Reproducibility of Execution Environments in Computational Science Using Semantics and Clouds

Idafen Santana-Perez[a,*], Rafael Ferreira da Silva[b], Mats Rynge[b], Ewa Deelman[b], María S. Pérez-Hernández[a], Oscar Corcho[a]

[a] *Ontology Engineering Group, Universidad Politécnica de Madrid, Madrid, Spain*
[b] *University of Southern California, Information Sciences Institute, Marina del Rey, CA, USA*

Abstract

One of the most common forms of addressing reproducibility in scientific workflow-based computational science in the past decades has consisted in 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 for describing the execution environment of scientific workflows, so as to conserve them, using semantic vocabularies. 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 on three distinguished 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 in science. 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 available to be distributed, 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, 8], and disease-related research [9, 10], 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.

Many efforts have been reported on studying the reproducibility of scientific results in life sciences. Some studies clearly show the difficulties when trying to replicate experimental results in biology [11]. The *Reproducibility Project: Cancer Biology* [12] 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 the whole of scientific 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 pursuit similar goals, there are several differences between them [13]. 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 the incremental research of scientific studies by modifying, improving and repurposing them.

*Corresponding address: Ontology Engineering Group (OEG), Laboratorio de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, Avda. Montepríncipe, 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 Rynge), 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 one, 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 [14], 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 [15], authors explain the problems they faced when they tried to reproduce an experiment [16] 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 advises, 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 [17]. 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 with the community research through a common repository, however the high storage demand of VM images remains a challenging problem [18, 19].

Inspired by the aforementioned ideas, exposed in [14], we aim to define means for authors to share the relevant information about the execution environment of a given scientific workflow. We argue that by explicitly describing this knowledge we increase the reproducibility degree of the environment and of the workflow therefore.

Semantics have been proposed as a way for attaining curation and conservation of the digital assets related to scientific experiments (e.g., biomedical research [20]). 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, using real scientific workflow applications, in which we describe the applications 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.

The semantics modeling of computational resources and some of the reproducibility tools were introduced and evaluated in [21] for a single astronomic scientific workflow application. In this work, we extend our previous work by introducing 1) a set of new features to our framework (as a result of our previous work), 2) a study of two new life sciences workflows based on genomic processing applications, and 3) a practical evaluation of the framework with the new features for the astronomic workflow as well as the two 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 for achieving the 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.

We propose the definition of 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.

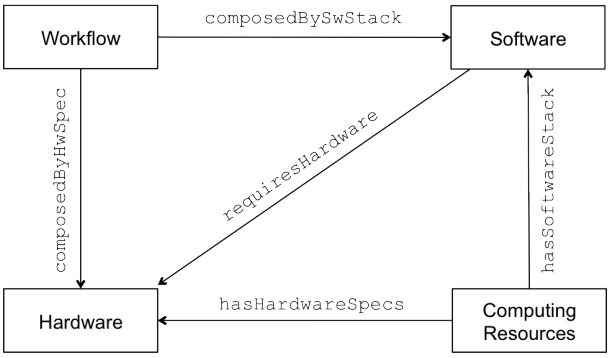

Figure 1: Overview of the ontology network (→ denotes inter-domain relation).

We have identified four main domains of interest for documenting computational scientific infrastructures [22]. We have 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*: it identifies the most common hardware information, including CPU, Storage and RAM memory, and their capacities.
- *Software domain*: it 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 required steps for deploying them.
- *Workflow domain*: it 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*: it 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.

3. Reproducibility in Scientific Workflows

In this section, we introduce the tools used in this work for the instantiation and evaluation of the aforementioned semantic models. We first describe the Pegasus Workflow Management System (WMS) [23, 24], which is used as our 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). A workflow execution includes data management, monitoring, and failure handling. Individual workflow tasks are managed by a task scheduler (HTCondor [25]), 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 [22], which comprises the semantic models described in Section 2 and a set of tools for annotating and consuming data; and the PRECIP [26] experiment management tool to manage the experiment. In addition, we use Vagrant [27], 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

The Workflow Infrastructure Conservation Using Semantics ontology (WICUS) is an OWL2 (Web Ontology Language) ontology network that implements the semantic models introduced in Section 2. This ontology network is available online [28] and its goal is to define the relevant and required properties for describing scientific computational infrastructures. The detailed description of the ontologies, including its mains terms and relation in the context of a workflow execution are provided in [22]. 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 for also including the output information of a configuration process (e.g., the resultant IP and port in which a recently deployed service will be listening).

Besides the ontology network, a set of components have been developed around it, for facilitating the annotation of the resources involved on the execution of a scientific workflow. These tools are not fully automated yet, but represent a first step on helping users to define the requirements of their experiments. Figure 2 shows the main modules, their flow and intermediate results involved in the process for achieving reproducibility, and 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 vocabulary, representing workflow transformations and the workflow infrastructure requirements.

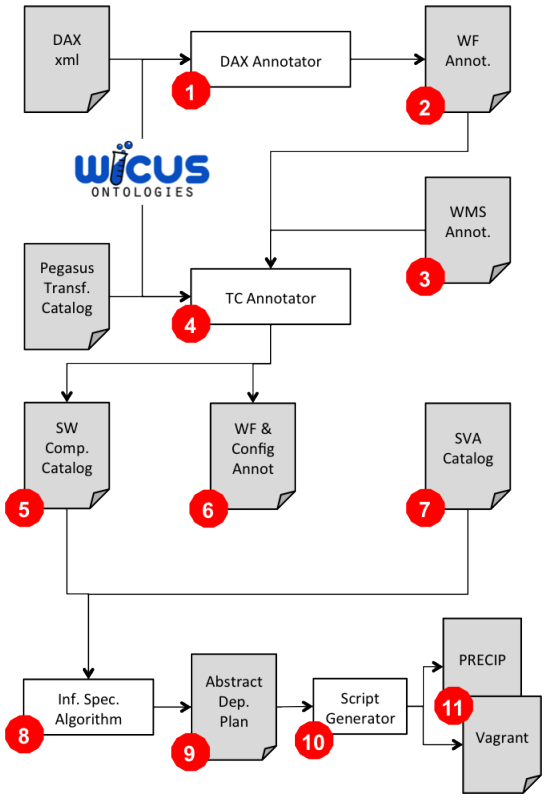

Figure 2: WICUS annotation modules and flow. White boxes represent tools for semantic annotation or algorithms, and grey boxes represent files used or generated by the framework.

2. **Workflow Annotations.** This RDF file contains the description of the workflow and its infrastructure requirements.
3. **WMS Annotations.** This RDF file contains the information of the WMS component and its dependencies. This information is added to the *Software Components Catalog*.
4. **Transformation Catalog Annotator.** This tool parses the Pegasus Transformation Catalog (which describes the binaries involved on the workflow execution and their locations) and the WMS annotations file, to generate two set of annotations: the *Software Components Catalog* and the *Workflow & Configuration Annotation* files.
5. **Software Components Catalog.** This RDF file contains the set of annotations about the binaries, dependencies, deployment plans and scripts, and configuration information of the software involved in the experiment.
6. **Workflow & Configuration Annotation File.** This RDF file contains the same information as in 2, but enriched with the configuration information for each workflow execution step, as specified in the transformation catalog.
7. **Scientific Virtual Appliances Catalog.** This RDF file contains available VM appliances. Information about the related infrastructure providers and the VM images that compose an appliance are included in this dataset.
8. **Infrastructure Specification Algorithm.** This process reads files 5, 6, and 7, and generates a configuration file, which describes VMs and software components to be created and deployed.
9. **Abstract Deployment Plan.** This plan contains information about the set of components and their associated deployment steps for configuring the execution infrastructure. This plan will be later enacted using a concrete language. This decoupled approach allows the integrations of new enactment systems easily.
10. **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.
11. **Executable Script.** This script creates a PRECIP/Vagrant experiment, which runs a VM, copies the required binaries, and executes deployment scripts to set the environment for the workflow execution. It also contains the original experiment commands in order to re-execute it.

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

3.2.2. PRECIP

The Pegasus Repeatable Experiments for the Cloud in Python (PRECIP) [26] is a flexible experiment management control API for running experiments on all types of Clouds, including academic Clouds such as FutureGrid [29] and the NSFCloud [30, 31] (through OpenStack), and commercial Clouds such as Amazon EC2 [32] and Google Compute Engine [33]. In PRECIP, interactions with the provisioned instances are done by tagging. 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. 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.

3.2.3. Vagrant

Vagrant [27] is an open-source and multi-platform solution for deploying development environments locally using virtualization. It relies on virtualization solutions such as Oracle VirtualBox [34] (also open-source) or VMWare [35], and support Amazon EC2-like server configurations. Since version 1.6 it also supports Docker [36] 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 customize [37].

In this work, we introduce how Vagrant can be used for achieving reproducibility at 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 three real scientific workflow applications. This process is an extension of the one introduced in [21], in which we evaluate the improvements on our approach to the Montage [38] workflow, and also evaluate the Epigenomics [39], and SoyKB [40, 41] workflows. We study and document these workflows and their execution environments, which include the application software components and the workflow management system.

The goal of this experiment is to reproduce original workflow executions into the three different Cloud scenarios introduced in this work: FutureGrid [29] and Amazon EC2 [32] using PRECIP, and a local execution environment by using Vagrant. FutureGrid is an academic Cloud test-bed 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.

4.1. Scientific Workflows

Montage. The Montage workflow [38] 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 3 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 [39] is currently involved in mapping the epigenetic state of human cells on a genome-wide scale. The Epigenomics workflow (Figure 4) 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

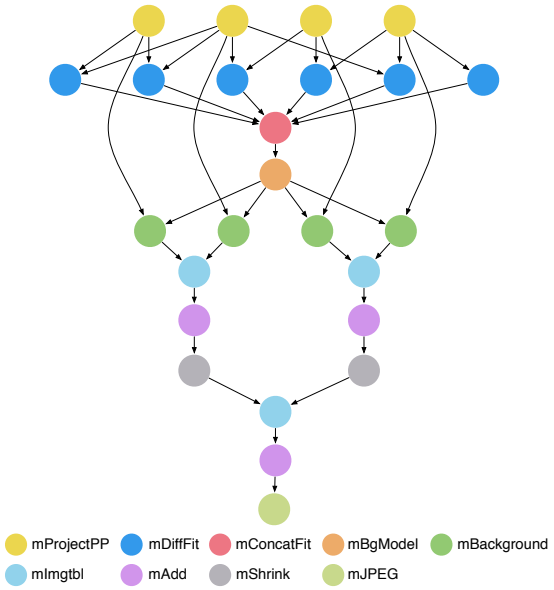

Figure 3: A small (20 node) Montage workflow.

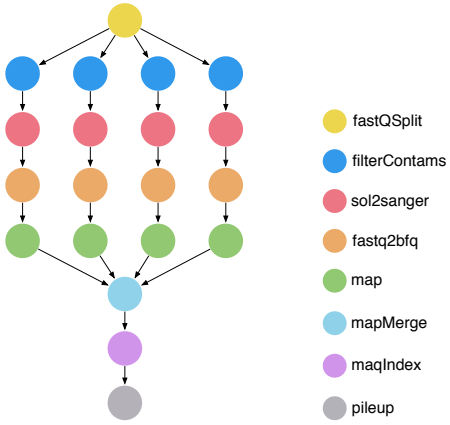

Figure 4: Epigenomics workflow.

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 [40, 41] 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 5) implements a Single Nucleotide Polymorphism (SNP) and injection/deletion (indel) identification and analysis pipeline using the GATK haplotype caller [42] 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

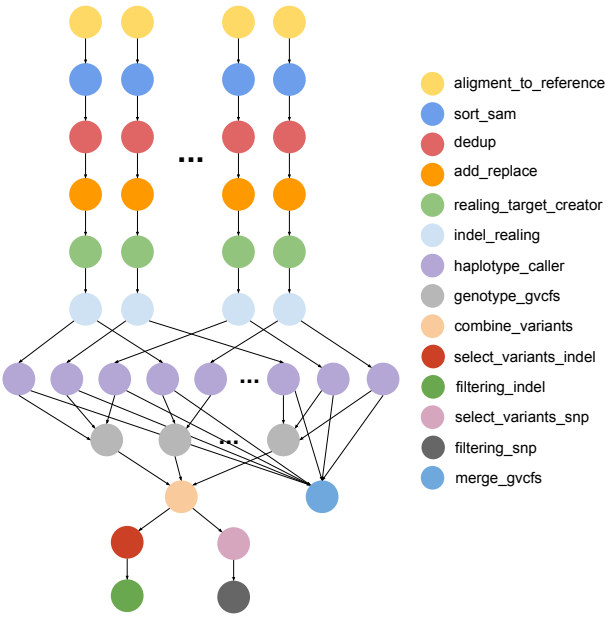

Figure 5: SoyKB workflow.

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 on the Research Object (RO) [43] associated to this paper [44].

As described in Figure 2, 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 and one for each sub-workflow, which defines the software components required by each transformation. Figure 6 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 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 are empty, as they are not yet related to their software components. Figure 7 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).

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 (Figure 8). We have then annotated 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 Perl [45] interpreter, the GNU libc [46] and Libstdc++ [47] 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 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

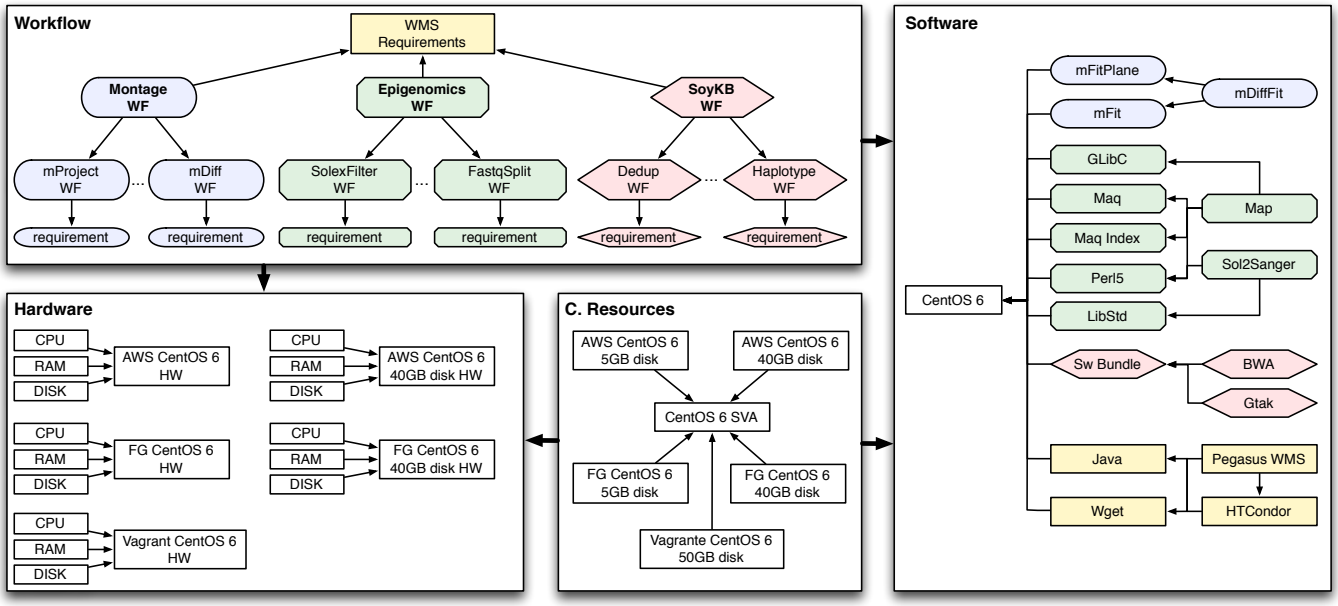

Figure 6: Overview of the generated annotations for the Montage, Epigenomics, and SoyKB workflows using the WICUS ontology network (yellow rectangles represent the workflow component; blue squashed rectangles represent the Montage workflow; green bevelled rectangles represent the Epigenomics workflows; and red hexagons represent the SoyKB workflow).

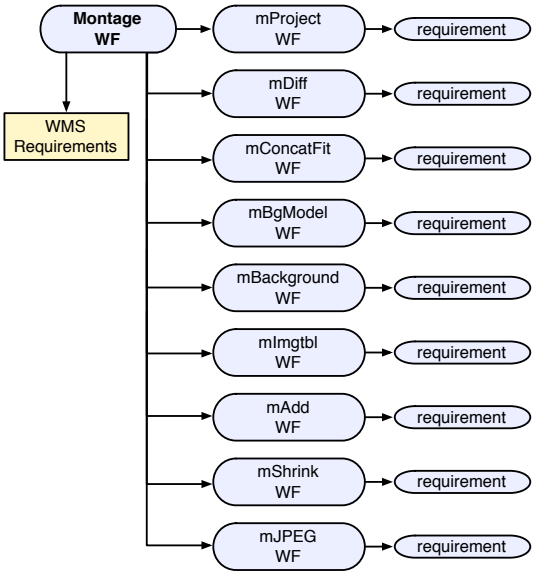

Figure 7: Annotations for the Montage workflows using the `Workflow` domain ontology.

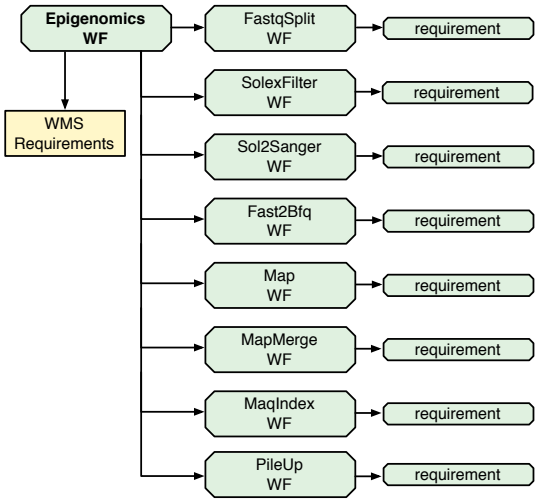

Figure 8: Annotations for the Epigenomics workflows using the `Workflow` domain ontology.

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 on a software bundle that is deployed on the computational nodes to be invoked by the wrappers. 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 one for the SoyKB workflow (requires larger disk space). In both cases, we generated conceptually equivalent VMs 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 for 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 (step 7 in Figure 2). We define an Image Appliance that groups all VMs appliances into one single Scientific Virtual Appliance (`CentOS 6 SVA`). As described in the Software domain ontology [22] a Scientific Virtual Appliance is defined by the set of software elements associated to it, while Image Appliances describe the combination of hardware characteristics and infrastructure providers. Thus, we will have five different Image Appliances grouped into one single Scientific Virtual Appliance, 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)	40	5	40	5	50
CPU (GHz)	2.4	2.4	2.9	2.9	2.83
CPU Arch.	64 bits		64 bits		64 bits
OS	CentOS 6		CentoOS 6		CentOS 6

Table 1: CentOS 6 Virtual 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 frequency.

	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.

The last step on the process for achieving reproducibility in scientific workflows (Figure 2) is to execute the Infrastructure Specification Algorithm (ISA) described in [22]. 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.

Infrastructure Specification 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 6). Requirements and software components are matched, and a dependency graph is built based on the relation between the requirements and the component dependencies. This graph is then used to compute the intersection between the set of software components from the Scientific Virtual Appliance (SVA) and the dependency graph of each sub-workflow. ISA selects the intersection where the value is maximized for each sub-workflow. Software components already available in the SVA are then removed from the chosen graph. To reduce the number of SVAs, the algorithm attempts to merge sub-workflow requirements into a single SVA. Requirements can be merged if all their software components are compatible. Finally, ISA generates a script (either using PRECIP or Vagrant) with the set of required instructions to instantiate, configure, and deploy the computational resources and software components on the corresponding provider. Listing 1 shows the pseudo-code of the algorithm.

Listing 1: 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 );
```

In this work, we have extended ISA to support 1) Image Appliance filtering, and 2) generation of Vagrant scripts. The algorithm filters Image Appliances from the selected SVAs that do not meet the hardware requirements specified for the workflow (lines 23–25). To enact support to different execution scripts, we created an intermediate phase (*Abstract Deployment Plan*, step 9 in Figure 2), which defines the steps and scripts to be executed, along with their configuration parameters. ISA then translates this plan into a PRECIP or Vagrant script depending on the resultant target provider (line 27).

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 [22] domain ontology, which defines the software stacks that should be deployed in the execution platform. Figure 9 shows the relations between the different elements that compose the `Stack` domain ontology. A `Software Stack` may be composed by one or more `Software Components`. Each of them has an associated `Deployment Plan` according to the target execution platform, which is composed by one or more `Deployment Steps`.

Listing 2 shows an example of the abstract plan for the SoyKB workflow generated by the ISA. The first section of the plan (lines 1–26) describes the deployment of the `Pegasus WMS` 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 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.

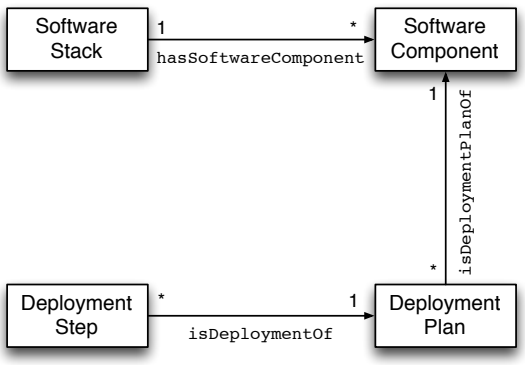

Figure 9: Overview of the WICUS software stack relation diagram.

Listing 2: 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
```

Execution Script. In all cases, ISA is 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.

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 in 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 tool [48] 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.

All the original and generated scripts are available as part of the experimental material included in the RO associated with this work [44]. This RO also contains pointers to the software and resources used in the experimental evaluation.

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 [49, 50]. Most of the works address 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 [51]. As a result, there is an increase on the number of publications providing associated experimental materials [52, 53].

A study to evaluate reproducibility in scientific workflows is conducted in [54]. The study evaluates a set of domain-specific workflows, available in myExperiment [55], to identify causes of workflow decay. The study shows 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. Note that some of those third-party resource issues could be also considered as execution environment problems, as many of them are remote services for information processing.

Recently, another comprehensive study has been published [56], surveying 601 papers from ACM conferences and 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 show 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 propose, 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, 8], disease-related experiments [9, 10] and many others. Several studies have exposed the difficulties of trying to reproduce experimental results on life sciences, such as biology [11]. and cancer analysis [12].

Replicability and reproducibility of computational experiments using cloud computing resources and software descriptions have been widely proposed as an approach for those studies in which performance is not a key experimental result [57].

The Executable Paper Grand Challenge [58] and the SIGMOD conference in 2011 [59] 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 [60, 61]. However, the high storage demand of VM images remains a challenging problem [18, 19]. Moreover, the cost of storing and managing data in the Cloud is still high, and 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 is exposed in [62]. ReproZip [63] is a provenance-based tool that tracks 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. Instead, our work uses semantic annotations to describe these dependencies.

Galaxy [64], a well know workflow system in the context of life sciences, proposes a web-service based system for achieving accessible and reproducible computations. Galaxy hides the implementation details of the underlying tools for workflow developers, providing a web-based interface for retrieving an analyzing genomic data. Even when this approach has proved to be successful in several cases, we argue that it does not cover local development of workflows, which is a common case on computational science.

Software components cannot be preserved just by maintaining their binary executable code, but by guaranteeing the performance of their features. In [65], the concept of adequacy is introduced to measure how a software component behaves relatively 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.

A recent and relevant contribution to the state of the art of workflow preservation has been developed within the context of the TIMBUS project [66]. 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 managements. 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 been also proposed in the area of biomedical research as a way for achieving reproducibility of published experiments [20]. In this work authors propose to annotate the software artifacts in the same way that gene products or phenotypes are annotated. In order to do so, authors propose the *Software Ontology* (SWO), a model for describing the software involved on the storage and management of data. The SWO have many concepts in common with the WICUS ontology network, but is specialized on the biomedical domain, focusing on modeling biomedical-related software specifically. WICUS aims to be a more generic ontology that can be applied in different scientific domains.

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 4 domain ontologies, aggregated in the the WICUS ontology network. From these models, we defined a process for documenting workflow applications, the workflow management system where they can be executed, and their dependencies.

We conducted and experimental process in which we studied three workflow applications from different areas of science (Montage, Epigenomics and SoyKB) using the Pegasus WMS. We executed the 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. The drawback of our approach is that it assumes the application and the workflow management system binaries are publicly available.

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 kind of library would be helpful. We plan to study how and when those libraries can be built, analyzing their degree of reuse.

We also aim to study other workflows, belonging to different scientific areas, as well as applying our approach to new workflow management systems. We will also work on increasing the degree of automation of the semantic annotation process to describe both the workflow application and the workflow management system. As introduced in this work, WICUS is a 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). 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. Fahey, 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. Rynge, 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. Elnitski, 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] Kepler/clotho integration, http://sourceforge.net/projects/keplerclotho, (accessed 2015-04-10).

[9] 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.

[10] 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.

[11] 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

[12] T. M. Errington, F. E. Tan, J. Lomax, N. Perfito, E. Iorns, W. Gunn, K. D. Evans, Reproducibility Project: Cancer Biology, osf.io/e81xl (2015).

[13] 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.

[14] G. King, Replication, replication, PS: Political Science and Politics 28 (3) (1995) 443–499.

[15] 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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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

[21] I. Santana-Perez, R. Ferreira da Silva, M. Rynge, 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.

[22] I. Santana-Perez, M. S. Pérez-Hernández, Towards reproducibility in scientific workflows: An infrastructure-based approach, Scientific Programming 2015. doi:10.1155/2015/243180.

[23] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, et al., Pegasus: A framework for mapping complex scientific workflows onto distributed systems, Scientific Programming 13 (3).

[24] E. Deelman, K. Vahi, G. Juve, M. Rynge, 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.

[25] 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.

[26] S. Azarnoosh, M. Rynge, 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.

[27] J. Palat, Introducing vagrant, Linux Journal 2012 (220) (2012) 2.

[28] The workflow infrastructure conservation using semantics ontology, http://purl.org/net/wicus, (accessed 2015-04-10).

[29] Futuregrid, (accessed 2015-04-10).
URL http://portal.futuregrid.org

[30] Chameleon cloud, http://www.chameleoncloud.org, (accessed 2015-04-10).

[31] Cloudlab, http://cloudlab.us, (accessed 2015-04-10).

[32] Amazon Elastic Compute Cloud: Amazon EC2, http://aws.amazon.com/ec2, (accessed 2015-04-10).

[33] Google Compute Engine (GCE), https://cloud.google.com/compute, (accessed 2015-04-10).

[34] J. Watson, Virtualbox: Bits and bytes masquerading as machines, Linux J. 2008 (166).
URL http://dl.acm.org/citation.cfm?id=1344209.1344210

[35] VMware, http://www.vmware.com, (accessed 2015-04-10).

[36] 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

[37] Vagrantbox, http://www.vagrantbox.es, (accessed 2015-04-10).

[38] 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.

[39] USC epigenome center, http://epigenome.usc.edu/, (accessed 2015-04-10).

[40] T. Joshi, B. Valliyodan, S. M. Khan, Y. Liu, J. V. Maldonado dos Santos, Y. Jiao, D. Xu, H. T. Nguyen, N. Hopkins, M. Rynge, N. Merchant, Next generation resequencing of soybean germplasm for trait discovery on xsede using pegasus workflows and iplant infrastructure, in: XSEDE 2014, 2014, poster.

[41] 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.

[42] GATK, https://www.broadinstitute.org/gatk, (accessed 2015-04-10).

[43] 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.

[44] http://purl.org/net/FGCS-Wicus-Pegasus-Repro, (accessed 2015-04-10).
[45] Perl, https://www.perl.org, (accessed 2015-04-10).
[46] The GNU C library, http://www.gnu.org/software/libc, (accessed 2015-04-10).
[47] Standard C++ library, http://gcc.gnu.org/libstdc++, (accessed 2015-04-10).
[48] pHash, http://www.phash.org, (accessed 2015-04-10).
[49] 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
[50] S. Arabas, M. R. Bareford, I. P. Gent, B. M. Gorman, M. Hajiarabderkani, T. Henderson, L. Hutton, A. Konovalov, L. Kotthoff, C. McCreesh, R. R. Paul, K. E. Petrie, A. Razaq, D. Reijsbergen, 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
[51] 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
[52] 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-printsarXiv:1203.4802.
[53] 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).
[54] 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.
[55] 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.
[56] 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
[57] 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
[58] Executable paper grand challenge (2011).
URL http://www.executablepapers.com/
[59] P. Bonnet, S. Manegold, M. Bjørling, 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.
[60] G. R. Brammer, R. W. Crosby, S. J. Matthews, T. L. Williams, Paper mâché: Creating dynamic reproducible science, Procedia Computer Science 4 (0) (2011) 658–667. doi:10.1016/j.procs.2011.04.069.
[61] 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.
[62] B. Howe, Virtual appliances, cloud computing, and reproducible research, Computing in Science Engineering 14 (4) (2012) 36–41. doi:10.1109/MCSE.2012.62.
[63] F. Chirigati, D. Shasha, J. Freire, Reprozip: Using provenance to support computational reproducibility.
[64] 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.
[65] B. Matthews, A. Shaon, J. Bicarregui, C. Jones, J. Woodcock, E. Conway, Towards a methodology for software preservation, California Digital Library.
[66] R. Mayer, T. Miksa, A. Rauber, Ontologies for describing the context of scientific experiment processes, in: 10th International Conference on e-Science, 2014.