scientists to design, document and execute their computational experiments. During workflow design, scientists use third party software or their own code to implement workflow components. This paper investigates the issues that arise when such software evolves in terms of how a scientist's workflow is affected.

There are many reasons for scientists to modify a workflow that they created, either by changing specific steps of the workflow (also called *workflow components*) or changing the workflow structure. Changes in software used to implement components are common and could happen for different reasons, e.g., a newer version is available, older software is not maintained. Also, data sources change, e.g. when datasets are updated with new formats, which may require adjustments in existing components and adding new ones. Thus, due to changes in software and data, workflows must be updated accordingly to avoid workflow decay [79] and reproducibility issues [36]. Another important reason to update workflows is when scientists are exploring alternative ways of performing a computational experiment. During these exploratory tasks, scientists often want to compare methods or try different approaches to implement a workflow component.

In current workflow systems, scientists manage these updates manually. However, updating a workflow is a complex and time-consuming task, as it requires tracking down information about the different versions of software and functions used in the components of the workflow and understanding the impact in other workflow steps. In previous work [15], we elicited a set of requirements for supporting the exploration and update of workflows motivated by hydrology workflows and their use of models with very different versions over the years. These requirements motivate the need for capturing additional metadata for better describing software used in workflows, such as software functionality and implementation changes over time. This paper revisits those requirements and makes them more specific through a detailed scenario in which a decades-old machine learning software is used in a workflow for weather prediction. Those requirements guided the

design and implementation of OntoSoft-VFF, a semantic software metadata catalog to help scientists to manage workflow evolution and updates. OntoSoft-VFF is based on a novel ontology for representing software metadata. Our goal is to use the information in OntoSoft-VFF to support scientists in selecting appropriate pieces of software to implement a given workflow component, to explore the use of alternative software in their workflow, and to keep track of all workflow changes. OntoSoft-VFF's catalog extends OntoSoft [38], an existing metadata catalog designed for fostering scientific software reuse and sharing.

The main contributions of this paper are the following:

- Through a scenario in which a scientist updates a computational experiment, we revisit requirements for software metadata to describe software that implements workflow components.

- A software metadata catalog developed for those requirements. The catalog is based on a novel ontology designed to describe software functionality and its evolution. The catalog supports comparing and searching semantic metadata for software.

To illustrate our work, we use a running example of machine learning workflows, since it is a domain with many alternative methods available where software changes frequently. This example is used to present the elements of our ontology and the features available in the catalog. We show how to create and update workflow components using the semantic metadata stored in our catalog, and the benefits of integrating the metadata catalog with a workflow system to support scientists in their exploratory tasks. Finally, we validated our approach showing how it addresses our requirements. OntoSoft-VFF has been designed to be generic and can be applied to any scientific domain.

Throughout this paper we adopt the following terminology:

- *Software* is a set of functions that perform similar or related computations and are delivered as a package by developers. An example of software is Weka [43], a decades-old open source Java software with a widely used collection of machine learning algorithms for data mining tasks. A more modern and also popular software is the Scikit-learn Python libraries [64].

- *Software version* is a unique state of a software as it is being released. For example, the latest Weka release is version 3.9.2.

- *Functionality* is a conceptual computation or operation that can be performed by a piece of software. For example, Weka implements classification, regression, and clustering functionalities.

- *Software function* is the implementation of a functionality in a software. An example is the Weka J48 function that implements a classification functionality using the C4.5 decision tree algorithm [65].

- Software change is a relevant modification associated with a software function over time. An example of a change is the improvement of accuracy in the result of a J48

classifier function. A new software version may imply software changes as well as modified, new, or deprecated functionalities or functions.

The rest of the paper is organized as follows. Section 5.2 introduces related research on workflow updates and software representation. Section 5.3 describes the main scenario we are addressing, expanding the requirements derived from previous work. Section 5.4 introduces OntoSoft-VFF. Section 5.5 shows how we exploit the data published in the catalog to facilitate workflow exploration and evolution. In Section 5.6 we validate our framework against the requirements in Section 5.3. Finally, we present our conclusions and future work in Section 5.7.

5.2 Related Work

Our discussion of related work covers two areas: approaches for workflow exploration and updates, and approaches for software and representation of software changes. The most common shortcoming of these approaches is the lack of appropriate metadata to account for and appropriately track software evolution, thereby placing a major burden to scientists in managing workflow evolution. Even when metadata exists, it is not designed to support workflow exploration and updates.

Workflow systems [39, 62, 1, 66] are mostly concerned with workflow construction, execution and provenance collection and inspection. Vistrails [47, 66] uses a software registry that stores the software name and version identification to support workflow upgrades. However, this registry does not track changes in terms of functions and functionality, nor store information about semantics of inputs and outputs, such as data types and data formats. Such kinds of metadata are necessary to support more robust approaches to update a workflow with components for data transformation and other upgrades.

In [76], the authors proposed a framework for management of knowledge associated with workflow evolution. The framework, however, does not track changes in the software used to implement the workflow components, thus missing the opportunity to relate the effects of changes in software to the outputs of workflow components.

Understanding how results have been produced requires knowledge of the software being used. Work such as [37, 52, 25] proposes mechanisms to represent software; however, they lack metadata for describing changes in software, and how to use specific software functionalities. Their representations define inputs and outputs for the software in general, rather than defining the inputs and outputs for a function. OntoSoft [38] is a software registry that is concerned with representation, sharing and reuse of software metadata. The software representation used does not capture information about software functions and software changes over time.

Regarding software updates, software version management systems [25] track changes in software code. However, changes represented in these systems do not provide enough information to allow a scientist to filter them by software function, for example, and track down the changes through time associated with a specific function or functionality. Also, it is hard for a scientist to track the issues and bug fixes related to a specific function in a software version, because version control systems do not explicitly represent the functions

available in the versioned software.

The most common shortcoming of these approaches is the lack of appropriate metadata to account for and appropriately track software evolution, thereby placing a major burden to scientists in managing workflow evolution. Even when metadata exists, it is not designed to support workflow exploration and updates.

5.3 Motivating scenario – revisiting requirements for workflow exploration and updates

In this section we show how scientists explore new functionalities of software, so they can upgrade workflow components using these new functionalities. In order to achieve this goal, scientists usually go through a series of tasks that are performed manually and without any support. They start by identifying the function and software used in a workflow component, and understanding the algorithms that they implement. Scientists also need to find and compare similar software versions and functions. Finally, when deciding either which workflow component to upgrade or which software function to use to implement a new component, scientists need to be able to create or upgrade components and to change the rest of the workflow as needed.

To illustrate the needs for supporting the management of workflow exploration and evolution, we use an example of a workflow designed to process weather data to make weather predictions. We chose this scenario for several reasons. First, it is a simple domain-independent scenario chosen to simplify our presentation, yet it captures the complexities that we have seen in our prior work with hydrology modeling software and workflows [15]. Scientific modeling software has the same issues but also additional subtleties as discussed in [15]. Second, the scenario involves machine learning algorithms, and consequently a large choice of options of algorithms (and thus many exploration and update options). Lastly, we had implemented a variety of workflows in previous work (e.g., [37, 44, 67]) using older versions of Weka, as well as new ones using Scikit-learn and other machine learning libraries [40]. The workflows run in the WINGS [39] workflow system.

In our scenario, Alice, a meteorologist in California, wants to predict weather for the city of Pasadena. She starts with a very simple workflow, shown in Figure 5.1, that had been used to process 2007 weather data from Santa Monica to make weather predictions for Pasadena, both cities located in California. The workflow no longer runs, and Alice would like to update it.

The workflow contains two workflow components (*J48Modeler* and *J48Classifier*) from Weka that use the C4.5 decision tree algorithm. The first component uses it to learn a decision tree model from training data (the *trainingData* input), while the *J48Classifier* uses this learned model to classify test data (the *testData* input). The *ClassIndex* parameter, used as input for both workflow components, specifies that the feature in a specific indexed column is the one we are trying to predict. The workflow also contains two components to ingest data since Weka uses a special comma separated data format called ARFF (Attribute-Relation File Format). In the discussion that follows, we will describe

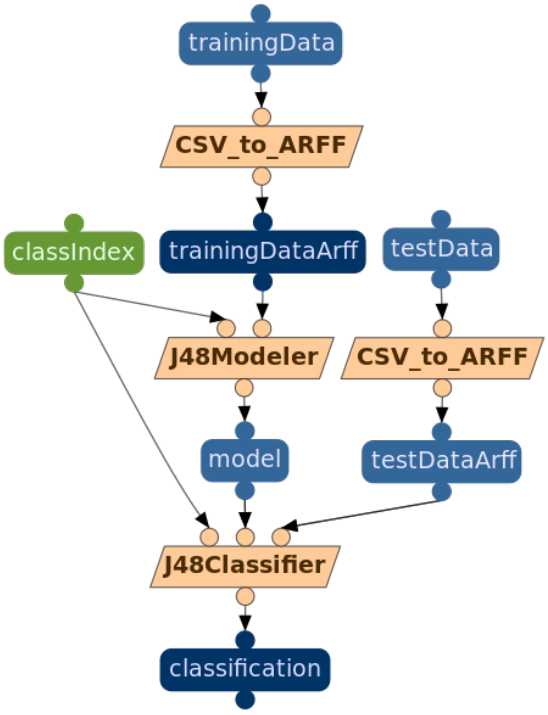


Figure 5.1: A very simple workflow using a decision tree machine learning algorithm for training and classification.

how in order to decide whether (and how) to upgrade the workflow, Alice needs to go through the documentation provided by Weka in quite a bit of detail.

5.3.1 Finding which software is used in workflow components

A workflow does not include much information about the software that implements each workflow component, such as software function and version invoked. This information could help a scientist like Alice to upgrade the workflow component – e.g., to decide whether and how to upgrade to a new version.

Alice found in the workflow documentation that Weka is used to implement the workflow and that both workflow components invoke the Weka *J48* functions. Alice has to read the Weka documentation to understand that these functions implement the C4.5 decision tree algorithm. To manage the workflow evolution, she needs to know not only the software and algorithm used to implement the workflow components, but also the specific software version and functionalities invoked in Weka to implement each of the workflow components. This information, however, is not explicitly captured by scientific workflow systems, which usually only store the code that invokes the software. So, either Alice relies on the documentation, or she looks at the actual component code to find out how the software has been invoked and which version is being used. The latter information, even if available, is not easy to infer from an invocation code.

In some cases, the function invocation can help to find out the code that is being used, but unfortunately this does not help a scientist to understand what the function does or the semantics of the inputs and outputs and of their data types or formats. This information is important for scientists to know what kind of data is needed to run a

component as well as to check compatibility between workflow components.

Alice found out by looking at the code that both components are implemented using Weka version 3.6.2, which was released in 2010. Through further analysis, she discovers that the *J48Modeler* and *J48Classifier* components are implemented invoking the J48 Java class in Weka, which is located in the *weka.classifiers.trees* package and uses the C4.5 decision tree algorithm. Now, she is ready for the next step in deciding whether to upgrade the workflow – assessing the impact of changing software versions.

5.3.2 Understanding differences between software versions

Important changes may have been made between software version releases, such as the addition of new functions and changes in function interfaces which affect their invocation. To check whether and how to upgrade a workflow, scientists often read release notes where software developers usually describe differences between software versions, e.g., bug fixes or performance improvements. However, release notes may be generic and are not designed to allow scientists to quickly determine how general software changes affect their particular workflows.

Alice is more interested in stable versions than development versions, since the latter are more likely to suffer from bugs. However, stable versions are likely to suffer of delays in receiving new functionalities since they are tested in development versions first. Minor and patch versions, compared to major versions, usually provide backward compatibility changes that should not affect function interfaces, making workflow upgrades much easier. Changes in software interfaces may in turn affect the implementation of the interfaces of the workflow components. The solution is either to recode the component, or to create additional workflow components for data transformation to make interfaces compatible between components again.

The latest Weka version is version 3.9.2, released in December 2017, a minor development version. If Alice wants a stable version, she needs to choose one from several available releases, ranging from 3.6.3 to 3.6.15 (13 versions) and 3.8.0 to 3.8.2 (3 versions). The major version upgrades are 3.8.0 and 3.9.0. The former is stable and the latter is a development version. None of this information is readily available, and Alice needs to spend time analyzing the Weka documentation.

The final step in assessing choices for workflow upgrades is to decide whether to modify the software functions adopted, choosing alternatives with similar characteristics.

5.3.3 Finding similar software functions

New software functions may implement new functionalities and use new algorithms, thus opening new opportunities for the design of scientific experiments. Scientists can explore such new functions, for instance, to carry out slightly different computations in the workflow and compare the execution results. In our scenario, Alice wants to try other software functions that implement the same classification functionality but use different algorithms.

She would like to find software functions similar to the ones implemented in J48Modeler

and J48Classifier. To do so, she will have to decide whether to check for alternative Weka implementations, whether to look in other software libraries. Weka functions are more likely to accept the same data format and data type for inputs and outputs, and for this reason she restricts herself to choosing from Weka options.

5.3.4 Understanding differences in software functions

To decide whether to upgrade a workflow (and which components should be updated), Alice needs to understand the differences between versions of Weka functions. Therefore, Alice needs to know what has changed in Weka since the release of version 3.6.2. Since the creation of a working workflow from scratch is a time-consuming task, especially when the workflow may contain many data preparation steps, an upgrade (and thus version comparison) is worth the effort.

Changes in functions may include function renaming, addition of input parameters, and support for different input data formats. All these kinds of changes may affect the implementation of existing workflow components.

Alice goes through the versions of the software functions in Weka 3.6.2 and 3.9.2 using the Weka command line interface, the software manual, release notes and the Javadoc documentation for help. Alice figures out that the only change needed is to update the Weka library used in the workflow. The function interfaces from Weka 3.6.2 and 3.9.2 versions are exactly the same, though this is not typically the case especially after several years have passed.

Versions may include other modifications beyond changes in software function invocation. For instance, important changes may be related to performance and accuracy, which should ideally be described in release notes.

5.3.5 Identifying known issues, bug fixes affecting software functions

A scientist may adopt a new version if it fixes some bug. However, a new version may have unintended side effects, such as affecting other functionalities in the workflow. Alice is interested in releases containing fixes to bugs that affect her workflow. Once again, she needs to read release notes and look at the version control repository used by the Weka project. This control version repository is used to track issues and create bug fixes associated with code.

5.3.6 Creating a workflow component to explore new software functions

Alice decided that she wants to use a different classification method the ID3 decision tree algorithm. To do this, she needs to create new workflow components to replace the ones that use the C4.5 decision tree algorithm in her workflow. For this, she needs to know how to implement a workflow component in the specific workflow system (in our case, WINGS) using the Weka software (i.e., which inputs and parameters to use, which

outputs to expect, and what code to invoke). She also needs to know how to invoke the appropriate Weka function.

She chooses the ID3 Java class, which implements the desired classifier. However, she does not know which inputs to use to implement the modeling and training components.

This Java class has several possible input parameters and datasets. The combination of inputs allows it to perform different functions according to the inputs used, such as create a model using a training dataset and classify a dataset using an existing trained model. The appropriate combination of inputs which to carry out a specific function can only be found by reading the manual of the software or talking to an expert in Weka. Using Weka version 3.9.2, she can create the desired component by invoking the ID3 java class in the package *weka.classifiers.trees*.

Then she needs to create the component's I/O and manually map the I/O to the corresponding function I/O. This kind of process is time-consuming and error prone. Any scientist who wants to create a new workflow component needs to go through the same process.

Alice learns about the Scikit-learn software, and wants to consider using it. This is very common, particularly in science where alternative models and libraries may provide numerous options that are often better than upgrades to older code. Alice consults the documentation of Scikit-learn and sees that it does not use the ARFF format. It is typical that using new functions from other libraries in a workflow may require additional data transformation components in the workflow. In this case, the conversion steps from CSV to ARFF in the workflow are no longer needed.

5.3.7 Requirements

In previous work we gathered several general requirements regarding workflow component metadata, workflow updates, and workflow comparisons [15]. Here, we focus on the first aspect: the requirements to capture the characteristics of the software that implements workflow components.

More specifically, we focus on describing the software used in these components, and its evolution through time in order to support workflow exploration and evolution. Building on the scenarios from [15] and summarizing the scenarios above, we formulate the following requirements:

- R1 – Workflow descriptions should capture the software, software version, and functions used in the implementation of workflow components.
- R2 – Scientists should be alerted about relevant updates of software used in their workflows.
- R3 – Version descriptions should capture metadata about differences between software functions, particularly about their interfaces.
- R4 – Given a software package that can be used to create many workflow components, scientists need to easily figure out how to implement a component and how to update an existing component with newer versions of that software.

- R5 – Scientists should get a summary of changes between two given software versions to understand their differences without having to understand the history of changes associated to each version in between the old and the chosen one.
- R6 – Version descriptions should capture bug fixes and known bugs and relate them to specific software functions.

These requirements highlight the need to have metadata associated with software packages, their versions, the software functionality and functions that they implement, and the software changes done to specific functions in new versions.

The scenario we described reflects the difficulty scientists face to assess how, when and whether to upgrade their workflows or not, even when they are code-savvy. We point out that these difficulties are not specific to the choice of Weka or any other software used by scientists. Rather, the scenarios highlight that such software package repositories are designed to support code sharing and tracking, presenting technical details to programmers rather than efficiently highlighting conceptual descriptions to scientists. In other words, scientists lack a more structured and function-based representation of software to help them to design, upgrade and understand workflows. Moreover, in version control repositories, documentation is not provided in machine-readable format that can be used by a workflow system to assist the scientist in exploring and managing the evolution of a workflow.

The next section describes OntoSoft-VFF, the framework we designed to address the requirements presented in this section.

5.4 Ontosoft-VFF: A framework to help scientists to explore and update workflows

We designed and developed OntoSoft-VFF (Ontology framework for Software Version, Function and Functionality) to address the requirements described in Section 5.3. This framework is based on a novel ontology, which is used to construct a semantic metadata catalog. OntoSoft-VFF extends OntoSoft [38], a framework composed of an ontology and a metadata catalog that aims to describe software metadata to support scientists to share and reuse software. Our extension to OntoSoft include both the novel ontology and associated services to describe software versions, functions, functionality and changes to software. OntoSoft-VFF provides the semantic information needed by scientists to explore their workflows, and to assess whether and how to update them, fully supporting the needs exemplified in the scenarios of Section 5.3.

Figure 5.2 shows an overview of our ontology. We represent the elements already present in OntoSoft using the namespace (*sw*: `http://ontosoft.org/software#`), whereas our new ontology uses the namespace (*vff*: `https://w3id.org/ontosoft-vff/ontology#`). The ontology contains terms to describe:

- Software metadata: represents software, its relations with software versions, and other relations such as with operating systems, programming languages, and any software dependencies.

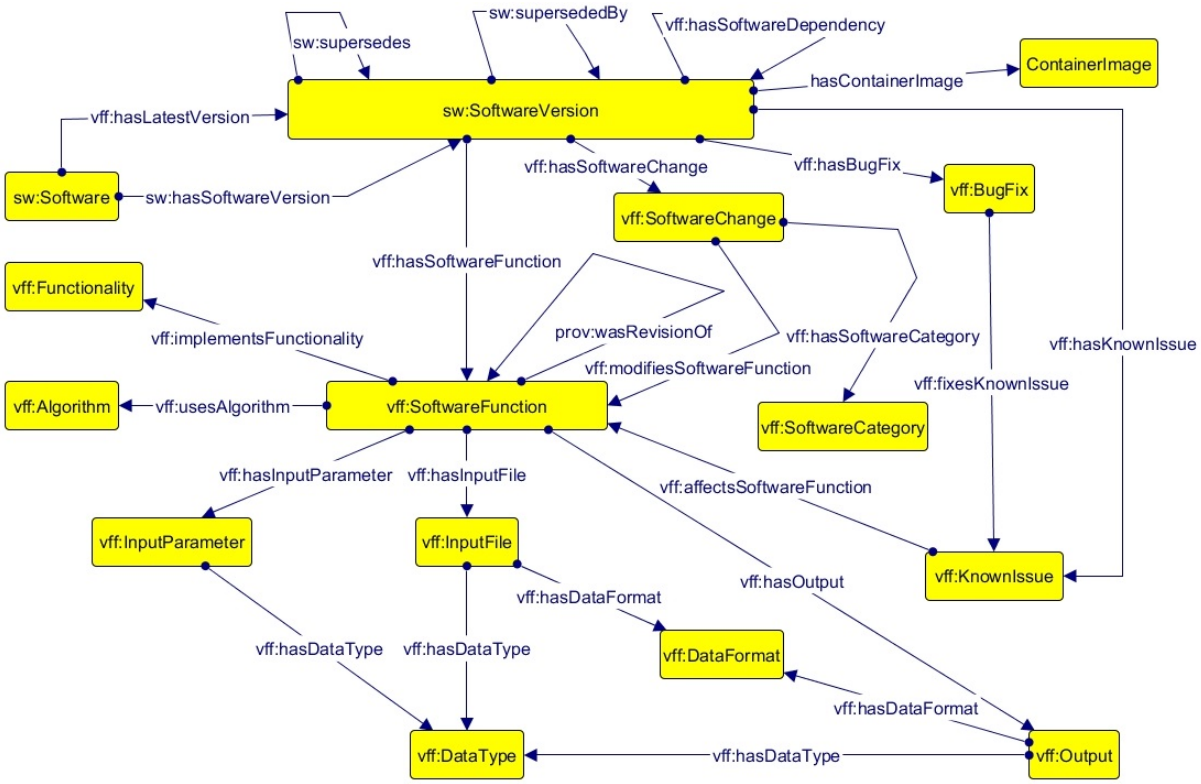


Figure 5.2: Diagram with the representation of the main classes and relations of our ontology.

- Software version metadata: includes the metadata for a given software version including new functionality and functions.
- Software function metadata: includes metadata regarding functions released in software versions and their inputs and outputs.
- Software change metadata: includes metadata for representing changes in software versions over time, including known issues and bug fixes.

A major contribution of our work is to model software used in workflow components with respect to its functionality and evolution over time. In the following sections we focus on the relevant classes and relations specified in the ontology to address our requirements. Due to space limitations, we will only illustrate and describe the classes and relations related to the goals of this paper. For the same reason, we have retained here only the parts of OntoSoft that are relevant to this discussion.

The ontology is domain independent and can be extended to address specific requirements of domain scientists. For example, we have found that for geosciences models it is important to capture environmental assumptions, variables and processes associated to modeling software [15, 35].

The ontology is available in OWL and documented in [11].

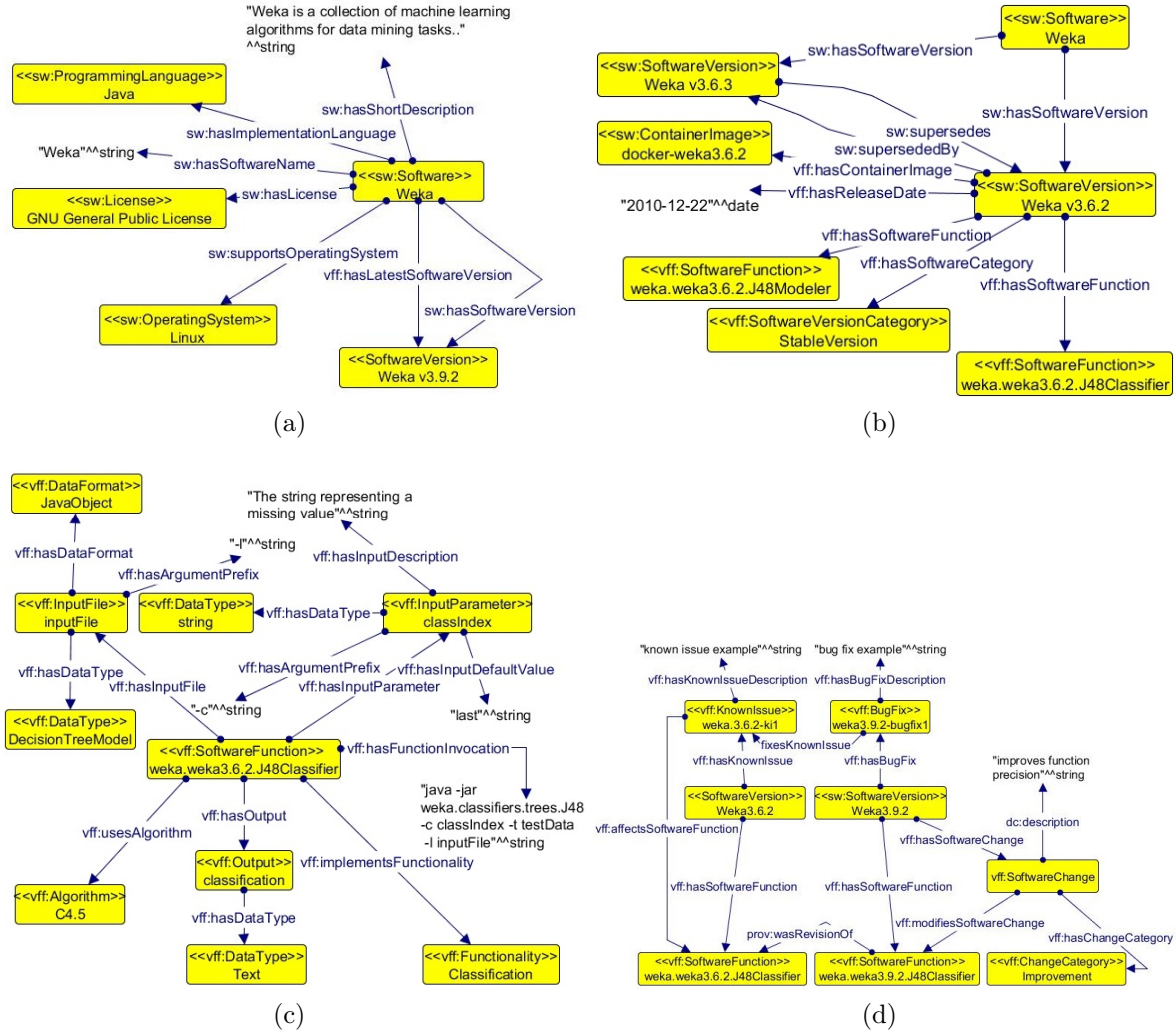


Figure 5.3: (a) An example of use of classes and relations from the ontology's software module to represent metadata associated with the Weka software. (b) An example of use of classes and properties from the ontology's software version module to represent metadata associated with the Weka 3.9.2 software version. (c) An example of metadata to represent the J48Classifier function from the Weka 3.6.2 version. (d) An example of metadata to represent changes to the J48Classifier function across Weka versions.

5.4.1 Describing software

Figure 5.3a illustrates an example of software metadata classes and properties for representing the Weka software. Weka is implemented using the Java programming language and uses a GNU license. Linux is one of the operating systems supported. Weka has the 3.9.2 version and this is its latest version. We use the OntoSoft `sw:Software` class. It has a property `sw:hasSoftwareVersion` that relates a software with each of its versions, while `vff:hasLatestSoftwareVersion` relates a `sw:Software` to its latest version in order to provide direct access to this specific version. `sw:Software` has other properties, such as `sw:hasLicense`, `sw:supportsOperatingSystem` and `sw:hasImplementationLanguage`. They represent important information to know when creating a workflow component using a software.

5.4.2 Describing software versions

Figure 5.3b illustrates the use of the classes and properties for Weka version 3.6.2. This is a stable version of Weka released in 2010. It has the *J48Classifier* and *ID3Classifier* functions and is superseded by the 3.6.3 version. We extended the OntoSoft `sw:SoftwareVersion` class with properties and classes to describe internal functions, versions, software dependencies, and version categories. `sw:SoftwareVersion` has properties `sw:supersededBy` and `sw:supersedes` to support navigation across software versions. To these, we added the property `vff:hasSoftwareFunction`, thereby linking `sw:SoftwareVersion` with `vff: SoftwareFunction` and thus representing functions released in a version. We introduced the notion of category of software versions (`vff:SoftwareVersionCategory`), whose values can be: major version, minor version, stable version and development version. Categories can help scientists to decide which version to use to implement a workflow component. `vff:ContainerImage` was designed to describe workflow components that use containers. Its properties such `vff:hasContainerLocation` and `vff:hasContainerInvocation` respectively specify its location in a container repository, and how to invoke the container image. This allows the isolation of a software version and its dependencies into a self-contained unit that can run anywhere independent of the environment.

5.4.3 Describing software functions

Figure 5.3c illustrates an example of the *J48Classifier* function in Weka 3.6.2 version. We recall from Section 5.1 that the difference between software and function can be subtle. A function represents a particular implementation of functionality of a software. A function is represented with the `vff:SoftwareFunction` class and implements a `vff:Functionality`, specified using the `vff:implementsFunctionality` property. A function might be implemented using a set of `vff:Algorithm`, which are specified with the `vff:usesAlgorithm` property. In Weka, a single Java class can implement several software functions with different functionalities. Here, a function has a unique name (`vff:hasFunctionName`) and a description to help identifying its objective (`vff:hasFunctionDescription`). Functions have unique invocation, inputs, parameters, and outputs (`vff:hasFunctionInvocation`, `vff:InputFile`, `vff:InputParameter` and `vff:`

`OutputFile`). The inputs and outputs have a description, argument prefix and are associated with `vff:DataType` and `vff:DataFormat`. We also represent the default value associated with an input parameter (`vff:hasInputDefaultValue`), since recommended defaults are often indicated in the documentation of scientific software.

5.4.4 Describing software changes

Figure 5.3d shows an example of a known issue, bug fix and change associated with the J48Classifier function in the Weka 3.6.2 and 3.9.2 versions.

A change is defined as a modification in a software function caused by `vff:BugFix` or improvements (`vff:SoftwareChange`). Change description includes `vff:KnownIssue` as well, to represent bugs or limitations associated with `vff:SoftwareFunction` and may be fixed by `vff:BugFix` in further versions. Bug fixes and known issues have descriptions to help scientists to understand how they affect `vff:SoftwareFunction`.

5.5 Using OntoSoft-VFF to store, compare and search semantic metadata for software

This section presents the services provided by OntoSoft-VFF. We designed these services by extending the OntoSoft catalog to use the ontology extensions introduced in Section 5.4.

The catalog includes the following important features, taking advantage of the semantic metadata and our ontology:

- Management of software functions and evolution metadata: provides means to obtain information about software, software version, functions and changes.
- Comparison mechanism: allows the comparison between different software, software versions and functions.
- Search mechanism: allows searching for software, software version and software functions.
- Mechanism for creation of workflow components: allows the creation of components by using the metadata associated with software functions.

The source code of OntoSoft-VFF can be found in [9].

5.5.1 Management of software functions and evolution metadata

In OntoSoft-VFF, software developers can add metadata about software, its versions, and available functions. Developers can also provide information about know issues, relevant changes and bug fixes associated with software functions. We envision an interactive system that extracts automatically some of this information and makes the burden minimal on the developers.

weka.weka3.6.2.ID3Classifier	weka.weka3.6.2.J48Classifier
[OPTIONAL] Functionality	
Classification	Classification
[OPTIONAL] What is the algorithm used in this function?	
ID3 Decision Tree	C4.5
[OPTIONAL] What is the invocation line for this function?	
java weka.classifiers.trees.Id3 -T testData -l inputFile -c classIndex > classification	java weka.classifiers.trees.J48 -T testData -l inputFile -c classIndex > classification
[OPTIONAL] What input files does this function require?	
Input File Name: inputFile **Input File Description:** Sets input model file. **Input File Argument:** -l **Input File Data Type:** Decision Tree Model **Input File Data Format:** Java Object	**Input File Name:** inputFile **Input File Description:** Sets model input file. **Input File Argument:** -l **Input File Data Type:** Decision Tree Model **Input File Data Format:** Java Object

Figure 5.4: Example of comparison between the ID3Classifier and J48Classifier functions.

When adding metadata for a new software version, our framework imports all the metadata of its previous version. The user only needs to provide information about the functions that have changed (e.g., algorithm, inputs, outputs, function name). When a function changes, OntoSoft-VFF creates a new URI for its metadata, and links it using the `prov:wasRevisionOf` property from the W3C PROV standard [60]. Through this URI, a workflow component can refer to the specific version of a function used in its implementation.

By adding bug fixes in new versions released, the user can provide information about known issues and associate them with specific functions in previous versions.

5.5.2 Comparison across versions and functions

The OntoSoft catalog allows the comparison of software via its metadata. We extended this to allow the comparison of software versions and functions as well. Our extension provides a simple comparison for software versions based on the functions they implement and the software version categories.

Function comparison is done by using metadata about functionality, algorithms, data types and data formats for inputs and outputs of functions, as well as relevant changes to functions such as bug fixes or improvements or known issues. Functions can be compared to other functions in the same software version, or to functions belonging to different software versions. This helps scientists understand the changes and differences in functions over time.

Figure 5.4 shows the comparison of functions using metadata of the functions *ID3 Classifier* and *J48Classifier* in Weka version 3.6.2. Due to space limitations, we only show functionality, algorithm, invocation line, and input files. As we can see, these functions have the same inputs and functionality. However, they use different algorithms and distinct function invocations, since they are implemented by different Java classes in Weka.

5.5.3 Search

The search capability allows scientists to find software, software versions, and software functions using their metadata. Search parameters include keywords related to the software and function names, functionality, license, data formats and data types of function inputs and outputs.

5.5.4 Creation of workflow components

OntoSoft-VVF facilitates the creation of workflow components using the semantic metadata associated with a software function. This helps scientists to create a workflow component from scratch. This mechanism is implemented as an external tool with access to the software function metadata. Our catalog exports the metadata in JSON format and also allows the use of a SPARQL endpoint to query the software metadata for the creation of workflow components. Other work can take advantage of OntoSoft-VFF to implement a similar mechanism for a different workflow management system.

We demonstrate this mechanism by integrating our catalog with the WINGS workflow system. Because WINGS uses semantic workflows, this system can take full advantage of semantic metadata available in OntoSoft-VFF's catalog to create workflow components. Our framework automatically creates in WINGS the inputs and outputs for the new workflow component and maps them to the software function's inputs and outputs. Using the metadata available, OntoSoft-VFF automatically configures the interface of the new component, including data types, and creates the invocation code to the function in the workflow component. We assume that there is a container image available from an online repository for each software version, which can be used to invoke the software function, thus avoiding the burden to install and configure the software dependencies. This tool creates the *ID3Classifier* component using the function with the same name from the Weka 3.9.2 version. This component can be used to replace the *J48Classifier* component in a workflow as in our scenarios.

5.6 Validation

To validate the ontology and services of OntoSoft-VFF, we demonstrate the ability of our framework to answer a series of competency questions on software functions and evolution, and the use of software functions in workflow components. The competency questions have been drawn from the requirements outlined in Section 5.3.

We designed queries to be evaluated against the structured metadata captured for the scenarios presented in Section 5.3. We present a description of each query, their translation into SPARQL, and the results obtained by evaluating them. The competency questions guided the development of our ontology and justified the creation of classes and properties. We use the namespace (ex: `https://w3id.org/ontosoft-vff/example`) to refer to the instance of our ontology in SPARQL.

Query 1: Given a software function invoked by a workflow component implementation, what is the software and software version of this function?

This query is useful to identify the software and software version of a function used to implement a workflow component. Retrieving metadata about software used in a workflow implementation addresses requirement R1.

Here, we are interested in retrieving the software function (`ex:weka3.6.2-J48Classifier`) responsible for implementing the J48Classifier workflow component. Specifically, the results point out that this software function is from the Weka software (`ex:Weka`) and was released in the Weka version 3.6.2 (`ex:Weka3.6.2`).

The SPARQL used for answering this query can be formulated as follows:

```
1 select ?sw ?swVersion where {
2  ?swVersion rdf:type sw:SoftwareVersion ;
3      vff:hasSoftwareFunction ex:weka3.6.2-J48Classifier .
4  ?sw rdf:type sw:Software ;
5     sw:hasSoftwareVersion  ?swVersion . }
```

Query 2: Are there any newer versions for a given function?

This query is useful to identify new versions of a given function used in a workflow component. Some software versions may not change a software function; thus, it is not the case of finding new software versions. This query retrieves a new function version of a given software function, which can be further compared to understand their differences. Retrieving metadata about software version releases addresses requirement R2.

Here, we are interested in showing the newest version of the J48Classifier function (`ex:weka3.6.2-J48Classifier`). The query results point out that the J48Classifier function has a new version (`ex:weka3.9.2-J48Classifier`) in Weka 3.9.2 version (`ex:weka 3.9.2`).

The SPARQL query used for answering this query can be formulated as follows:

```
1 select ?swVersionNew ?swFunctionNew where {
2   ?swVersionNew rdf:type sw:SoftwareVersion ;
3       vff:hasSoftwareFunction ?swFunctionNew .
4   ?swFunctionNew rdf:type sw:SoftwareFunction ;
5       prov:wasRevisionOf  ex:weka3.6.2-J48Classifier . }
```

Query 3: What are the differences between two versions of a given software function?

This query is useful to detect the differences between two version of a software function, particularly their interfaces, to use that information to upgrade a workflow component. Detecting differences between two versions of a software function addresses requirement R3.

Here, we are interested in the J48Classifier function from the 3.6.2 version (`ex: weka3.6.2-J48Classifier`) and the 3.9.2 version (`ex:weka3.9.2-J48Classifier`). We

run a separate query for each software function and then compare their results to compare the functions I/O. The results point out that there is no difference between their interfaces (i.e., their inputs and outputs).

The SPARQL used for answering this query can be formulated as follows:

```
select ?inputName ?inputDataFormat ?inputDataType ?inputParamName ?inputParamType
  ↪  ?outputName ?outputDataFormat ?outputDataType where {
ex:weka3.6.2-J48Classifier rdf:type vff:SoftwareFunction ;
     vff:hasInputFile  ?inputFile ;
     vff:hasInputParameter ?inputParam ; vff:hasOutputFile ?output .
  ?inputFile vff:hasInputDataFormat ?inputDataFormat ;
    vff:hasInputDataType ?inputDataType ;
    vff:hasInputName ?inputName .
  ?inputParam vff:hasInputParameterDataType ?paramType ;
       vff:hasInputParamName ? inputParamName .
  ?output vff:hasOutputDataFormat ?outputDataFormat ;
       vff:hasOutputDataType  ?outputDataType ;
       vff:hasOutputName ?outputName . }
```

Query 4: Are there any similar functions to a given function in newer software versions?

This query is useful to find similar functions in newer software versions based on their functionalities. We designed the query to find software functions that implement the same functionality or use the same algorithm than a given software function used in a workflow component. We can filter the functions by software and software version or find software function across different software. Detecting differences between two software versions, particularly about new software functions available addresses requirement R4.

Here, we are interested in finding a similar function to J48Classifier (`ex:weka3.6.2-J48Classifier`) that implements the same functionality in the same software version (`ex:weka3.6.2`). The query results point out that the ID3Classifier function (`ex:weka 3.6.2- ID3Classifier`) implements the same functionality the J48Classifier function does.

The SPARQL used for answering this query can be formulated as follows:

```
select ?swFunction where {
  ex:weka3.6.2-J48Classifier rdf:type vff:SoftwareFunction ;
      vff:implementsFunctionality ?functionality .
  ?swFunction rdf:type vff:SoftwareFunction ;
      vff:implementsFunctionality ?functionality .
  ex:weka3.9.2 vff:hasSoftwareFunction ?swFunction .
  FILTER(ex:weka3.9.2-J48Classifier != ?swFunction) . }
```

Query 5: How to invoke a given software function?

This query is useful to implement the invocation code of a workflow component based on the specification of an existing software function. Retrieving metadata about software functions, particularly their invocation code addresses requirement R5.

Here, we are interested in retrieving metadata associated with invocation of the ID3Classifier function (`ex:weka3.6.2-ID3Classifier`), such as function invocation and container invocation. By retrieving this information, we can create the invocation code in a workflow component. The query results point out that the function invocation is *"java -jar weka.classifiers.trees.Id3 -T testData -l inputFile -c classIndex > classification"* and the container invocation is *"docker run lucasaugustomcc/weka3.6.2"*.

The SPARQL used for answering this query can be formulated as follows:

```
select ?functionInvocation ?containerInv  where {
  ex:weka3.9.2-ID3Classifier rdf:type vff:SoftwareFunction ;
      vff:hasSoftwareFunctionInvocation  ?functionInvocation .
  ?swVersion rdf:type sw:SoftwareVersion ;
   vff:hasContainerImage ?containerImg ;
       vff:hasSoftwareFunction ex:weka3.9.2-ID3Classifier .
  ?containerImg vff:hasContainerImageInvocation  ?containerInv . }
```

Query 6: a) Are there any known issues that affect a given software function?

This query is useful to find out known issues that can affect the performance or results of software functions.

Here, we are interested in retrieving known issues associated with the ID3Classifier function (`ex:weka3.6.2-ID3Classifier`). The query results point out that no known issues are associated with this function.

The SPARQL used for answering this query can be formulated as follows:

```
select ?bug ?bugDescription where {
   ?bug rdf:type sw:KnownIssue ;
       vff:hasKnownIssueDescription ?bugDescription ;
       vff:affectsSoftwareFunction ex:weka3.6.2-J48Classifier . }
```

b) Are there any important changes associated with new versions of a given software function?

This query is useful to find out which software version they should upgrade a workflow component to take advantage of improvements associated with versions of software functions.

Here, we are interested in retrieving changes associated with the J48Classifier function. The query results point out that there are no bug fixes associated with the J48Classifier function in Weka 3.9.2.

The SPARQL used for answering this query can be formulated as follows:

```
select ?bugFix ?bugFixDescription where {
  ?bugfix rdf:type vff:BugFix ;
       vff:hasBugFixDescription ?description ;
   vff:fixesKnownIssue ?knownIssue .
  ?knowIssue
       vff:affectsSoftwareFunction ex:weka3.6.2-J48Classifier . }
```

Retrieving metadata about known issues and bug fixes associated with different versions of software functions used in a workflow component addresses requirement R6.

In summary, these queries show that the requirements are fully supported by OntoSoft-VFF, and that when they are not supported directly the framework provides the information necessary to address them. OntoSoft-VFF can answer questions that have been traditionally answered by scientists with great effort. The competency questions and the results obtained by evaluating the queries can be found in [10]

5.7 Conclusion

This paper presented OntoSoft-VFF, a semantic software catalog designed and developed to help scientists to manage workflow exploration and evolution, while they update or investigate alternatives for their computational experiments. OntoSoft-VFF relies on an ontology we designed to capture software versions, functionality, and functions and their evolution over time. This ontology is used in the construction of OntoSoft-VFF's underlying semantic metadata for software.

We showed that when a workflow is semantically linked to such metadata, scientists can explore the workflow to understand its evolution, and to compare among several software implementations to select one to implement a workflow's component. While related work is mostly concerned with workflow design, evolution, or provenance information, our goal is to help scientists to understand the evolution of the software used in the workflow components.

OntoSoft-VFF was built to meet requirements found through exploration of scenarios based on our experience using a variety of machine learning software libraries as well as diverse hydrology models. We demonstrate through competency questions that OntoSoft-VFF successfully meets those requirements. The competency questions and the scenarios are additional contributions of our work, since they describe very common scientific practices which are taken for granted and thus seldom explicitly formulated.

There are several possibilities for extending our work. One of them is to further explore the scenarios and competency questions in order to set up a benchmark for research on workflow evolution. A limitation of OntoSoft-VFF is that the addition of software metadata was manually done. This could be done semi-automatically in the future. Also, we plan to integrate OntoSoft-VFF with a workflow system to support scientists to efficiently update their workflows as the underlying application software evolves, and to easily explore new designs for their computational experiments. Finally, we plan to align to other ontologies such as the SWO, which contains thousands of instances.

Chapter 6

Conclusions and Future Work

6.1 Overview

The research presented in this thesis concerns challenges in data-intensive science, in particular to overcome the problem of supporting experiment reuse and reproducibility. Our motivation came from interdisciplinary research environments, in which each scientist works in a distinct domain, using different vocabularies, methodologies, perspectives of solving a problem and granularity of objects of interest. Moreover, they often use scripts, which are difficult to understand by third parties who were not involved in their development (and sometime even by the same scientists who developed them), and they are as such not amenable to reuse and reproduce. This scenario usually requires a means to provide access to the description of an experiment and its results to reproduce those results as well as speed up the construction of new experiments, and foster collaboration through reuse, and/or adaptation and repurposing of (parts of) experiments.

To this end, we proposed W2Share, a framework to support a script-to-reproducible research methodology. It drives the development of workflow 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 WROs as a means to ensure reproducibility.

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.

Also, we presented OntoSoft-VFF, a semantic software catalog designed and developed to help scientists manage workflow exploration and evolution, while they update or investigate alternatives for their computational experiments. OntoSoft-VFF relies on an ontology we designed to capture software versions, functionality, and functions and their evolution over time. This ontology is used in the construction of OntoSoft-VFF's underlying semantic metadata for software. We showed that when a workflow is semantically linked to such metadata, scientists can explore the workflow to understand its evolution, and to compare among several software implementations to select one to implement a

workflow's component.

Three different case studies – in Molecular Dynamics, Bioinformatics and Weather Forecasting – were analyzed to gather the requirements of script-to-workflow conversion, workflow exploration and evolution, and to validate our research. All these cases can clearly benefit from our research to enable reuse and reproducibility of experiments. We also believe that our solution can be extended and adopted by other domains with similar requirements.

Any extension to other domains will require not only validation from experts, but also their manual input when inserting YesWorkflow tags, and quality annotations. Curation for reuse and reproducibility is recognizably a labor-intensive task, involving many profiles of scientists and curators.

Besides this need for human annotations, there are other limitations to our work. For instance, as mentioned in Chapter 5, one limitation of our proposal to create workflow variants is its dependence on specific software ontologies. We created our own ontology, but it may need to be extended to cater to additional needs.

6.2 Main Contributions

This thesis had as a main objective to support reproducibility and reuse of computational experiments. To this effect, we tried to answer two questions: (i) how to understand a computational experiment; and (ii) how to extend a computational experiment.

We answered question 1 through the design and implementation of W2Share, a framework to support a methodology for conversion of scripts into reproducible workflow research objects. We answered question 2 through the design and implementation of OntoSoft-VFF, a software metadata catalog with information and services to support scientists during composition and evolution of scientific workflows.

Summing up, our first contribution, presented in Chapter 2, was to present the requirements for the conversion of script to reproducible research.

The second contribution of this thesis, also presented in Chapter 2, is to propose a methodology that guides the scientists through the process of conversion of script-based experiments into reproducible workflow research objects. This is proposed to fill the gap of the absense of a principled way to convert script-to-workflows.

The third contribution, introduced in Chapter 3, is to present W2Share and its features for quality assessment of computational experiments.

The fourth contribution, presented in Chapter 3 and 4, is the design and implementation of W2Share. It exploits tools and standards that have been developed by the scientific community to promote reuse and reproducibility. A prototype implementation of W2Share is available at `https://w3id.org/w2share`.

The fifth contribution, presented in Chapter 5, is the design and implementation of OntoSoft-VFF for capturing information about software and workflow components to support scientists manage workflow exploration and evolution. OntoSoft-VFF is composed by a novel ontology and a software metadata catalog. A prototype implementation of OntoSoft-VFF is available at `https://w3id.org/ontosoft-vff`.

The last contribution, presented in Chapters 2, 3 and 4, is to analyze two real world examples from Molecular Dynamics and Bioinformatics, showing how they can benefit from W2Share framework.

6.3 Future Work

This research can be extended in different practical/implementation and theoretical aspects, in close cooperation with domain experts. They correspond to some open questions – some of which appeared while we were developing our work. Some possibilities and open questions are listed below.

- Evaluate our script to reproducible research framework with other use cases, from fields other than Molecular Dynamics and Bioinformatics.
- Promote 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.
- Evaluate the cost effectiveness of our proposal, in particular since in some cases it may require extensive involvement of scientists.
- Provide support to other SWfMS. Our implementation only supports mapping to Taverna workflows; mapping to other system will require recoding.
- Provide support to data citation standards, such as integrating W2Share's WRO repository to Digital Object Identification (DOI).
- Automatically compare the quality of the experiment results based on the original script and the workflow.
- Explore the Common Workflow Language (CWL)[1] to create executable workflows by using a standard that works across multiple SWfMS.
- Explore the scenarios and competency questions in order to set up a benchmark for research on workflow evolution.
- Investigate a semi-automatic approach to add software metadata to the OntoSoft-VFF catalog.
- Integrate OntoSoft-VFF to W2Share's architecture to support scientists to efficiently update their workflows as the underlying application software evolves, and to easily explore new designs for their computational experiments.
- Align the OntoSoft-VFF ontology to other ontologies such as the Software Ontology (SWO), which contains thousands of instances, and design new ontologies to characterize software functions and functionality, in close cooperation with domain experts.

[1] `https://www.commonwl.org/`

- Investigate managing and tracking of workflow variants and their differences. This includes how to compare workflow components and workflow variants regarding their interfaces and functions, and present these results in a useful way for scientists to understand their differences and the implications on experiment results.
- Design an interactive framework to support scientists in the exploration and experimentation process through workflow variants. This includes how to leverage workflow reuse and composition to support the creation of workflow variants. For example, given a new component that needs to replace an existing one in a workflow, suggest what other components may need to be added or removed from the workflow.
- Work with more complex scripts, e.g., that are composed of multiple files. The scripts used as case studies are self-contained, and thus the script-to-workflow conversion does not require checking multiple files.
- Work with non-DAG (Directed Acyclic Graph) workflows, e.g., those that support loops; Taverna, to the best of our knowledge, does not support non-DAG workflows.
- Provide a structural means to assess quality, e.g., creating test cases under software engineering methodologies and principles.

Bibliography

[1] Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludascher, and Steve Mock. Kepler: an extensible system for design and execution of scientific workflows. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, pages 423–424. IEEE, 2004.

[2] Elaine Angelino, Daniel Yamins, and Margo Seltzer. Starflow: A script-centric data analysis environment. In *Provenance and Annotation of Data and Processes*, pages 236–250. Springer, 2010.

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

[4] Mikołaj Baranowski, Adam Belloum, Marian Bubak, and Maciej Malawski. Constructing workflows from script applications. *Scientific Programming*, 20(4):359–377, 2012.

[5] Sean Bechhofer, David De Roure, Matthew Gamble, Carole Goble, and Iain Buchan. Research objects: Towards exchange and reuse of digital knowledge. In *The Future of the Web for Collaborative Science (FWCS 2010)*. Nature Precedings, 2010.

[6] Khalid Belhajjame, Oscar Corcho, Daniel Garijo, Jun Zhao, Paolo Missier, David Newman, Raúl Palma, Sean Bechhofer, Esteban García Cuesta, José Manuel 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.

[7] Khalid Belhajjame, Jun Zhao, Daniel Garijo, Matthew Gamble, Kristina Hettne, Raul Palma, Eleni Mina, Oscar Corcho, José Manuel Gómez-Pérez, Sean 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*, 32:16–42, 2015.

[8] Jacek Cala, Yaobo Xu, Eldarina Azfar Wijaya, and Paolo 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*, pages 694–700. IEEE, 2014.

[9] Lucas Carvalho, Daniel Garijo, Claudia Bauzer Medeiros, and Yolanda Gil. lucasaugustomcc/ontosoft: Ontosoft-vff v1.0.0, September 2018. https://doi.org/10.5281/zenodo.1414544.

[10] Lucas Carvalho, Daniel Garijo, Claudia Bauzer Medeiros, and Yolanda Gil. Ontosoft-vff evalution: Competency questions, September 2018. https://doi.org/10.5281/zenodo.1414552.

[11] Lucas Carvalho, Daniel Garijo, Claudia Bauzer Medeiros, and Yolanda Gil. The ontosoft-vff ontology v1.0.0, September 2018. https://w3id.org/ontosoft-vff/ontology.

[12] Lucas A. M. C. Carvalho, Khalid Belhajjame, and Claudia Bauzer Medeiros. Converting scripts into reproducible workflow research objects. In *Proc. of the IEEE 12th Int. Conf. on eScience, October 23-26*, pages 71–80, Baltimore, MD, USA, 2016. IEEE.

[13] Lucas A. M. C. Carvalho, Khalid Belhajjame, and Claudia Bauzer Medeiros. A prov-compliant approach to script-to-workflow process (under review). *The Semantic Web Journal*, 2018.

[14] Lucas A. M. C. Carvalho, Bakinam T. Essawy, Daniel Garijo, Claudia Bauzer Medeiros, and Yolanda 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)*, pages 1–8, 2017.

[15] Lucas A. M. C. Carvalho, Bakinam T. Essawy, Daniel Garijo, Claudia Bauzer Medeiros, and Yolanda Gil. Requirements for supporting the iterative exploration of scientific workflow variants. In *Proceedings of the ACM Workshop on Capturing Scientific Knowledge (SciKnow)*, 2017.

[16] Lucas A. M. C. Carvalho, Daniel Garijo, Claudia Bauzer Medeiros, and Yolanda Gil. Semantic Software Metadata for Workflow Exploration and Evolution. In *Proc. of the IEEE 14th Int. Conf. on eScience, Oct 29-Nov 1*, Amsterdam, Netherlands, 2018. IEEE.

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

[18] Lucas A. M. C. Carvalho and Claudia Bauzer Medeiros. W2share case study: Workflow research object (wro), October 2018. https://doi.org/10.5281/zenodo.1465897.

[19] Lucas A. M. C. Carvalho and Claudia Bauzer Medeiros. W2share evalution: Competency questions, October 2018. https://doi.org/10.5281/zenodo.1465893.

[20] Lucas A. M. C. Carvalho, Rodrigo L. Silveira, Caroline S. Pereira, Munir S. Skaf, and Claudia Bauzer Medeiros. Provenance-based retrieval: Fostering reuse and reproducibility across scientific disciplines. In *Proc. of the 6th IPAW, June 7-8, 2016*, pages 183–186. Springer, 2016.

[21] Lucas A. M. C. Carvalho, Regina Wang, Daniel Garijo, and Yolanda 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*, pages 1–8, Austin, TX, USA, 2017.

[22] Fernando Chirigati, Rémi Rampin, Dennis E. Shasha, and Juliana Freire. Reprozip: Computational reproducibility with ease. In *SIGMOD Conference*, pages 2085–2088. ACM, 2016.

[23] Sarah Cohen-Boulakia, Khalid Belhajjame, Olivier Collin, Jérôme Chopard, Christine Froidevaux, Alban Gaignard, Konrad Hinsen, Pierre Larmande, Yvan Le Bras, Frédéric Lemoine, et al. Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities. *Future Generation Computer Systems*, 75:284–298, 2017.

[24] Sarah Cohen-Boulakia and Ulf Leser. Search, adapt, and reuse: the future of scientific workflows. *ACM SIGMOD Record*, 40(2):6–16, 2011.

[25] Reidar Conradi and Bernhard Westfechtel. Version models for software configuration management. *ACM Computing Surveys (CSUR)*, 30(2):232–282, 1998.

[26] Víctor 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.

[27] Susan B Davidson and Juliana Freire. Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1345–1350. ACM, 2008.

[28] David De Roure, Carole Goble, and Robert Stevens. Designing the myexperiment virtual research environment for the social sharing of workflows. In *IEEE Int. Conf. on e-Science and Grid Computing*, pages 603–610. IEEE, 2007.

[29] Ewa Deelman, Tom Peterka, Ilkay Altintas, Christopher D Carothers, Kerstin Kleese van Dam, Kenneth Moreland, Manish Parashar, Lavanya Ramakrishnan, Michela Taufer, and Jeffrey Vetter. The future of scientific workflows. *The International Journal of High Performance Computing Applications (IJHPCA)*, 32(1):159–175, 2018.

[30] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, et al. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015.

[31] Saumen Dey, Khalid Belhajjame, David Koop, Meghan Raul, and Bertram Ludäscher. Linking prospective and retrospective provenance in scripts. In *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)*, 2015.

[32] Daniel de Oliveira 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.

[33] Juliana Freire, David Koop, Emanuele Santos, and Cláudio T Silva. Provenance for computational tasks: A survey. *Computing in Science & Engineering*, 10(3), 2008.

[34] Juliana Freire, Cláudio T. Silva, Steven P. Callahan, Emanuele Santos, Carlos E. Scheidegger, and Huy T. Vo. Managing rapidly-evolving scientific workflows. In Luc Moreau and Ian Foster, editors, *Provenance and Annotation of Data*, pages 10–18, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[35] Daniel Garijo, Deborah Khider, Yolanda Gil, Lucas Carvalho, Bakinam Essawy, Suzanne Pierce, Daniel H. Lewis, Varun Ratnakar, Scott D. Peckham, Chris Duffy, and Jonathan Goodall. A semantic model registry to support comparison and reuse. In *Proceedings of the Ninth International Congress on Environmental Modeling and Software*, Fort Collins, CO, 2018.

[36] Daniel Garijo, Sarah Kinnings, Li Xie, Lei Xie, Yinliang Zhang, Philip E Bourne, and Yolanda Gil. Quantifying reproducibility in computational biology: the case of the tuberculosis drugome. *PloS one*, 8(11):e80278, 2013.

[37] Yolanda Gil, Pedro A Gonzalez-Calero, Jihie Kim, Joshua Moody, and Varun Ratnakar. A semantic framework for automatic generation of computational workflows using distributed data and component catalogues. *Journal of Experimental & Theoretical Artificial Intelligence*, 23(4):389–467, 2011.

[38] Yolanda Gil, Varun Ratnakar, and Daniel Garijo. Ontosoft: Capturing scientific software metadata. In *Proceedings of the 8th ACM International Conference on Knowledge Capture*, Palisades, NY, 2015. ACM.

[39] Yolanda Gil, Varun Ratnakar, Jihie Kim, Pedro A González-Calero, Paul Groth, Joshua Moody, and Ewa Deelman. Wings: Intelligent workflow-based design of computational experiments. *IEEE Intelligent Systems*, 26(1):62–72, 2011.

[40] Yolanda Gil, Ke-Thia Yao, Varun Ratnakar, Daniel Garijo, Greg Ver Steeg, Pedro Szekely, Rob Brekelmans, Mayank Kejriwal, Fanghao Luo, and I-Hui Huang. P4ml: A phased performance-based pipeline planner for automated machine learning. In *Proceedings of Machine Learning Research, ICML 2018 AutoML Workshop*, 2018.

[41] Antoon Goderis, Ulrike Sattler, Phillip Lord, and Carole Goble. Seven bottlenecks to workflow reuse and repurposing. In *The Semantic Web–ISWC 2005*, pages 323–337. Springer, 2005.

[42] Thomas R Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, 43(5-6):907–928, 1995.

[43] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[44] Matheus Hauder, Yolanda Gil, and Yan Liu. A framework for efficient data analytics through automatic configuration and customization of scientific workflows. In *E-Science (e-Science), 2011 IEEE 7th International Conference on*, pages 379–386. IEEE, 2011.

[45] Tony Hey, Stewart Tansley, Kristin M Tolle, et al. *The fourth paradigm: data-intensive scientific discovery*, volume 1. Microsoft research Redmond, WA, 2009.

[46] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90, 2016.

[47] David Koop, Carlos E Scheidegger, Juliana Freire, and Cláudio T Silva. The provenance of workflow upgrades. In *International Provenance and Annotation Workshop*, pages 2–16. Springer, 2010.

[48] Timothy Lebo, Satya Sahoo, Deborah 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.

[49] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, 13(4):457–493, 2015.

[50] Bertram Ludäscher, Mathias Weske, Timothy McPhillips, and Shawn Bowers. Scientific workflows: Business as usual? In *International Conference on Business Process Management*, pages 31–47. Springer, 2009.

[51] Ketan Maheshwari and Johan Montagnat. Scientific workflow development using both visual and script-based representation. In *6th World Congress on Services (SERVICES-1)*, pages 328–335. IEEE, 2010.

[52] James Malone, Andy Brown, Allyson L Lister, Jon Ison, Duncan Hull, Helen Parkinson, and Robert Stevens. The software ontology (swo): a resource for reproducibility in biomedical data analysis, curation and digital preservation. *Journal of biomedical semantics*, 5(1):25, 2014.

[53] Marta Mattoso, Jonas Dias, Kary ACS Ocaña, Eduardo Ogasawara, Flavio Costa, Felipe Horta, Vítor Silva, and Daniel de Oliveira. Dynamic steering of hpc scientific workflows: A survey. *Future Generation Computer Systems*, 46:100–113, 2015.

[54] Brian McBride. The resource description framework (rdf) and its vocabulary description language rdfs. In *Handbook on ontologies*, pages 51–65. Springer, 2004.

[55] Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.

[56] Timothy McPhillips, Shawn Bowers, Khalid Belhajjame, and Bertram Ludäscher. Retrospective provenance without a runtime provenance recorder. In *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)*, 2015.

[57] Timothy McPhillips, Tianhong Song, Tyler Kolisnik, Steve Aulenbach, Khalid Belhajjame, R Kyle Bocinsky, Yang Cao, James Cheney, Fernando Chirigati, Saumen Dey, et al. Yesworkflow: A user-oriented, language-independent tool for recovering workflow information from scripts. *International Journal of Digital Curation*, 10(1):298–313, 2015.

[58] Paolo Missier, Simon Woodman, Hugo Hiden, and Paul Watson. Provenance and data differencing for workflow reproducibility analysis. *Concurrency and Computation: Practice and Experience*, 28(4):995–1015, 2016.

[59] Johan Montagnat, Benjamin Isnard, Tristan Glatard, Ketan Maheshwari, and Mireille Blay 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*, page 7. ACM, 2009.

[60] Luc Moreau, Paolo Missier, Khalid Belhajjame, Reza B'Far, James Cheney, Sam Coppens, Stephen Cresswell, Yolanda Gil, Paul Groth, Graham Klyne, et al. Prov-dm: The prov data model. *World Wide Web Consortium (W3C) Recommendation*, 2013.

[61] Leonardo Murta, Vanessa Braganholo, Fernando Chirigati, David Koop, and Juliana Freire. noworkflow: Capturing and analyzing provenance of scripts. In *Provenance and Annotation of Data and Processes*, pages 71–83. Springer, 2014.

[62] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R Pocock, Anil Wipat, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[63] Raúl Palma, Piotr Hołubowicz, Oscar Corcho, José Manuel Gómez-Pérez, and Cezary Mazurek. Rohub—a digital library of research objects supporting scientists towards reproducible science. In *Semantic Web Evaluation Challenge*, pages 77–82. Springer, 2014.

[64] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[65] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[66] Carlos E Scheidegger, Huy T Vo, David Koop, Juliana Freire, and Claudio T Silva. Querying and re-using workflows with vstrails. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1251–1254. ACM, 2008.

[67] Ricky J Sethi, Hyunjoon Jo, and Yolanda Gil. Re-using workflow fragments across multiple data domains. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 90–99. IEEE, 2012.

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

[69] Renato Beserra Sousa. Quality flow: a collaborative quality-aware platform for experiments in escience. Master's thesis, Institute of Computing - University of Campinas, 2015.

[70] Renato Beserra Sousa, Daniel Cintra Cugler, Joana Esther Gonzales Malaverri, and Claudia Bauzer Medeiros. A provenance-based approach to manage long term preservation of scientific data. In *2014 IEEE 30th Int. Conf. on Data Eng. Workshops (ICDEW)*, pages 162–133. IEEE, 2014.

[71] Renan Souza, Vitor Silva, Alvaro LGA Coutinho, Patrick Valduriez, and Marta Mattoso. Data reduction in scientific workflows using provenance monitoring and user steering. *Future Generation Computer Systems*, 2017.

[72] W. Souza, B. Carvalho, D. Dogini, and I. Lopes-Cendes. Identification of differentially methylated genes potentially associated with neurological diseases. In *ASHG 64th Annual Meeting, October 18-22, 2014*. American Society of Human Genetics, 2014.

[73] Yannis Tzitzikas, Nikos Minadakis, Yannis Marketakis, Pavlos Fafalios, Carlo Allocca, Michalis Mountantonakis, and Ioanna Zidianaki. *MatWare: Constructing and Exploiting Domain Specific Warehouses by Aggregating Semantic Data*, pages 721–736. Springer International Publishing, 2014.

[74] Victoria Uren, Philipp Cimiano, José Iria, Siegfried Handschuh, Maria Vargas-Vera, Enrico Motta, and Fabio Ciravegna. Semantic annotation for knowledge management: Requirements and a survey of the state of the art. *Web Semantics: science, services and agents on the World Wide Web*, 4(1):14–28, 2006.

[75] Michael Wilde, Mihael Hategan, Justin M Wozniak, Ben Clifford, Daniel S Katz, and Ian Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652, 2011.

[76] Eran Chinthaka Withana, Beth Plale, Roger Barga, and Nelson Araujo. Versioning for workflow evolution. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 756–765. ACM, 2010.

[77] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, et al. The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Research*, 41(W1):W557–W561, 2013.

[78] Ustun Yildiz, Adnene Guabtni, and Anne HH Ngu. Business versus scientific workflows: A comparative study. In *Services-I, 2009 World Conference on*, pages 340–343. IEEE, 2009.

[79] Jun Zhao, José Manuél Gómez-Pérez, Khalid Belhajjame, Graham Klyne, Esteban Garcia-Cuesta, Austin Garrido, Kristina Hettne, Maree Roos, David De Roure, and Carole Goble. Why workflows break—understanding and combating decay in taverna workflows. In *E-Science (e-Science), 2012 IEEE 8th International Conference on*, pages 1–9. IEEE, 2012.

Appendix A

Listings of Chapter 4

Listings of the Molecular Dynamics case study in Chapter 4.

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.
```

```
13
14 <abs-workflow/Setup_MD/>
15     prov:wasDerivedFrom <files/Setup_MD/script.sh> ;
16     prov:wasAttributedTo [
17         a prov:Agent ;
18         foaf:name "Lucas Carvalho" ] .
19
20 <abs-workflow/Setup_MD/processor/split/>
21   prov:wasDerivedFrom [
22     a prov:Entity, oa:TextPositionSelector;
23     oa:start "1644"^^xsd:Integer;
24     oa:end "1786"^^xsd:Integer;
25   ] .
26
27 <workflow/Setup_MD/> a wfdesc:Workflow, prov:Entity ;
28     prov:wasDerivedFrom <files/Setup_MD/script.sh> ;
29     prov:wasDerivedFrom <abs-workflow/Setup_MD/> ;
30     wfdesc:hasSubProcess <workflow/Setup_MD/processor/split> .
31
32 <workflow/Setup_MD/processor/split/>
33     prov:wasDerivedFrom <abs-workflow/Setup_MD/processor/split/> .
34
35 <workflow/Setup_MD/variant> a wfdesc:Workflow, prov:Entity ;
36     prov:wasDerivedFrom <workflow/Setup_MD/> .
```

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

```
1  @base <https://w3id.org/w2share/wro/md-setup/>
2  @prefix prov:  <http://www.w3.org/ns/prov#> .
3  @prefix wfprov:  <http://purl.org/wf4ever/wfprov#> .
4  @prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
5  @prefix wfdesc:  <http://purl.org/wf4ever/wfdesc#> .
6  @prefix tavernaprov:  <http://ns.taverna.org.uk/2012/tavernaprov/> .
7  @prefix owl:  <http://www.w3.org/2002/07/owl#> .
8  @prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
9  @prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
10 @prefix dct:   <http://purl.org/dc/terms/> .
11
12 <run/e0fa2f25-0755/>
13     rdf:type                    wfprov:WorkflowRun ;
14     dct:hasPart                 <run/e0fa2f25-0755/process/f0a0bd65-78d3/> ;
15     wfprov:describedByWorkflow  <workflow/Setup_MD/> ;
16     prov:used                   <data/5c65c151-0333/ref/61f8795e-e650> ;
17     dct:hasPart                 <run/e0fa2f25-0755/process/c06ff05e-eceb/> ;
18     prov:endedAtTime            "2016-06-16T11:25:24.549-03:00"^^xsd:dateTime ;
19     prov:startedAtTime          "2016-06-16T11:25:12.838-03:00"^^xsd:dateTime ;
20     wfprov:usedInput            <data/5c65c151-0333/ref/61f8795e-e650>;
21
22 <data/5c65c151-0333/ref/61f8795e-e650>
23     tavernaprov:content         <data/4e0baa1f-fc0f/input/structure.pdb> ;
24     wfprov:describedByParameter
   ↪   <workflow/Setup_MD/processor/split/in/initial_structure> ;
```

```
25      wfprov:describedByParameter  <workflow/Setup_MD/processor/in/initial_structure> ;
26      prov:wasGeneratedBy          <run/4e0baa1f-fc0f/process/c3a0e8c0-dcb0/> ;
27      rdf:type                     wfprov:Artifact ;
28      rdf:type                     prov:Entity .
29
30  <data/e0fa2f25-0755/ref/55269975-380f>
31      tavernaprov:content          <data/4e0baa1f-fc0f/output/bglc.pdb> ;
32      wfprov:describedByParameter  <workflow/Setup_MD/processor/psgen/in/bglc_pdb> ;
33      wfprov:describedByParameter  <workflow/Setup_MD/processor/split/out/bglc_pdb> ;
34      wfprov:wasOutputFrom         <run/4e0baa1f-fc0f/process/c3a0e8c0-dcb0/> ;
35      prov:wasGeneratedBy          <run/4e0baa1f-fc0f/process/c3a0e8c0-dcb0/> ;
36      rdf:type                     wfprov:Artifact ;
37      rdf:type                     prov:Entity .
38
39  <run/4e0baa1f-fc0f/process/c3a0e8c0-dcb0/>
40      wfprov:describedByProcess    <workflow/Setup_MD/processor/split/> ;
41      wfprov:usedInput             <data/5c65c151-0333/ref/61f8795e-e650> ;
42      prov:wasAssociatedWith       <#taverna-engine> ;
43      rdf:type                     wfprov:ProcessRun ;
44      prov:endedAtTime             "2017-03-10T08:19:32.405-03:00"^^xsd:dateTime ;
45      prov:startedAtTime           "2017-03-10T08:19:31.075-03:00"^^xsd:dateTime ;
46      prov:used                    <data/5c65c151-0333/ref/61f8795e-e650> ;
47      wfprov:wasPartOfWorkflowRun  <run/e0fa2f25-0755/> .
```

Listing A.5: Excerpt of the WRO manifest.

```
1  @base <https://w3id.org/w2share/wro/md-setup/>.
2  @prefix ro: <http://purl.org/wf4ever/ro#> .
3  @prefix ore: <http://www.openarchives.org/ore/terms/> .
4  @prefix wf4ever: <http://purl.org/wf4ever/wf4ever#> .
5
6  <files/Setup_MD/script.sh> a ro:Resource, wf4ever:Script .
7  <workflow/executable-workflow.t2flow> a ro:Resource, wf4ever:Workflow .
8  <workflow/refined-workflow.t2flow> a ro:Resource, wf4ever:Workflow .
9  <data/4e0a1f-fc0f/input/structure.pdb> a ro:Resource, wf4ever:Dataset .
10 <data/4e0a1f-fc0f/output/bglc.pdb> a ro:Resource, wf4ever:Dataset .
11
12 <> a ro:ResearchObject ;
13    ore:aggregates <files/Setup_MD/script.sh>,
14                   <workflow/executable-workflow.t2flow>,
15                   <workflow/refined-workflow.t2flow>,
16                   <data/4e0a1f-fc0f/input/structure.pdb>,
17                   <data/4e0a1f-fc0f/output/bglc.pdb> .
```