

Reproducibility of Execution Environments in Computational Science Using Semantics and Clouds

Idafen Santana-Perez^{a,*}, Rafael Ferreira da Silva^b, Mats Ryngé^b, Ewa Deelman^b, María S. Pérez-Hernández^a, Oscar Corcho^a

^aOntology Engineering Group, Universidad Politécnica de Madrid, Madrid, Spain

^bUniversity of Southern California, Information Sciences Institute, Marina del Rey, CA, USA

Abstract

In the past decades, one of the most common forms of addressing reproducibility in scientific workflow-based computational science has consisted of tracking the provenance of the produced and published results. Such provenance allows inspecting intermediate and final results, improves understanding, and permits replaying a workflow execution. Nevertheless, this approach does not provide any means for capturing and sharing the very valuable knowledge about the experimental equipment of a computational experiment, i.e., the execution environment in which the experiments are conducted. In this work, we propose a novel approach based on semantic vocabularies that describes the execution environment of scientific workflows, so as to conserve it. We define a process for documenting the workflow application and its related management system, as well as their dependencies. Then we apply this approach over three different real workflow applications running in three distinct scenarios, using public, private, and local Cloud platforms. In particular, we study one astronomy workflow and two life science workflows for genomic information analysis. Experimental results show that our approach can reproduce an equivalent execution environment of a predefined virtual machine image on all evaluated computing platforms.

Keywords: Semantic Metadata, Scientific Workflow, Reproducibility, Life Sciences.

1. Introduction

Reproducibility of results of scientific experiments is a cornerstone of the scientific method. Therefore, the scientific community has been encouraging researchers to publish their contributions in a verifiable and understandable way [1, 2]. In computational science, or *in-silico* science, reproducibility often requires that researchers make code and data publicly available so that the data can be analyzed in a similar manner as in the original work described in the publication. Code must be made available, and data must be accessible in a readable format [3].

Scientific workflows are a useful representation for managing the execution of large-scale computations. Many scientists now formulate their computational problems as scientific workflows running on distributed computing infrastructures such as campus Clusters, Clouds, and Grids [4]. Researchers in bioinformatics have embraced workflows for a whole range of analyses, including protein folding [5], DNA and RNA sequencing [6, 7], and disease-related research [8, 9], among others. The workflow representation not only facilitates the creation and management of the computation but also builds a foundation upon which results can be validated and shared.

Much research has focused on studying the reproducibility of scientific results in life sciences. Some studies clearly show the difficulties when trying to replicate experimental results in biology [10]. The *Reproducibility Project: Cancer Biology* [11] is an active project aiming to independently reproduce the experimental results of 50 high-impact cancer biology studies, evaluating the degree of reproducibility of those results and the main issues related to them.

Since workflows formally describe the sequence of computational and data management tasks, it is easy to trace the origin of the data produced. Many workflow systems capture provenance at runtime, what provides the lineage of data products and as such underpins scientific understanding and data reuse by providing the basis on which trust and understanding are built. A scientist would be able to look at the workflow and provenance data, retrace the steps, and arrive at the same data products. However, this information is not sufficient for achieving full reproducibility.

Reproducibility, replicability, and repeatability are often used as synonyms. Even when they pursue similar goals, there are several differences between them [12]. In this work we consider them as separated concepts. While replicability can be defined as a strict recreation of the original experiment, using the same method, data and equipment, reproducibility implies that at least some changes have been introduced in the experiment, thus exposing different features. While being a less restrictive term, reproducibility is a key concept in science, as it allows incremental research by modifying, improving and repurposing the experimental methods and conditions.

*Corresponding address: Ontology Engineering Group (OEG), Laboratorio de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, Avda. Boadilla del Monte, s/n, Boadilla del Monte, 28660, Spain

Email addresses: isantana@fi.upm.es (Idafen Santana-Perez), rafsilva@isi.edu (Rafael Ferreira da Silva), rynge@isi.edu (Mats Ryngé), deelman@isi.edu (Ewa Deelman), mperez@fi.upm.es (María S. Pérez-Hernández), ocorcho@fi.upm.es (Oscar Corcho)

In this work, we address the reproducibility of the execution environment for a scientific workflow, as we do not aim to obtain an exact incarnation of the original setup, but rather an environment that is able to support the required capabilities exposed by the former environment. In order to reproduce or replicate any digital artifact we need to properly handle its conservation. According to [13], to achieve conservation one needs to guarantee that “*sufficient information exists with which to understand, evaluate, and build upon a prior work if a third party could replicate the results without any additional information from the author*”. Hence, we address workflow conservation in order to attain its reproducibility.

In [14], authors explain the problems they faced when they tried to reproduce an experiment [15] for mapping all putative FDA and European drugs to protein receptors within the scope of a given proteome. For each identified problem, they enumerate a set of suggestions for addressing the related issues. In four out of the total six cases, execution environment problems are mentioned.

Currently, most of the approaches in computational science conservation, in particular for scientific workflow executions, have been focused on data, code, and the workflow description, but not on the underlying infrastructure—which is composed of a set of computational resources (e.g., execution nodes, storage devices, and networking) and software components. We identify two approaches for conserving the environment of an experiment: 1) *physical conservation*, where the real object is conserved due to its relevance and the difficulty in obtaining a counterpart; and 2) *logical conservation*, where objects are described in such a way that an equivalent one can be obtained in a future experiment.

The computational environment is often conserved by using the physical approach, where computational resources are made available to scientists over a sustained period of time. As a result, scientists are able to reproduce their experiments in the same environment. However, such infrastructures demand huge maintenance efforts, and there is no guarantee that it will not change or suffer from a natural decay process [16]. Furthermore, the infrastructure may be subjected to organization policies, which restrict its access to a selective group of scientists, thereby limiting reproducibility to this restricted group. On the other hand, data, code, and workflow descriptions can be conserved by using a logical approach, which is not subjected to natural decay processes.

Accordingly, we propose a logical-oriented approach to conserve computational environments, where the capabilities of the resources (virtual machines (VM)) are described. From this description, any scientist, interested in reproducing an experiment, will be able to reconstruct the former infrastructure (or an equivalent one) in any Cloud computing infrastructure (either private or public). One may argue that it would be easier to keep and share VM images (or containers) with the community research through a common repository, however the high storage demand of VM images (in particular for Big Data applications) may be a challenging problem [17, 18].

Inspired by the aforementioned ideas, exposed in [13], we aim to define means for authors to share the relevant informa-

tion about the execution environment of a given scientific workflow. We argue that by explicitly describing this knowledge we increase the degree of reproducibility of the environment and of the workflow.

Semantics have been proposed as a way of attaining curation and conservation of the digital assets related to scientific experiments (e.g., biomedical research [19]). Our approach uses semantic-annotated workflow descriptions to generate lightweight scripts for an experiment management API that can reconstruct the required infrastructure. We propose to describe the resources involved in the execution of the experiment using a set of semantic vocabularies, and use those descriptions to define the infrastructure specification. This specification can then be used to derive the set of instructions that can be executed to obtain a new equivalent infrastructure.

We conduct a practical experimentation process in which we describe a set of workflows and their environments using a set of semantic models. Then, we use an experiment management tool to reproduce a workflow execution in different Cloud platforms. In this work, we study three real and representative production workflows, which have been disseminated in several scientific publications. Many scientific workflows allow their execution infrastructure to be escalated, depending on the processing needs of the application. In this work, we analyze the reproduction of workflows which were formerly configured to run into single-node infrastructures.

The first iteration of the semantic models for representing computational resources, and some of the reproducibility tools were introduced and evaluated in [20] for a single astronomy workflow. In this work, we extend our previous work by introducing 1) a set of new features to our framework based on the result of our previous work, including a new version of our models and features in our algorithmic process; 2) a study of two new life sciences workflows based on genomic processing applications and their related challenges with respect to previous cases; and 3) a practical evaluation of the framework with the new features for the astronomy workflow as well as to the new life science workflows.

The paper is organized as follows. Section 2 describes our semantic approach for documenting computational infrastructures. Section 3 presents the description of the tools used to implement the semantic models and manage the experiment. Section 4 describes the experimentation process. Section 5 presents the related work, and Section 6 summarizes our results and identifies future works.

2. Semantic Modeling of Computational Resources

In this work, we argue that in order to achieve reproducibility of a scientific workflow, enough information about the computational resources should be provided. These descriptions allow the target audience, usually another computational scientist in the same domain, to better understand the underlying components involved in a workflow execution.

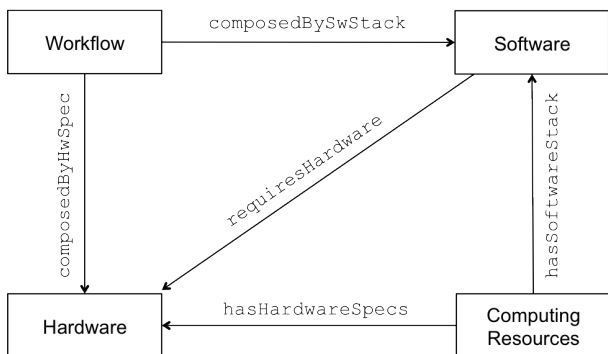


Figure 1: Overview of the ontology network (→ denotes inter-domain relation).

2.1. WICUS ontology network

We define semantic models for describing the main domains of a computational infrastructure, and for defining the taxonomy of concepts and the relationships between them. These models describe software components, hardware specifications, and computational resources (in the form of VMs). They also capture infrastructure dependencies of the workflows (e.g., services that must be running, available libraries, etc.).

As a result, this process facilitates experiment’s reusability since a new experiment, which may reuse parts of the workflow previously modeled, or a reproduction of a workflow, would benefit from the infrastructure dependencies already described.

We have identified four main domains of interest for documenting computational scientific infrastructures [21], and developed a set of models, one for each domain, and an ontology network that defines the inter-domain relations between these models (Figure 1):

- *Hardware domain*: identifies the most common hardware information, including CPU, Storage and RAM memory, and their capacities.
- *Software domain*: defines the software components involved on the execution. It includes the pieces of executable software (e.g., scripts, binaries, and libraries) used in the experiment. In addition, dependencies between those components and configuration information are also defined, as well as the steps required for their deployment.
- *Workflow domain*: describes and relates workflow fragments (a.k.a transformations) to their dependencies. Therefore, scientists can understand which are the relevant infrastructure components for each part of the workflow.
- *Computing Resources domain*: expresses the information about the available computing resources. In this domain, only virtualized resources are currently considered (i.e., virtual machine). It includes the description of the VM image, its provider, and specifications.

The Workflow Infrastructure Conservation Using Semantics ontology (WICUS) is an OWL2 (Web Ontology Language) ontology network that implements the conceptualization of these domains. This ontology network is available online¹ and its

goal is to define the relevant and required properties for describing scientific computational infrastructures. The detailed description of the ontologies, including their main terms and relation in the context of a workflow execution are provided in [21]. Currently, two versions of the ontology network have been released. The latest one, released in August 2014, includes a set of new properties for better describing software and hardware requirements, and also for including the output information of a configuration process (e.g., the resultant IP and port on which a recently deployed service will be listening).

These models have been documented and made available online, and have also been aligned with several of well-known vocabularies, such as P-Plan² and DCMI Metadata Terms³. For example, the class `wreqs:Workflow` that represents a scientific workflow in our domain, has been aligned as a subclass of `p-plan:Plan`, which turns to be a subclass of `prov:Plan` from the PROV vocabulary⁴. These alignments would allow us to integrate our generated data with other datasets describing scientific workflows and their related data and computation resources. Further versions of the WICUS ontology network might also be aligned with new vocabularies.

Listing 1 illustrates a running example that summarizes the main annotations of the SoyKB workflow requirements. SoyKB is a genomics workflow that will be described in detail later (Section 4.1). Lines 1–5 state the individual `Workflow:soykb_WF` as a member of the `wreqs:Workflow` class and indicates that it requires the `SoftwareRequirements:soykb_WF_SOFT_REQ` requirement as part of its execution environment. Lines 6–11 define the different components of the main workflow, which defines their own requirements. Lines 13–16 define the workflow software requirement `SoftwareStack:PEGASUS_WMS_CENTOS_6_5_SOFT_STACK`, which depends on other software stacks such as Condor, SSH or the Java SDK (lines 23–28).

Listing 1: WICUS workflow annotations example.

```

1 Workflow : soykb_WF
2   a wreqs : Workflow ;
3   wreqs : requiresExecutionEnvironment
4     HardwareRequirements : soykb_WF_HW_REQ .
5     SoftwareRequirements : soykb_WF_SOFT_REQ .
6 wreqs : hasSubworkflow
7   ConcreteWorkflow : SELECT_VARIANTS_INDEL_CONC_WF ,
8   ConcreteWorkflow : SORT_SAM_CONC_WF ,
9   ConcreteWorkflow : INDEL_REALIGN_CONC_WF ,
10  ...
11   ConcreteWorkflow : FILTERING_SNP_CONC_WF ,
12
13 SoftwareRequirements : soykb_WF_SOFT_REQ
14   a wreqs : SoftwareRequirements ;
15   wicus : composedBySoftwareStack
16     SoftwareStack : PEGASUS_WMS_CENTOS_6_5_SOFT_STACK .
17
18 SoftwareStack : PEGASUS_WMS_CENTOS_6_5_SOFT_STACK
19   a wstack : SoftwareStack ;
20   wstack : dependsOn
21     SoftwareStack : CENTOS_6_5_OS_SOFT_STACK
22     ...
23     SoftwareStack : JAVA_1.7.0_OPENJDK_X86_64_SOFT_STACK
24     SoftwareStack : CONDOR_CENTOS_6_5_SOFT_STACK
25     SoftwareStack : OPEN_SSH_SERVER_SOFT_STACK
  
```

²<http://purl.org/net/p-plan>

³<http://dublincore.org/documents/dcmi-terms/>

⁴<http://www.w3.org/ns/prov>

¹<http://purl.org/net/wicus>

```

26 SoftwareStack : OPEN_SSH_CLIENTS_SOFT_STACK ;
27 wstack : hasSoftwareComponent
28 SoftwareComponent : PEGASUS_WMS_CENTOS_6_5_SOFT_COMP .

```

To retrieve this information, mechanisms such as the Jena API⁵ or SPARQL queries can be used to traverse the individuals. In this work, we have used SPARQL as it is the most standard and flexible way of accessing RDF data. Listing 2 shows a query example for retrieving the software dependencies of a given workflow. We first inquire for the SoyKB workflow requirements (lines 6–8), and then its requirement’s stacks and their dependencies (lines 9–11).

Listing 2: WICUS SPARQL query example.

```

1 PREFIX wreqs : <http://purl.org/net/wicus-reqs#>
2 PREFIX wicus : <http://purl.org/net/wicus#>
3 PREFIX wstack : <http://purl.org/net/wicus-stack#>
4
5 select ?wf ?stack ?depstack where {
6 <http://purl.org/net/wicus-reqs/resource/
7 Workflow/soykb_WF>
8 wreqs:requiresExecutionEnvironment ?req .
9 ?req a wreqs:SoftwareRequirements .
10 ?req wicus:composedBySoftwareStack ?stack .
11 ?stack wstack:dependsOn ?depstack
12 }

```

In this work, we have mainly focused on software artifacts, arguing that they are the primary resources in terms of computational reproducibility. Nevertheless, we have also modeled the main hardware capabilities required by our applications such as CPU, memory, and storage. We do not address network requirements since the workflows studied used single node configurations and network performance is not a major concern for the reproducibility of the experiments. We acknowledge that network may impact experiment repeatability, hence network is already included as part of the WICUS vocabulary.

2.2. Abstract Deployment Plan

This layer allows WICUS to generate abstract deployment plans regardless of the underlying execution tool. The abstract plan is based on the WICUS Software [21] domain ontology, which defines the software stacks that should be deployed in the execution platform. Figure 2 shows the relationships between the different elements that compose the Stack domain ontology. A Software Stack may be composed of one or more Software Components. Each of them has an associated Deployment Plan according to the target execution platform, which is composed of one or more Deployment Steps.

Listing 3 shows an example of the abstract plan for the SoyKB workflow. The first section of the plan (lines 1–26) describes the deployment of the workflow management system (Pegasus [22]) and its related dependencies. Note that this section is common across all deployment plans for the workflows covered in this work. The remaining lines describe how the SoyKB software is deployed. The SOFTWARE.TAR.GZ stack, which is a dependency for all SoyKB wrappers, is the first component to be deployed (lines 27–29). Finally, the last section of the plan (lines 30–45) describes how the four SoyKB wrappers

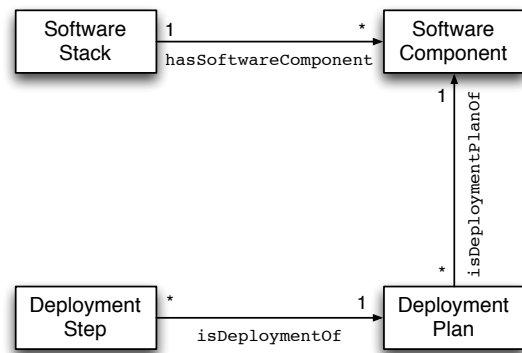


Figure 2: Overview of the WICUS software stack relation diagram.

are deployed. For each wrapper, two deployment steps are required: 1) copy of the program execution binary, and 2) the granting of proper execution permissions.

Listing 3: Abstract deployment plan of the SoyKB WF.

```

1 OPEN_SSH_CLIENTS_SOFT_STACK stack
2 OPEN_SSH_CLIENTS_SOFT_COMP component
3 OPEN_SSH_CLIENTS_DEP_STEP step
4 OPEN_SSH_SERVER_SOFT_STACK stack
5 OPEN_SSH_SERVER_SOFT_COMP component
6 OPEN_SSH_SERVER_DEP_STEP step
7 WGET_SOFT_STACK stack
8 WGET_SOFT_COMP component
9 WGET_DEP_STEP step
10 CONDOR_CENTOS_6_5_SOFT_STACK stack
11 CONDOR_CENTOS_6_5_SOFT_COMP component
12 STOP_CONDOR_DEP_STEP step
13 ADD_CONDOR_REPO_DEP_STEP step
14 CONDOR_YUM_INSTALL_DEP_STEP step
15 CLEAN_AND_SET_CONDOR_DEP_STEP step
16 RESTART_DAEMONS_DEP_STEP step
17 JAVA-1.7.0-OPENJDK.X86_64.SOFT_STACK stack
18 JAVA-1.7.0-OPENJDK.X86_64.SOFT_COMP component
19 JAVA-1.7.0-OPENJDK.X86_64_DEP_STEP step
20 JAVA-1.7.0-OPENJDK-DEVEL.X86_64.SOFT_STACK stack
21 JAVA-1.7.0-OPENJDK-DEVEL.X86_64.SOFT_COMP component
22 JAVA-1.7.0-OPENJDK-DEVEL.X86_64_DEP_STEP step
23 PEGASUS_WMS_CENTOS_6_5_SOFT_STACK stack
24 PEGASUS_WMS_CENTOS_6_5_SOFT_COMP component
25 ADD_PEGASUS_REPO_DEP_STEP step
26 PEGASUS_YUM_INSTALL_DEP_STEP step
27 SOFTWARE_TAR_GZ_SOFT_STACK stack
28 SOFTWARE_TAR_GZ_SOFT_COMP component
29 SOFTWARE_TAR_GZ_DEP_STEP step
30 PICARD_WRAPPER_SOFT_STACK stack
31 PICARD_WRAPPER_SOFT_COMP component
32 PICARD_WRAPPER_DEP_STEP step
33 PICARD_WRAPPER_2_DEP_STEP step
34 SOFTWARE_WRAPPER_SOFT_STACK stack
35 SOFTWARE_WRAPPER_SOFT_COMP component
36 SOFTWARE_WRAPPER_DEP_STEP step
37 SOFTWARE_WRAPPER_2_DEP_STEP step
38 GATK_WRAPPER_SOFT_STACK stack
39 GATK_WRAPPER_SOFT_COMP component
40 GATK_WRAPPER_DEP_STEP step
41 GATK_WRAPPER_2_DEP_STEP step
42 BWA_WRAPPER_SOFT_STACK stack
43 BWA_WRAPPER_SOFT_COMP component
44 BWA_WRAPPER_DEP_STEP step
45 BWA_WRAPPER_2_DEP_STEP step

```

3. Reproducibility in Scientific Workflows

In this section, we introduce the tools used for the instantiation and evaluation of the aforementioned semantic models.

⁵<https://jena.apache.org>

We first describe the Pegasus Workflow Management System (WMS) [22], which is used as the workflow engine, and then a set of reproducibility tools for semantic annotations and experiment management.

3.1. Scientific Workflow Execution

The Pegasus WMS can manage workflows comprised of millions of tasks, recording data about their execution and intermediate results. In Pegasus, workflows are described as abstract workflows, that is, they do not contain resource information, or the physical locations of data and executables. Workflows are described as directed acyclic graphs (DAGs), where nodes represent individual computational tasks and the edges represent data and control dependencies between tasks. The abstract workflow description is represented as a DAX (DAG in XML), capturing all the tasks that perform computations, the execution order of these tasks, and for each task the required inputs, expected outputs, and the arguments with which the task should be invoked.

During a workflow execution, Pegasus translates an abstract workflow into an executable workflow, determining the executables, data, and computational resources required for the execution. Pegasus maps executables to their installation paths or to a repository of stageable binaries defined in a Transformation Catalog (TC). The TC maps logical transformations to physical executables on the system. It also includes information about profiles and environment variables required by transformations, as well as which system they are compiled for, among others. A workflow execution includes data management, monitoring, and failure handling. Individual workflow tasks are managed by a task scheduler (HTCondor [23]), which supervises their execution on local and remote resources.

3.2. Reproducibility Artifacts

To conduct the experimentation on scientific workflows reproducibility, we use the WICUS framework [21], which comprises the semantic models described in Section 2 and a set of tools for annotating and consuming data; and the PRECIP [24] experiment management tool to manage the experiment. In addition, we use Vagrant [25], a tool for deploying virtual deployment environments, to achieve local reproducibility of the experiments. Below, we describe each of these tools in detail.

3.2.1. WICUS

Besides the ontology network introduced before, a set of components have been developed around it, for facilitating the annotation of the resources involved in the execution of a scientific workflow. These tools are not fully automated yet, but represent a first step in helping users define the requirements of their experiments. Figure 3 shows the main modules involved in the process for achieving reproducibility. It also describes the process of data generation and consumption. Below, we provide an overview of each of these modules:

1. **DAX Annotator.** This tool parses a DAX (Pegasus' workflow description) and generates a set of annotations using the terms of the WICUS ontology network, representing

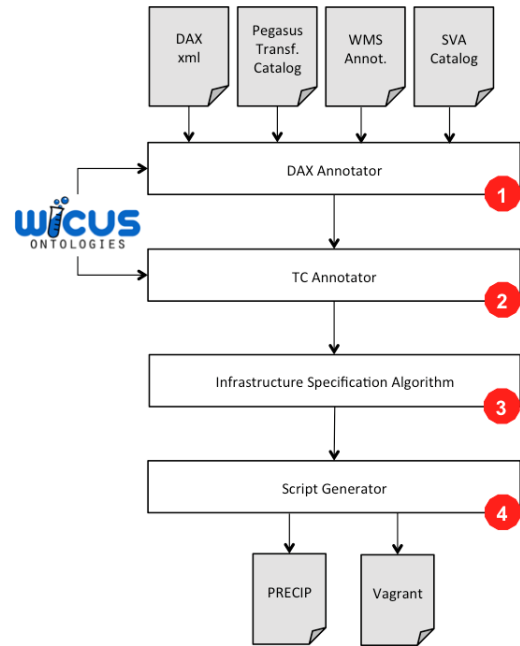


Figure 3: WICUS annotation modules and flow. White boxes represent tools for semantic annotation or algorithms, and grey boxes represent files used or generated by the framework.

workflow transformations and the workflow infrastructure requirements. As a result of this process, a file containing the description of the workflow and its infrastructure requirements is generated.

2. **Transformation Catalog Annotator.** This tool parses the Pegasus Transformation Catalog and the WMS annotations file, to generate two set of annotations: the *Software Components Catalog* file, which contains the set of annotations about the binaries, dependencies, deployment plans and scripts, and the *Workflow & Configuration Annotation* file, containing the configuration information for each workflow execution step, as specified in the transformation catalog.
3. **Infrastructure Specification Algorithm.** This process reads files generated on the previous steps and generates a file containing the *Abstract Deployment Plan* describing the VMs to be created, and the software components to be deployed. This plan will be enacted later using a concrete language.
4. **Script Generator.** This module concretizes the abstract deployment plan using the selected language (either PRECIP or Vagrant in this case) to generate an executable script. New script syntaxes may be added in the future.

The main entries of the WICUS process include the *DAX XML* file (DAG in XML), and the *Pegasus Transformation Catalog* file, which contains the information about the software components required by the workflow. It also requires the *WMS Annotations* file, which describes the Pegasus system, and the *SVA Catalog* file, containing information about the available computational resources. Both files are generated manually us-

ing the WICUS ontologies. Upon completion, the process outputs execution scripts, which create a VM, copy the required binaries, and execute the deployment scripts to set the environment for the workflow execution. It also contains the original experiment commands in order to re-execute it.

In this work, we introduce our current use case using PRECIP and Vagrant as enactment systems for recreating the execution environments. Thanks to the decoupled design of the system, additional solutions could be easily incorporated by extending the *Script Generator* component. For instance, the *Abstract Deployment Plan* could be translated into new syntaxes to explore new technologies such as Docker containers [26], or TOSCA [27] templates.

In the experimentation process (Section 4), we will present a detailed description and the applicability of each module for the target scientific workflow applications.

3.2.2. Infrastructure Specification Algorithm

The third step on the process for achieving reproducibility in scientific workflows (Figure 3) is to execute the Infrastructure Specification Algorithm (ISA). The algorithm retrieves the corresponding information for the workflow and its dependencies from the annotation datasets, and calculates the dependencies and compatibility between requirements and the available computational resources. It also considers the software already installed on the resources to avoid unnecessary installation steps.

Listing 4 shows the pseudo-code of the algorithm. ISA combines the annotated data based on the 4 domain ontologies in order to find a suitable infrastructure specification that meets the requirements of the workflow. The algorithm retrieves and propagates the WMS requirements of the top-level workflow (Workflow domain ontology) to its related sub-workflows (as defined in Figure 7). Requirements and software components are matched, and a dependency graph is built based on the relationship between the requirements and the component dependencies (line 13). ISA then calculates the intersection between the set of components installed on the SVA and the set of components from the dependency graph, selecting the SVA that maximizes the value of that intersection for each sub-workflow (line 17). Software components already available in the SVA are then removed from the dependency graph, as they do not have to be installed (line 19). To reduce the number of SVAs, we have extended ISA to support *Image Appliance* filtering. The algorithm attempts to merge sub-workflow requirements into a single SVA (line 21). Requirements can be merged if all their software components are compatible. Thus, the algorithm filters those that do not meet the hardware requirements specified for the workflow (lines 23–25). Finally, ISA generates a script with the set of required instructions to instantiate, configure, and deploy the computational resources and software components on the corresponding provider (line 27). To enact support to different execution scripts, we introduced an intermediate phase (*Abstract Deployment Plan*), which defines the steps and scripts to be executed, along with their configuration parameters. We also extended ISA to generate Vagrant scripts in addition to PRECIP. A complete description and evaluation of the algorithm can be found in [21].

Listing 4: Pseudo-code overview of the Infrastructure Specification Algorithm (ISA).

```

1 WorkflowRequirementsDataset.load();
2
3 SVADataset.load();
4
5 SoftwareCatalogDataset.load();
6
7 Map<Workflow, List<Requirements>> wfSwReqs =
  retrieveSwRequirements(
    WorkflowRequirementsDataset, WorkflowID );
8
9 Map<Workflow, List<Requirements>>
  propagatedWfSwReqs = propagateSwReqs(
    wfSwReqs );
10
11 List<List<List<SWComponents>>>
  softwareComponents = getSoftwareComponents(
    propagatedWfSwReqs );
12
13 Map<Requirement, D-Graph<SWComponents>>
  softwareComponentsDependencyGraph =
  getSoftwareDependencies(softwareComponents );
14
15 List<SVA> availableSvas =
  getAvailableSvas( providersList );
16
17 Map<Requirements, SVA> maxCompatibilities =
  getCompatibilityIntersection(
    softwareComponentsDependencyGraph,
    availableSvas );
18
19 Map<Requirement, D-Graph<SWComponents>>
  substractedSwComponentsDepGraph =
  substractSoftwareComponents(
    softwareComponentsDependencyGraph,
    maxCompatibilities );
20
21 Map<SVA, List<Requirments>>mergedSvas=
  mergeSubworkflows( propagatedWfSwReqs,
    maxCompatibilities );
22
23 Map<Workflow, List<Requirements>> wfHwReqs =
  retrieveHwRequirements(
    WorkflowRequirementsDataset, WorkflowID );
24
25 Map<SVA, List<Requirments>>filteredSvas=
  getCompatibleHwImageAppliances( mergedSvas,
    wfHwReqs );
26
27 generateScript( filteredSvas,
  substractedSwComponentsDepGraph );

```

3.2.3. PRECIP

The Pegasus Repeatable Experiments for the Cloud in Python (PRECIP) [24] is a flexible experiment management control API for running experiments on all types of Clouds, including academic Clouds such as FutureGrid⁶ and the NSFCloud^{7,8} (through OpenStack), and commercial Clouds such as Amazon EC2⁹ and Google Compute Engine¹⁰. In PRECIP,

⁶<http://portal.futuregrid.org>

⁷<http://www.chameleoncloud.org>

⁸<http://cloudlab.us>

⁹<http://aws.amazon.com/ec2>

¹⁰<https://cloud.google.com/compute>

when an instance is provisioned, the scientist can add arbitrary tags to that instance in order to identify and group the instances in the experiment. Then, future interactions can be performed by using the given tags. API methods such as running remote commands, or copying files, all use tags to specify which instances to target. PRECIP does not force the scientist to use a special VM image, and no PRECIP components need to be pre-installed in the image. Scientists can use any basic Linux image and PRECIP will bootstrap instances using SCP and SSH commands. PRECIP provides functionality to run user-defined scripts on the instances to install/configure software and run experiments, and also manages SSH keys and security groups automatically.

In this work, we use PRECIP to define a script able to reproduce the execution environment of the former experiment, and run it on a Cloud platform.

3.2.4. Vagrant

Vagrant [25] is an open-source and multi-platform solution for deploying development environments locally using virtualization. It relies on virtualization solutions such as Oracle VirtualBox [28] (also open-source) or VMWare¹¹, and support Amazon EC2-like server configurations. Since version 1.6 it also supports Docker [26] containers. Vagrant provides a set of commands and configuration files to enact and customize virtual machines (also referred to as boxes). It allows defining the set of commands and/or scripts to be executed during the different stages of the booting process. Several base images are publicly available for users to download and customize¹².

In this work, we introduce how Vagrant can be used for achieving reproducibility in a local execution environment—usually a scientist’s laptop/desktop computer. As a result, users are able to repeat and modify their original experiment, repurposing or improving it, which is a highly desirable goal of any reproducibility process. By executing Vagrant with the resultant *Vagrantfile* generated by the Infrastructure Specification Algorithm, the user will create a virtual machine on its own computer and automatically execute the workflow, being also able to access it and modify the environment.

4. Experimentation Process

In this section, we instantiate the semantic models introduced in Section 3 (Figure 1) for real scientific workflow applications. This process is an extension of the one introduced in [20], in which we evaluate the improvements on our approach to the Montage [29] workflow. In addition, we extend this evaluation to the Epigenomics¹³ and SoyKB [30, 31] workflows. We study and document these workflows and their execution environments, which include the application software components and the workflow management system. A detailed characterization of these workflows can be found in [32, 33].

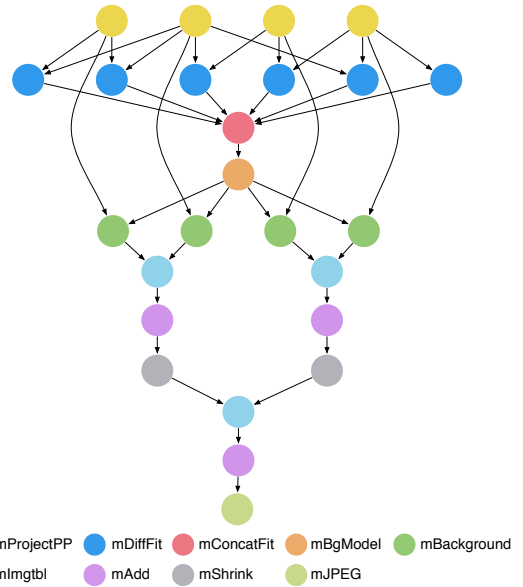


Figure 4: A small (20 node) Montage workflow.

The goal of this experiment is to reproduce original workflow executions on the three different Cloud scenarios: FutureGrid and Amazon EC2 using PRECIP, and a local execution environment by using Vagrant. FutureGrid is an academic Cloud testbed facility that includes a number of computational resources at distributed locations. Amazon Web Services EC2 is a public infrastructure provider, and the *de facto* standard for IaaS Cloud platforms. Since most of Cloud platforms are based on OpenStack¹⁴, our system can be used by most of them.

4.1. Scientific Workflows

Montage. The Montage workflow [29] was created by the NASA Infrared Processing and Analysis Center (IPAC) as an open source toolkit that can be used to generate custom mosaics of astronomical images in the Flexible Image Transport System (FITS) format. In a Montage workflow, the geometry of the output mosaic is calculated from the input images. The inputs are then re-projected to have the same spatial scale and rotation, the background emissions in the images are corrected to have a uniform level, and the re-projected, corrected images are co-added to form the output mosaic. Figure 4 illustrates a small (20 node) Montage workflow. The size of the workflow depends on the number of images required to construct the desired mosaic.

Epigenomics. The USC Epigenome Center is currently involved in mapping the epigenetic state of human cells on a genome-wide scale. The Epigenomics workflow (Figure 5) processes multiple sets of genome sequences in parallel. These sequences are split into subsets, the subsets are filtered to remove contaminants, reformatted, and then mapped to a reference genome. The mapped sequences are finally merged and

¹¹<http://www.vmware.com>

¹²<http://www.vagrantbox.es>

¹³<http://epigenome.usc.edu>

¹⁴<https://www.openstack.org>

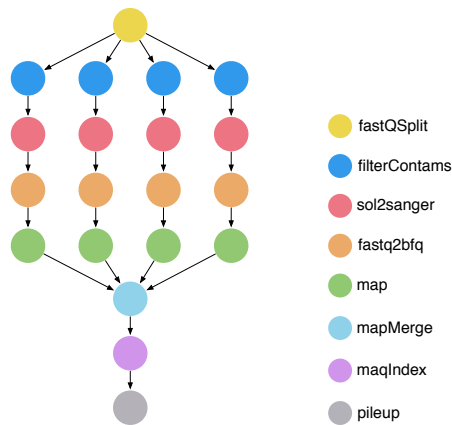


Figure 5: Epigenomics workflow.

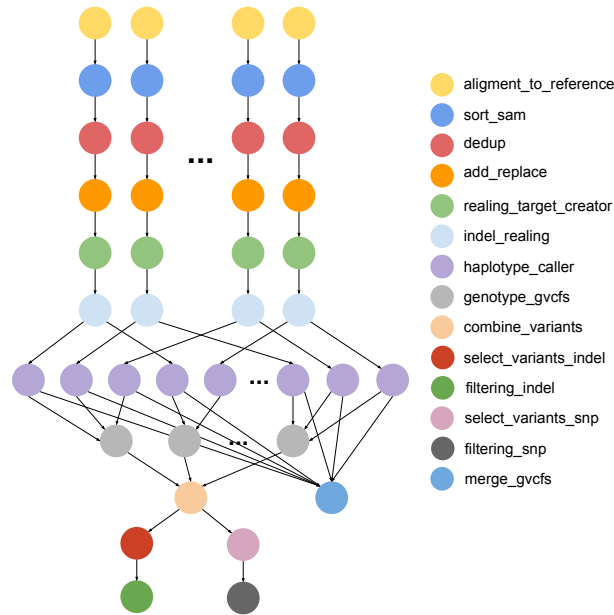


Figure 6: SoyKB workflow.

indexed for later analysis. In this work, the Epigenomics workflow was used to align genome sequence reads to a reference genome for human chromosome 21. The size of the workflow depends on the chunking factor used on the input data, which determines the number of sequence reads in each chunk.

SoyKB. The SoyKB workflow [30, 31] is a genomics pipeline that re-sequences soybean germplasm lines selected for desirable traits such as oil, protein, soybean cyst nematode resistance, stress resistance, and root system architecture. The workflow (Figure 6) implements a Single Nucleotide Polymorphism (SNP) and injection/deletion (indel) identification and analysis pipeline using the GATK haplotype caller¹⁵ and a soybean reference genome. The workflow analyzes samples in parallel to align them to the reference genome, to de-duplicate the data, to identify indels and SNPs, and to merge and filter the results. The results are then used for genome-wide association studies (GWAS) and genotype to phenotype analysis. The workflow instance used in this paper is based on a sample dataset that requires less memory than a full-scale production workflow, however it carries out the same process and requires the same software components.

4.2. Generating Semantic Annotations

In this subsection, we present the annotations generated for each of the scientific workflows presented above and the Pegasus WMS using the WICUS ontology network. All the annotations generated in this work are available through the Research Object (RO) [34] associated to this paper¹⁶.

As described in Figure 3, the first step in the process of documenting a workflow is the annotation of the workflow DAX file. We use the Workflow domain ontology to describe a workflow as 1) an individual that represents the top level workflow, and 2) a set of individuals representing its sub-workflows, one for each transformation. We then generate the necessary requirements, one for the top level workflow, which specifies the WMS

requirements, and one for each sub-workflow, which defines the software components required by each transformation. The data generated using this process follows the Linked Data [35] principles. It uses HTTP URIs exposed in accordance with the RDF standards.

Figure 7 shows a simplified overview of the annotations generated using the WICUS ontology network for the Montage, Epigenomics, and SoyKB workflows as well as for the Pegasus WMS. Below, we describe each of these semantic annotations in detail:

Workflow Management System. We use the Software domain ontology to describe the components that compose the workflow engine (in this case Pegasus) as individuals, and to represent its dependencies. Pegasus relies on HTCondor as the task manager, and both depend on Java and wget. In addition, all components also depend on the operating system, which in our case is CentOS. The process to describe the deployment of the WMS components is based on the installation of and configuration processes as specified in their documentation. As a result, we will define a set of installation scripts for each of the components. These scripts are included as part of the deployment plan along with their configuration information. The WMS components are defined as a requirement (WMS Requirements) using the Workflow domain ontology. This requirement is then linked to each of the workflows included in this work. As a result, Java, wget, HTCondor, and Pegasus WMS should be installed on the target computational resource.

Montage Workflow. We use the Workflow domain ontology to describe the Montage workflow as an individual that represents the top level workflow, and another 9 individuals representing its sub-workflows, one for each transformation. We also generate 9 requirements, which define the software components required by each transformation. At this point, these requirements

¹⁵<https://www.broadinstitute.org/gatk>

¹⁶<http://purl.org/net/FGCS-Wicus-Pegasus-Repro>

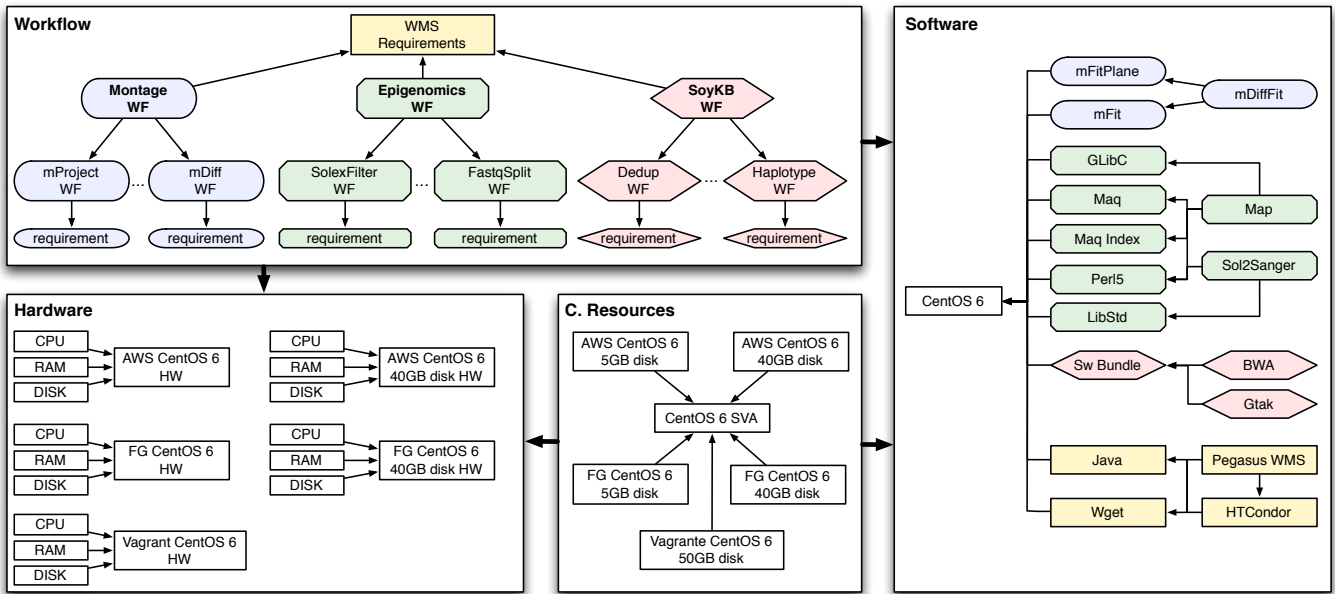


Figure 7: Overview of the generated annotations for the Montage, Epigenomics, and SoyKB workflows using the WICUS ontology network (yellow rectangles represent the workflow component; blue squashed rectangles represent the Montage workflow; green bevelled rectangles represent the Epigenomics workflows; and red hexagons represent the SoyKB workflow).

are empty, as they are not yet related to their software components. Figure 8 shows the set of generated individuals for the Montage workflow.

Application components are described in the Montage workflow’s Transformation Catalog, where the binary file, version, and destination path are defined. These components are also described as individuals using the Software domain ontology. We use this information to generate the configuration parameters of the deployment script, which in this case is the same for all components. The script downloads the binary files from an online repository and copies them to the specified destination path. This process identified 59 software components for the Montage workflow that are annotated and included in the Software Components Catalog. Then, the Transformation Catalog Annotator module relates each transformation requirement, defined using the Workflow domain ontology, to the application component, and therefore to the deployment information. In this experiment, we define 9 Montage components that are linked to the requirements, and another two sub-components that are defined as dependencies in the software catalog (*mDiffFit* depends on the *mDiff* and *mFitPlane* components).

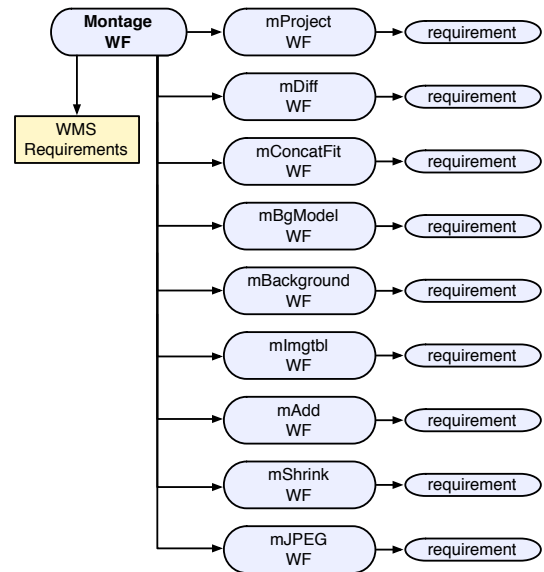


Figure 8: Annotations for the Montage workflows using the Workflow domain ontology.

Epigenomics Workflow. Following the same approach as in the previous case, we use the Workflow domain ontology to describe the Epigenomics workflow as an individual that represents the top level workflow, and another 8 individuals representing its sub-workflows, one for each transformation. We then annotate the components described in the Epigenomics’ Transformation Catalog as individuals using the Software domain ontology. We have also identified and annotated 6 software dependencies related to the workflow, which include the

Perl¹⁷ interpreter, the GNU libc¹⁸ and Libstdc++¹⁹ libraries, and two other binaries from the Epigenomics distribution, *maq* and *maqindex*.

SoyKB Workflow. We describe the SoyKB workflow as an individual that represents the top level workflow, and another 14

¹⁷<https://www.perl.org>

¹⁸<http://www.gnu.org/software/libc>

¹⁹<http://gcc.gnu.org/libstdc++>

individuals representing each transformations. For the sake of simplicity, we do not show the annotations for this workflow. Although SoyKB is the largest workflow in terms of its number of steps (around 670) among the other workflows included in this work, it defines only four software components as dependencies (`bwa-wrapper`, `gatk-wrapper`, `picard-wrapper`, and `software-wrapper`). These components are software wrappers that invoke different libraries and binaries depending on the parameters used for the execution of the workflow. The components are included in a software bundle that is deployed on the computational nodes. Hence, a dependency for this bundle has been included in the Software Components Catalog.

Computational Resources. We use the Computing Resources and Hardware domain ontologies to describe computational resources. For each Cloud resource (Amazon EC2 and FutureGrid), we defined two virtual machines: one that meets the requirements for the Montage and Epigenomics workflows (requires smaller disk space); and another for the SoyKB workflow (requires larger disk space). In both cases, we generated conceptually equivalent appliances, as they both provide the CentOS 6 operating system, but differ in the hardware configuration. We attempt to reduce the resource consumption of Cloud resources due to the cost of storing/transferring VM images. Since Vagrant execution is performed locally, we generated a single VM appliance that meets the requirements of all workflows.

The description of the appliances is then included in the Scientific Virtual Appliances catalog (SVA, Figure 3). According to the SVA model, an SVA is supported by one or more Image Appliances, which represent the set of VMs belonging to a single infrastructure provider that are able to enact the characteristics (software and hardware) specified for the SVA. Thus, we created five Image Appliances that are grouped into a single Scientific Virtual Appliance (CentOS 6 SVA), which is defined by the CentOS 6 software stack. Table 1 summarizes the main characteristics of the five appliances we have annotated.

	Amazon EC2		FutureGrid		Vagrant
	Small	Large	Small	Large	
RAM (GB)	7	7	8	8	4
Disk (GB)	5	40	5	40	50
CPU (GHz)	2.4	2.4	2.9	2.9	2.83
CPU Arch.	64 bits		64 bits		64 bits
OS	CentOS 6		CentOS 6		CentOS 6

Table 1: CentOS 6 Image Appliances.

Hardware Requirements. For each scientific workflow, we have also analyzed the hardware specifications required for their execution. Table 2 shows the minimum threshold per workflow for each requirement. During the computational resource selection process (described in the following section), we will consider that any resource that meets the requirements specified by a workflow will be a valid candidate for running the workflow. Since we do not target workflow execution performance, but a correct execution of the workflow, we have not identified any specific capacity regarding CPU clock speed.

	CPU (GHz)	CPU Arch.	RAM (GB)	Disk (GB)
Montage	-	64 bits	4	4
Epigenomics	-	64 bits	4	4
SoyKB	-	64 bits	4	10

Table 2: Workflow hardware requirements.

4.3. Reproducing Workflow Executions

In all cases, after executing the ISA over the described annotations for the three workflows, we were able to map the abstract plan either into a PRECIP or a Vagrant script (depending on the specified provider). Each generated script is composed of the following main sections:

- *Experiment Creation:* generates a new experiment using the given VM image ID and the user credentials for the selected infrastructure provider;
- *Software Deployment:* executes the set of instructions defined on the deployment plan of each software component to install and configure the required software to execute the workflow. In this section, both the workflow management system and the application are deployed with their dependencies;
- *User Setup:* creates a user account on the VM (if it does not exist) and configures the necessary pair of SSH keys to enable file transfer and execution. This account will be used to run the workflow;
- *Data Stage and Workflow Execution:* stages all the input data required for the workflow execution on the VM, and launches the workflow execution. Since our work is focused on infrastructure reproducibility, data and workflow management are not covered in our approach. This part of the script is generated ad-hoc for each workflow.

Note that all the configuration and deployment commands (first 3 sections) require superuser privileges on the VM. The workflow execution, however, is performed under the user account created in the third section.

4.4. Results and Discussion

We executed the resultant scripts for the three workflows over their corresponding platforms. That is, a total of 9 different executions, as each workflow is executed on Futuregrid and Amazon EC2 using PRECIP, and in a local Vagrant execution environment. All the executions were compared to their original one in a predefined VM image, where the execution environment was already in place.

Results show that the VM execution environments deployed by all scripts are able to fully execute their related workflows. To check that not only the workflows are successfully executed but also that the results are correct and equivalent, we checked their produced output data. In the case of Montage, which produces an image as output, we used a perceptual hash tool²⁰

²⁰<http://www.phash.org>

to compare the resulting image (0.1 degree image of the sky) against the one generated by the baseline execution, obtaining a similarity factor of 1.0 (over 1.0) with a threshold of 0.85. In the Epigenomics and SoyKB workflows, the output data is non-deterministic due to the existence of probabilistic steps. In this case, the use of a hash method is unfeasible. Hence, we validated the correct execution of the workflow by checking that correct output files were actually produced, and that the standard errors produced by the applications did not contain any error message. In both cases the results obtained in each infrastructure were equivalent in terms of their size (e.g., number of lines) and content.

Experimental results show that scientific experiments in form of computational workflows can be reproduced based on semantic annotations using a proper vocabulary. The set of tools introduced in this work enables the generation of semantic annotations, which ease the effort of documenting the execution environment. However, those tools do not provide a fully automated process, thus some manual interactions are still required. Two of the entry files of the reproducibility process (*WMS Annotations* and *SVA Catalog*, Figure 3) are manually generated. Although these files require some effort to elaborate, they are created once and reused for all workflows of a workflow management system. Identifying components and dependencies of a WMS usually requires specialized knowledge, however this information is not often explicitly described.

When reproducing workflows from different WMS, a new set of annotations would be required, whereas the *SVA Catalog* could be reused. This catalog is extended as new computational resources are added to the system. Most of the software involved on the execution of the three workflows studied in this work was, and still is, available online either through distribution packages repositories or hosted online by different institutions. Other software applications, such as the ones designed for the workflow, have been hosted in a public repository for this work. We decided to upload them to GitHub²¹ to ensure that they could be remotely retrieved.

All the original and generated scripts are available as part of the experimental material included in the RO associated with this work. This RO also contains pointers to the software and resources used in the experimental evaluation.

The reproducibility of a computational environment presupposes that at least some differences may appear when compared to the former counterpart. In this work, we assume that these differences occur when using virtualization technologies and dynamic deployment of software components. Therefore, we obtain an equivalent infrastructure, which might not be an exact copy of the original one, but exposes the required characteristics for executing the workflow. This approach allows scientists to understand how the underlying infrastructure is defined, and to evaluate the impact of different configurations on the experiment, as the configuration parameters of the environment are explicitly defined in the semantic annotations.

5. Related Work

A computational experiment involves several elements that must be conserved to ensure reproducibility. In the last year several studies and initiatives have been conducted for solving its associated challenges [36, 37]. Most of the works addressed the conservation of data and the workflow description, however the computational environment is often neglected. Recent studies have exposed the necessity of publishing adequate descriptions of the runtime environment of experiments to avoid replication hindering [38]. As a result, there is an increased focus on the number of publications providing associated experimental materials [39, 40].

A study to evaluate reproducibility in scientific workflows was conducted in [41]. The study evaluated a set of domain-specific workflows, available in myExperiment [42], to identify causes of workflow decay. The study showed that nearly 80% of the workflows cannot be reproduced, that about 12% of these reproducibility issues are due to the lack of information about the execution environment, and that 50% of them are due to the use of third-party resources such as web services and databases that are not available anymore.

Recently, another comprehensive study has been published [43], where 601 papers from ACM conferences were surveyed, studying how authors share the data and code supporting their results. Authors found that 402 of those papers were supported by code. In this study authors tried to obtain the code related to each publication, looking for links within the paper itself, searching on code repositories, and contacting the authors when necessary. After the code was obtained, several students were asked to try to build it. This whole process was limited by experimental design to a period of 30 minutes. Results showed that in 32.3% of the 402 papers students were able to obtain the code and build the code within the given period. In 48.3% of the cases, code was built with some extra effort, and in 54% of the papers code was either built or the authors stated the code would build with reasonable effort. Authors proposed, as a result of this study, a *sharing specification* for publications that allow to state the level of sharing of each paper.

The workflow paradigm has been widely adopted in the bioinformatics community, for studying genome sequencing [6, 7], disease-related experiments [8, 9] and many others. Several studies have exposed the difficulties of trying to reproduce experimental results on life sciences, such as biology [10] and cancer analysis [11].

Replicability and reproducibility of computational experiments using Cloud computing resources and software descriptions have been proposed as an approach for those studies in which performance is not a key experimental result [44].

The Executable Paper Grand Challenge [45] and the SIGMOD conference in 2011 [46] highlighted the importance of allowing the scientific community to reexamine experiment execution. The conservation of virtual machine (VM) images emerges as a way of preserving the execution environment [47, 48]. However, the high storage demand of VM images remains a challenging problem [17, 18]. Moreover, the cost of storing and managing data in the Cloud is still high, and

²¹<https://github.com>

the execution of high-interactivity experiments through a network connection to remote virtual machines is also challenging. A list of advantages and challenges of using VMs for achieving reproducibility was exposed in [49].

In [50], authors introduce ReproZip, a provenance-based tool for tracking the operating system calls to identify the libraries and data dependencies, as well as the configuration parameters involved in an experiment. The tool combines all these dependencies into a single package that can be used to reproduce an experiment. Although this approach avoids storing VM images, it still requires storing the application binaries and their dependencies. In this work, we use semantic annotations to describe these dependencies. In our approach, we do not try to capture and package the original elements of the former infrastructure (copy the files and libraries). Instead, we describe them in a way that an available counterpart can be retrieved and tuned to expose the same characteristics. We claim that we do not only provide a technical solution, but also increase the reproducibility degree of the experiment by exposing the knowledge about the underlying infrastructure in a structured way. ReproZip would attain better dynamism if an approach as the one proposed in this work is used, since it abstracts the description of the infrastructure from the concrete elements that are involved in the original experiment.

Galaxy [51] is a web-based WMS that aims to bring data analysis capabilities to non-expert users in the biological sciences domain, hiding the implementation details of the underlying tools for workflow developers. These workflows can be deployed on a Galaxy server locally or on the Cloud, through the Galaxy CloudMan component. The main goals of the Galaxy framework are accessibility to biological computational capabilities and reproducibility of the analysis result by tracking the information related to the full process. Even when this approach has proved to be successful in many cases, it does not consider infrastructure aspects.

Software components cannot be preserved just by maintaining their binary executable code, but by guaranteeing the performance of their features. In [52], the concept of adequacy is introduced to measure how a software component behaves relative to a certain set of features. Our work is based on this same concept, where we build a conceptual model to semantically annotate the relevant properties of each software component. Then, we use scripting to reconstruct an equivalent computational environment using these annotations.

Some efforts have been undertaken to define syntaxes for describing the execution environment of scientific applications. The recently released TOSCA²² standard has been proposed to specify the components and life cycle of scientific workflows, including their dependencies. TOSCA is a Cloud management standard that creation of declarative templates for defining the workflow as well as the information required for enacting its environment. As a result, workflows are portable to any TOSCA-compliant provider [27]. Although this approach significantly increases the reproducibility degree of the experiment, it requires the workflow to be rewritten. In this work, however, our

goal is to describe the workflow to obtain an equivalent execution environment depending on the available providers.

Several tools have been developed to enact the supporting environment. For instance, Cloudify²³ (a tool based on the TOSCA standard) allows to define the configuration of a given application and its dependencies. These descriptions can be later used to state the application's installation process as well as its lifecycle. Infrastructure Manager [53] is a tool that adds an abstraction layer for selecting, deploying, and configuring VMs in several providers, such as OpenNebula [54], Microsoft Azure²⁴, and OpenStack. In this work, we explore PRECIP and Vagrant, but thanks to WICUS decoupled architecture, other tools can be easily incorporated.

A recent and relevant contribution to the state of the art of workflow preservation was developed within the context of the TIMBUS project [55]. The project aimed to preserve and ensure the availability of business processes and their computational infrastructure, aligned with the enterprise risk and the business continuity management. They also proposed a semantic approach for describing the execution environment of a process. However, even though TIMBUS has studied the applicability of their approach to the eScience domain, their approach is mainly focused on business processes.

Semantics have also been proposed in the area of biomedical research as a way for achieving reproducibility of published experiments.

For instance, in [19] authors propose a taxonomy for classifying software artifacts in the context of biological computations in the same way that gene products or phenotypes are classified. To build the classification, they developed the *Software Ontology* (SWO), a model for cataloguing the software involved on the storage and management of data. The SWO is a comprehensive catalogue of software-related concepts for composing description of processing artifacts. It contains more than 4000 classes distributed in several modules for describing the different areas of software information. Many of these concepts share commonalities with the concepts used in the WICUS network, in particular on the Software Domain ontology. We identify a conceptual equivalences between `swo:Software` and `wstack:SoftwareComponent` classes, or `swo:hasVersion` and `wstack:hasVersion`, even though their context and purpose are not the same. While the SWO ontology focuses on describing the properties of a biology software, aiming to make it more understandable and potentially reproducible, the WICUS ontologies focus on the deployment of software artifacts and stating their necessary dependencies. We aim to develop a more generic and flexible ontology that can be applied in different scientific domains.

The EDAM ontology²⁵ is a more concise ontology for describing bioinformatics resources. It includes concepts for annotating different data types, identifiers and formats, as well as topics and operations. Even when concept of operations can be related to the concept of software components, they are more

²²<http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html>

²³<http://getcloudify.org/>

²⁴<http://azure.microsoft.com>

²⁵<http://edamontology.org/page>

related to data transformations rather than to executable deployment and computations. The ontologies proposed in this work are not strictly focused on data descriptions and analysis.

A plethora of models have also been proposed to describe Cloud environments [56]. Many initiatives have been undertaken to define Cloud Computing interfaces, such as the Open Cloud Computing Interface (OCCI²⁶), one of the pioneering initiatives, and Unified Cloud Interface (UCI²⁷), which provides several ontologies describing Cloud infrastructures besides defining an interface. In [57], Cloud taxonomies for describing services and providers from the three main layers of Cloud Computing are proposed. The CoCoOn ontology [58] describes the main terms of Infrastructure as a Service (IaaS) providers, including functional (e.g., memory or pricing) and non-functional (e.g., quality of service) concepts. Even though we characterize the main properties of computational resources, which belong to the IaaS layer, we do not aim to model the Cloud providers themselves neither the quality of services provided by them. We acknowledge that those considerations are relevant for the scheduling problem, however they are out of the scope of this work.

6. Conclusion and Future Work

In this work, we proposed a semantic modeling approach to conserve computational environments in scientific workflow executions, where the resources involved in the execution of the experiment are described using a set of semantic vocabularies. We defined and implemented four domain ontologies, aggregated in the the WICUS ontology network. From these models, we defined a process for documenting workflow applications, the workflow management system, and their dependencies.

We conducted experiments with three real workflow applications (Montage, Epigenomics, and SoyKB) from different sciences using the Pegasus WMS. We used the Infrastructure Specification Algorithm (ISA) to obtain a set of PRECIP and Vagrant scripts to describe and execute the experiment. Experimental results show that our approach can reproduce an equivalent execution environment of a predefined VM image on academic, public, and local Cloud platforms.

Semantic annotations of the computational environment, combined with the ISA and the scripting functionality provided by PRECIP and Vagrant, is a powerful approach for achieving reproducibility of computational environments in future experiments, and at the same time addresses the challenges of high storage demand of VM images. In this work, we have demonstrated that an equivalent computational environment (that fulfills the requirements of a workflow) can be obtained from simple base VM images, which are already available in most Cloud platforms. Consequently, there is no need to create and store a new VM image for each workflow. Therefore, this approach significantly diminishes the costs and efforts in maintaining a VM image that may have limited usage. We acknowledge that

this approach may be limited to the online (publicly) availability of the software, however most of the scientific tools developed nowadays are hosted as open-source projects.

In this work, we have focused on the execution aspects of a workflow, which do not consider the experimental data as part of the reproducibility process. Including it during the description of the workflow, and using it as a decision criteria when generating an infrastructure specification would lead to a more effective process. We then aim to study how network capabilities or extra data volumes can be included as part of the specification process, reducing the data-related issues of an experiment from its infrastructure point of view.

To attain reproducibility, the proposed model requires that information about the software components involved on the workflow execution should be available and as explicit as possible. In Pegasus, the transformation catalog provides all necessary information. Similar catalogs (or files) are also available in other workflow systems. Also, knowledge about semantic technologies is required for generating some of the annotations for describing the underlying infrastructure. Currently, the description of the WMS and the computational resources are manually generated, whereas the remaining annotations are performed by tools of the WICUS system. Future work include the development of tools to fully (or partially) automate this process. The annotation process may be assisted by experts, in the same way librarians aid book authors and publishers to conserve their contributions.

The results of this work also show how components, such as the workflow system, can be annotated once and then reused among workflows. We envision a library of workflow descriptions in which components and tools can be easily reused, even during the development process of the workflow. Many workflows are built upon previous workflows, especially within the context of a scientific domain, and hence having such type of library would be helpful. We plan to study how and when those libraries can be built, analyzing their degree of reuse. As introduced in this work, our descriptions of the environment, generated using the WICUS ontology network, is compliant with the Linked Data principles. This would allow its future publication and integration with other information sources, which fit the idea of a distributed and structured library of descriptions that could be publicly available worldwide.

We are currently studying the applicability of our approach to other workflow applications (from different scientific areas) and systems. Although the Cloud systems included in this work cover a representative spectrum of Cloud providers, we are also considering other Cloud scenarios and emerging container-based solutions.

Currently, our approach is able to represent multi-node infrastructures, even though in this work we have only analyzed and reproduced workflows that were configured to use a single node environment. We then plan to extend the scope of our approach to enable reproducibility of more complex and larger execution infrastructures, such as large clusters and topology-specific network platforms.

We will also work on increasing the degree of automation of the semantic annotation process to describe both the workflow

²⁶<http://occi-wg.org>

²⁷<http://code.google.com/p/unifiedcloud>

application and the workflow management system. As introduced in this work, WICUS is an ongoing effort, thus we also plan to extend the ontology network to include new concepts and relations such as software variants, incompatibilities, and user policies for resource consumption.

Acknowledgements

This material is based upon work supported in part by the National Science Foundation under Grant No. 0910812 to Indiana University for “FutureGrid: An Experimental, High-Performance Grid Test-bed” and the FPU grant from the Spanish Science and Innovation Ministry (MICINN), and the Ministerio de Economía y Competitividad (Spain) project “4V: Volumen, Velocidad, Variedad y Validez en la Gestión Innovadora de Datos” (TIN2013-46238-C4-2-R). This research was also supported by the National Science Foundation under the SI²–SSI program, award number 1148515. We also thank Gideon Juve and Karan Vahi for their valuable help.

References

- [1] Reproducible research: Addressing the need for data and code sharing in computational science, <http://www.stanford.edu/~vcs/Conferences/RoundtableNov212009/RoundtableOutputDeclaration.pdf> (2009).
- [2] D. James, N. Wilkins-Diehr, V. Stodden, D. Colbry, C. Rosales, M. Fahy, J. Shi, R. Ferreira da Silva, K. Lee, R. Roskies, L. Loewe, S. Lindsey, R. Kooper, L. Barba, D. Bailey, J. Borwein, O. Corcho, E. Deelman, M. Dietze, B. Gilbert, J. Harkes, S. Keele, P. Kumar, J. Lee, E. Linke, R. Marciano, L. Marini, C. Mattman, D. Mattson, K. McHenry, R. McLay, S. Miguez, B. Minsker, M. Perez-Hernandez, D. Ryan, M. Rynga, I. Santana-Perez, M. Satyanarayanan, G. S. Clair, K. Webster, E. Hovig, D. Katz, S. Kay, G. Sandve, D. Skinner, G. Allen, J. Cazes, K. W. Cho, J. Fonseca, L. Hwang, L. Koesterke, P. Patel, L. Pouchard, E. Seidel, I. Suriarachchi, Standing together for reproducibility in large-scale computing: Report on reproducibility@XSEDE (2014). URL <http://arxiv.org/abs/1412.5557>
- [3] V. Stodden, F. Leisch, R. D. Peng (Eds.), *Implementing Reproducible Research*, Chapman & Hall, 2014.
- [4] I. Taylor, E. Deelman, D. Gannon, M. Shields, *Workflows for e-Science: Scientific Workflows for Grids*, Springer-Verlag New York, Inc., 2007.
- [5] T. Craddock, P. Lord, C. Harwood, A. Wipat, E-science tools for the genomic scale characterisation of bacterial secreted proteins, in: All hands meeting, 2006, pp. 788–795.
- [6] D. Blankenberg, G. V. Kuster, N. Coraor, G. Ananda, R. Lazarus, M. Mangan, A. Nekrutenko, J. Taylor, Galaxy: a web-based genome analysis tool for experimentalists, *Current protocols in molecular biology* (2010) 19–10.
- [7] B. Giardine, C. Riemer, R. C. Hardison, R. Burhans, L. Elmitski, P. Shah, Y. Zhang, D. Blankenberg, I. Albert, J. Taylor, W. Miller, W. Kent, A. Nekrutenko, Galaxy: a platform for interactive large-scale genome analysis, *Genome research* 15 (10) (2005) 1451–1455.
- [8] P. Fisher, H. Noyes, S. Kemp, R. Stevens, A. Brass, A systematic strategy for the discovery of candidate genes responsible for phenotypic variation, in: *Cardiovascular Genomics*, Springer, 2009, pp. 329–345.
- [9] R. Gaizauskas, N. Davis, G. Demetriou, Y. Guod, I. Roberts, Integrating biomedical text mining services into a distributed workflow environment, in: *Proceedings of UK e-Science All Hands Meeting*, 2004.
- [10] J. P. A. Ioannidis, D. B. Allison, C. A. Ball, I. Coulibaly, X. Cui, A. C. Culhane, M. Falchi, C. Furlanello, L. Game, G. Jurman, J. Mangion, T. Mehta, M. Nitzberg, G. P. Page, E. Petretto, V. van Noort, Repeatability of published microarray gene expression analyses, *Nat Genet* 41 (2) (2009) 149–155. URL <http://dx.doi.org/10.1038/ng.295>
- [11] T. M. Errington, F. E. Tan, J. Lomax, N. Perfito, E. Iorns, W. Gunn, K. D. Evans, Reproducibility Project: Cancer Biology, osf.io/e81x1/ (2015).
- [12] C. Drummond, Replicability is not reproducibility: Nor is it good science, in: *Proceedings of the Evaluation Methods for Machine Learning Workshop at the 26th ICML*, 2009.
- [13] G. King, Replication, replication, *PS: Political Science and Politics* 28 (3) (1995) 443–499.
- [14] D. Garijo, S. Kinnings, L. Xie, L. Xie, Y. Zhang, P. E. Bourne, Y. Gil, Quantifying reproducibility in computational biology: The case of the tuberculosis drugome, *PLoS ONE* 8 (11) (2013) e80278. doi:10.1371/journal.pone.0080278. URL <http://dx.doi.org/10.1371%2Fjournal.pone.0080278>
- [15] S. L. Kinnings, L. Xie, K. H. Fung, R. M. Jackson, L. Xie, P. E. Bourne, The mycobacterium tuberculosis drugome and its polypharmacological implications, *PLoS computational biology* 6 (11) (2010) e1000976.
- [16] M. Gavish, D. Donoho, A universal identifier for computational results, *Procedia Computer Science* 4 (2011) 637 – 647, proceedings of the ICCS’11. doi:<http://dx.doi.org/10.1016/j.procs.2011.04.067>.
- [17] B. Mao, H. Jiang, S. Wu, Y. Fu, L. Tian, Read-performance optimization for deduplication-based storage systems in the cloud, *ACM Transactions on Storage (TOS)* 10 (2) (2014) 6. doi:10.1145/2512348.
- [18] X. Zhao, Y. Zhang, Y. Wu, K. Chen, J. Jiang, K. Li, Liquid: A scalable deduplication file system for virtual machine images, *IEEE Transactions on Parallel and Distributed Systems* 25 (5) (2014) 1257–1266. doi:10.1109/TPDS.2013.173.
- [19] J. Malone, A. Brown, A. Lister, J. Ison, D. Hull, H. Parkinson, R. Stevens, The Software Ontology (SWO): a resource for reproducibility in biomedical data analysis, curation and digital preservation, *Journal of Biomedical Semantics* 5 (1) (2014) 25+. doi:10.1186/2041-1480-5-25. URL <http://dx.doi.org/10.1186/2041-1480-5-25>
- [20] I. Santana-Perez, R. Ferreira da Silva, M. Rynga, E. Deelman, M. S. Pérez-Hernández, O. Corcho, A semantic-based approach to attain reproducibility of computational environments in scientific workflows: A case study, in: *Euro-Par 2014: Parallel Processing Workshops*, Vol. 8805 of Lecture Notes in Computer Science, 2014, pp. 452–463. doi:10.1007/978-3-319-14325-5_39.
- [21] I. Santana-Perez, M. S. Pérez-Hernández, Towards reproducibility in scientific workflows: An infrastructure-based approach, *Scientific Programming* 2015. doi:10.1155/2015/243180.
- [22] E. Deelman, K. Vahi, G. Juve, M. Rynga, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, K. Wenger, Pegasus, a workflow management system for science automation, *Future Generation Computer Systems* 46 (2015) 17–35. doi:10.1016/j.future.2014.10.008.
- [23] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: The condor experience: Research articles, *Concurr. Comput. : Pract. Exper.* 17 (2-4) (2005) 323–356. doi:10.1002/cpe.v17:2/4.
- [24] S. Azarnoosh, M. Rynga, G. Juve, E. Deelman, M. Niec, M. Malawski, R. Ferreira da Silva, Introducing PRECIP: an API for managing repeatable experiments in the cloud, in: *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, Vol. 2 of CloudCom, 2013, pp. 19–26. doi:10.1109/CloudCom.2013.98.
- [25] J. Palat, Introducing vagrant, *Linux Journal* 2012 (220) (2012) 2.
- [26] D. Merkel, Docker: Lightweight linux containers for consistent development and deployment, *Linux J.* 2014 (239). URL <http://dl.acm.org/citation.cfm?id=2600239.2600241>
- [27] R. Qasha, J. Cala, P. Watson, Towards automated workflow deployment in the cloud using toasca, in: *8th IEEE International Conference on Cloud Computing, New York, USA (IEEE CLOUD)*, 2015.
- [28] J. Watson, Virtualbox: Bits and bytes masquerading as machines, *Linux J.* 2008 (166). URL <http://dl.acm.org/citation.cfm?id=1344209.1344210>
- [29] G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, D. S. Katz, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, M.-H. Su, Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand, in: *Astronomical Telescopes and Instrumentation*, Vol. 5493, International Society for Optics and Photonics, 2004, pp. 221–232. doi:10.1117/12.550551.
- [30] T. Joshi, B. Valliyodan, S. M. Khan, Y. Liu, J. V. Maldonado dos Santos, Y. Jiao, D. Xu, H. T. Nguyen, N. Hopkins, M. Rynga, N. Merchant,

- Next generation resequencing of soybean germplasm for trait discovery on xseede using pegasus workflows and iplant infrastructure, in: XSEDE 2014, 2014, poster.
- [31] T. Joshi, M. R. Fitzpatrick, S. Chen, Y. Liu, H. Zhang, R. Z. Endacott, E. C. Gaudiello, G. Stacey, H. T. Nguyen, D. Xu, Soybean knowledge base (soykb): a web resource for integration of soybean translational genomics and molecular breeding, *Nucleic Acids Research* 42 (D1) (2014) D1245–D1252. doi:10.1093/nar/gkt905.
- [32] R. Ferreira da Silva, G. Juve, E. Deelman, T. Glatard, F. Desprez, D. Thain, B. Tovar, M. Livny, Toward fine-grained online task characteristics estimation in scientific workflows, in: 8th Workshop on Workflows in Support of Large-Scale Science, WORKS '13, 2013, pp. 58–67. doi:10.1145/2534248.2534254.
- [33] R. Ferreira da Silva, G. Juve, M. Rynge, E. Deelman, M. Livny, Online task resource consumption prediction for scientific workflows, *Parallel Processing Letters* 25 (3). doi:10.1142/S0129626415410030.
- [34] Ó. Corcho, D. Garijo Verdejo, K. Belhajjame, J. Zhao, P. Missier, D. Newman, R. Palma, S. Bechhofer, E. García Cuesta, J. M. Gómez-Pérez, et al., Workflow-centric research objects: First class citizens in scholarly discourse., in: Proc. Workshop on the Semantic Publishing (SePublica), Informatica, 2012.
- [35] T. Heath, C. Bizer, *Linked Data: Evolving the Web Into a Global Data Space*, Synthesis Lectures on Web Engineering Series, Morgan & Claypool, 2011.
URL <https://books.google.es/books?id=0Fv59Wcfkx8C>
- [36] T. Hothorn, F. Leisch, Case studies in reproducibility, *Briefings in Bioinformatics* 12 (3) (2011) 288–300. arXiv:<http://bib.oxfordjournals.org/content/12/3/288.full.pdf+html>, doi:10.1093/bib/bbq084.
URL <http://bib.oxfordjournals.org/content/12/3/288.abstract>
- [37] S. Arabas, M. R. Bareford, I. P. Gent, B. M. Gorman, M. Hajjarabderkani, T. Henderson, L. Hutton, A. Konovalov, L. Kotthoff, C. McCreesh, R. R. Paul, K. E. Petrie, A. Razaq, D. Reijtsbergen, An open and reproducible paper on openness and reproducibility of papers in computational science, *CoRR abs/1408.2123*.
URL <http://arxiv.org/abs/1408.2123>
- [38] N. D. Rollins, C. M. Barton, S. Bergin, M. A. Janssen, A. Lee, A computational model library for publishing model documentation and code, *Environmental Modelling and Software* 61 (0) (2014) 59 – 64. doi:<http://dx.doi.org/10.1016/j.envsoft.2014.06.022>.
URL <http://www.sciencedirect.com/science/article/pii/S1364815214001959>
- [39] C. Titus Brown, A. Howe, Q. Zhang, A. B. Pyrkosz, T. H. Brom, A Reference-Free Algorithm for Computational Normalization of Shotgun Sequencing Data, *ArXiv e-prints arXiv:1203.4802*.
- [40] C. Titus Brown, A. Howe, Q. Zhang, A. B. Pyrkosz, T. H. Brom, A Reference-Free Algorithm for Computational Normalization of Shotgun Sequencing Data, <http://ged.msu.edu/papers/2012-diginorm>, (accessed 2015-04-10).
- [41] J. Zhao, J. M. Gomez-Perez, K. Belhajjame, G. Klyne, E. Garcia-Cuesta, A. Garrido, K. Hettne, M. Roos, D. De Roure, C. Goble, Why workflows break? understanding and combating decay in taverna workflows 0 (2012) 1–9. doi:<http://doi.ieeecomputersociety.org/10.1109/eScience.2012.6404482>.
- [42] D. D. Roure, C. Goble, R. Stevens, Designing the myexperiment virtual research environment for the social sharing of workflows, in: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, 2007, pp. 603–610. doi:10.1109/E-SCIENCE.2007.29.
- [43] C. Collberg, T. Proebsting, A. M. Warren, Repeatability and Benefaction in Computer Systems Research, A Study and a Modest Proposal.
URL <http://reproducibility.cs.arizona.edu/v2/RepeatabilityTR.pdf>
- [44] T. Crick, B. A. Hall, S. Ishtiaq, K. Takeda, "share and enjoy": Publishing useful and usable scientific models, *CoRR abs/1409.0367*.
URL <http://arxiv.org/abs/1409.0367>
- [45] Executable paper grand challenge (2011).
URL <http://www.executablepapers.com/>
- [46] P. Bonnet, S. Manegold, M. Björning, W. Cao, J. Gonzalez, J. Granados, N. Hall, S. Idreos, M. Ivanova, R. Johnson, et al., Repeatability and workability evaluation of sigmod 2011, *ACM SIGMOD Record* 40 (2) (2011) 45–48. doi:10.1145/2034863.2034873.
- [47] G. R. Brammer, R. W. Crosby, S. J. Matthews, T. L. Williams, Paper m^âch^é: Creating dynamic reproducible science, *Procedia Computer Science* 4 (0) (2011) 658–667. doi:10.1016/j.procs.2011.04.069.
- [48] P. V. Gorp, S. Mazanek, Share: a web portal for creating and sharing executable research papers, *Procedia Computer Science* 4 (0) (2011) 589 – 597, proceedings of the International Conference on Computational Science, {ICCS} 2011. doi:10.1016/j.procs.2011.04.062.
- [49] B. Howe, Virtual appliances, cloud computing, and reproducible research, *Computing in Science Engineering* 14 (4) (2012) 36–41. doi:10.1109/MCSE.2012.62.
- [50] F. Chirigati, D. Shasha, J. Freire, Reprozip: Using provenance to support computational reproducibility.
- [51] J. Goecks, A. Nekrutenko, J. Taylor, T. G. Team, Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences, *Genome Biol* 11 (8) (2010) R86.
- [52] B. Matthews, A. Shaon, J. Bicarregui, C. Jones, J. Woodcock, E. Conway, Towards a methodology for software preservation, *California Digital Library*.
- [53] M. Caballer, I. Blanquer, G. Moltó, C. de Alfonso, Dynamic management of virtual infrastructures, *Journal of Grid Computing* 13 (1) (2015) 53–70. doi:10.1007/s10723-014-9296-5.
URL <http://dx.doi.org/10.1007/s10723-014-9296-5>
- [54] D. Milojevic, I. Llorente, R. S. Montero, Opennebula: A cloud management tool, *Internet Computing, IEEE* 15 (2) (2011) 11–14. doi:10.1109/MIC.2011.44.
- [55] R. Mayer, T. Miksa, A. Rauber, Ontologies for describing the context of scientific experiment processes, in: 10th International Conference on e-Science, 2014.
- [56] L. Youseff, M. Butrico, D. Da Silva, Toward a unified ontology of cloud computing, in: Grid Computing Environments Workshop, 2008. GCE '08, 2008, pp. 1–10. doi:10.1109/GCE.2008.4738443.
- [57] C. Hoefer, G. Karagiannis, Taxonomy of cloud computing services, in: GLOBECOM Workshops (GC Wkshps), 2010 IEEE, 2010, pp. 1345–1350. doi:10.1109/GLOCOMW.2010.5700157.
- [58] M. Zhang, R. Ranjan, A. Haller, D. Georgakopoulos, M. Menzel, S. Nepal, An ontology-based system for cloud infrastructure services' discovery, in: Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on, IEEE, 2012, pp. 524–530.