A Semantic-Based Approach to Attain Reproducibility of Computational Environments in Scientific Workflows: A Case Study

Idafen Santana-Perez[1], Rafael Ferreira da Silva[2], Mats Rynge[2], Ewa Deelman[2], María S. Pérez-Hernández[1] and Oscar Corcho[1]

[1] Ontology Engineering Group, Universidad Politécnica de Madrid, Madrid, Spain
{isantana,mperez,ocorcho}@fi.upm.es
[2] USC Information Sciences Institute, Marina Del Rey, CA, USA
{rafsilva,rynge,deelman}@isi.edu

Abstract. Reproducible research in scientific workflows is often addressed by tracking the provenance of the produced results. While this approach allows inspecting intermediate and final results, improves understanding, and permits replaying a workflow execution, it does not ensure that the computational environment is available for subsequent executions to reproduce the experiment. In this work, we propose describing the resources involved in the execution of an experiment using a set of semantic vocabularies, so as to conserve the computational environment. We define a process for documenting the workflow application, management system, and their dependencies based on 4 domain ontologies. We then conduct an experimental evaluation using a real workflow application on an academic and a public Cloud platform. Results show that our approach can reproduce an equivalent execution environment of a predefined virtual machine image on both computing platforms.

1 Introduction

Reproducibility of results of published 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 [18]. 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 [21].

In the context of scientific experiments, terms such as reproducibility, replicability and repeatability are sometimes used as synonymous. Even though there is no a clear consensus on how to define both (definitions may vary over different scientific areas), in this work we understand them as different concepts [9]. In this work we address the reproducibility of the execution environment for a scientific workflow, as we do not aim to obtain necessarily an exact incarnation of the original one, but rather an environment that is able to support the required capabilities exposed by the former environment.

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 [23]. This representation not only facilitates the creation and management of the computation but also builds a foundation upon which results can be validated and shared. 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, which 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.

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: *physical conservation*, where the real object is conserved due to its relevance and the difficulty in obtaining a counterpart; and *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 [12]. Furthermore, the infrastructure may be subjected to organization policies, which restricts its access to a selective group of scientists, thus limiting reproducibility to this restricted group. On the other hand, data, code, and workflow description can be conserved by using a logical approach that 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 [15,26].

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 evaluation for a real scientific workflow application in which we describe the application and its environment using a set of semantic models, and use an experiment management tool to reproduce a workflow execution in two different Cloud platforms.

In this work we entail reproducibility from the execution environment point of view. For the sake of showing how our approach works, we provide an example in which we have also include the data and the workflow execution of the experiment.

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

2 Semantic Modeling of Computational Resources

Scientific workflows are also used for preserving and sharing scientific experiments in science. Research efforts focused on describing the workflow structure and the experimental data, both input data and results. In this work, we argue that the information about the computational resources should be also provided for achieving full reproducibility. These descriptions allow the target audience, usually another scientist in the same domain, to 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 the software components, hardware specifications, and the available computational resources (in the form of VMs). They also capture infrastructure dependencies of the workflows. 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. We have developed a set of models, one for each domain, and an ontology network that defines the inter-domain relations between these models (Fig. 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 required steps for deploying them.

– *Workflow domain*: describes and relates workflow fragments (a.k.a transformations) to their dependencies. Therefore, scientists can understand what 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. VMs). It includes the description of the VM image, its provider, and specifications.

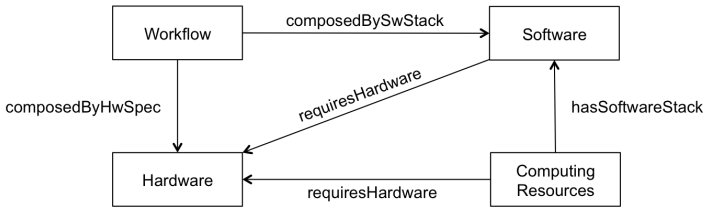

Fig. 1. Overview of the ontology network (→ denotes inter-domain relation).

3 Reproducibility in Scientific Workflows

In this section, we conduct a practical evaluation through experimentation in which we instantiate the semantic models aforementioned for a real scientific workflow application. We study and document the Montage [4] workflow and its execution environment, which includes the application software components and the workflow management system. Montage is and astronomy workflow application that is widely used by many astronomers to construct large image mosaics of the sky.

3.1 Scientific Workflow Execution

Scientific workflows allow users to easily express multi-step computational tasks, for example retrieve data from an instrument or a database, reformat the data, and run an analysis. Scientific workflows are described as high-level abstraction languages which conceal the complexity of execution infrastructures to the user. In most cases workflows are described as directed acyclic graphs (DAGs), where the nodes represent individual computational tasks and the edges represent data and control dependencies between tasks. Workflow interpretation and execution are handled by a workflow management system (WMS) that manages the execution of the application on the distributed computing infrastructure.

In this work, we use the Pegasus WMS [8] as our workflow engine. The Pegasus WMS can manage workflows comprised of millions of tasks, recording data about the execution and intermediate results. In Pegasus, workflows are described an abstract workflows, which do not contain resource information, or

the physical locations of data and executables. The abstract workflow description is represented as a DAX (DAG in XML), capturing all the tasks that perform computation, 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, the Pegasus WMS 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 [24]), which supervises their execution on local and remote resources.

3.2 Reproducibility Tools

To conduct the experimental evaluation, we use the WICUS [20] framework, which comprises the semantic models described in Section 2 and a set of tools for annotating and consuming data, and the PRECIP [2] experiment management tool to manage the experiment. Below, we describe each of these tools in detail.

WICUS. The Workflow Infrastructure Conservation Using Semantics ontology (WICUS) [20] is an OWL2 [17] (Web Ontology Language) ontology network that implements the semantic models introduced in Section 2. This ontology network is available online[1] and it is a continuous effort to discover and define the relevant and required properties for describing scientific computational infrastructures.

Besides the ontology network, WICUS has a set of modules that facilitates 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. Fig. 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.

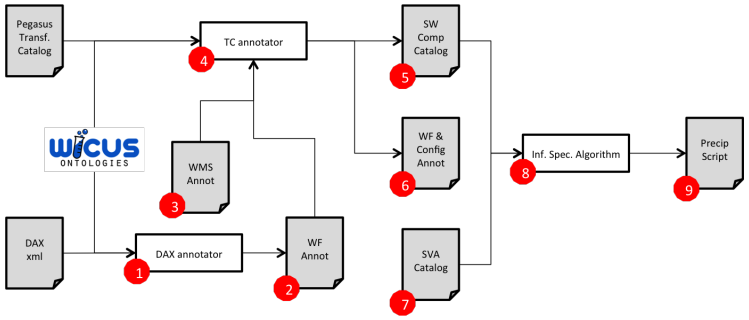

Fig. 2. WICUS annotation modules and flow.

[1] `http://purl.org/net/wicus`

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.

2. Workflow annotations. An RDF file containing the description of the workflow and its infrastructure requirements.

3. WMS annotations. An RDF file containing the information of the WMS component and its dependencies. This information will be 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. An RDF file containing 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. An RDF file containing 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. An RDF file describing 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 (e.g. a PRECIP script), which describes VMs and software components to be created and deployed.

9. PRECIP script. This script creates a PRECIP 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 PRECIP commands from the original experiment in order to re-execute it.

PRECIP. The Pegasus Repeatable Experiments for the Cloud in Python (PRECIP) [2] is a flexible experiment management control API for running experiments on all types of Clouds, including academic Clouds such as FutureGrid [11], and commercial Clouds such as Amazon EC2 [1]. 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, PRECIP usage is twofold. First, the tool is used to describe and perform a workflow execution using the Pegasus WMS on a predefined VM image. Second, the WICUS annotation modules use PRECIP to generate a script able to reproduce the execution environment of the former experiment, and run it on different Cloud platforms.

3.3 Experimental Evaluation

The goal of this experiment is to reproduce an original workflow execution in two different Cloud infrastructures: FutureGrid [11] and Amazon EC2 [1]. 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.

Generating Semantic Annotations. Fig. 3 shows a simplified overview of the annotations generated for the Montage workflow using the WICUS ontology network.

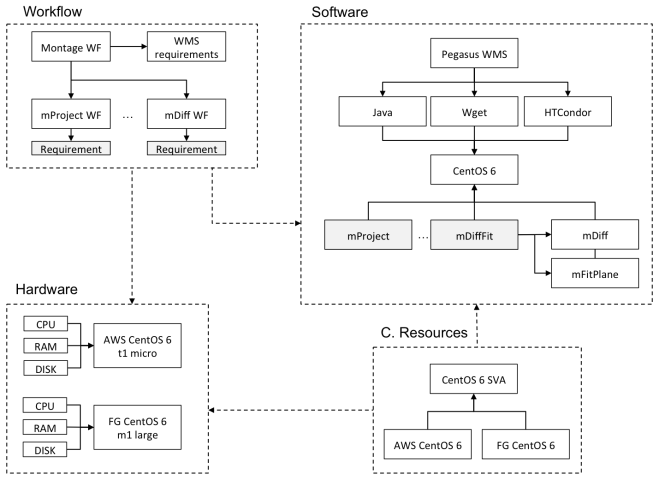

Fig. 3. Annotations for the Montage workflow using the WICUS ontology network.

As shown in Fig. 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 the Montage workflow as 1) an individual that represents the top level workflow, and 2) another 9 individuals representing its sub-workflows, one for each transformation. We also generate 10 requirements, one for the top level

workflow, which specifies the WMS requirements, and the remaining for defining the software components required by each transformation. At this point, these requirements are empty, as they are not yet related to their software components.

In this experiment, we address two types of components: the WMS and the application related components. The WMS components include the workflow engine, in our case the Pegasus WMS, and its dependencies. Pegasus uses HTCondor as task manager, and also depends on Java. We use the `Software` domain ontology to describe these components as individuals, and to represent their dependencies. The 3 components also depend on the operating system, which in our case is CentOS.

To describe the deployment of the WMS components, we studied their installation processes according to their documentation. We then defined a set of installation bash scripts for each of them. These scripts are included on the deployment plans of the components along with their configuration information.

Application components are described from 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).

To describe computational resources we use the `Computing Resources` and `Hardware` domain ontologies. The Scientific Virtual Appliances Catalog includes the description of two virtual machine images, one for FutureGrid and another for Amazon EC2. These two images are conceptually equivalent, as they both provide CentOS 6 operating system. Therefore, we generate two Image Appliances (FG CentOS 6 and AWS CentOS 6) that are grouped into one single Scientific Virtual Appliance (CentOS 6 SVA). Depending on which providers are available, one or the other will be selected.

Reproducing Workflow Executions. The last step on the process for achieving reproducibility in scientific workflows (Fig. 2) is to execute the Infrastructure Specification Algorithm (ISA). The ISA combines the annotated data based on the 4 domain ontologies in order to find a suitable infrastructure specification that is able to run the workflow. The algorithm retrieves and propagates the WMS requirements of the top-level workflow (`Workflow` domain ontology) to its related sub-workflows. Requirements and software components are matched, and a dependency graph is built based on the relation between the require-

ments and the component dependencies. This graph is then used to compute the intersection between the set of software components from the 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-workflows requirements into a single SVA. Requirements can be merged if all their software components are compatible. Finally, ISA generates a PRECIP script with the set of required instructions to instantiate, configure, and deploy the computational resources and software components.

In this experiment, we execute ISA over the annotated data in a scenario where FutureGrid is the only available platform for resource provisioning, and in a scenario where the available platform is Amazon EC2. In both cases, the algorithm is able to obtain a PRECIP script for each infrastructure. Each generated script is composed by 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 SSH keys to enable file transfers and execution. This account will be used to run the workflow;
- *Data Stage and Workflow Execution*: stages all the input data of the Montage workflow 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.

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 scripts on their corresponding platforms. Both executions succeeded on deploying and running the Montage workflow, the Pegasus WMS, and their dependencies. We also performed the same execution of the Montage workflow in a predefined VM image, where the execution environment is already in place. Results show that the VM execution environments deployed by both scripts are equivalent to the execution environment of the predefined VM image. In addition, we used a perceptual hash tool[2] to compare the resulting image (0.1 degree image of the sky) generated by both executions against the one generated by the baseline execution. We obtained a similarity factor of 1.0 (over 1.0) with a threshold of 0.85, which means the images are identical.

All the original and generated scripts are available as part of the experimental material included in the Research Object (RO) [3] associated with this

[2] pHash - `http://www.phash.org`

paper[3]. This RO also contains pointers to the software and resources used in this experiment.

4 Related Work

A computational experiment involves several elements that must be conserved to ensure reproducibility. Most of the works addresses the conservation of data and the workflow description, however the computational environment is often neglected. An study to evaluate reproducibility in scientific workflows is conducted in [25]. The study evaluates a set of domain-specific workflows, available in the myExperiment [19] collaborative environment, to identify causes of workflow decays. The study shows that nearly 80% of the workflows cannot be reproduced, and that about 12% are due to the lack of information about the execution environment, and that 50% are due to the use of third-party resources such as web services and databases. Note that some of those third-party resource issues could be also considered as execution environment problems.

The Executable Paper Grand Challenge [10] and the SIGMOD conference in 2011 [5] highlighted the importance of allowing the scientific community to reexamine an experiment execution. The conservation of virtual machine (VM) images emerges as a way of preserving the execution environment [6,13]. However, the high storage demand of VM images remains a challenging problem [15,26]. 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 [14]. ReproZip [7] 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.

Software components cannot be preserved just by maintaining their binary executable code, but by guaranteeing the performance of their features. In [16], 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 is being developed within the context of the TIMBUS project [22]. The project aims 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 propose a semantic approach for

[3] `http://pegasus.isi.edu/publications/reppar`

describing the execution environment of a process. Even though TIMBUS has studied the applicability of their approach to the eScience domain, their approach is mainly focused on business processes.

5 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 a workflow application (Montage), a workflow management system (the Pegasus WMS), and their dependencies. We then used the PRECIP experiment management tool to describe and execute the experiment. Experimental results show that our approach can reproduce an equivalent execution environment of a predefined VM image on an academic and a public Cloud platforms.

The semantic annotations of the computational environment combined with the scripting functionality provided by PRECIP 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.

In the future we plan to apply our approach in a larger set of scientific workflows and involve users from different scientific areas, aiming to automate the generation process of the semantic annotations to describe both the workflow application and the workflow management system. We also plan to extend the WICUS 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", 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. Amazon Elastic Compute Cloud: Amazon EC2, `http://aws.amazon.com/ec2`
2. Azarnoosh, S., Rynge, M., et al.: Introducing precip: an api for managing repeatable experiments in the cloud. In: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science. CloudCom, vol. 2, pp. 19–26 (2013)
3. Belhajjame, K., Corcho, O., , et al.: Workflow-centric research objects: First class citizens in scholarly discourse. In: Proc. Workshop on the Semantic Publishing (SePublica). Proc. Workshop on the Semantic Publishing (SePublica), Crete, Greece (2012)

4. Berriman, G.B., Deelman, E., et al.: Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. In: SPIE Conference on Astronomical Telescopes and Instrumentation. vol. 5493, pp. 221–232 (2004)
5. Bonnet, P., Manegold, S., et al.: Repeatability and workability evaluation of sigmod. SIGMOD Rec. 40(2), 45–48 (2011)
6. Brammer, G.R., Crosby, R.W., et al.: Paper mache: Creating dynamic reproducible science. Procedia Computer Science 4(0), 658 – 667 (2011), proceedings of the International Conference on Computational Science
7. Chirigati, F., Shasha, D., Freire, J.: Reprozip: Using provenance to support computational reproducibility (2013)
8. Deelman, E., Singh, G., et al.: Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Scientific Programming 13(3) (2005)
9. Drummond, C.: Replicability is not reproducibility: Nor is it good science. In: Proceedings of the Evaluation Methods for Machine Learning Workshop at the 26th ICML (2009)
10. Executable paper grand challenge (2011), `http://www.executablepapers.com/`
11. Futuregrid, `http://portal.futuregrid.org`
12. Gavish, M., Donoho, D.: A universal identifier for computational results. Procedia Computer Science 4, 637 – 647 (2011), proceedings of the ICCS'11
13. Gorp, P.V., Mazanek, S.: Share: a web portal for creating and sharing executable research papers. Procedia Computer Science 4(0), 589 – 597 (2011), proceedings of the International Conference on Computational Science, {ICCS} 2011
14. Howe, B.: Virtual appliances, cloud computing, and reproducible research. Computing in Science Engineering 14(4), 36–41 (2012)
15. Mao, B., Jiang, H., et al.: Read-performance optimization for deduplication-based storage systems in the cloud. Trans. Storage 10(2) (2014)
16. Matthews, B., Shaon, A., et al.: Towards a methodology for software preservation (2009)
17. Owl 2 web ontology language, `http://www.w3.org/TR/owl2-overview/`
18. Reproducible research: Addressing the need for data and code sharing in computational science. `http://www.stanford.edu/~vcs/Conferences/RoundtableNov212009/RoundtableOutputDeclaration.pdf` (2009)
19. Roure, D.D., Goble, C., Stevens, R.: 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. pp. 603–610 (2007)
20. Santana-Pérez, I., Pérez-Hernández, M.S.: Towards reproducibility in scientific workflows: An infrastructure-based approach. IEEE Computing in Science & Engineering p. submitted (2014)
21. Stodden, V., Leisch, F., Peng, R.D. (eds.): Implementing Reproducible Research. Chapman & Hall (2014)
22. Strodl, S., Mayer, R., Antunes, G., Draws, D., Rauber, A.: Digital preservation of a process and its application to e-science experiments (2013)
23. Taylor, I., Deelman, E., et al.: Workflows for e-Science. Springer (2007)
24. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: The condor experience: Research articles. Concurr. Comput. : Pract. Exper. 17(2-4), 323–356 (2005)
25. Zhao, J., Gomez-Perez, J.M., et al.: Why workflows break - understanding and combating decay in taverna workflows. 2012 IEEE 8th International Conference on E-Science 0, 1–9 (2012)
26. Zhao, X., Zhang, Y., et al.: Liquid: A scalable deduplication file system for virtual machine images. IEEE Trans. on Paral. and Distr. Syst. PP(99) (2013)