Hindawi Publishing Corporation
Scientific Programming
Volume 2015, Article ID 243180, 11 pages
http://dx.doi.org/10.1155/2015/243180

Research Article

Towards Reproducibility in Scientific Workflows: An Infrastructure-Based Approach

Idafen Santana-Perez and María S. Pérez-Hernández

Ontology Engineering Group (OEG), Universidad Politécnica de Madrid, Avenida Montepríncipe, s/n, 28660 Boadilla del Monte, Spain

Correspondence should be addressed to Idafen Santana-Perez; isantana@fi.upm.es

Received 12 September 2014; Accepted 3 February 2015

Academic Editor: Gianluigi Greco

It is commonly agreed that in silico scientific experiments should be executable and repeatable processes. Most of the current approaches for computational experiment conservation and reproducibility have focused so far on two of the main components of the experiment, namely, data and method. In this paper, we propose a new approach that addresses the third cornerstone of experimental reproducibility: the equipment. This work focuses on the equipment of a computational experiment, that is, the set of software and hardware components that are involved in the execution of a scientific workflow. In order to demonstrate the feasibility of our proposal, we describe a use case scenario on the Text Analytics domain and the application of our approach to it. From the original workflow, we document its execution environment, by means of a set of semantic models and a catalogue of resources, and generate an equivalent infrastructure for reexecuting it.

1. Introduction

Reproducibility is a goal that every scientist developing a research work should take into account during the experimental and publication processes. Any scientific publication is meant to both announce a result of interest and convince readers that the exposed claims are true [1]. Therefore, the scientific community is encouraging authors and editors to publish their contributions in a verifiable and understandable way. However, reproducibility in computational sciences is a goal hard to achieve due to the complexity of computational experiments, usually involving many steps and combining several data sources [2].

In the context of scientific experiments, reproducibility and replicability 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 [3]. On one hand, replicability can be defined as an exact incarnation of the original experiment, considering the exact same environment, and performed over the same individuals using the same original experiment configuration. On the other hand, reproducibility implies that even when the goal of the experiment is the same, at least part of the experiment has been modified to obtain a new result or has been adapted to fit a new scenario. In this work, we address the reproducibility of the execution environment for a scientific workflow, as we do not aim to necessarily 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 computational science, or *in silico* science, experiments are widely designed as scientific workflows. These workflows are being published more and more frequently as digital artifacts, along with their related publications, including not only the description of the experiment but also additional materials to understand them [4].

A scientific workflow is a precise, executable description of a scientific procedure [5]. Therefore, it is advisable to enrich them with a description of a proper execution environment, that, along with the workflow and the information about the rest of its components, would enable its reproduction. We distinguish three main components on a computational scientific experiment, which must be properly conserved to achieve its reproducibility:

(i) data: the input and output data of the experiment. The input data represents the information of the domain we study (e.g., light from the stars, molecule composition, etc.), and output data is the result of carrying out the experiment (e.g., charts, statistic deviation, plots, etc.) that enables verifying the experimental hypothesis;

(ii) scientific procedure: the description of all the steps of the experiment and how they must be performed, including as many details as possible for guaranteeing its traceability and repeatability. A computational workflow can be considered as a precise description of a scientific procedure, as it contains the details of each step of the process, relating how those steps are connected and how the data flows through them;

(iii) equipment: all the tools and materials involved in the experimental process. In computational science, the equipment is defined by the computational infrastructure, that is, the set of resources (computers, storage devices, networking, etc.) and software components necessary for executing the experiment.

The reproducibility of any object, either physical or digital, is achieved by its proper conservation. Conservation can be defined as the action of prolonging the existence of significant objects by researching, recording, and retaining the relevant information related to the object.

Currently, most of the approaches dealing with computational science conservation have been centered on the data and procedure components, leaving the computational equipment out of the scope. Hence, reproducibility problems related to execution environments are not being handled properly. The lack of approaches for conserving the infrastructure employed in an experiment makes scientists willing to repeat it to guess which original set of tools it was and how it was set up. This may be even impossible to do if the former components were insufficiently described and therefore it is not feasible to identify a counterpart offering the same capabilities.

In this paper, we try to identify the most relevant current approaches and their limitations. Taking into account these ones, we have developed an infrastructure-aware approach for computational execution environment conservation and reproducibility based on documenting the components of the infrastructure. We introduce here the main contributions of our work: (i) the set of semantic models for describing workflow execution environments, (ii) a catalogue documenting the resources involved on a real use case scenario, and (iii) an algorithm that generates an infrastructure specification based on this information.

The reminder of this paper is organized as follows. In Section 2, we show the main current approaches in the area of execution environment preservation. We then introduce in Section 3 our approach and then, in Section 4, we describe our semantic model. In Section 5, we establish a use case scenario in the context of Text Analytics workflows and expose the datasets we have generated using our models. In Section 6, we introduce an algorithm for generating an equivalent infrastructure specification and in Section 7 we summarize how we applied our ideas to the use case and, finally, in Section 8, we outline the main conclusions and define future lines of work.

2. Current Approaches

A computational experiment involves several elements, each of which must be conserved in order to ensure reproducibility. Conserving data and the workflow specification is not enough. As pointed out in [6] *"without the ability of properly consuming the conserved information we are left with files full of rotting bits."* Therefore, it is mandatory to maintain the operability of the tools for accessing, consuming, and interpreting the components of a scientific experiment (either input data or the description of the method). In this section, we survey the main conservation approaches regarding the experimental equipment.

An interesting study about issues in computational scientific reproducibility is exposed in [7], where authors conducted a study about workflow decay over a set of biological workflows from myExperiment [8] designed for the Taverna [9] platform. Authors define four different categories for workflow decay causes: *volatile third-party resources*, *missing example data*, *missing execution environment*, and *insufficient descriptions about workflows*. The study shows that nearly 80% of the workflows failed to be reproduced, and that around 12% of these failures happened due to *missing execution environment* issues and that 50% are due to *volatile third-party resources*. Taking into account that around 22% of the tasks in Taverna are related to web services [10], some of those third-party resources issues could be considered also as execution environment problems.

Data and workflow conservation has been widely addressed in recent years. Initiatives such as CrowdLabs [11], the Galaxy project [12], or GenePattern [13] aim to conserve and share the knowledge about scientific computational experiments and provide means for reproducing them. However, so far, a complete and integral approach for documenting and conserving the execution environment as a whole has not been developed.

In [14], authors expose how software must be preserved. As a complex and dynamic entity, software can not be preserved just by maintaining its binary executable code. Authors claim that a software component is more likely to be preserved by guaranteeing the performance of its features rather than conserving the same *physical* binary code. To this end, they introduce the concept of adequacy, as a way of measuring how a software component performs related to a certain set of features. The aim is to build a conceptual model that allows to capture the relevant properties of each software, enhancing the possibilities of successfully conserving them.

In 2011, the Executable Paper Grand Challenge [15] pointed out the importance of allowing the scientific community to reexamine the execution of an experiment. As a result of this challenge, some authors proposed the use of virtual machines as a way of preserving the execution environment of an experiment [16, 17]. Also, as part of the SIGMOD conference on 2011, a study was carried out to evaluate how a set of repeatability guidelines proposed to the authors submitting a paper (i.e., using virtual machines, pre- and post-conditions, and provenance-based workflow infrastructures) could help reviewers to reproduce the experiments described on the submitted paper [18].

A list of advantages and challenges of using virtual machines for achieving reproducibility is exposed in [19], arguing that availability of a highly distributed and automated solution for computing such as Cloud Computing allows cost reduction, efficient and reliable lifecycle management, large scale processing, and cost sharing. However, authors expose that using Cloud solutions implies issues that are not yet fully solved, such as the high cost of storing data in the Cloud or the problems of dealing with high interactive experiments through a network connection to remote virtual machines.

Authors also claim that provenance tracking inside a virtual machine or the reuse and repurpose of the infrastructure are real issues when using Cloud solutions. In our opinion, these claims are not major issues within the scope of our work. Provenance of a process executed on a virtual machine can be traced by using a workflow management system in the same way it can be traced on a local cluster. Regarding repurposing an infrastructure, this is out of the scope of our work, as we are trying to achieve its conservation and not looking for any kind of improvement.

Recently, some authors [20] have clearly exposed the necessity of capturing and preserving the execution environment of an experiment, providing tools for analyzing and packaging the resources involved on it. ReproZip [21] and CDE [22] are promising tools in this direction, which are aligned with some of the principles of our work, as they aim to capture the information about an infrastructure and try to reproduce this in a new environment. These tools read the infrastructure components involved on the execution (files, environment variables, etc.) and store this information into a package. This package can be later unpackaged into another machine in order to repeat the experiment.

These approaches differ from ours as we do not try to capture the real elements of the infrastructure (copy the files and libraries) but rather we try to describe them and obtain an available counterpart that can be tuned to expose the same features. We agree with ReproZip authors that packaging the physical infrastructure components limits the scope of applicability, as the packages require most of the target machine to be the same. We also argue that the knowledge and understanding of the infrastructure, as well as the dynamism of this solution, would be higher using an approach like the one exposed on our work, as we abstract the description of the infrastructure from the concrete elements that are involved in the former experiment.

Another recent and relevant contribution to the state of the art is being developed within the context of the TIMBUS project [23], which aims to preserve and ensure the availability of business processes and their computational infrastructure, aligning it with enterprise risk management and business continuity management. They propose, as we do, a semantic approach for describing the execution environment of a process.

Our approach differs from the TIMBUS one as we propose a more lightweight and explicit way of annotating the infrastructure information based on our ontology network. Even though TIMBUS has studied the applicability of their approach to the eScience scenario, their contributions are mainly focused on business processes. Our approach focuses on scientific processes, which are dataflow-oriented and usually do not contain loops or branch structures.

Finally, we highlight deployment tools, such as Puppet [24], Chef [25], and PRECIP [26], which are able to take an infrastructure specification and deploy it on a Cloud provider and are highly useful for recreating an execution environment. We will introduce in this work how PRECIP can be used as an enactment system within the reproducibility process.

3. Infrastructure-Aware Approach

The equipment used in a scientific experiment plays a key role on its reproducibility. Without the proper set of tools, it is hard to ensure the execution of the same process obtaining consistent results. In order to guarantee it, we need to document and describe the tools involved (types, brand, provider, version, etc.) and the information for setting it up (calibration, configuration, handling, etc.).

We identify two different approaches for conserving the equipment of an experiment, depending on how relevant this equipment is and how hard it is to obtain an equivalent individual of the involved tools:

(i) physical conservation: that is, conserving the real object, due to its relevancy and the difficulty of getting a counterpart. The Large Hadron Collider (LHC http://lhc.web.cern.ch/lhc/) or specialized telescopes for high-energy astronomy are examples of this kind of equipment, due to its singularity. In those cases, it is mandatory to conserve the real equipment and bring access to them to the research community;

(ii) logical conservation: usually the equipment used in experimental science can be obtained by most of the research community members. Commonly, it is easier to obtain a counterpart than accessing the original tool. Sometimes accessing the original is even impossible due to its natural decay (individuals used in experiments such as plants or animals). In those cases, it is more suitable to describe the object so an equivalent one can be obtained in a future experiment: buying the same model and version of a microscope, cultivating a plant of the same species, and so forth. This requires a precise and understandable description of those elements.

As pointed out before, in computational science, the main tools for carrying on an experiment are computational infrastructures, either virtual or physical, where the high amount and variety of requirements of a computational experiment imply a highly heterogeneous environment. In this work, we define a computational infrastructure as the set of computational nodes and software components that are set to execute a computational experiment.

Classical reproducibility approaches in computational science aim to share the infrastructure by bringing access to it within a community of users with the same interests. This approach clearly fits the physical conservation case; an organization or a set of them sets up an infrastructure (supercomputers, clusters, and grids) for a specific goal and allows some users to have access to their resources under some conditions. These are usually big and expensive infrastructures that require a lot of effort of maintenance in the long term. These infrastructures have proved to be a significant contribution in computational science.

However, there are some challenges regarding reproducibility issues that these kinds of approaches cannot face. Within the context of this work, we identify the following:

(i) static infrastructures: classical infrastructures require a huge technical maintenance effort, as they must be tuned and configured in order to fulfill the requirements of the different experiments developed by the community. The process of adapting these kind of infrastructures to a new experiment is not trivial and usually is restricted by the policies of the organization hosting it. This also makes the reexecution of an old experiment more difficult once the infrastructure has been modified;

(ii) vendor locking: even though the main purpose of most of these infrastructures is to be shared with as many users as possible, it is not feasible to assume that any organization can guarantee access to their infrastructure to everyone interested on executing or repeating an experiment;

(iii) long term conservation: guaranteeing the conservation of an infrastructure is not a trivial task. Issues such as projects ending or funding cuts may challenge its future availability. Moreover, any infrastructure suffers from a natural decay process [27]. Machines get eventually broken and as new tools appear and new software and hardware requirements are needed, so machines must be replaced.

To solve the above-mentioned challenges, we propose in this work a different approach that contributes to the previous one rather than substituting it. We aim to face those challenges from a logical-oriented conservation point of view.

Instead of trying to conserve and share a physical infrastructure, we propose to describe its capabilities and, based on that description, reconstruct the former infrastructure (or an equivalent one) using virtualization techniques.

Virtualization is a mature technique that has been used for the last three decades and that has lately gained momentum. By using these techniques, we are facing the long term conservation challenge, as virtualization solutions are mature enough to assume that they will be available in the future. By introducing the concept of an infrastructure-aware approach, we aim to develop a solution that is not tied to any specific virtualization solution, allowing the system to be adapted to new solutions as they emerge.

This approach implies some restrictions and assumptions on the scope of applicability of this work, mainly related to performance aspects, as it is hard to guarantee the performance of a resource when using virtualization. Therefore, we leave out the scope of this work those experiments that take into account the performance as part of their goals.

In our approach, we propose to define the capabilities of the machines involved in the experiment, rather than using just virtual machine images with those capabilities installed on them. Based on the description, we would be able to generate a set of virtual machines exposing those capabilities. As Cloud computing (either private or public) is meant to be a public facility, allowing almost everyone to create virtual resources, we claim that our approach faces the vendor-lock problem by implementing a common and shared pool of resources in which every researcher could execute scientific applications. We assume that research communities have access to those resources and that they can be hosted on the Cloud.

As mentioned before, virtualization allows the customization of the resources, so we can define a specific infrastructure configuration for each experiment. This eases the configuration and maintenance process of classical approaches.

Our approach aims to separate the description of the infrastructure capabilities from the specific solutions that could be used to generate it, defining a more adaptive solution that increases the chances of reproducing the execution environment. This approach also simplifies the storage requirements of the experiment equipment, as it is more feasible to store and preserve a description of a computational infrastructure than preserving it physically, as suggested in other approaches that aim to store and preserve virtual machine images.

We assume that with this approach it is not always possible to reproduce the execution environment, as the necessary resources may not be available and an equivalent counterpart may not be found. However, we think that this is a more versatile and flexible approach and therefore increases the chances of achieving the infrastructure reproduction. Also, we consider that initiatives such as WINGS or myExperiment, which maintain a shared catalog of components accessible in the long term, support our goal.

We identify the following main technical contributions that must be implemented to achieve the goals of this approach:

(i) models/vocabularies: we need to define a way for representing and relating all the relevant information about the capabilities of the infrastructure. We propose the use of semantic techniques to develop a set of interrelated ontologies defining the necessary aspects and relations about all the components involved in the execution of a computational experiment. Semantic technologies are a standard and integrable way of sharing information, which is an important feature when trying to share and conserve the knowledge of an entity. The description of the infrastructure and the resources in the catalogue will be described using these vocabularies;

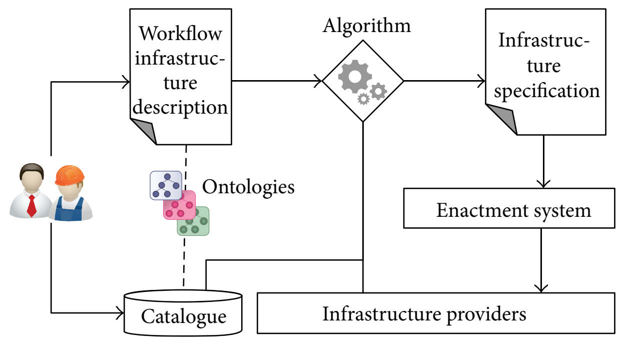

Figure 1: System overview.

(ii) catalogue: it is not feasible to assume that we could be able to deal with every kind of virtual appliance and to install and manage every kind of application. We need to develop a scientific appliances catalogue, including a set of representative virtual appliances, and applications for supporting computational experiments within the context of a scientific community. This catalogue will be dynamically populated by the members of the community involved in the experiment design and development and will serve as part of the input information for the process of generating the infrastructure configuration;

(iii) methodological framework: it is composed by a set of tools and methodologies for capturing all the necessary knowledge for conserving the infrastructure. This framework will define who is responsible for the annotation and curation of the information, describing the components of the infrastructure, and will guide the users in the process of identifying the elements that must be documented and which details about them must be included.

Figure 1 depicts an overall view of the main components of our contribution and how they are related. The diagram shows how the different users (scientists, IT staff members, etc.) interact with the *Catalogue*, querying it to obtain the identifiers of the resources used in the experiment and adding new resources in case they do not already exist. These identifiers are used within the *Workflow Infrastructure Description* to link each part of the workflow to its dependencies (software packages, libraries, hardware requirements, etc.) in the *Catalogue*, and therefore these dependencies are added to the description of the infrastructure. All the components included in the description and the catalogue are defined using the terms and relations included in the *Ontologies* of our system.

The *Workflow Infrastructure Description* serves as the input of the *Infrastructure Specification Algorithm*, which is invoked whenever an infrastructure must be reproduced. This process also queries the state of the available *Infrastructure Providers* in order to get the resource availability and the *Catalogue* to retrieve which tools and appliances can be used. With this information about the former infrastructure and providers, the algorithm generates an *Infrastructure Specification*, which defines a deployment plan detailing the resources to be created and how they must be configured. Finally, the *Enactment System* reads the *Infrastructure Specification* and carries out the actions defined on it over the *Infrastructure Providers*, producing the target infrastructure that the experiment would use to be executed.

In this work, we assume that the software binaries are either available online, as part of Open Source project and/or on a public repository, or available on the user filesystem or organization's repository. We argue that this assumption holds for most of the scientific projects, where the necessary software components are available within the context of the scientific community. Even though we have included some concepts related to software licensing in our ontologies, issues related to license and software rights are out of the scope of this work.

4. Workflow Infrastructure Representation

In this work, we introduce the idea of describing computational resources involved in the execution of a scientific workflow. To this end, we propose the uses of semantic technologies for defining the necessary set of conceptual models that allow us to generate descriptions of the workflow and its environment. Semantic technologies include standardized languages such as OWL2 (http://www.w3.org/TR/owl2-overview/), data models such as RDF (http://www.w3.org/RDF/) and query languages such as SPARQL (http://www.w3.org/TR/rdf-sparql-query/), along with a wide range of development and management tools, that make them a mature and proved solution.

The WICUS ontology network (available at http://purl.org/net/wicus) describes the main concepts of a computational infrastructure in a scientific workflow. This network is composed of five ontologies written in OWL2, four domain ontologies, describing the different concepts of a scientific workflow from the point of view of its infrastructure, and another ontology for linking them.

The four ontologies that compose the network are the `Software Stack Ontology` (Section 4.1), the `Hardware Specs Ontology` (Section 4.2), the `Scientific Virtual Appliance Ontology` (Section 4.3), and the `Workflow Execution Requirements Ontology` (Section 4.4).

As mentioned, we join these ontologies by means of the `Wicus ontology`, which defines five object properties relating concepts from those ontologies, as depicted in Figure 2. In this section, we introduce the main concepts and properties of the network. A more detailed description of each ontology can be accessed through their corresponding URIs.

4.1. Software Stack Ontology. This ontology describes the software elements of a computational resource. These descriptions can be used to describe both the already deployed software components and the software requirements of a workflow, depending on whether it is being used for describing requirements of a virtual appliance.

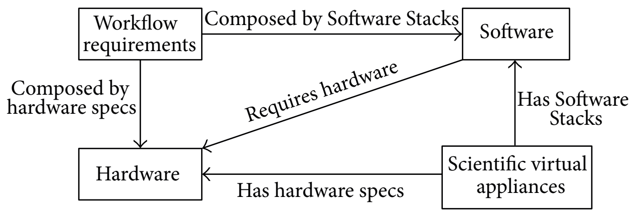

Figure 2: WICUS ontology network overview.

The main concept of this ontology is the `wstack:SoftwareStack` class, which groups a set of `wstack:SoftwareComponent`. The `wstack:SoftwareComponent` class encodes the information of a software package, a library, a script, or, in general, any piece of executable software.

The information about how to deploy and configure a software element is also included in this ontology. The `wstack:DeploymentPlan`, `wstack:DeploymentStep` and `wstack:ConfigurationParameter` classes encode this information.

4.2. Hardware Specs Ontology. An ontology is meant for describing the hardware characteristics of a computational infrastructure and the hardware requirements of a software component.

This ontology describes the set of hardware characteristics/requirements of an element by means of the class `whw:HardwareSpec`, which aggregates a set of `whw:HardwareComponent` such as the CPU or the RAM memory and details its capabilities using the `whw:Feature`.

4.3. Scientific Virtual Appliance Ontology. This ontology defines the main concepts related to virtualized computational resources provided by an infrastructure provider (e.g., IaaS Cloud providers). This ontology focuses on Scientific Virtual Appliances, that is, the assembly of virtual hardware and software components designed within the scope of a scientific process.

This ontology describes the concept of `wsva:ScientificVirtualAppliance`, a computational resource described as a virtual machine with a set of software components deployed on it. These resources define a set of features specifically designed for supporting the execution of a scientific workflow.

`wsva:ScientificVirtualAppliance` is based on an `wsva:ImageAppliance` that aggregates a set of `wsva:VMImage` that can be used for enacting a virtual machine in a certain `wsva:InfrastructureProvider` and then configured to expose the capabilities of the appliance.

4.4. Workflow Execution Requirements Ontology. We have implemented this ontology for describing and relating a scientific workflow with its execution requirements.

This ontology relates the concept of `wreq:Workflow`, representing a scientific workflow, to a `wreq:ExecutionEnvironment`, defining its dependencies. Depending on whether the steps of the workflow are fully specified in terms of infrastructure or not, we distinguish between `wreq:ConcreteWorkflow` and `wreq:AbstractWorkflow`.

These concepts are related to the `wreq:requires` object property. For the purpose of describing the execution infrastructure, we consider that a wokflow can be composed of several subworkflows, defining for each one of them their own infrastructure description. We do not describe the execution order of these workflows or their inner steps, as this information is not relevant from the infrastructure point of view.

5. Annotations Catalogue

To exemplify the ideas of our approach, we introduce here a use case scenario involving a set of workflow templates from the WINGS platform along with their related software components. We have studied and annotated 4 workflow templates from the Text Analytics domain [28].

These templates contain abstract steps, that is, steps that can be implemented using different software components. Combining all the possible software components, we obtain 56 different workflow implementations.

In this section, we explain how these templates are annotated and then how this information is consumed by our algorithm to generate an infrastructure specification. We include three different datasets, namely, the *Workflow Requirements Dataset*, the *Software Stacks Dataset*, and the *Scientific Virtual Appliance Dataset*. All of them are included in the *TextAnalyticsWF-RO* Research Object [4] (available at http://purl.org/net/TextAnalyticsWF-RO), a bundle containing the resources and documentation of our use case.

For these templates, we generated the *Workflow Requirements Dataset*, a *Software Stacks Dataset*, and a *Scientific Virtual Appliance Dataset*, which all together compose the *Catalogue* component of the system depicted in Figure 1.

To annotate the requirements of each template, we have developed a prototype of an annotation tool (available at http://github.com/idafensp/WicusAnnotationTool), which takes a workflow template as input and generates a web form. This form includes some parameter suggestions based on the information retrieved from the WINGS component catalogue, using string similarity between the workflow step and component's name.

This tool takes advantage of the similar structure of most of the components. All of them include a shell script that invokes an external script (Java, MATLAB, or Weka), so we can suggest to the user the most probable parameters. All those suggestions can be modified by the user, correcting them and adding new ones. This tool improves the usability of our approach, reducing the amount of effort for annotation. Improvements of this tool, such as integrating systems like CDE [22] or SystemTap [29] to trace the execution and crawl the software components involved, are a part of our future work.

Using this tool, we have documented the mentioned 4 templates and their related 27 software components. This information includes the configuration parameters values and version required (when available). Our system allows to record the information about versions, but do not enforce it for all component. In this case we had information for describing the version of the Java VM and Ubuntu components, but not for the JAR components as developers did not include this information. We consider that is important to allow both, versioned and unversioned components in our system, as many times scientific software is not developed and published using a versioning cycle.

We also have generated a catalogue including the available Software Stacks, detailing the Software Binaries location of each software component and what parameters can be specified. In this catalogue, we also include a deployment plan for each component, that specifies a set of steps and scripts for installing it in its future target location.

For describing the available computational resources, we have generated a dataset for Scientific Virtual Appliances. In this dataset, we describe the set of virtual machine images that can be used to enact the appliance and the Software Stacks installed on them.

These 3 datasets, along with the Software Binaries and the VM Images, compose the 3 catalogues that our system queries for generating the new infrastructure configuration. The *TextAnalyticsWF-RO* research object associated with this paper contains the RDF files of each catalogue, as well as a set of sample queries for interacting with them.

6. Infrastructure Specification Algorithm

In this section, we introduce the Infrastructure Specification Algorithm (ISA), a process for generating an infrastructure specification, which defines the set of virtual machines that must be created and the software components that will be deployed on each one of them.

The inputs of this algorithm are the three datasets explained in Section 5 and the identifier, as a URI, of the workflow whose infrastructure we want to reproduce.

We have developed this proof-of-concept implementation of the algorithm in Java, using Jena (http://jena.apache.org/) for managing the semantic information of our system, which is stored locally. Jena provides the SPARQL endpoint we query for retrieving information.

This version of the algorithm aims to find all the possible matches between the requirements of a workflow, including their dependencies, and the available virtual appliances.

Firstly, after loading the datasets (lines 1–5), we query the *Workflow Requirements Dataset* to load the requirements of each step of the workflow and their related Software Stacks (line 7). Then, we query the *Software Stacks Dataset* to recursively calculate the Requirement Dependency Graph, a directed graph that relates Software Stacks with their dependent ones (lines 10–15).

This graph may contain circular dependencies. Even though these dependencies are a bad practice in software design and development, they usually occur in functional programming and among nonexpert developers, which may be the case of a scientist developing a computational experiment without sufficient skills on programming. In our datasets, there are no circular dependencies; however, our algorithm is able to deal with them avoiding infinite loops by keeping track of the visited nodes and avoiding them over the recursive iterations.

Once we have generated the Requirement Dependency Graph, we retrieve the information of all the available Scientific Virtual Appliances (line 17). We then calculate the compatibility between each requirement and appliance (line 19). To do that, we calculate the intersection of the set of Software Stacks of the graph with the set of stacks of the appliance. We require the intersection to be greater than a certain threshold. Appliances below this threshold are removed from the candidates list. In this work, we have used a threshold parameter with value 0, requiring that at least both sets have 1 stack in common.

We then sort the resultant appliances for obtaining the one with the greater intersection for each requirement, arguing that the more components they have in common, the less deployment effort would be needed, making the appliance more suitable.

Finally, we use this intersection to remove the unnecessary stacks from the Requirement Dependency Graph, as they are already deployed in the appliance. We remove each stack included in the intersection and its derived dependencies recursively (line 21). That is, the dependencies that have no other stack depending on them and therefore get isolated once the stack from the intersection has been removed.

We finally merge subworkflows that share common requirements (line 23). This version implements a simple policy that aims to reduce the deployment effort. This is a policy among many others that may be implemented considering many other aspects, such as performance (favoring local Cloud providers) or economic aspects (encouraging the system to select the cheapest available SVA).

A pseudocode overview of the main steps of the ISA is listed in Listing 1.

In the last step, we generate a PRECIP experiment which creates the necessary computational resources based on the SVAs (line 23). We traverse the software components that must be deployed to generate a set of PRECIP commands for executing the deployment plans of each component.

An online demo of the current implementation is available at http://github.com/idafensp/WicusISADemos/tree/master/v.0.2.

7. Putting It Together

Once we have all the appliances and their associated requirement's dependency graphs, we generate the infrastructure specification using the stacks' deployment plans. In Listing 2, we see how our algorithm has included scripts and configuration parameters of the stacks for generating a PRECIP file that can be enacted on the Amazon EC2 Infrastructure Provider. This specification corresponds to the *Feature Generation* workflow included in our dataset. This workflow processes a file containing a list of words and generates a vector format data structure with a filtered version of the original file.

In this specification file, we see the deployment description of one experiment. The depicted fragment of the file corresponds to the *StopsWords* step of the *FeatureGeneration* workflow. In this example, the user specified that this step required the *StopWords.jar* component (lines 31–60).

```
WorkflowRequirementsDataset.load();

SVADataset.load();

SoftwareCatalogDataset.load();

Map<Workflow,List<Requirements>> wfReqs =
  retrieveRequirements(WorkflowRequirementsDataset, WorkflowID);

Map<Workflow,List<Requirements>> propagatedWfReqs =
  propagateReqs(wfReqs);

List<List<List<SWComponents>>> softwareComponents =
     getSoftwareComponents(propagatedWfReqs);

Map<Requirement,D-Graph<SWComponents>>
     softwareComponentsDependencyGraph = getSoftwareDependencies(
     softwareComponents);

List<SVA> availableSvas = getAvailableSvas(providersList);

Map<Requirements,SVA> maxCompatibilities =
     getCompatibilityIntersection(softwareComponentsDependencyGraph,
     availableSvas);

Map<Requirement,D-Graph<SWComponents>>
     substractedSwComponentsDepGraph = substractSoftwareComponents(
     softwareComponentsDependencyGraph, maxCompatibilities);

Map<SVA, List<Requirments>>mergedSvas= mergeSubworkflows(
     propagatedWfReqs, maxCompatibilities);

generatePrecipScript(mergeSubworkflows,
     substractedSweComponentsDepGraph);
```

Listing 1: Pseudocode overview of the ISA.

According to the *Software Stacks Dataset,* this component depends on another JAR named *RemoveStopWords.jar*, which depends on the Java VM version 1.7.0_21, and on the Ubuntu 12.04 OS, and therefore the algorithm has included both dependencies (lines 9–29). The Java VM included in the catalogue corresponds to the available Ubuntu implementation.

In the annotations, it was specified that the *JAVA_HOME* variable must be set to an specific path and that both JAR files should be located on the "DIR:" folder. We have defined two configurable variables in our system, "DIR:" and "REPO:" that the user can define. "DIR:" represents the execution folder of the workflow that usually depends on the WMS, while "REPO:" defines the URL (either FTP or HTTP) where the software binaries can be located.

Our systems selects in this case the "ami-967edcff" Amazon Machine Image (lines 1–9), as it appears annotated in the *Scientific Virtual Appliance Dataset* with the Ubuntu 12.04 software stack, and therefore it matches the requirements of the step. Notice that the specification on Listing 2 does not include Ubuntu 12.04, as it is already described as installed on the selected appliance.

In this example, we have defined the Ubuntu 12.04 software stack as a bundle that includes a virtual machine image that includes that version of the OS. A more detailed description of it, including specific libraries and tools would be useful. Also defining a more expressive dependency relationship, that could define that the component, depends on the Ubuntu OS or any variant of it under some restrictions and would be desirable. These two aspects are a part of the future work in which we plan to generate more fine-grained annotations and add new and more expressive object properties based on the work described in [14].

With this specification along with the scripts referenced on it, we will be able to reproduce an infrastructure for reexecuting the former workflow.

8. Conclusions and Future Work

In this work, we motivate and expose how an infrastructure-aware approach could ease the experiment reproduction and argue how it should be done. As described in this paper, we propose to describe the computational resources,

```
exp = EC2Experiment(
  os.environ['AMAZON_EC2_REGION'],
  os.environ['AMAZON_EC2_URL'],
  os.environ['AMAZON_EC2_ACCESS_KEY'],
  os.environ['AMAZON_EC2_SECRET_KEY'])



exp.provision("ami-967edcff", instance_type="t1.micro", tags=["inst0"
    ], count=1)

# Wait for all instances to boot and become accessible.
exp.wait()

···

# [STACK] Deployment of wstack:JAVA_SOFT_STACK stack

# [COMPONENT] Deployment of wstack:JAVA_SOFT_COMP component(version
    : 1.7.0_21)

# [STEP] Execution of wstack:JAVA_DEP_STEP step

# copy JAVA_script.sh to the instance
exp.put(["inst0"], "JAVA_script.sh", "/home/cloud-user/JAVA_script.
    sh", user="root")

# granting execution for JAVA_script.sh
exp.run(["inst0"], "chmod␣755␣/home/cloud-user/JAVA_script.sh", user
    ="root")

# executing the JAVA_script.sh script
exp.run(["inst0"], "/home/cloud-user/JAVA_script.sh␣JAVA_HOME␣/usr/
    lib/jvm/", user="root")

# [STACK] Deployment of wstack:REMOVESTOPWORDS_SOFT_STACK stack

# [COMPONENT] Deployment of wstack:REMOVESTOPWORDS_SOFT_COMP
    component

# [STEP] Execution of wstack:REMOVESTOPWORDS_DEP_STEP step

# copy deploy_jar.sh to the instance
exp.put(["inst0"], "deploy_jar.sh", "/home/cloud-user/deploy_jar.sh"
    ,user="root")

# granting execution for deploy_jar.sh
exp.run(["inst0"], "chmod␣755␣/home/cloud-user/deploy_jar.sh", user=
    "root")

# executing the deploy_jar.sh script
exp.run(["inst0"], "/home/cloud-user/deploy_jar.sh␣␣JAR_FILE␣REPO:
    jar/RemoveStopWords.jar␣DEST_PATH␣DIR:␣", user="root")

# [STACK] Deployment of wstack:STOPWORDS_SOFT_STACK stack

```

Listing 2: Continued.

```
# [COMPONENT] Deployment of wstack:STOPWORDS_SOFT_COMP component

# [STEP] Execution of wstack:STOPWORDS_DEP_STEP step

# copy deploy_jar.sh to the instance
exp.put(["inst0"], "deploy_jar.sh", "/home/cloud-user/deploy_jar.sh"
    ,user="root")

# granting execution for deploy_jar.sh
exp.run(["inst0"], "chmod␣755␣/home/cloud-user/deploy_jar.sh", user=
    "root")

# executing the deploy_jar.sh script
exp.run(["inst0"], "/home/cloud-user/deploy_jar.sh␣JAR_FILE REPO:jar
    /StopWords.jar␣␣DEST_PATH␣DIR:␣", user="root")

···
```

Listing 2: Part of the PRECIP specification file for the FeatureGeneration workflow.

from hardware specification to software configuration, rather than physically conserving and sharing them. We argue that documenting the features and characteristics of these resources in an abstract way improves the expressiveness of the approach, increasing the chances of recreating an equivalent execution environment in the future.

We have described a use case scenario on Text Analytics domain, in which we documented 4 workflow templates and their related resources. Based on those descriptions, we have explained how our *Infrastructure Specification Algorithm* is able to generate an equivalent infrastructure definition that can be later enacted on an infrastructure provider.

We are planning to extend our *Infrastructure Specification Algorithm*, providing new policies and including new criteria for selecting and combining computational resources. Even though the running time of our algorithm is not expensive (around 4 seconds), we plan to study different heuristics to our selection process in order to reduce its complexity, which is important when dealing with large collections of resources.

We will also extend the WICUS ontology network, adding new properties and classes for increasing its expressiveness. These capabilities will allow us to define infrastructures with a deeper level of detail and include more dimensions for describing software components, including properties such as *package*, *release,* or *variant*, as described in [14].

The annotations about the workflow infrastructure requirements are a key component of our approach. In order to obtain a coherent description of the execution environment and also to reduce the amount of effort for the scientists, we need to improve and expand the usability of our annotation tools.

We are planning to add features for automatic capture and suggest the components involved in the execution. In addition to this tool, we need to provide a set of guidelines that allow users to understand and evaluate what are the necessary and relevant properties of their execution environments. We will define a set of methodological steps for experiment designers to know how to populate and for scientists in general to know how to reproduce an execution environment using our approach.

We are also planning to apply our approach to another science domains (such as astrophysics, genetics, etc.), using different workflow management systems, in order to validate the ideas exposed in this paper on different contexts involving different kinds of workflows.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to thank the WINGS Project (http://wings-workflows.org/) and the PRECIP (http://pegasus.isi.edu/precip) team for their support and materials and the FPU Grant Program (Formacion de Profesorado Universitario) from the Spanish Science and Innovation Ministry (MICINN).

References

[1] J. P. Mesirov, “Accessible reproducible research,” *Science*, vol. 327, no. 5964, pp. 415–416, 2010.

[2] Y. Gil, E. Deelman, M. Ellisman et al., “Examining the challenges of scientific workflows,” *Computer*, vol. 40, no. 12, pp. 24–32, 2007.

[3] 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.

[4] K. Belhajjame, O. Corcho, D. Garijo et al., “Workflow-centric research objects: first class citizens in scholarly discourse,” in *Proceedings of the Workshop on the Semantic Publishing*, Crete, Greece, 2012.

[5] D. de Roure, K. Belhajjame, P. Missier et al., “Towards the preservation of scientific workflows,” in *Proceedings of the the 8th International Conference on Preservation of Digital Objects (iPRES ’11)*, Singapore, 2011.

[6] V. G. Cerf, “Avoiding ‘bit rot’: long-term preservation of digital information,” *Proceedings of the IEEE*, vol. 99, no. 6, pp. 915–916, 2011.

[7] J. Zhao, J. M. Gomez-Perez, K. Belhajjame et al., “Why workflows break—understanding and combating decay in Taverna workflows,” in *Proceedings of the IEEE 8th International Conference on E-Science*, pp. 1–9, October 2012.

[8] D. de Roure, C. Goble, and R. Stevens, “Designing themyexperiment virtual research environment for the social sharing of workflows,” in *Proceedings of the 3rd IEEE International Conference on E-Science and Grid Computing (E-SCIENCE ’07)*, pp. 603–610, IEEE, Washington, DC, USA, December 2007.

[9] T. Oinn, M. Greenwood, M. Addis et al., “Taverna: Lessons in creating a workflow environment for the life sciences,” *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1067–1100, 2006.

[10] I. Wassink, P. E. van der Vet, K. Wolstencroft et al., “Analyzing scientific workflows: why workflows not only connect web services,” in *IEEE Congress on Services 2009*, L. J. Zhang, Ed., pp. 314–321, IEEE Computer Society Press, Los Alamitos, Calif, USA, July 2009.

[11] P. Mates, E. Santos, J. Freire, and C. T. Silva, “Crowdlabs: social analysis and visualization for the sciences,” in *Proceedings of the 23rd International Conference on Scientific and Statistical Database Management (SSDBM ’11)*, pp. 555–564, Springer, Berlin, Germany, 2011.

[12] B. Giardine, C. Riemer, R. C. Hardison et al., “Galaxy: a platform for interactive large-scale genome analysis,” *Genome Research*, vol. 15, no. 10, pp. 1451–1455, 2005.

[13] M. Reich, T. Liefeld, J. Gould, J. Lerner, P. Tamayo, and J. P. Mesirov, “GenePattern 2.0,” *Nature Genetics*, vol. 38, no. 5, pp. 500–501, 2006.

[14] B. Matthews, A. Shaon, J. Bicarregui, J. Catherine, J. Woodcock, and E. Conway, *Towards a Methodology for Software Preservation*, 2009.

[15] 2011, http://www.executablepapers.com/.

[16] P. Van Gorp and S. Mazanek, “Share: a web portal for creating and sharing executable research papers,” *Procedia Computer Science*, vol. 4, pp. 589–597, 2011, Proceedings of the International Conference on Computational Science, 2011.

[17] G. R. Brammer, R. W. Crosby, S. J. Matthews, and T. L. Williams, “Paper mache: creating dynamic reproducible science,” *Procedia Computer Science*, vol. 4, pp. 658–667, 2011, Proceedings of the International Conference on Computational Science (fICCSg '11).

[18] P. Bonnet, S. Manegold, M. Bjørling et al., “Repeatability and workability evaluation of SIGMOD 2011,” *SIGMOD Record*, vol. 40, no. 2, pp. 45–48, 2011.

[19] B. Howe, “Virtual appliances, cloud computing, and reproducible research,” *Computing in Science and Engineering*, vol. 14, no. 4, Article ID 6193081, pp. 36–41, 2012.

[20] Reproducibility in computational and experimental mathematics, 2012.

[21] F. Chirigati, D. Shasha, and J. Freire, “ReproZip: using provenance to support computational reproducibility,” in *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance*, 2013.

[22] P. J. Guo, “CDE: run any Linux application on-demand without installation,” in *Proceedings of the 25th International Conference on Large Installation System Administration (LISA ’11)*, USENIX Association, Berkeley, Calif, USA, December 2011.

[23] S. Strodl, R. Mayer, G. Antunes, D. Draws, and A. Rauber, “Digital preservation of a process and its application to e-science experiments,” in *Proceedings of the 10th International Conference on Preservation of Digital Objects (IPRES ’13)*, 2013.

[24] Puppet Labs, Puppet, http://projects.puppetlabs.com/projects/puppet.

[25] Opscode, Chef, http://www.opscode.com/chef/.

[26] S. Azarnoosh, M. Rynge, G. Juve et al., “Introducing precip: an api for managing repeatable experiments in the cloud,” in *Proceedings of the IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom ’13)*, pp. 19–26, Bristol, UK, December 2013.

[27] M. Gavish and D. Donoho, “A universal identifier for computational results,” *Procedia Computer Science*, vol. 4, pp. 637–647, 2011, Proceedings of the International Conference on Computational Science (fICCSg '11).

[28] M. Hauder, *Efficient text analytics with scientific workflows [M.S. thesis]*, University of Augsburg, Institute for Software and Systems Engineering, Augsburg, Germany, 2011.

[29] Systemtap, http://sourceware.org/systemtap/.