Computational Environment: An ODP to Support Finding and Recreating Computational Analyses

Michelle Cheatham[1], Charles Vardeman II[2], Nazifa Karima[1], and Pascal Hitzler[1]

[1] Wright State University
{michelle.cheatham,karima.2,pascal.hizler}@wright.edu
[2] University of Notre Dame
cvardema@nd.edu

Abstract. The Computational Environment ontology design pattern models the environment in which a computational analysis was conducted down to the hardware level. The pattern is intended to support comparison and reproducibility of computational analyses. Due to the concrete nature of the pattern, it makes use of several controlled vocabularies, which are kept current through an automated script that leverages the manually curated information on Wikipedia.

Keywords: computational environment, ontology design pattern, reproducibility

1 Introduction

Recently there has been a push towards ensuring the reproducibility of scientific results presented in scholarly publications. Journals have relaxed page restrictions on "methodology" sections, and funding agencies have begun to require investigators to publish any data they collect to data repositories. This is key to allowing scientists to build on the work of others and to avoid wasted funds and effort unnecessarily repeating work or re-collecting data. These efforts alone are not likely to be sufficient, however. To truly enable discoverability and reproducibility of previous scientific results as required by the scientific method, as well as to understand the context of those results so that the results can be correctly interpreted and extended, there is a need to preserve the underlying computations and analytical process that led to those results in a generic machine-readable format.

Significant work has already been done on modeling a computational analysis process, but this prior work tends to stop short of capturing the underlying hardware and operating system details necessary to replicate the procedure. In this paper we therefore present an ontology design pattern to represent a Computational Environment. The ODP was developed over the course of several working sessions by a group of ontological modeling experts, library scientists, and domain scientists from different fields, including computational chemists and high-energy physicists interested in preserving analysis of data collected from the Large Hadron Collider at CERN. Our goal was to arrive at an ontology design pattern that is capable of answering the following competency questions:

- What environment do I need to put in place in order to replicate the work in Paper X?
- There has been an error found in Script Y. Which analyses need to be re-run?
- Based on recent research in Field Z, what tools and resources should new students work to become familiar with?
- Are the results from Study A and Study B comparable from a computational environment perspective?

2 Related Work

Because the goal of reproducing computational analyses is recognized as very important, there has been a large body of work on this topic. In this section we focus on the existing work that is most similar to our goals. The primary difference between these previous efforts and our current one is the type of software and hardware **environment** in which a computational analysis takes place. For instance, Oscar Corcho and his colleagues have been working on Research Objects [1], which are somewhat based on Carol Goble's Taverna workflows [3]. This work assumes that the activities within the workflows are web services. As a result, Research Objects do not model machine- and platform-specific aspects of the computational environment, such as processor speed, amount of memory, operating system version, etc.

Other work comes from the field of digital object preservation. PREMIS, for instance, is a data dictionary associated with the Open Archival Information System [2]. The focus here is slightly different from our goals – rather than attempting to preserve the environment necessary to reproduce or understand a series of computational activities, PREMIS seeks to preserve the environment necessary to render a document or program (such as a video game) that has been archived. The focus on preserving an *entity* rather than a process, as well as the need to remain compatible with the rest of PREMIS and the OAIS, mean that the model is not suitable for our current use case. However, the PREMIS model does consider hardware, software, and external dependencies, and it can serve as a useful basis for the Computational Environment model we are developing.

Another related effort of which we are aware is work by Secure Business Austria, together with a research group at Karlsruhe, to develop a "process context model" [4]. This work is in a sense a proper superset of our current goals: their model considers hardware, operating system, software, and third party libraries and services, but it goes far beyond that by also including the organizational environment in which the process was executed, the people involved, licenses, authorizations, etc. This wide-ranging coverage area and the sheer size of the model (it contains 240 entities arranged in 25 major groups) make this model unwieldy for our current effort.

Very recent work by researchers from Ghent University and the University of Bonn is available online as a technical report [5]. The ontology presented there models software components and their configuration with the intent of enabling reproduction of computational experiments and analysis. Important entities in their model include a software *bundle*, such as a library or application, a *module*, which is a particular version of a library, and a *component*, which is a specific part of a module that can be called with particular parameters, such as a constructor in an object-oriented application. This approach to modeling a computational environment somewhat overlaps the one presented here. It lacks the details about the hardware and some aspects of the operating system environment that this effort seeks to capture, but it does represent the software portions of the computational environment important for reproducibility. It would be possible to align the two models or to replace a portion of the model presented here with the one from [5]. We will discuss this in more detail in Section 5.

3 Scope

Our goal in this work is to model the actual environment present during a computational analysis. Representing all possible environments in which it is feasible for the analysis to be executed is outside of the scope of our current effort. For example, statements such as 'This analysis was done on a computer running Ubuntu version 14.0.4' are in scope, while knowledge such as 'For this analysis, Ubuntu version 12.0 or greater is required' is not.

Figure 1 shows a general representation of the hardware, software, and external resources associated with a computational analysis. It is possible to consider all of this part of the Computational Environment. Instead, we have decided to define the boundaries somewhat more narrowly: we do not include the runtime configuration and parameters as part of the environment. The rationale is that to some extent the same environment should be applicable to many computational analyses in the same field of study, but this would not be true if we included such analysis-specific information as runtime parameters as part of the environment. Data sources were considered outside of the confines of the computational environment for similar reasons. External web services and similar resources were not included because they are not inter-related in the same way as the environmental elements are. For instance, deciding to use a different operating system often necessitates using a different version of drivers, libraries, and software applications, whereas in most cases the entire blue section of Figure 1 could be changed with no impact on the external services.

These decisions are obviously subjective to some degree, but they make practical sense in this case because numerous knowledge representations for web services and data resources already exist, and it does not seem productive to recreate them here. Although highly relevant to scientific workflows, we have further decided to leave the modeling of parallel and grid computing environments for future work.

With the scope now well defined, the question of how to represent the environmental components can be addressed.

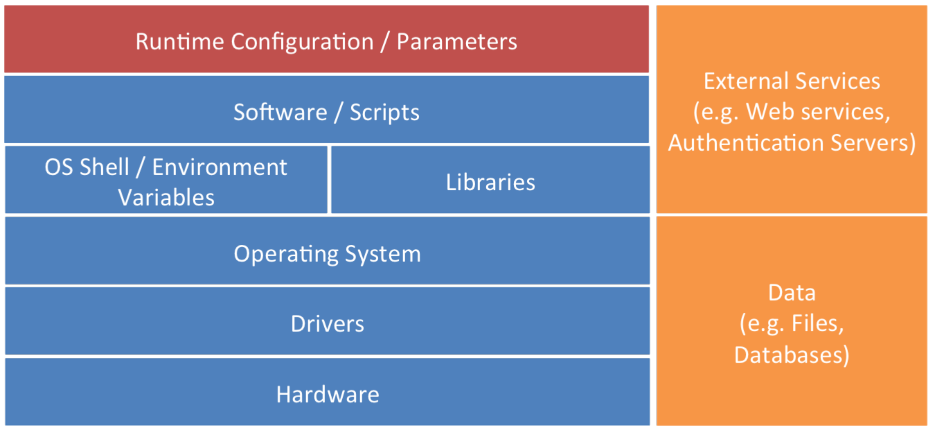

Runtime Configuration / Parameters
Software / Scripts
OS Shell / Environment Variables
Libraries
Operating System
Drivers
Hardware
External Services (e.g. Web services, Authentication Servers)
Data (e.g. Files, Databases)

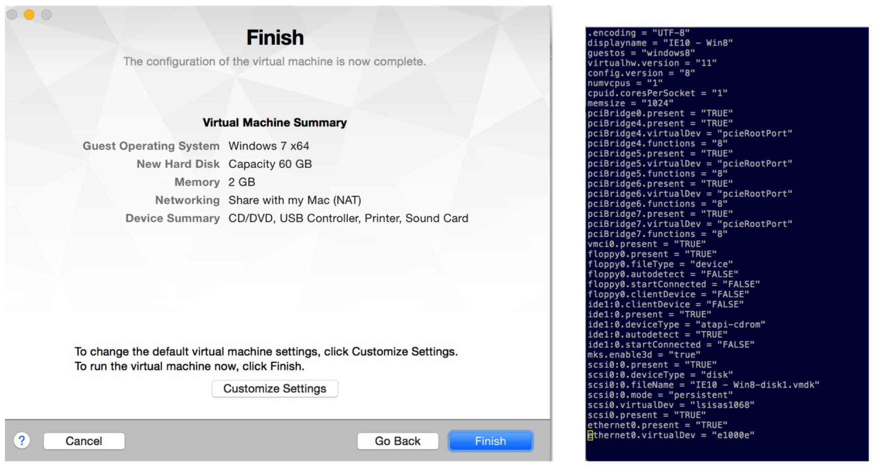
Fig. 1: Defining the boundary of a Computational Environment

4 Design

There are two general ways to approach this modeling task: abstract-to-concrete and concrete-to-abstract. In the former, a environmental component is recursively broken down into its constituent components until the desired granularity is reached. For instance, introductory computer science textbooks traditionally state that a computer (i.e. the hardware) consists of processors, memory, I/O devices, and network interfaces. I/O devices can be categorized as disk drives, visualization technologies, etc. The second way to approach the modeling task is to begin from concrete data, such as from tools that collect or utilize information about a computational environment, and generalize from that to more abstract concepts. We have chosen the later approach for this effort, using the tools VMware Player and Docker as our data sources, and sanity-checking our results using the abstract-to-concrete viewpoint and the competency questions presented above.

VMware Player is a hardware virtualization tool available free for non commercial use. It enables many aspects of a computational environment, including the operating system, libraries, environment variables, software, and local files, to be configured and saved as a virtual machine that can then be downloaded and run on any computer with an x86 architecture. Creating a new virtual machine involves indicating which hardware resources of the host computer system should be made available to the virtual machine, and then installing and configuring the guest operating system along with any other desired software. A screenshot of the dialog for provisioning a new virtual machine and a snippet of the resulting configuration file are shown in Figure 2.

Docker is a free open source application that has become very popular among scientists for reproducing computational analyses. In contrast to the hardware virtualization provided by VMware Player, Docker provides operating system virtualization by using abstractions built into operating system kernels such as



Finish

The configuration of the virtual machine is now complete.

Virtual Machine Summary

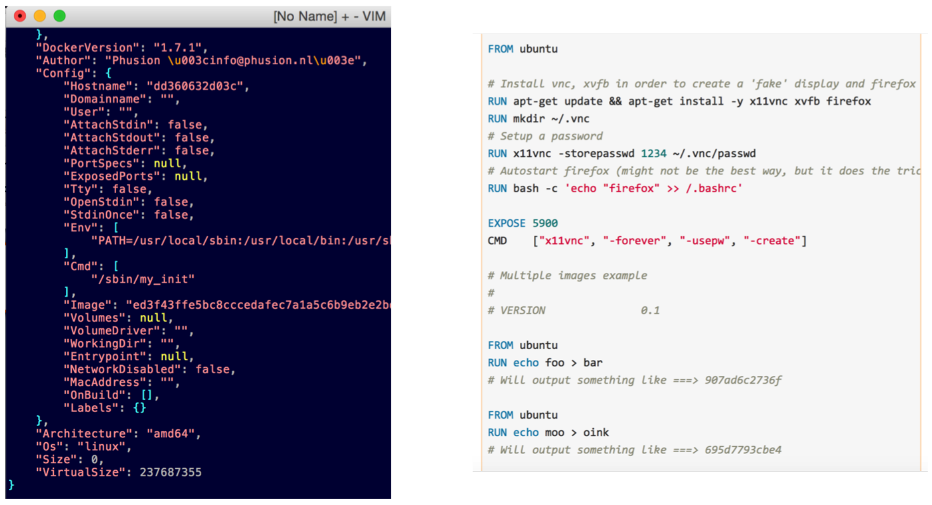
Guest Operating System	Windows 7 x64
New Hard Disk	Capacity 60 GB
Memory	2 GB
Networking	Share with my Mac (NAT)
Device Summary	CD/DVD, USB Controller, Printer, Sound Card

To change the default virtual machine settings, click Customize Settings.
To run the virtual machine now, click Finish.

```
.encoding = "UTF-8"
displayname = "IE10 - Win8"
guestos = "windows8"
virtualhw.version = "11"
config.version = "8"
numvcpus = "1"
cpuid.coresPerSocket = "1"
memsize = "1024"
pciBridge0.present = "TRUE"
pciBridge4.present = "TRUE"
pciBridge4.virtualDev = "pcieRootPort"
pciBridge4.functions = "8"
pciBridge5.present = "TRUE"
pciBridge5.virtualDev = "pcieRootPort"
pciBridge5.functions = "8"
pciBridge6.present = "TRUE"
pciBridge6.virtualDev = "pcieRootPort"
pciBridge6.functions = "8"
pciBridge7.present = "TRUE"
pciBridge7.virtualDev = "pcieRootPort"
pciBridge7.functions = "8"
vmci0.present = "TRUE"
floppy0.present = "TRUE"
floppy0.fileType = "device"
floppy0.autodetect = "FALSE"
floppy0.startConnected = "FALSE"
floppy0.clientDevice = "FALSE"
ide1:0.clientDevice = "FALSE"
ide1:0.present = "TRUE"
ide1:0.deviceType = "atapi-cdrom"
ide1:0.autodetect = "TRUE"
ide1:0.startConnected = "FALSE"
mks.enable3d = "true"
scsi0:0.present = "TRUE"
scsi0:0.deviceType = "disk"
scsi0:0.fileName = "IE10 - Win8-disk1.vmdk"
scsi0:0.mode = "persistent"
scsi0.virtualDev = "lsisas1068"
scsi0.present = "TRUE"
ethernet0.present = "TRUE"
ethernet0.virtualDev = "e1000e"
```

Fig. 2: VMWare configuration data

the linux container system (LXC). Starting from a base OS layer, Docker allows the user to specify the series of steps necessary to configure the environment and run a program. It is currently restricted to Linux applications, but as of the time of this writing Microsoft is said to be working on Windows capability. Figure 3 shows a Docker description file along with a snippet of a script describing the configuration and commands to execute.



```
    },
    "DockerVersion": "1.7.1",
    "Author": "Phusion \u003cinfo@phusion.nl\u003e",
    "Config": {
        "Hostname": "dd360632d03c",
        "Domainname": "",
        "User": "",
        "AttachStdin": false,
        "AttachStdout": false,
        "AttachStderr": false,
        "PortSpecs": null,
        "ExposedPorts": null,
        "Tty": false,
        "OpenStdin": false,
        "StdinOnce": false,
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/s
        ],
        "Cmd": [
            "/sbin/my_init"
        ],
        "Image": "ed3f43ffe5bc8cccedafec7a1a5c6b9eb2e2b
        "Volumes": null,
        "VolumeDriver": "",
        "WorkingDir": "",
        "Entrypoint": null,
        "NetworkDisabled": false,
        "MacAddress": "",
        "OnBuild": [],
        "Labels": {}
    },
    "Architecture": "amd64",
    "Os": "linux",
    "Size": 0,
    "VirtualSize": 237687355
}
```

```
FROM ubuntu

# Install vnc, xvfb in order to create a 'fake' display and firefox
RUN apt-get update && apt-get install -y x11vnc xvfb firefox
RUN mkdir ~/.vnc
# Setup a password
RUN x11vnc -storepasswd 1234 ~/.vnc/passwd
# Autostart firefox (might not be the best way, but it does the tric
RUN bash -c 'echo "firefox" >> /.bashrc'

EXPOSE 5900
CMD    ["x11vnc", "-forever", "-usepw", "-create"]

# Multiple images example
#
# VERSION               0.1

FROM ubuntu
RUN echo foo > bar
# Will output something like ===> 907ad6c2736f

FROM ubuntu
RUN echo moo > oink
# Will output something like ===> 695d7793cbe4
```

Fig. 3: Docker configuration data

Based on these computational environment preservation tools, we developed the following set of data items relevant to the definition of a computational environment. It is important to note that not every item need be defined in all cases. In fact, if a particular item is not relevant to the successful duplication of a computational analysis, it might be preferable to omit it rather than overly constraining those attempting to replicate the work.

- Hardware
  - Processor
    - Architecture
    - Number of cores
    - Frequency
  - Memory
    - Amount
  - Disk space
    - Amount
  - I/O Device
  - Network Interfaces
    - Virtual MAC address
- Operating System
  - Kernel
    - Name
    - Version
  - Distribution
    - Name
    - Version
- OS Shell/Environment
  - Environment variable
    - Name
    - Value
- Software
  - Name
  - Version
  - Location[3],[4]

5 Formalization

Based on the analysis described in the previous section, we developed the Computational Environment ODP shown graphically in Figure 4. Classes are depicted as yellow rectangles with solid borders, and properties are shown as labeled arrows from domain to range. Literal datatypes are shown as rounded blue rectangles. Subclass relationships are depicted as white-headed arrows. The purple rectangles with dashed borders indicate controlled vocabularies, which are connected to their corresponding class by a dashed line. These will be discussed in more detail in the following section.

[3] Examples of instances of 'location' include mount point, URL, etc.
[4] We did not find a need to treat libraries or device drivers differently from standalone software.

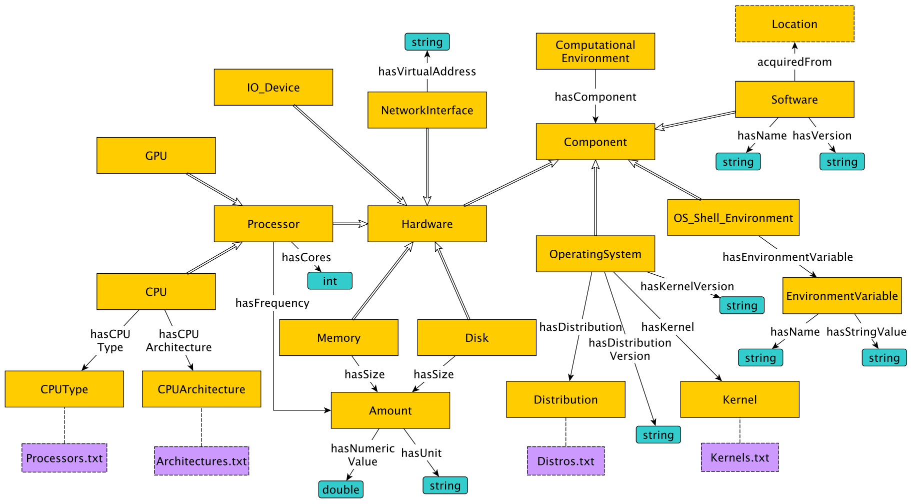


Fig. 4: The Computational Environment ODP

The axiomization of this pattern is relatively straightforward. The controlled vocabularies (i.e. list of acceptable values for a particular type of entity) are represented as named instances of the appropriate class, with an accompanying OWL oneOf axiom to restrict the associated property to one of the allowable instances, as shown below for the case of ProcessorArchitecture.

```
<ClassAssertion>
    <Class IRI="#CPUArchitecture"/>
    <NamedIndividual IRI="#AMD64"/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI="#CPUArchitecture"/>
    <NamedIndividual IRI="#PowerPC"/>
</ClassAssertion>

...

<ClassAssertion>
    <Class IRI="#CPUArchitecture"/>
    <NamedIndividual IRI="#X86"/>
</ClassAssertion>

<EquivalentClasses>
    <Class IRI="#CPUArchitecture"/>
    <ObjectOneOf>
        <NamedIndividual IRI="#AMD64"/>
```

```
            <NamedIndividual IRI="#PowerPC"/>
                          ...
            <NamedIndividual IRI="#X86"/>
        </ObjectOneOf>
    </EquivalentClasses>
```

Both domain and range are specified for hasCores, hasDistributionVersion and hasKernelVersion, but only the range is constrained for all other datatype properties. Regarding object properties, we have scoped the domain and ranges for hasCPUArchitecture, hasEnvironmentVariable, hasKernel and hasDistribution, but scoped ranges only for hasSize, hasFrequency, and hasComponent, and neither domain nor range for acquiredFrom.

The listing below shows instance data for the pattern based on running Wireshark on the first author's laptop in order to analyze network traffic. IP addresses have been obscured. While the ODP is commented to describe the classes and properties, this example is included in order to more fully convey the meaning and intended use of the entities within the pattern.

```
@base <http://dase.cs.wright.edu/ontologies/ComputationalEnvironment#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>

:exComputationalEnv a :ComputationalEnvironment .

:exMemory a :Memory ;
            :hasSize :exMemSize .

:exMemSize a :Amount ;
             :hasNumericValue "16"^^xsd:double ;
             :hasUnit "GB" .

:exDisk a :Disk ;
          :hasSize :exDiskSize .

:exDiskSize a :Amount ;
              :hasNumericValue "500"^^xsd:double ;
              :hasUnit "GB" .

:exProcessor a :Processor ;
               :hasCPUType :Intel_Core_i5 ;
               :hasCPUArchitecture :X86 ;
               :hasCores "2"^^xsd:int ;
               :hasFrequency :exFrequency .

:exFrequency a :Amount ;
               :hasNumericValue "2.6"^^xsd:double ;
               :hasUnit "GHz" .

:exOperatingSystem a :OperatingSystem ;
                     :hasKernel :macOS ;
                     :hasKernelVersion "10.12.5" .

:exNetInterface a :NetworkInterface ;
                  :hasVirtualAddress "xxx.xxx.xxx.xxx" .

:exKeyboard a :IO_Device ;

:exMonitor a :IO_Device ;

:exSoftware a :Software ;
              :hasName "wireshark" ;
              :hasVersion "2.2.7" ;
              :acquiredFrom exLocation .

:exLocation :hasName "https://1.na.dl.wireshark.org/osx/
                  Wireshark%202.2.7%20Intel%2064.dmg" .

:exShellEnv a :OS_Shell_Environment ;
                :hasEnvironmentVariable :exEnvVariable .

:exEnvVariable a :EnvironmentVariable ;
                 :hasName "SSH_CONNECTION" ;
                 :hasStringValue "xxx:xxx:xxx:xxx 60123 yyy:yyy:yyy:yyy" .

:exComputationalEnv :hasComponent :exMemory ,
                                  :exDisk ,
                                  :exProcessor ,
                                  :exOperatingSystem ,
                                  :exNetInterface ,
                                  :exKeyboard ,
                                  :exMonitor ,
                                  :exSoftware ,
                                  :exShellEnv .
```

Our intention is that related ontologies, such as those used to capture a computational analysis workflow, could refer to the computational environment ODP presented here. Returning to the competency questions outlined at the beginning of this paper, we see that this ODP specifies the underlying hardware, operating system, and software that are fundamental to replicating a computational analysis. Because the pattern includes the software involved in an analysis, critical and/or frequently used software can be identified. Additionally, instance data in this pattern could be queried in response to any errors discovered later in a piece of software and any related analyses could be redone. Meanwhile, modeling the hardware aspects of the environment allow one to determine when two studies are directly comparable from a computational perspective.

The pattern is intended to be flexible enough to support modeling of a computational environment under various conditions. For instance, sometimes it is helpful for replicating the performance of parallelized algorithms if the single core performance of the original system is known. This can be captured by modeling the environment of a multi-core system with numCores equal to one (and the rest of the instance data updated accordingly).

The pattern is available on the Ontology Design Patterns website at `http://ontologydesignpatterns.org/wiki/Submissions:ComputationalEnvironment`. In addition, pattern development is hosted at Github[5] where it is kept up to date via the automated process described in the next section. Community comments and contributions are always welcome.

The software class in this model is roughly equivalent to the concept of a software bundle in [5], and it would be possible to align the two models at this point if more granular modeling of the software aspects of the computational environment were desired.

6 Maintenance

Reproducing a computational analysis is a relatively exacting endeavor, and this has some implications for the Computational Environment ODP. One of these is that this ODP is more concrete than most, down to the level of controlled vocabularies for some of the entities most critical for replicating certain types of computational analyses, such as those that involve assessing computation time. Of course, controlled vocabularies are a double-edged sword: they are useful for constraining possible values, but they require regular maintenance to make sure that they remain up to date. This is particularly true in the case of computational environments, where the pace of technology evolution can quickly make controlled vocabularies for entities such as processors and operating system distributions obsolete.

In order to leverage the benefits of controlled vocabularies while mitigating the maintenance effort, we have written a script that automatically runs each month to update the controlled vocabularies in the ODP. This is done by using the data on various Wikipedia pages. Wikipedia is manually curated by thousands of people on a consistent basis, so it is an ideal source of data in this case. When the script runs, it accesses the Wikipedia pages shown in Table 1 and uses basic text processing to extract the list of processor architectures, processors, operating system kernels, and operating system distributions. It then compares these lists to those present in the current version of the ODP. If there are new items in the list, the pattern is updated and automatically pushed to the GitHub repository. In order to avoid invalidating any previously valid linked data published according to this ODP, the script does not remove any values

[5] `https://github.com/mcheatham/computationalEnvironmentODP`

from the controlled vocabularies in the pattern, even if they are removed from Wikipedia.[6]

We note that there is some difference between the technical definitions of the terms 'kernel' and 'distribution' and the way they are often used in practice. For example, a computer may be running Ubuntu, in which case the the kernel is Linux, and the distribution is Ubuntu. In the case of Mac OS, the kernel is technically XNU (a variant of Mach), the distribution is macOS, and the distribution version may be something like 10.12 (Sierra). However, it is relatively common to refer to macOS as a kernel rather than a distribution. The same issue sometimes comes up with CPUs and CPU architectures, albeit less frequently. In situations like these, we employ the same categorization as that used on Wikipedia.

A copy of the update script is stored in the same GitHub repository as the ontology design pattern. The script automatically executes at 3:00 am EST on the first of each month.

Entity	Wikipedia Page(s)
CPU Architecture	List of CPU architectures
CPU	List of microprocessors
OS Kernel	List of operating systems
OS Distribution	List of operating systems & List of Linux distributions

Table 1: Wikipedia pages for population of the controlled vocabularies within the Computational Environment ODP

7 Conclusions and Future Work

There has been significant work on developing semantic models to enhance the reproducibility of computational analyses; however, most existing models begin at the level of software or services. In this work we model a computational environment down to the "bare metal" of the computer on which an analysis is performed. This is particularly important for reproducing results involving things such as computation time (or that may timeout or cause memory thrashing). The pattern makes extensive use of controlled vocabularies in order to achieve a high level of comparability between instances, while using automated scripts to extract data from Wikipedia in order to limit the maintenance effort required to keep these controlled vocabularies current.

We plan to utilize this model to capture the details of computational environments mentioned in academic research papers, as well as those inherent in virtual machine description files. Our eventual goal is to develop an application

[6] In the case of misspellings, mistakes, or malicious Wikipedia edits, values can be removed manually.

capable of searching for an appropriate virtual machine with which to replicate the computational analysis described in a given paper.

There are some limitations to this pattern that we hope to address in our future work on this topic. The pattern currently does not model grid-based computational environments. In addition, the pattern captures only the specific environment in which a computational analysis occurred, not any in which it could be repeated without impacting the results.

Acknowledgments. The first, third and fourth author acknowledge partial support by the National Science Foundation under award 1440202 *EarthCube Building Blocks: Collaborative Proposal: GeoLink – Leveraging Semantics and Linked Data for Data Sharing and Discovery in the Geosciences.* The first, second and fourth authors acknowledge partial support by the National Science Foundation under award PHY-1247316 *DASPOS: Data and Software Preservation for Open Science*

References

1. Corcho, O., Garijo Verdejo, D., Belhajjame, K., Zhao, J., Missier, P., Newman, D., Palma, R., Bechhofer, S., García Cuesta, E., Gomez-Perez, J.M., et al.: Workflow-centric research objects: First class citizens in scholarly discourse. (2012)
2. Dappert, A., Peyrard, S., Delve, J., Chou, C.: Describing digital object environments in premis. In: 9th International Conference on Preservation of Digital Objects (iPRES2012). pp. 69–76 (2012)
3. Goble, C.A., De Roure, D.C.: myexperiment: social networking for workflow-using e-scientists. In: Proceedings of the 2nd workshop on Workflows in support of large-scale science. pp. 1–2. ACM (2007)
4. Mayer, R., Rauber, A., Neumann, M.A., Thomson, J., Antunes, G.: Preserving scientific processes from design to publications. In: International Conference on Theory and Practice of Digital Libraries. pp. 113–124. Springer (2012)
5. Taelman, R., Van Herwegen, J., Capadisli, S., Verborgh, R.: Reproducible software experiments through semantic configuration (May 2017), https://linkedsoftwaredependencies.org/articles/reproducibility/, accessed: 2017-06-22