Provenance-Based Reproducibility in the Semantic Web

Luc Moreau

School of Electronics and Computer Science,
University of Southampton, Southampton, SO17 1BJ UK

Abstract

Reproducibility is a crucial property of data since it allows users to understand and verify how data was derived, and therefore allows them to put their trust in such data. Reproducibility is essential for science, because the reproducibility of experimental results is a tenet of the scientific method, but reproducibility is also beneficial in many other fields, including automated decision making, visualization, and automated data feeds. To achieve the vision of reproducibility, the workflow-based community has strongly advocated the use of provenance as an underpinning mechanism for reproducibility, since a rich representation of provenance allows steps to be reproduced and all intermediary and final results checked and validated. Concurrently, multiple ontology-based representations of provenance have been devised, to be able to describe past computations, uniformly across a variety of technologies. However, such Semantic Web representations of provenance *do not have any formal link* with execution. Even assuming a faithful and non-malicious environment, *how can we claim that an ontology-based representation of provenance enables reproducibility, since it has not been given any execution semantics*, and therefore has no formal way of expressing the reproduction of computations? This is the problem that this paper tackles by defining a denotational semantics for the Open Provenance Model, which is referred to as the *reproducibility semantics*. This semantics is used to implement a *reproducibility service*, leveraging multiple Semantic Web technologies, and offering a variety of reproducibility approaches, found in the literature. The reproducibility capabilities are evaluated by means of a series of empirical experiments aimed at highlighting novel functionality. In particular, we demonstrate the ability to reproduce computations involving multiple technologies, as is commonly found on the Web.

Keywords: provenance, reproducibility, denotational semantics, primitive environment

1. Introduction

The envisaged applications of science and technology are far reaching, from Government personalised services for the citizen[1] to personalised medicine[2], from understanding climate change, to its tackling by smart energy usage [1]. Technology is revolutionising the way scientists undertake science, as illustrated by Grid Computing [2], e-Science [3], or the Fourth Paradigm [4]. While science is becoming computation and data intensive, the fundamental tenet of the scientific method remains unchanged: experimental results need to be *reproducible* [5].

The fundamental principle of reproducibility is particularly important given the importance of science in our life. This importance is illustrated by the scientific advice on climate change that has helped shape governmental policies. The mail controversy of the Climatic Research Unit "Climate-Gate"[3] highlights how global the impact of science has become. As a result, calls for more transparency in climate science have been issued; specifically, a parliament committee called for the release of both the raw data and the computer code used in research[4]. This is in no way limited to climate science: in social science, evidence-based policy refers to public policy that is informed by rigorously established objective evidence [6]. Similar calls exist for transparency in clinical trial results used in drug approval[5]. It is no surprise that novel initiatives [7] and recommendations [8] that encourage the publication of data sets are emerging. These are steps in the right direction, but by themselves, they do not ensure reproducibility of the scientific process.

Provenance refers to the source or origin of something. In a computational setting, provenance of a data item is an explicit representation of the processes that led to that data item [9]. The workflow-based scientific community has strongly advocated the use of provenance as an underpinning mechanism for reproducibility: "*reproducibility requires rich provenance information, so that researchers can repeat techniques and analysis methods to obtain scientifically similar results . . . In order to support reproducibility, workflow management systems must capture and generate provenance information as a critical part of the workflow-generated data.*"[10]. The strong belief that provenance can support reproducibility is also echoed by the

---

[1] http://www.telegraph.co.uk/technology/news/7484600/Every-citizen-to-have-personal-webpage.html
[2] http://www.futuremedicine.com/doi/abs/10.2217/17410541.5.1.55
[3] http://en.wikipedia.org/wiki/Climatic_Research_Unit_email_controversy

[4] http://blogs.nature.com/news/thegreatbeyond/2010/03/parliament_committee_calls_for.html
[5] http://www.sciencebasedmedicine.org/?p=215

provenance community, with more than twenty papers cited by a recent survey on provenance [11] mentioning reproducibility in their abstract.

To relate reproducibility to provenance, various approaches have emerged in which provenance is defined in the context of specific execution semantics. Why-provenance [12] and lineage [13] identify source tuples that "contributed" to a result returned by a database query (within the relational and xml models). By applying the same query to such source tuples, the result could be reproduced. Souilah *et al.* [14] define a denotation of provenance in the context of a π-calculus variant: this denotation can replay the sending and receiving of data values across processes. Cheney *et al.* [15] define a notion of traces faithful to a program if they record enough information to recompute the program when the inputs change; this definition is proposed in the context of the Nested Relational Calculus. In workflow systems, the Virtual Data Systems [16] views provenance as consisting of two components: all the aspects of the procedure or workflow for creating a data object (referred to as prospective provenance), and the information about the runtime environment in which these procedures were executed (retrospective provenance), the combination of both offering reproducibility. By means of a translation from provenance to its workflow language [17], Taverna is also able to reproduce past results. All of these approaches have in common that reproducibility is directly linked to a specific execution semantics, whether from a given database engine, workflow system, or calculus. Unfortunately, this is crucially problematic when provenance is to be made available on the Web, where multiple execution technologies are inevitably involved.

The Open Provenance Vision [11] is a call for architectural guidelines to support provenance inter-operability on the Web, by means of open models, open serialization formats and open APIs. As envisaged in the Open Provenance Vision, the provenance from individual systems or components can be expressed, connected in a coherent fashion, and queried seamlessly. Several models for provenance have emerged to tackle this vision, including Provenir [18, 19], the Provenance Vocabulary [20], PASOA [21], and the Open Provenance Model [22]. All rely on Semantic Web definitions, consisting of ontologies, vocabularies or abstract models. Assuming that provenance based on these models was generated faithfully and non-maliciously, *how can we claim provenance enables reproducibility, given that there is no execution semantics attached to these models?* This is precisely the problem that this paper tackles, offering a definition of reproducibility for the Open Provenance Model (OPM) [22].

To address this problem, this paper offers the following contributions:

1. Adopting a Semantic Web perspective to the problem, we extend the Open Provenance Model with minimum executional information and assumptions related to execution. In particular, we introduce the class of primitive procedures and the notion of primitive environment that maps such primitive procedures to something that can be executed. OPM processes are themselves caused by primitive procedure invocations.

2. We present the reproducibility semantics, a denotational semantics for OPM graphs. This mathematical (and therefore technology independent) definition formulates how an OPM graph can be seen as a mathematical function, taking some inputs and a primitive environment, and resulting in another OPM graph. This semantics is novel because it tackles a substantial subset of OPM, and in particular, its notion of account. With this semantics, we also identify a class of OPM graphs that are reproducible, and at the same time recognize that not all OPM graphs are by default reproducible. This mathematical formulation also allows us to specify variants of reproducibility, which help us provide formal groundings for other reproducibility proposals found in the literature.

3. We propose a Semantic Web based architecture for a *reproducibility service*, which can take OPM graphs and check their reproducibility. This architecture, which relies on an extended OWL ontology for OPM, SWRL rules, and a set of SPARQL queries, is intended to act as a reference for the reproducibility service.

4. We provide an evaluation of the approach by demonstrating: *(i)* how the reproducibility service is capable of reproducing the results of the first provenance challenge [23]; *(ii)* how the reproducibility service can easily be customized to invoke multiple execution technologies, such as command line and web services, simply by changing its primitive environment; *(iii)* how it can be used with different inputs.

There is no strong consensus on what reproducibility means, and how it can be achieved. Hence, we start by surveying related work, by providing several definitions of reproducibility, and explain how this work compares against them (Section 3). We then introduce the reproducibility semantics in Section 4, first considering account-less OPM graphs, and then multi-account OPM graphs. Next, we study the idea of a reproducibility service in the context of the Semantic Web, overview its architecture, and discuss the Semantic Web techniques that we leveraged for its implementation (Section 5). We then undertake a range of empirical evaluations aiming to demonstrate the capability of the reproducibility service, and its suitability for deployment in a multi-technology environment such as the Web (Section 6). This is followed by a discussion of the approach (Section 7) before we conclude the paper and summarize possible future work (Section 8). Beforehand, we summarize the OPM terminology.

2. OPM Terminology in One Paragraph and Figure

We assume that the reader is familiar with the OPM specification [22]; a tutorial on OPM is also available from `openprovenance.org/tutorial`. The following example acts as a reminder for the OPM terminology. Figure 1 illustrates an OPM graph describing the evaluation of a numeric expression $(10 + 20) \times 30/9$ resulting in value 100. Ovals represent artifacts and are here associated with numeric values; black rectangles denote processes. Plain edges represent data derivations

(referred to as was-derived-from dependencies); dotted edges represent the dependencies between processes and artifacts, denoting the consumption of the latter by the former (used edges) or the generation of the latter by the former (was-generated-by edges); they are annotated by their roles in bracket. Gray "post-it" rectangles are annotations.

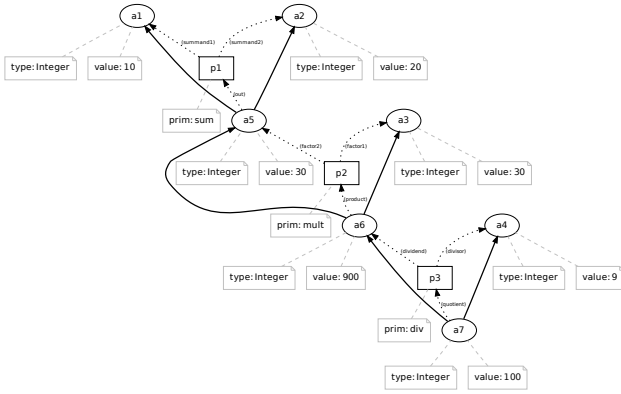Figure 1: OPM graph for Numeric Expression

3. Related Work

In this section, we first review several definitions of reproducibility. We then discuss work on reproducibility that is not provenance specific, before focusing on provenance based approaches. Finally, given that our work consists of a novel formalization of OPM, we review extant efforts in that field.

3.1. What is reproducibility?

There is no strong consensus on what reproducibility means in the context of computational science or computer-based systems. Wikipedia defines reproducibility generally as follows: "*Reproducibility* is one of the main principles of the scientific method, and refers to the ability of a test or experiment to be accurately reproduced, or replicated, by someone else working independently to see if the reproduced experiments gives similar results to those originally reported". Wikipedia further contrasts reproducibility from *repeatability*, which measures the success rate in successive experiments, possibly conducted by the same experimenters. Reproducibility relates to the agreement of test results with different operators, test apparatus, and laboratory locations.

In computer systems, experiments are encoded as programs or workflows [10] that, like recipes, describe the various steps of execution, and can be executed time and time again. In computer systems, however, extensive logs of past activities, which we will refer to as provenance, can provide an accurate description of what occurred in the past, and can be used to reproduce experiments: the difference is that reproduction can be based on the logs, rather than the recipe.

Bechhofer *et al.* [24] see the need for a framework that facilitates the reuse and exchange of digital knowledge. They put forward the idea of Research Objects as containers for a principled aggregation of resources, produced and consumed by common services and shareable between scientists. In this context, they distinguish the following terms:

- *Repeatability*: relies on sufficient information for the original researchers or others to be able to repeat the study. This may involve access to data or execution of services.

- *Reproducibility* of a result consists of starting with the same materials and methods and see if a prior result can be confirmed. It is a special case of repeatability, since it contains complete information such that a final or intermediate result can be verified.

- *Replayability* allows the investigator to "go back and see what happened". It does not necessarily involve execution or enactment of processes and services. It places a requirement on provenance of data.

From the above definitions, it is not entirely clear whether Bechhofer's repeatability and reproducibility draw on the original recipe or provenance of a past execution, or a combination of both. On the other hand, replayability seems to rely explicitly on provenance.

In their classification of provenance requirements, Miles *et al.* [25] identify a use case (Use Case 17) that distinguishes *reenactment*, i.e. performing the same experiment, but using contemporary data and services, from *repetition*, which means performing the same experiment with the same data and services as before, e.g. to test that the results can be reproduced. Whilst framed in the context of provenance, reenactment seems to apply equally to provenance and workflows.

So, reproducibility is a multi-dimensional problem, where several issues need to be taken into consideration: *(i) which scripts?*: is this workflow or provenance based reproducibility? *(ii) which inputs?*: is the experiment reproduced with the same inputs or others? (by inputs, we include not only experimental data, but also parameters) *(iii) which primitives?*: are the original primitives or services invoked? *(iv) which results?*: are intermediary and final results comparable to the original ones? In this paper, we fix the first dimension, focusing on provenance-based reproducibility, while considering all other dimensions of the problem.

3.2. Reproducibility without Provenance

Reproducibility has initially been researched without taking provenance into consideration. We review some salient outcomes, before focusing on provenance-based reproducibility.

Claerbout pioneered the concept of *really reproducible research*, "*An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.*" [26].

3

This pioneering approach has led to the more recent Re-Doc [27], a system for reproducing scientific computations in electronic documents. It consists of three components, make-files, make rules and naming conventions. Such an environment is akin to a workflow system, where the workflow script is a makefile, which can be executed over any input. It however does not describe a past execution and how a past result was achieved. But this shows that under the term "reproducibility", one can find two very different understandings: regenerate a result by applying a recipe on arbitrary inputs vs reproduce all the steps found in an evidence of a past execution.

In forensic investigations, a key factor is reproducibility, defined as the ability to achieve a consistent level of quality throughout the investigative process, no matter how many times it is repeated under the same conditions [28]. Pan and Batten [28] propose a model based on read and write operations, and associated timestamps, allowing them to be ordered in a linear time flow. This model is targeted to forensic investigations and is not aimed to generic computations. An alternative approach, which is provenance-based, is proposed by Levine and Liberatore [29] and discussed in Section 3.3.

Stodden [30] defines reproducibility as the ability of others to recreate and verify computational results, given appropriate software and computing resources. Stodden investigates the legal impediment to scientific reproducibility, and proposes a reproducible research standard. We do not denigrate the importance of legal issues, but this article only focuses on the technical aspects of reproducibility.

3.3. Provenance-based Reproducibility

Davidson and Freire explain that a key benefit for maintaining provenance of computational results is reproducibility: a detailed record of the steps followed to produce a result allows others to reproduce and validate these results [31]. Specifically, with an explicit representation of provenance, so-called provenance queries can be expressed to identify all the data objects and the sequence of steps that have been used to produce a result [32].

Levine and Liberatore [29] seek to improve the reproducibility and comparison of digital forensic evidence. They propose a simple canonical description of digital evidence provenance that explicitly states the set of tools and transformations that led from acquired raw data to the resulting product. This provenance representation allows for the comparison and the reproduction of results. Inspired by the principle of N-version programming, their approach allows multiple tools/libraries to be used to reproduce a result, hereby increasing the confidence that can be put in investigations.

Silva et al. [33] argue about the importance of reproducibility in visualization. Leveraging the Vistrails systems, they exploit the provenance it generates to ensure that users will be able to reproduce the visualizations and let them easily navigate through the space of visualization pipelines created for a given exploration task. Vistrails adopts an action-based provenance model, intended to help reproducibility. We conjecture that it is based on replaying such actions, but it fails to include an explicit description of how it can be achieved outside the context of Vistrails itself. Koop et al. [34] discuss the problem of managing upgrades of tools and libraries, while still being able to run a previous computation in a new environment; this work is specifically focused on the Vistrails system and methods necessary to automatically update workflows and provenance.

Mesirov [5] proposes a Reproducible Research System (RRS), consisting of two components. The first is the Reproducible Research Environment (RRE), which provides computational tools together with the ability to automatically track the provenance of data, analyses, and results and to package them (or pointers to persistent versions of them) for redistribution. The second element is a Reproducible Research Publisher (RRP), which is a document-preparation system, such as standard word-processing software, that provides an easy link to the RRE.

Cheney et al. [15] provide an operational semantics for the Nested Relational Calculus (NRC) that generate *traces*, intented to capture the execution history of a query. Such a notion of trace is a representation of provenance which is directly inspired by the syntax of NRC constructs. They define properties of such traces and NRC programs, such as consistency and fidelity. A trace is said to be consistent to a program, if it is an explanation of what happened when the program was evaluated. Fidelity is the property that holds when the trace records enough information to recompute a program when the inputs change.

The provenance approaches that have been reviewed in this section are all grounded in a specific execution environment. Hence, because of their dependencies to a specific technology or execution semantics, they fail to meet a key requirement of the Open Provenance Vision [11] for provenance on the Web. Alternative provenance definitions are ontology-based and not specific to an execution technology: Provenir [18, 19], the Provenance Vocabulary [20], PASOA [21], and the Open Provenance Model [22]. None of them, however, has been given a semantics that is suitable for reproducibility purpose. This is a shortcoming that we address for OPM in this paper. This work complements other endeavours aiming to provide a formal underpinning to OPM, which we survey in the following section.

3.4. Formal definitions of OPM

Moreau et al. [35] provide a set-theoretic definition of OPM, as well as an illustration of how an abstract machine execution can generate OPM-based provenance traces. The mapping from execution to OPM graphs is not formally characterized, and no attempt is made to provide a converse mapping. In the subsequent version, Kwasnikowska et al. [36] provide a temporal interpretation of OPM graphs, defined as the set of temporal inequalities implies by its edges. OPM inferences combined with graph patterns are shown to be sound and complete with respect to inferences that can be made over graph interpretations. This temporal interpretation does not provide an understanding of OPM from an execution perspective either.

Cheney [37] investigates the use of structural causal models as a semantics for provenance graphs, and relates some OPM concepts to notions of actual cause and explanation proposed by Halpern and Pearl [38, 39]. At some level, the semantics

we propose here bears some similarity to Cheney's since it is also a denotation of a provenance graph, i.e., it sees a graph as a mathematical function, resulting in a new provenance graph. In practice, they differ for several reasons: *(i)* our semantics conforms to OPM v1.1 and in particular handles OPM accounts, whereas Cheney's is account-less and regards single-step derivation edges as inferrable, when they can only be asserted in OPM; *(ii)* our semantics builds on the Semantic Web philosophy, where globally unique names are meant to capture well understood concepts (in this case, primitives), which are explicitly captured within a notion of primitive environment; *(iii)* Cheney's semantics attempts the more ambitious goal of providing a global approximation (using the predictive nature of causal models) for the program being executed (without having its explicit code), so that its behaviour can be repeated for any arbitrary input; *(iv)* our semantics allows us to explore and characterize variants of reproducibility.

Missier and Goble [17] address the question of whether, for any OPM graph, there exists a plausible workflow in the Taverna workflow language, which could have generated the graph. To this end, they identify the extra information that should be captured as part of an OPM graph so that the mapping from OPM to a workflow representation can be derived. Whilst this work focuses on some specificities of the Taverna workflow language, such as the implicit iterator semantics, it is similar in spirit to ours, since it derives an executable semantics for OPM. In their case, it is obtained by composing their translation to the Taverna semantics [40]. It however does not tackle OPM in full, ignoring accounts, and does not define reproducibility itself.

Various ontological definitions of the Open Provenance Model have emerged. The OPM toolbox[6] supports bidirectional conversions between XML and RDF serializations, respectively defined according to an XML schema and an OWL ontology [41]. This ontology was inspired by the OWL definition compatible with the OPM implementation of Tupelo[7] [42]. During the third provenance challenge, the Tetherless team also defined an OPM OWL ontology[8].

As part of the W3C Provenance Incubator activity [43], mappings of multiple provenance ontologies to OPM were defined [44]. These mappings showed that concepts such as processes, artifacts, and agents can be mapped quite naturally between the models. The mappings however do not characterize the computational implications associated with those models, and do establish whether reproducibility is preserved during translation across models.

4. Reproducibility Semantics

In this section, we specify the *reproducibility semantics* for OPM graphs, which we define as the mathematical meaning of an OPM graph, seen as a program and whose execution results

in a new OPM graph. First, we study reproducibility in account-less OPM graphs (Sections 4.2, 4.3 and 4.4) and then in multi-account graphs (Sections 4.5 and 4.6).

4.1. Intuition of the Reproducibility Semantics

Before delving into the technical details of a denotational semantics, we provide some intuition of how we propose to reproduce the execution on an OPM graph. We assume that each process in an OPM graph is annotated with the name of a primitive, and that there is a primitive environment that maps primitive names to actual functions, which take some inputs and produce some outputs. Here, function arguments are not identified by their position in a sequence of arguments but by their roles; likewise, outputs can be multiple and are identified by their role.

The inputs of an OPM graph are all the artifacts for which there is no was-generated-by edge; likewise, its outputs are all the artifacts that are not adjacent to a used edge. Given an acyclic OPM graph, we assume the existence of a function that returns a list of all its processes, sorted by order of execution: by this, we mean that a process in the sorted list does not use any artifact generated by a process that is subsequent in the list.

Reproducibility is achieved by recursively traversing the sorted list of processes, executing each of them in turn. Execution of a process is achieved by invoking its associated primitive function on the values of artifacts (paired with roles) it uses, and results in values (also paired with roles), for which new artifacts are created. The reproducibility semantics constructs a new OPM graph at the same time, describing the re-execution of the graph. The new graph is initialized with the set of input artifacts (with the same value as in the original graphs, or different values, depending on the kind of reproducibility one wants to achieve). Each process execution adds the process and the output artifacts it generated, and all associated edges with roles, where appropriate.

For comparing the original graph and the new graph, but also for book-keeping, a mapping from nodes of the original graph to those of the new graph is constructed. It allows us to check whether all nodes have been mapped, and whether they have the same associated values. Furthermore, in the case of multi-account OPM graphs, the mapping allows us to decide whether we are processing a node that has already been encountered.

So, in summary, the reproducibility semantics takes a set of input artifacts (with their associated values), a primitive environment, and an input graph, and returns a mapping and a new OPM graph, which describes the reexecution of the input graph, and a mapping. We now formalize this semantics.

4.2. Preliminary Definitions

Figure 2 displays the definition of an account-less OPM graph. It consists of a set of nodes and a set of edges. Nodes can be artifacts or processes[9], whereas edges are of four permitted types. Artifacts and processes respectively belong to primitive sets *Artifact* and *Process*. Figure 2 is a stylised version of a

---

---

[9] Like [35], the semantics presented in this paper ignores agents, since their role as catalyst does not directly affect computational reproducibility.

Standard ML definition of this semantics, available for download[10].

$$
\begin{aligned}
\textit{Artifact} &= \text{primitive set of artifacts} \\
\textit{Process} &= \text{primitive set of processes} \\
\textit{Role} &= \text{primitive set of roles} \\
\textit{Value} &= \text{primitive set of values} \\
\textit{PrimitiveName} &= \text{primitive set of primitive names} \\
\textit{Node} &= \mathsf{art\ of\ } \textit{Artifact}\ |\ \mathsf{proc\ of\ } \textit{Process} \\
\textit{Edge} &= \mathsf{used\ of\ } (\textit{Process} \times \textit{Role} \times \textit{Artifact}) \\
&\quad |\ \mathsf{wgb\ of\ } (\textit{Artifact} \times \textit{Role} \times \textit{Process}) \\
&\quad |\ \mathsf{wdf\ of\ } (\textit{Artifact} \times \textit{Artifact}) \\
&\quad |\ \mathsf{wtb\ of\ } (\textit{Process} \times \textit{Process}) \\
\textit{OPMGraph} &= \mathbb{P}(\textit{Node}) \times \mathbb{P}(\textit{Edge}) \\
\textit{AResolver} &= \textit{Artifact} \rightarrow \textit{Value} \\
\textit{PResolver} &= \textit{Process} \rightarrow \textit{PrimitiveName} \\
\textit{FullOPMGraph} &= \textit{OPMGraph} \times \textit{AResolver} \times \textit{PResolver}
\end{aligned}
$$

Figure 2: Account-Less OPM Graph

Artifacts are associated with values (belonging to a primitive set of values), whereas each process is associated with the name of a primitive, whose invocation resulted in this process. The association is defined by the mappings *AResolver* and *PResolver*. A *FullOPMGraph* then refers to an *OPMGraph* accompanied by the artifact and the process resolvers.

The OPM graph of Figure 1 can be formalized by $G^v = (G, \mathcal{V}^a, \mathcal{V}^p)$ as follows:

$G = \langle \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, p_1, p_2, p_3\}, \{\mathsf{used}(p_1, \mathsf{summand1}, a_1), \mathsf{used}(p_1, \mathsf{summand2}, a_2), \mathsf{wgb}(a_5, \mathsf{out}, p_1), \mathsf{used}(p_2, \mathsf{factor1}, a_3), \mathsf{used}(p_2, \mathsf{factor2}, a_5), \mathsf{wgb}(a_6, \mathsf{product}, p_2), \mathsf{used}(p_3, \mathsf{divisor}, a_4), \mathsf{used}(p_3, \mathsf{dividend}, a_6), \mathsf{wgb}(a_7, \mathsf{quotient}, p_3), \mathsf{wdf}(a_5, a_1), \mathsf{wdf}(a_5, a_2), \mathsf{wdf}(a_6, a_3), \mathsf{wdf}(a_6, a_5), \mathsf{wdf}(a_7, a_4), \mathsf{wdf}(a_7, a_6)\}\rangle$.

$\mathcal{V}^a = \{(a_1, 10), (a_2, 20), (a_3, 30), (a_4, 9), (a_5, 30), (a_6, 900), (a_7, 100)\}$

$\mathcal{V}^p = \{(p_1, \mathsf{prim:sum}), (p_2, \mathsf{prim:mult}), (p_3, \mathsf{prim:div})\}$.

The *inputs of an OPM graph* are the artifacts that are not adjacent to a was-generated-by edge, whereas graph *outputs* are the artifacts that not adjacent to a used edge. In Figure 1, a_1, a_2, a_3, a_4 are inputs, while a_7 is an output.

For the purpose of reproducibility, some topological constraints are introduced on OPM graphs. Ways of relaxing these constraints are discussed further in Section 7.

Definition 1 (Reproducibility Graph Constraints).

1. *OPM graphs are supposed to be well-formed as per OPM v1.1 [22]: an artifact can be generated by at most one process, and there exists no cycle formed of edges of type was-derived-from.*
2. *For the purpose of the reproducibility semantics, we assume that an OPM graph is fully acyclic, i.e., no cycle can be formed with any edge.*
3. *We assume that there is no edge was-triggered-by.*
4. *We also assume that inputs and outputs are uniquely identified by a role, for a given process: for any edges* $\mathsf{used}(p, r, a_1), \mathsf{used}(p, r, a_2)$, *then* $a_1 = a_2$; *likewise, for any edges* $\mathsf{wgb}(a_1, r, p), \mathsf{wgb}(a_2, r, p)$, *then* $a_1 = a_2$.

$\square$

Our semantics relies on partial maps, mapping roles to some set (e.g., role-values or role-artifacts). For instance $rv^* \in Role \rightarrow Value$. Given a role $r \in DOM(rv^*)$, then $rv^*(r)$ denotes the value associated with r. For convenience, our notation also allows for such a partial map to be seen as a finite sequence of role-value pairs $\mathbb{P}(RoleValue)$, over which we can perform a *map* operation.

4.3. Reproducibility of an Account-Less OPM Graph

Key to reproducibility is a definition for each of the primitives referred to by processes of an OPM graph. To this end, we introduce the concept of a *primitive environment* associating primitive names with *primitives*, which essentially produce some output values for some input values. In OPM, values (whether input or output of a process) are associated with a role. Hence, primitives take sets of role-value pairs, and produce sets of role-value pairs (cf. Figure 3, line 2).

Furthermore, in OPM, edges of type *was-derived-from* (wdf) need to be asserted and cannot be inferred. The only component that has knowledge of such dependencies is the primitive itself. So we expect a primitive not only to return a set of role-value pairs, but also which *was-derived-from* dependency exists between which output (identified by its role) and which input (similarly identified). Such a pair of roles, an element of *EdgeSpec* (cf. Figure 3, line 4), can be used to reconstruct the appropriate *was-derived-from* edge in the resulting graph. The intent of the type *InvocationResult* is similar, except that it refers to role-artifact pairs rather than role-value pairs (cf. Figure 3, line 5).

Reproducing an OPM graph results in a new OPM graph. There is some mundane activity involved in constructing such a new OPM graph: how should artifacts and processes be created? Hence, we assume the presence of factories: given a node from the old graph and the current new graph that we are in the process of building, such a factory results in a new node and a new OPM graph containing that node (line 8–10). Parameterizing the reproducibility function by such factories allows us to consider a range of options, such as the resulting graph has the same nodes as the original or the resulting graph has fresh nodes.

Finally, to be able to compare the results of the reproduced computation and the original results (whether final or intermediary), we introduce a mapping function that maps nodes of the original graph to nodes of the resulting graph. The pair of mappers *AMapper* and *PMapper* is conveniently referred to as *Mappers* (cf. Figure 3, lines 1–3).

The reproducibility semantics has a signature of type *Reproduce* (cf. Figure 3, lines 16–18). Given a graph factory (i.e. how we construct nodes), a primitive environment, an input OPM graph and some input artifacts, a reproducibility function must produce a resulting OPM graph and mappings from the input graph to the output graph. Such a reproducibility function recursively traverses the input graph, reproducing the invocation of every primitive; to this end, it relies on an aux-

$$
\begin{array}{rlll}
1 & PrimitiveEnv & = & PrimitiveName \rightarrow Primitive \\
2 & Primitive & = & \mathbb{P}(RoleValue) \rightarrow (\mathbb{P}(RoleValue) \times \mathbb{P}(EdgeSpec)) \\
3 & RoleValue & = & Role \times Value \\
4 & EdgeSpec & = & Role \times Role \\
5 & InvocationResult & = & \mathbb{P}(Role \times Artifact) \times \mathbb{P}(EdgeSpec) \\
6 & Invocation & = & PrimitiveName \times \mathbb{P}(Role \times Artifact) \times \mathbb{P}(Role \times Artifact) \\
7 & & & \\
8 & ArtifactFactory & = & Artifact \rightarrow OPMGraph \rightarrow Artifact \times OPMGraph \\
9 & ProcessFactory & = & Process \rightarrow OPMGraph \rightarrow Process \times OPMGraph \\
10 & GraphFactory & = & ArtifactFactory \times ProcessFactory \\
11 & & & \\
12 & Execute & = & GraphFactory \rightarrow PrimitiveEnv \\
13 & & & \rightarrow Invocation \rightarrow Process \rightarrow FullOPMGraph \rightarrow Mappers \\
14 & & & \rightarrow InvocationResult \times FullOPMGraph \times Mappers \\
15 & & & \\
16 & Reproduce & = & GraphFactory \rightarrow PrimitiveEnv \\
17 & & & \rightarrow FullOPMGraph \rightarrow InputArtifacts \\
18 & & & \rightarrow Mappers \times FullOPMGraph
\end{array}
$$

$$
\begin{array}{rll}
AMapper & = & Artifact \rightarrow Artifact \\
PMapper & = & Process \rightarrow Process \\
Mappers & = & (AMapper \times PMapper) \\
 & & \\
InputArtifacts & = & AResolver
\end{array}
$$

Figure 3: Reproducibility in a single Account

iliary function of type *Execute*, reproducing the invocation of a single process (cf. Figure 3, lines 12–14).

Having defined all the necessary sets, the reproducibility-semantics can be expressed as in Figure 4. The reproducibility-semantics is captured by function $reproduce_1$ of type *Reproduce*, which relies on the auxiliary function $reproduce_2$ to recursively execute each process of its input graph G_1^v. Given that OPM graphs are directed and acyclic, we assume the existence of *sortProcesses* a function that can obtain an ordered list of processes, such that the invocation of a process in the list does not require the outputs of any subsequent processes in the list. The auxiliary function $reproduce_2$ relies on *execute* to invoke primitives, and for each such invocation, ensures that new edges of the appropriated type are accumulated in the graph G_2^v.

The auxiliary function *execute* of type *Execute* invokes a primitive, as per defined in the primitive environment $\mathcal{E}$ and extends its input graph G^v with new artifacts and processes, created with the respective factories. It also ensures that the mappers are suitably extended with new mappings and valuations for the process and its output artifacts.

4.4. Reproducibility Properties

In this section, we establish some properties regarding the reproducibility of OPM graphs. First, we define role-value map equality and deterministic primitive.

Definition 2 (Role-Value Map Equality). *Two role-value maps rv_1^* and rv_2^* are equal if $DOM(rv_1^*) = DOM(rv_2^*)$ and $rv_1^*(r) = rv_2^*(r)$ for any $r \in DOM(rv_1^*)$.* □

Definition 3 (Deterministic Primitive). *A primitive pr is a deterministic function, if for any rv_1^*, rv_2^*, if $rv_1^* = rv_2^*$, then $pr(rv_1^*) = pr(rv_2^*)$.* □

Reproducibility of a graph can be defined as follows. Given the same inputs, the reproducibility function produces another graph that is equal "up to the mapping" between nodes, for a given primitive environment.

Definition 4 (Reproducible Graph). *Let G_1^v be an OPM graph. Let $(\mathcal{F}^a, \mathcal{F}^p)$ be artifact/process factories; let $\mathcal{E}$ be a primitive environment, let $in(G_1^v)$ be the values of input artifacts in G_1^v. Let $(\mathcal{M}_2, G_2^v) = reproduce_1\ (\mathcal{F}^a,\ \mathcal{F}^p)\ \mathcal{E}\ G_1^v\ in(G_1^v)$.*
The graph G_1^v is reproducible in $\mathcal{E}$, if the following holds:

$$
G_1^v \approx^{\mathcal{M}_2} G_2^v.
$$

□

Graph equality "up to mapping" is satisfied if graphs are isomorphic and have the same values for corresponding nodes, formalized as follows.

Definition 5 (Graph Equality "up to mapping"). *Let $G_1^v = (G_1, \mathcal{V}_1^a, \mathcal{V}_1^p)$ and $G_2^v = (G_2, \mathcal{V}_2^a, \mathcal{V}_2^p)$ be two full OPM graphs. Let $\mathcal{M}$ be a pair $\langle \mathcal{M}^a, \mathcal{M}^p \rangle$ of bijections such that their domains are the nodes in G_1 and their range the nodes in G_2. Two graphs G_1^v, G_2^v are equal "up to mapping $\mathcal{M}$", noted*

$$
G_1^v \approx^{\mathcal{M}} G_2^v,
$$

if the following conditions hold:

- *For any $a \in G_1$, $\mathcal{M}^a(a) \in G_2$ and $\mathcal{V}_1^a(a) = \mathcal{V}_2^a(\mathcal{M}^a(a))$.*

- *For any $p \in G_1$, $\mathcal{M}^p(a) \in G_2$ and $\mathcal{V}_1^p(p) = \mathcal{V}_2^p(\mathcal{M}^p(p))$.*

- *For any edge $\langle n_1, n_2 \rangle \in G_1$ (or $\langle n_1, r, n_2 \rangle \in G_1$), $\langle \mathcal{M}(n_1), \mathcal{M}(n_2) \rangle \in G_2$ (or $\langle \mathcal{M}(n_1), r, \mathcal{M}(n_2) \rangle \in G_2$).*

- *For every edge $\langle n_1', n_2' \rangle \in G_2$, there exists $\langle n_1, n_2 \rangle \in G_1$ such that $\langle \mathcal{M}(n_1), \mathcal{M}(n_2) \rangle = \langle n_1', n_2' \rangle$ (and likewise, for edges with roles).*

□

$reproduce_1 : Reproduce$

$reproduce_1\ (\mathcal{F}^a,\ \mathcal{F}^p)\ \mathcal{E}\ G_1^v\ \theta =$

$let\ (\_,\mathcal{M}_2,G_2^v)\ =\ initGraphForInputs\ \mathcal{F}^a\ (graphInputs\ G_1^v)\ \mathcal{M}_{Init}\ \theta$

$in\ reproduce_2\ (\mathcal{F}^a,\ \mathcal{F}^p)\ \mathcal{E}\ (sortProcesses\ G_1^v)\ G_1^v\ \mathcal{M}_2\ G_2^v$

end

$reproduce_2\ \mathcal{F}\ \mathcal{E}\ []\ G_1^v\ \mathcal{M}_1\ G_2^v = (\mathcal{M}_1,G_2^v)$

$reproduce_2\ \mathcal{F}\ \mathcal{E}\ (p :: l)\ G_1^v\ \mathcal{M}_1\ G_2^v =$

$let\ inv$	$=$	$extractInvocation\ p\ G_1^v\ \mathcal{M}_1$
$(\_,ru^*,\_)$	$=$	inv
$((rg^*,sp^*),G_3^v,\mathcal{M}_2)$	$=$	$execute\ \mathcal{F}\ \mathcal{E}\ inv\ p\ G_2^v\ \mathcal{M}_1$
$(G_3,\ \mathcal{V}_3^a,\ \mathcal{V}_3^p)$	$=$	G_3^v
p_2	$=$	$\mathcal{M}_2\ p$
$used^*$	$=$	$map\ (\lambda(r,a).\ \mathtt{used}(p_2,r,a))\ ru^*$
wgb^*	$=$	$map\ (\lambda(r,a).\ \mathtt{wgb}(a,r,p_2))\ rg^*$
wdf^*	$=$	$map\ (\lambda(r_1,r_2).\ \mathtt{wdf}(rg^*(r_1),\ ru^*(r_2)))\ sp^*$
G_4^v	$=$	$(G_3\ \cup\ wdf^*\ \cup\ wgb^*\ \cup\ used^*,\ \mathcal{V}_3^a,\ \mathcal{V}_3^p)$

$in\ reproduce_2\ \mathcal{F}\ \mathcal{E}\ l\ G_1^v\ \mathcal{M}_2\ G_4^v$

end

$execute : Execute$

$execute\ (\mathcal{F}^a,\mathcal{F}^p)\ \mathcal{E}\ inv\ p_0\ (G,\mathcal{V}^a,\mathcal{V}^p)\ (\mathcal{M}^a,\mathcal{M}^p) =$

$let\ (n,ru^*,rg^*)$	$=$	inv
pr	$=$	$\mathcal{E}(n)$
(rv^*,sp^*)	$=$	$pr\ (map\ (\lambda(r,a).\ (r,\mathcal{V}^a(a)))\ ru^*)$
(rg_2^*,G_1)	$=$	$mapWithGraph\ (\lambda(r,v).\lambda G.\ let\ (a,G) = \mathcal{F}^a\ rg^*(r)\ G$
		$\qquad in\ ((r,a),G)$
		$\qquad end)$
		$\qquad\qquad\qquad\quad rv^*\ G$
(p_2,G_2)	$=$	$\mathcal{F}^p\ p_0\ G_1$
$\mathcal{V}_2^a$	$=$	$\mathcal{V}^a[rg_2^* \to^{r^*} rv^*]$
$\mathcal{V}_2^p$	$=$	$\mathcal{V}^p\ [p_2 \to n]$
$\mathcal{M}_2^a$	$=$	$\mathcal{M}^a[rg^* \to^{r^*} rg_2^*]$
$\mathcal{M}_2^p$	$=$	$\mathcal{M}^p\ [p_0 \to p_2]$

$in\ ((rg_2^*,sp^*),(G_2,\mathcal{V}_2^a,\mathcal{V}_2^p),(\mathcal{M}_2^a,\mathcal{M}_2^p))$

end

$initGraphForInputs\ \mathcal{F}^a\ a^*\ (\mathcal{M}^a,\mathcal{M}^p)\ \mathcal{V}^a =$

$let\ (a_2^*,G)$	$=$	$mapWithGraph\ \mathcal{F}^a\ a^*\ G_{Init}$
$\mathcal{M}_2^a$	$=$	$\mathcal{M}^a[a^* \to^* a_2^*]$
$\mathcal{V}_2^a$	$=$	$\mathcal{V}_{Init}^a[a_2^* \to^* (map\ \mathcal{V}^a\ a^*)]$

$in\ (a_2^*,(\mathcal{M}_2^a,\mathcal{M}^p),(G,\mathcal{V}_2^a,\mathcal{V}_{Init}^p))$

end

$extractInvocation\ p\ (G,\mathcal{V}^a,\mathcal{V}^p)\ (\mathcal{M}^a,\mathcal{M}^p) : Invocation =$

$let\ used^*$	$=$	$getUsed(p,G)$
a^*	$=$	$map\ (\lambda(\mathtt{used}(p,r,a)) => a)\ used^*$
a_2^*	$=$	$map\ \mathcal{M}^a\ a^*$
ru_2^*	$=$	$map_2\ (\lambda(\mathtt{used}(p,r,a),a_2).(r,a_2))\ used^*\ a_2^*$
rg_2^*	$=$	$map\ (\lambda(\mathtt{wgb}(a,r,p)).(r,a))\ getGeneratedBy(p,G)$

$in\ (\mathcal{V}^p(p),ru_2^*,rg_2^*)$

end

p	$\in$	$Process$	process
a	$\in$	$Artifact$	artifact
a^*	$\in$	$\mathbb{P}(Artifact)$	artifact set
G	$\in$	$OPMGraph$	OPM graph
G^v	$\in$	$FullOPMGraph$	Full OPM graph
$\mathcal{E}$	$\in$	$PrimitiveEnv$	primitive env.
pr	$\in$	$Primitive$	primitive
$\mathcal{F}^a$	$\in$	$ArtifactFactory$	artifact factory
$\mathcal{F}^p$	$\in$	$ProcessFactory$	process factory
$\mathcal{F}$	$\in$	$GraphFactory$	graph factory
sp	$\in$	$EdgeSpec$	edge specification
rv	$\in$	$RoleValue$	role-value pair
rv^*	$\in$	$Role \to Value$	role value map
rg	$\in$	$Role \times Artifact$	role-generated artifact pair
rg^*	$\in$	$Role \to Artifact$	role generated artifact map
ru	$\in$	$Role \times Artifact$	role-used artifact pair
ru^*	$\in$	$Role \to Artifact$	role used artifact map
$\mathcal{V}^a$	$\in$	$AResolver$	artifact resolver
$\mathcal{V}^p$	$\in$	$PResolver$	process resolver
$\mathcal{M}^a$	$\in$	$AMapper$	artifact mapper
$\mathcal{M}^p$	$\in$	$PMapper$	process mapper

$f[x \to y]$	$=$	$\lambda v.\ if\ v = x\ then\ y\ else\ f(v)$
$f[x :: x^* \to^* y :: y^*]$	$=$	$f[x \to y][x^* \to^* y^*]$
$f[[] \to^* []]$	$=$	f

extension by role

$f[(r,x) :: x^* \to^{r^*} y^*]$	$=$	$f[x \to y^*(r)][x^* \to^{r^*} y^*]$
$f[[] \to^{r^*} y^*]$	$=$	f

$mapWithGraph : (\alpha \to \beta \to \gamma * \beta) \to \mathbb{P}(\alpha) \to \beta \to \mathbb{P}(\gamma) \times \beta$

$mapWithGraph\ f\ []\ G\ = ([],G)$

$mapWithGraph\ f\ (x :: x^*)\ G =$

$let\ (y,G_2)$	$=$	$f\ x\ G$
(y^*,G_3)	$=$	$mapWithGraph\ f\ x^*\ G_2$

$in\ (y :: y^*,G_3)$

end

$getUsed(p,G) = \{\mathtt{used}(p,r,a) \in G\}$

$getGeneratedBy(p,G) = \{\mathtt{wgb}(a,r,p) \in G\}$

$sortProcesses : FullOPMGraph \to list(Process)$

Figure 4: Reproducibility Semantics

Based on Definition 4, for given factories and primitive environment, we can identify the class of reproducible OPM graphs. We note that not all graphs are reproducible. Indeed, the reproducibility function may not be defined, for different reasons, which we discuss below and illustrate with simple examples pertaining to Figure 1. *(i)* An incomplete set of inputs is provided: e.g., $\{(a_1, 10), (a_2, 20), (a_3, 30)\}$. *(ii)* Inputs or intermediary results are not in the domain of some primitives: e.g., $\{(a_1, 10), (a_2, 20), (a_3, 30), (a_4, 0)\}$. *(iii)* Incorrect number of inputs, incorrect roles or incorrect types are provided to a primitive; e.g., for primitive environment mapping `prim:sum` to the unary log function. We note that some of these failure reasons can be checked statically without re-executing primitives, if primitive signature and arity are available.

Checking that a graph is reproducible helps users to get confidence in data. In Section 5, we investigate the implementation of this semantics in a reproducibility service. Beforehand, we focus on reproducibility in multi-account graphs.

4.5. Multi-Account OPM Graphs

So far, we have tackled account-less OPM graphs. In this section, we provide a definition of multi-account graphs with a view of defining the associated reproducibility semantics. Figure 5 displays a definition of a multi-account graph *MAccOPMGraph*. It is a triple consisting of a set of accounts, a partial function mapping an account to an OPM graph, and a refinement function.

$$
\begin{aligned}
Account &= \text{primitive set} \\
Refinement &= Account \to Process \to Account \\
MAccOPMGraph &= \mathbb{P}(Account) \\
&\quad \times(Account \to OPMGraph) \\
&\quad \times Refinement \\
FullMAccOPMGraph &= MAccOPMGraph \times AResolver \\
&\quad \times PResolver
\end{aligned}
$$

Figure 5: Multi-Account OPM Graph

With this definition, the set of accounts known to a *MAccOPMGraph* is given by the first component of the triple. For each account, the multi-account OPM graph provides us with one OPM graph. A *FullMAccOPMGraph* includes artifact and process resolvers, mapping them to values or primitive names, respectively.

Two OPM graphs in a multi-account OPM graph overlap if they share some artifact or process. We see that in this definition a single value can be associated with an artifact in a multi-account OPM graph.

The third component of a *MAccOPMGraph* is a refinement function, which captures a very specific form of refinement, corresponding to *procedural abstraction*. A process is allowed to be refined into a subgraph (itself consisting of processes and artifact and associated edges); if p is the process, α the account in which p occurs, the subgraph must correspond to an account, say α'. In that case, the refinement function ρ is such that $\rho \, \alpha \, p = \alpha'$. To ensure reproducibility, we set some strict constraints on the refinement function.

Definition 6 (Account Constraints). *Let* $\rho \, \alpha_1 \, p_1 = \alpha_2$ *and* $\rho \, \alpha_3 \, p_2 = \alpha_4$. *We have that* $\alpha_2 = \alpha_4$ *if and only if* $\alpha_1 = \alpha_3$ *and* $p_1 = p_2$. $\square$

This ensures that an account corresponds to the activation of one and only one process.

4.6. Reproducibility of a Multi-Account OPM Graph

Figure 6 displays the reproducibility semantics for multi-account OPM graphs, expressed by a function of type *MAccReproduce*. Provided with the appropriate graph factories and a primitive environment, for a given account and input artifacts, it takes a FullMAccOPMGraph and returns another FullMAccOPMGraph and a set of mapper functions. The account provided as input is the account in which computation needs to be reproduced. Mappers now can map accounts of the original graph to accounts in the resulting graph. Likewise, factories comprise a factory for accounts.

The function $reproduce_4$ encodes the reproducibility semantics, invoking the recursive function $reproduce_3$ after initializing a *FullMAccOPMGraph* acting as an accumulator. The function $reproduce_3$ relies on $reproduce_2$ (defined in Figure 4) iteratively reproducing each account (and its refinements) ensuring that nodes are properly shared across accounts in the resulting graph.

To this end, we rely on the following operator, defining a factory function that creates a new artifact only for those that have not been mapped yet.

$$
\mathcal{F}^a \setminus \mathcal{M}^a = \lambda x G. \begin{array}{l} (\mathcal{M}^a x, \{\mathcal{M}^a x\} \cup G) \text{ if } \mathcal{M}^a x \neq \bot, \\ \mathcal{F}^a \, x \, G \text{ otherwise .} \end{array}
$$

We also introduce a combining operator, that takes the "union" of two functions, preferring the first over the second.

$$
\theta_1 \uplus \theta_2 = \lambda x. \begin{array}{l} \theta_1 x \text{ if } \theta_1 x \neq \bot \\ \theta_2 x \text{ otherwise.} \end{array}
$$

Let α be an account, and α_r a subaccount of α. The inputs to α_r (defined as θ_2 in Figure 6) is obtained by combining the inputs to α and the values of any artifact in α.

5. Semantic Web and Reproducibility

In this section, we investigate how to leverage the reproducibility-semantics of Section 4 in order to define a reproducibility service for the Semantic Web. First, we outline an architecture for a reference implementation of this service. Then, we survey the Semantic Web techniques that are exploited in this implementation. Finally, we summarize the assumptions underlying OPM graphs for reproducibility in the Semantic Web.

5.1. Reproducibility Service Architecture

Figure 7 displays the architecture of a reproducibility service. It consists of four key components: *(i)* the *reproducibility engine*, which implements the reproducibility semantics of Section 4; *(ii)* a *triple store* containing representations of the OPM

$$
\begin{aligned}
\textit{AccMapper} &= \textit{Account} \rightarrow \textit{Account} \\
\textit{AccountFactory} &= \textit{Account} \rightarrow \textit{MAccOPMGraph} \rightarrow \textit{Account} \times \textit{MAccOPMGraph} \\
\textit{MAccGraphFactory} &= \textit{GraphFactory} \times \textit{AccountFactory} \\
\textit{MAccMapper} &= \textit{Mappers} \times \textit{AccMapper}
\end{aligned}
$$

$$
\begin{aligned}
\textit{MAccReproduceIter} &= \textit{MAccGraphFactory} \rightarrow \textit{PrimitiveEnv} \rightarrow \mathbb{P}(\textit{Account} \times \textit{InputArtifacts}) \\
&\quad \rightarrow \textit{FullMAccOPMGraph} \rightarrow \textit{MAccMapper} \rightarrow \textit{FullMAccOPMGraph} \\
&\quad \rightarrow \textit{MAccMapper} \times \textit{FullMAccOPMGraph}
\end{aligned}
$$

$$
\begin{aligned}
\textit{MAccReproduce} &= \textit{MAccGraphFactory} \rightarrow \textit{PrimitiveEnv} \rightarrow (\textit{Account} \times \textit{InputArtifacts}) \rightarrow \textit{FullMAccOPMGraph} \\
&\quad \rightarrow \textit{MAccMapper} \times \textit{FullMAccOPMGraph}
\end{aligned}
$$

$reproduce_3 : MAccReproduceIter$
$reproduce_3\ \mathcal{F}\ \mathcal{E}\ []\ (\mathcal{G}, \mathcal{V}^a, \mathcal{V}^p)\ \mathcal{M}\ (\mathcal{G}_1, \mathcal{V}_1^a, \mathcal{V}_1^p) = (\mathcal{M},\ (\mathcal{G}_1, \mathcal{V}_1^a, \mathcal{V}_1^p))$

$reproduce_3\ \mathcal{F}\ \mathcal{E}\ ((\alpha, \theta) :: l)\ (\mathcal{G}, \mathcal{V}^a, \mathcal{V}^p)\ \mathcal{M}\ (\mathcal{G}_1, \mathcal{V}_1^a, \mathcal{V}_1^p) =$

$$
\begin{aligned}
\mathbf{let}\qquad (\alpha_0^*, \Gamma, \rho) &= \mathcal{G} \\
G &= \Gamma\ \alpha \\
(\alpha_2, \mathcal{G}_2) &= \mathcal{F}^\alpha\ \alpha\ \mathcal{G}_1 \\
\mathcal{M}_2^\alpha &= \mathcal{M}^\alpha[\alpha \rightarrow \alpha_2] \\
p^* &= sortProcesses(G, \mathcal{V}^a, \mathcal{V}^p) \\
(\_, (\mathcal{M}_0^a, \mathcal{M}_0^p), G_0^v) &= initGraphForInputs\ (\mathcal{F}^a\ \backslash\ \mathcal{M}^a)\ (graphInputs\ (G, \mathcal{M}^a, \mathcal{M}^p))\ (\mathcal{M}^a, \mathcal{M}^p)\ \theta \\
((\mathcal{M}_2^a, \mathcal{M}_2^p), (G_2, \mathcal{V}_2^a, \mathcal{V}_2^p)) &= reproduce_2\ (\mathcal{F}^a\ \backslash\ \mathcal{M}_0^a, \mathcal{F}^p)\ \mathcal{E}\ p^*\ (G, \mathcal{V}^a, \mathcal{V}^p)\ (\mathcal{M}_0^a, \mathcal{M}_0^p)\ G_0^v \\
\alpha^* &= map\ (\rho\ \alpha)\ (processes\ \mathcal{G}\ \alpha) \\
\theta_2 &= \theta\ \uplus\ (\mathcal{V}_2^a \circ \mathcal{M}_2^a)
\end{aligned}
$$
$\mathbf{in}\ reproduce_3\ \mathcal{F}\ \mathcal{E}\ (l@(map\ (\lambda\alpha.\ (\alpha, \theta_2))\ \alpha^*))\ (\mathcal{G}, \mathcal{V}^a, \mathcal{V}^p)\ ((\mathcal{M}_2^a, \mathcal{M}_2^p), \mathcal{M}_2^\alpha)\ (\mathcal{G}_2[\alpha_2 \rightarrow G_2], \mathcal{V}_1^a\ \uplus\ \mathcal{V}_2^a, \mathcal{V}_1^p\ \uplus\ \mathcal{V}_2^p)$
$\mathbf{end}$

$reproduce_4 : MAccReproduce$
$reproduce_4\ \mathcal{F}\ \mathcal{E}\ (\alpha, \theta)\ (\mathcal{G}, \mathcal{V}^a, \mathcal{V}^p) =$
$reproduce_3\ \mathcal{F}\ \mathcal{E}\ [(\alpha, \theta)]\ (\mathcal{G}, \mathcal{V}^a, \mathcal{V}^p)\ \mathcal{M}_{Init}\ (\mathcal{G}_{Init}, \mathcal{V}_{Init}^a, \mathcal{V}_{Init}^p)$

$\mathcal{F}^a$	$\in$	$ArtifactFactory$	artifact factory	ρ	$\in$ $Refinement$	refinement
$\mathcal{F}^p$	$\in$	$ProcessFactory$	process factory	Γ	$\in$ $Account \rightarrow OPMGraph$	account graph map
$\mathcal{F}^\alpha$	$\in$	$AccountFactory$	account factory	θ	$\in$ $InputArtifacts$	input artifacts
$\mathcal{F}$	$\in$	$GraphFactory$	graph factory			

Figure 6: Reproducibility of a Multi-Account OPM Graph

graph to reproduce, and of the OPM graph being generated, and semantic declarations of primitives *(iii)* the *primitive environment*, which maps primitive names to actual primitives; *(iv)* the *execution engines*, which are implementations of the primitive. We now discuss them below.
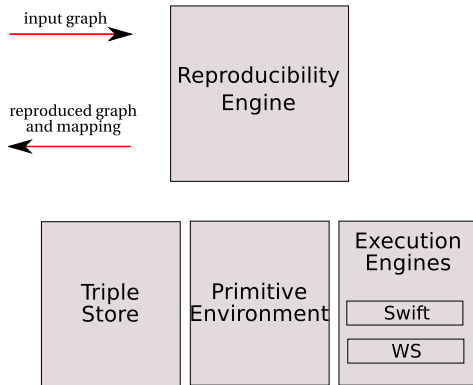

Figure 7: Reproducibility Architecture

The reproducibility engine implements the reproducibility semantics. Given an OPM graph, it extracts processes in their invocation order (as specified by dependencies). Each process is annotated with a primitive name. For instance, in the First Provenance Challenge [23] workflow (discussed in Section 6), process p1 results from the activation of the primitive prim:align_warp.

```
@prefix opm:  http://openprovenance.org/ontology#
@prefix prim: http://openprovenance.org/primitives#
@prefix pc1:  http://www.ipaw.info/pc1/

pc1:p1 opm:annotation pc1:an1_p1 .

pc1:an1_p1 a opm:Annotation;
           opm:property pc1:pr_9 .

pc1:pr_9 a opm:Property ;
    opm:uri   prim:primitive;
    opm:value prim:align_warp.
```

Each primitive name is associated with an implementation of that primitive by the primitive environment. For instance, the default environment contains a mapping from prim:align_warp to the name http://openprovenance.org/reproducibility/swift#align_warp, which is itself mapped to an implementation procedure in the execution engine. We have developed two types of execution engines, inline in the Java virtual machine, or delegated to the command line by means of the Swift workflow engine [45].

Swift encompasses Swift script, a declarative language allowing the type of procedures to be declared, as well as the necessary "plumbing" to invoke the executable on the command line. For instance, Figure 8 contains a declaration for align_warp specified as taking four inputs (i1, i2, i3, i4) and producing an output (o). Each input/output element is associated with its OPM role, unique for this primitive. For instance, the output o has role out, whereas the anatomy image i1 has

role img. We note that the actual invocation makes explicit use of the image files i1 and i2. On the other hand, the header files are explicitly declared with roles hdr and hdrRef, but are not passed explicitly to the executable align_warp since it derives them automatically by replacing the extension img by hdr.

```
<procedure name="align_warp">

 <output name="o" type="WarpParameters"    opr:role="out"/>

 <input  name="i1" type="AnatomyImage"     opr:role="img"/>
 <input  name="i2" type="ReferenceImage"   opr:role="imgRef"/>
 <input  name="i3" type="AnatomyHeader"    opr:role="hdr"/>
 <input  name="i4" type="ReferenceHeader" opr:role="hdrRef"/>

 <binding>
   <application>
     <executable>align_warp</executable>
     <function name="filename">
        <variableReference>i1</variableReference>
     </function>
     <function name="filename">
        <variableReference>i2</variableReference>
     </function>
     <function name="filename">
        <variableReference>o</variableReference>
     </function>
     <stringConstant>-m</stringConstant>
     <stringConstant>12</stringConstant>
     <stringConstant>-q</stringConstant>
   </application>
 </binding>
</procedure>
```

Figure 8: Swift Script for align_warp

The invocation engine identifies all used artifacts. For instance, pc1:a1 is declared as an artifact with type File, at location /home/pc1/reference.img.

```
pc1:a1 a opm:Artifact ;
opm:account pc1:black ;
opm:label "Reference Image" ;
opm:type prim:File.

pc1:a1 opm:annotation pc1:an1_a1 .

pc1:an1_a1 a opm:Annotation;
           opm:property pc1:pr_23 .

pc1:pr_23 a opm:Property ;
opm:uri prim:path ;
opm:value "/home/pc1/reference.img" .
```

The artifact pc1:a1 is an input artifact to the reproducibility function. We may want to use this exact file or another one. The reproducibility engine also needs to be provided with the actual value of this input, which is done by means of Swift configuration containing the actual location where the file must be retrieved from.

```
<variable name="var_i1" type="ReferenceImage">
    <file name="//home/user/pc1/reference.img"/>
</variable>
```

Likewise, we do not necessarily want to write output files at the same location, because we do not want to overwrite files,

or because we do not have the rights to do so. So, the reproducibility engine also specifies the actual location where output files must be stored into. In this example, they take place in the current directory.

```
<variable name="var_o" type="WarpParameters">
    <file name="./params1.warp"/>
</variable>
```

5.2. Semantic Web Techniques for Reproducibility

Multiple Semantic Web technologies have been exploited to build a reference implementation of the reproducibility service. Our baseline is a pre-existing OWL ontology for OPM [41]. We discuss the technologies that have been used, the purpose for which they were used, and their limitations.

Queries. Having represented OPM graphs in RDF, it is natural to use SPARQL [46] to express queries. This use of SPARQL is well documented in the literature (e.g., MINDSWAP [47] and Wings [48] in the first Provenance Challenge, Tetherless [49] in the third Challenge). Two difficulties worth noting emerge. Identifying all inputs of an OPM graph requires negation by failure, which can be encoded in SPARQL as illustrated in Figure 9. Querying transitive properties is discussed next in this section.

```
SELECT ?a
WHERE
  ?a a opm:Artifact
  OPTIONAL {?a opm:_wasGeneratedBy ?p}
  FILTER (!bound(?p))
```

Figure 9: SPARQL query: Inputs to an OPM Graph

Transitive Closures. When multiple or disconnected OPM graphs co-exist in a triple store, and we wish to reproduce a specific result, transitive closures are useful to identify the inputs that indirectly cause some specific outputs. Figure 10 displays a possible definition of the multi-step edge WasDerivedFrom* as a transitive property in OWL. First, we note that a WasDerivedFrom OPM edge is expressed as an OWL class and not an OWL property. Such an encoding facilitates the expressiveness of other OPM properties, such as account membership, time information, and OPM annotations.

We then define an OWL property, _wasDerivedFrom with Artifact as domain and range. Such a property can be inferred[11] by OWL by means of a property chain: if there is an artifact that is effect of a WasDerivedFrom edge, itself with another artifact as cause, then we can infer a property _wasDerivedFrom between these two artifacts.

Then, property _wasDerivedFrom_star is defined as transitive, with _wasDerivedFrom declared as a subproperty of _wasDerivedFrom_star. Similar definitions can be adopted for all multi-step inferences permitted by OPM.

Using OWL to encode OPM inferences presents some further challenges. First, we note that OPM completion rules (artifact and process introductions) are not expressible in OWL since they require the inference of novel instances. Second, the transitive closure defined in Figure 10 ignores accounts: it could infer a _wasDerivedFrom_star property by composing (properties inferred from) edges declared in two separate accounts, which is not a legal inference in OPM. A solution to this problem is to consider named graphs [50] to capture assertions related to an account, and ensure that OWL inferences are limited to a graph [51].

Ontological Definitions. An ontology has been designed and is being used at design time. It allows us to express core concepts, such as common artifacts (files with their path, numbers, collections), processes (with a reference to a primitive name), and kinds of primitives. This ontology (with prefix prim) extends the OPM OWL ontology, by subclassing its core classes artifacts and processes. It allows us to check for consistency of the various concepts.

Furthermore, still at design time, the ontology allows us to define common *derivations*, corresponding to *EdgeSpec* in the reproducibility semantics, and corresponding subtypes of the WasDerivedFrom relation, and associate them with the corresponding primitives. For instance, the addition primitive PrimitivePlus has two derivations Summand0Derivation and Summand1Derivation from its output (identified by role out0) to its respective inputs identified by roles summand0 and summand1. Figure 11 illustrate an excerpt of the Primitive ontology.

Hence, the use of ontologies facilitates the typing of primitives (seen as functions operating over typed role-value pairs) and their associated typed derivations. The kind of static type checking of OPM graphs discussed in Section 4.4 can be implemented by means of this ontology, and was referred to as semantic validity by Miles *et al.* [52].

Runtime Rules. The reproducibility semantics expresses how OPM edges can be constructed for every enacted process. Such edges can be asserted by new SWRL rules [12]. For instance, in Figure 12, a new edge _wasDerivedFrom is added to an OPM graph by means of a SWRL rule that checks the existence of an input artifact a2 and an output artifact a1, respectively used

---

[11]To specify this ontology, it was necessary to sacrifice elegance for decidability. Instead of defining properties effect and cause from Edge to Node, we had to define effectWasDerivedFrom and causeWasDerivedFrom from Wasderivedfrom to Node; similar properties were also defined for other OPM edges.

[12]It is important to note that OPM WasDerivedFrom edges cannot be inferred from other edges, but must be asserted. By constructing WasDerivedFrom edges by SWRL rules, we are not misinterpreting OPM semantics. Indeed, these edges are constructed according to an ontological definition of primitives, characterizing their semantics, and in particular, the associated dependencies. We note that the current version of this ontology only supports static WasDerivedFrom edges, always between the same outputs and inputs (identified by their roles), for whatever invocation of the primitive. An alternative design supports conditional edges, according to the inputs to the primitive.

```
// Class: http://openprovenance.org/ontology#WasDerivedFrom

SubClassOf(WasDerivedFrom Edge)
SubClassOf(WasDerivedFrom ObjectSomeValuesFrom(effectWasDerivedFrom Artifact))
SubClassOf(WasDerivedFrom ObjectSomeValuesFrom(causeWasDerivedFrom Artifact))
SubClassOf(WasDerivedFrom ObjectAllValuesFrom(effectWasDerivedFrom Artifact))
SubClassOf(WasDerivedFrom ObjectAllValuesFrom(causeWasDerivedFrom Artifact))

// Object property: http://openprovenance.org/ontology#effectWasDerivedFrom-1

InverseObjectProperties(effectWasDerivedFrom-1 effectWasDerivedFrom)
InverseFunctionalObjectProperty(effectWasDerivedFrom-1)
ObjectPropertyDomain(effectWasDerivedFrom-1 Artifact)
ObjectPropertyRange(effectWasDerivedFrom-1 WasDerivedFrom)

// Object property: http://openprovenance.org/ontology#_wasDerivedFrom

SubObjectPropertyOf(_wasDerivedFrom _wasDerivedFrom_star)
ObjectPropertyDomain(_wasDerivedFrom Artifact)
ObjectPropertyRange(_wasDerivedFrom Artifact)

// Sub property chain axiom

SubObjectPropertyOf(SubObjectPropertyChain(effectWasDerivedFrom-1 causeWasDerivedFrom) _wasDerivedFrom)

// Object property: http://openprovenance.org/ontology#_wasDerivedFrom_star

TransitiveObjectProperty(_wasDerivedFrom_star)
ObjectPropertyDomain(_wasDerivedFrom_star Artifact)
ObjectPropertyRange(_wasDerivedFrom_star Artifact)
```

Figure 10: Transitive _wasDerivedFrom_star

and generated by a process under some roles, process which is associated with a primitive, declared to produce a derivation between those roles.

5.3. Semantic Web Assumptions for Reproducibility

In this section, we summarize the assumption that underpin OPM graphs for reproducibility in the Semantic Web.

- *Well defined global names*: Primitive names should be defined by a global unique name, with a precise meaning. Likewise, their implementations must be uniquely identified. In the Semantic Web tradition, such primitive names and implementations can be identified by URIs. For instance, we previously named the primitive align_warp with the URI prim:align_warp.

- *Explicit representation of information*: Each execution engine may itself be configurable, and such configurations need to be made explicit. For instance, Figure 8 displays the Swift script for the implementation of align_warp. Furthermore, the Swift engine relies on a file tc.data to map executable name to a specific executable in the file system; this file must also be made explicit.

6. Evaluation

We have undertaken a range of empirical evaluations aiming to demonstrate the capability of the reproducibility service, and its suitability for deployment in a multi-technology environment such as the Web. We discuss them in turn.

1. Feasibility: First Provenance Challenge. Our first experience is designed to demonstrate that the reproducibility service is capable of reproducing the results of a significant computation. We used the First Provenance Challenge workflow [23], which has become a de-facto benchmark in the provenance community.

By hand, we constructed pc1trace, an OPM provenance trace of the First Provenance Challenge (see Figure A.17, in Appendix). The trace pc1trace is minimal in the sense that it is not annotated or decorated with any information that is not necessary for reproducibility. Reproducing pc1trace resulted in the following:

- an output trace with the same structure as pc1trace;

- a precise mapping of pc1trace node ids to the output trace ids;

- a set of intermediary and final output files.

We have checked that the resulting trace is identical up to the node ids, the locations of produced files, and the file contents.

Figure 13 summarizes the outcome of this experiment. The input artifacts were chosen to be the same files (at a location of our choice). The output artifacts a11 to a24 were shown to be the same. We however observe that the artifacts following the slicer stage differed. For instance, the artifact atlas-x.pgm had 8 bytes that differed by one unit. Our only interpretation of this difference is that we were using a more recent version of the fsl library.

The outcome of this experiment highlights the benefits of this approach. By keeping explicit provenance and intermediary re-
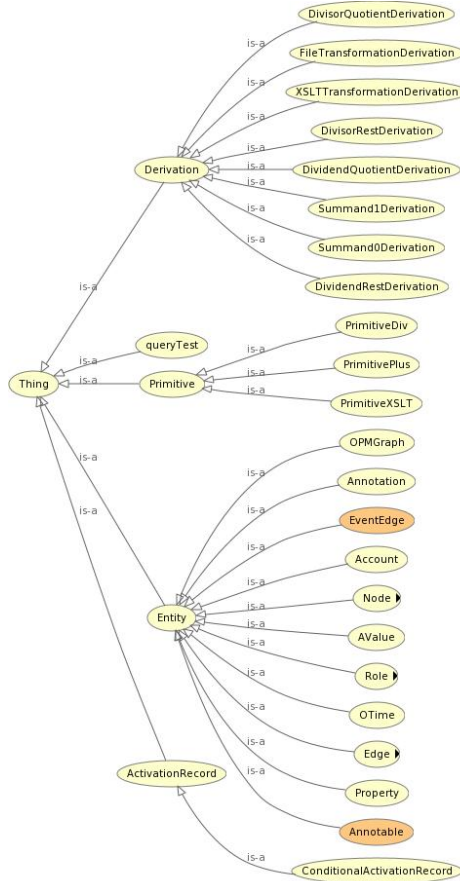
Figure 11: Primitive Ontology

```
Artifact(?a1),
Artifact(?a2),
_wasGeneratedBy(?a1,?p),
_used(?p,?a2),
effectWasGeneratedBy-1(?a1,?g),
cause(?g,?p),
role(?g,?r1),
isKindOf(?r1,?rt1),
effectUsed-1(?p,?u),
cause(?u,?a2),
role(?u,?r2),
isKindOf(?r2,?rt2),
isKindOf(?p,?pt),
hasDerivation(?pt,?d),
effects(?d,?rt1),
causedBy(?d,?rt2)
  -> _wasDerivedFrom(?a1,?a2)
```

Figure 12: SWRL rule to construct derivations according to the semantics of primitives

	artifact id	success?	comments
inputs	a1, ..., a10	✔	copies of originals of 2006-05-31
outputs	a11, ..., 24	✔	checked to be the same as originals of 2006-05-31
outputs	a25, ..., 30	✘	different

Figure 13: Experiment 1: Result Comparability

sults (which in this example were *four years old*), we were able to rerun the experiment and compare results. Reproducing the experiment allowed us to identify a specific library, whose installed version was more recent, as the cause of the divergent results.

2. Reproducibility Variants. In the second experiment, we establish that the reproducibility service can easily be customized to invoke multiple execution technologies, such as command line and Java code, simply by changing its primitive environment. Furthermore, the reproducibility service can be parameterized with different graph factories, to customize execution or to ensure the uniqueness of the generated graph.

The results of this experiment are summarized in Figure 14. The "Java Inline" execution refers to the ability to invoke Java code directly. This was achieved by reproducing the OPM graph of Figure 1, where the invoked primitives were implemented in Java directly. Alternatively, the PC1 OPM graph (Figure A.17) was reproduced using primitives implemented in Swift, invoking command lines. Work is in progress to support Web Services implementations of primitives; to this end, we are planning to make use of the D-Profile [53] to minimize the size of the OPM graph. We have also demonstrated that the reproducibility service can be configured with mock-up implementation of primitives, which are hardwired and return specific out-

puts for specific inputs (similarly to Bechhofer's [24] replayability). Such replayability was demonstrated for the PC1 OPM graph, with primitives returning directly the URLs as per specified in the First Challenge; such primitives were then described as "dummies" [23].

A driver for this paper is to provide a reproducibility semantics for an ontology-based representation of provenance, allowing a uniform representation of provenance, despite multiple execution technologies being involved in executions across the Web. To demonstrate this capability, we have configured the reproducibility service, with implementation of PC1 primitives using different technologies, e.g. Swift and Java, and have successfully reproduced the experiment.

The second part of Figure 14 demonstrates how the reproducibility service can be configured with various graph factories. The graph factory can be used to generate new ids for nodes (and edges), but also to change the location of files to be generated by the command line executables, so that they do not overwrite previously existing files. Both were successfully

Java Inline Execution	✔	demonstrated with numeric wflow
WebService Execution	✘	implementation in progress
Swift Execution	✔	demonstrated with PC1 workflow
Replayability [24]	✔	PC1 mock up with dummy primitives
Multi Technology	✔	demonstrated with PC1 workflow
Graph factory variant	✔	output graph with different artifact ids
Graph factory variant	✔	output with different file locations

Figure 14: Experiment 2: Reproducibility Variants

demonstrated using the PC1 OPM graph.

3. Other Inputs. In the third experiment, we establish that the reproducibility service can be used to reproduce experiments with inputs that differ from those used in the original experiment.

New inputs	✔	demonstrated with numeric wflow
Differently encoded inputs	✔	demonstrated with numeric wflow
Change of parameters	✔	Provenance Challenge 1

Figure 15: Experiment 3: Other Inputs

One should note that in this experiment we do not have the original workflow but just a trace of its past execution. Given the numeric expression OPM graph (Figure 1), one can recompute the expression with alternate inputs. When the original process makes decisions on its inputs, the outcome of such decision-making may differ when new inputs are provided. In that case, the provenance trace may not contain enough information to reproduce the original process (essentially alternate branches may be missing). This is an issue that Cheney *et al.* tackle under their "fidelity property" [15], which relies on a form of "continuation" [54], a data structure that combines computational state and program structure, to allow computations to be resumed and continued.

OPM requires the encoding of artifact values to be made explicit. Hence, alternate encodings of a same input can be supported (e.g., an integer passed by reference in a file, instead of by value). We note that this type of conversion, referred to as a "shim" by Duncan *et al.* [55], can be handled automatically and systematically in a number of cases [56].

Finally, parameter sweeps are possible by changing workflow parameters, considered as an "input" by the reproducibility service.

7. Discussion

7.1. OPM

This paper is the first to provide an executable semantics for a substantial subset of OPM, independently of a given execution technology. This formalization complements the ones discussed in Section 3.4: Cheney's causal perspective of OPM [37], Moreau *et al.*'s set-theoretic definition of OPM [35], Kwasnikowska *et al.*'s temporal interpretation of OPM [36], and Missier and Goble's translation of OPM to a workflow language [17]. The fact that each formalization covers a different subset of OPM, and that no equivalence between interpretations has been established yet, is indicative of a lack of a "grand theory of OPM".

OPM introduced interesting features, such as the notions of accounts and refinements. This paper has proposed a novel definition for these, which corresponds to the nested invocation of procedures in programming languages: a process can be refined into a subaccount (to some extent, the equivalent of an activation frame in language runtime systems). Alternate

definitions have been proposed, and their implication for reproducibility need to be investigated. Kwasnikowska and Van Den Bussche [57] propose a methodology to accommodate hierarchical refinements in OPM. Their notion of refinement allows for an OPM subgraph to be refined into another OPM subgraph. Groth and Moreau [53] propose the D-profile, a profile to express details of execution in distributed systems, such as communication and messages; the D-profile introduces an alternative form of refinement, where an artifact is refined into a subgraph. Whilst the notions of refinement defined in these proposals are more general than the one presented here, no reproducibility semantics of such refinements has been proposed.

This paper has introduced constraints on the topology of OPM graphs to ensure their reproducibility (cf. Definitions 1 and 6). It is our belief that the acyclicity constraint could be relaxed whilst still preserving reproducibility, for networks of processes exchanging artifacts. Indeed, provided that there is no cycle with *was-derived-from* edges, we can identify processes in subaccounts that exchange such artifacts. The current semantics would have to be extended in two different ways to support these: procedures would have to be called by "name" and no longer "by value", and processes would have to be ordered across multiple accounts. A number of edges have been ignored in the reproducibility semantics, because they hide execution "details", such as wtb and all multi-step edges. It would be interesting to investigate how their temporal interpretation [36] can be folded into the reproducibility semantics.

7.2. OPM and Semantic Web Technologies

McGrath and Futrelle [42] show limitations of SWRL and OWL in expressing OPM inferences. They did not consider property chains as we did in this paper. They propose a hybrid approach combining OWL, SWRL, RDF with extra tools to handle all OPM requirements. We are following a similar approach here.

Zhao's Open Provenance Model Vocabulary (OPMV) [58] aims to encode OPM in RDF, attempting to leverage existing vocabularies and ontologies such as Dublin Core, FOAF, and the Provenance Vocabulary [59], its predecessor. OPMV is work in progress, and does not support the full expressivity of OPM yet. It may benefit from some of the encoding of relations introduced by this paper.

A challenge brought by this work was putting Semantic Web technologies into action, in order to implement the reproducibility service. The challenge was both conceptual and implementational. First, there is not a single Semantic Web technology that allows us, today, to tackle all the issues we have encountered: *(i)* SPARQL does not support recursive queries over multi-step OPM edges; *(ii)* multi-step edges can be inferred by SWRL rules or OWL property chains; *(iii)* OPM n-ary relations are not naturally encoded in RDF; *(iv)* RDF Named graphs go some way capturing OPM features [51] such as account; *(v)* OPM completion rules require the inference of individuals, which can only be supported by some non-standardized extensions. Adopting all these technologies together result in a framework, whose semantics are not clear, and good properties such as decidability are lost. From a

practical point of view, at the time of writing, only a few reasoner could support the property chains described in this paper (TROWL and Pellet were successful, whilst FACT++ and HermiT failed). Pellet supported many of the above technologies, and was complemented by Java code, but performance of the overall approach remains a serious concern.

7.3. Reproducibility

In this paper, we have essentially regarded an OPM graph as a workflow, interpretable according to the reproducibility semantics. Therefore, this work bears relation with the workflow literature [60]. Techniques such a workflow abstraction and elaboration [61], scheduling [60], and collection-support are also applicable here [61].

The reproducibility semantics has been implemented using the OPM toolbox[13]. Its wrapping as a *reproducibility service* service remains to be undertaken. We envisage this service of being capable of taking OPM graphs, and reproducing their execution, timestamping and signing the resulting provenance trace, hereby confirming, in a non-forgeable way, that it is reducible. Such a service would need to be scalable, and is obviously a good candidate for parallelization.

In Section 3.1, we introduced dimensions to the problem of reproducibility: inputs, primitives, and results. They are captured by θ (inputs), $\mathcal{E}$, $\mathcal{V}^p$ (primitives), and $\mathcal{V}^a$ (results) in the reproducibility semantics. Figure 16 categorizes the various kinds of provenance-based reproducibility found in the literature according to these dimensions.

Inputs	Primitives	Results	
same	same	same	repetition (Miles *et al.* [25])
different	different	different	reenactment (Miles *et al.* [25])
same	same	—	repeatability (Bechhofer *et al.* [24])
same	same	same	reproducibilty (Bechhofer *et al.* [24])
same	mockup	same	replayability (Bechhofer *et al.* [24])
same	multiple variants	same	N-version reproducibility (Levince *et al.* [29])
—	different	—	upgrades (Koop *et al.* [34])
same	same	same	consistency (Cheney *et al.* [15])
different	different	different	fidelity (Cheney *et al.* [15])

Figure 16: Classification of Reproduciblity Approaches

In multiple publications [9, 21, 11], our preferred definition of provenance stated that it is an "explicit representation of the processes that *led* to that data item". In particular, we used the past tense to indicate that some processes *produced* a data item. In this paper, we looked at provenance as a *program*, which can be executed in the future. Hence, to accomodate this new perspective on provenance, we propose the following revised definition: provenance of a data item is an explicit representation of a computational activity, which in the past *led* to that data item, and which can be reproduced in the future, to derive possibly similar new data items. We are not the first to consider provenance as a program. Cheney [37] considers a subset of OPM as a series of nested `let` expressions, and Miles [62] introduces

POEM, a textual notation to create OPM graphs. None of them are investigating the problem of reproducibility.

Davidson *et al.* [63] study the problem of providing workflow data provenance without revealing the functionality of any module. To this end, they focus on the Secure View problem, which consists in ensuring privacy of all modules in a workflow, by hiding the smallest amount of data. The problem is established to be NP-hard, and they propose a polynomial-time approximation. We conjecture that there is a trade-off between full-reproducibility and full-privacy, since the reproducibility semantics expects primitive names (and implementations) to be shared. However, there may be a useful class of reproducibility behaviour, possibly similar to replayability [24], that can be performed on privacy-preserving provenance. Such an investigation has also to take into account the specific OPM graph structure, including *was-derived-from* edges, which partially reveal the private behaviour of processes.

8. Conclusion

Results reproducibility is crucial in scientific and nonscientific contexts to gain confidence in results and ensure their quality. It is particularly important when such results are derived from computations that make use of third-party services across the Web. In this context, ontology-based representations of provenance offer a uniform description of past executions across such services. Provenance is usually considered as a strong foundation for ensuring reproducibility, since its rich representation encompasses the necessary details to reproduce execution steps, and check all results, whether intermediary or final. However, ontology-based representations of provenance lack any formal link with execution, which makes it unclear why provenance is a sound foundation for reproducibility. We have tackled this problem by providing the reproducibility semantics for the Open Provenance Model; this semantics takes the form of a denotational semantics, which assigns well-formed OPM graphs to a function, which for some inputs, produces an OPM graph describing the reproduction of the result.

The benefits of the reproducibility semantics are multifold.

1. It provides a strong, technology-neutral, understanding of provenance by defining the mathematical meaning of OPM graphs. It allows us to define reproducibility formally, and classes of reproducible graphs. It is therefore the basis of a theory of provenance-based reproducibility.

2. It is a specification of a *reproducibility service*, which we envision as deployable on the Web or on Intranets. It allows users who publish results and their provenance, to check that their results are reproducible, and users who discover data, to verify how they were produced. Hence, it permits users to increase their confidence in such data.

3. From a methodological viewpoint, one always wonders what should be included in provenance. The semantics provides an algorithmic way to decide what needs to be recorded in provenance to ensure past computation reproducibility.

---

[13]

Our future work will address several concerns. From a theoretical perspective, we will aim to relax the topological constraints that we set on OPM graphs, and define a broader class of reproducible OPM graphs. Better and more scalable Semantic Web reasoning techniques are required to support the OPM specific inferences, and the necessary inferences required for reproducibility. Finally, we will seek to deploy a reproducibility service in the context of the Fourth Provenance Challenge, as a means to validate, automatically, the provenance traces produced by the participating teams.

9. Acknowledgement

References

[1] P. Vytelingum, T. D. Voice, S. D. Ramchurn, A. Rogers, N. R. Jennings, Agent-based micro-storage management for the smart grid, in: Autonomous Agents And MultiAgent Systems (AAMAS 2010), 2010. URL http://eprints.ecs.soton.ac.uk/18360/

[2] I. Foster, C. Kesselman (Eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufman Publishers, 1998.

[3] T. Hey, A. Trefethen, Cyberinfrastructure for e-science, Science 308 (5723) (2005) 817–821.

[4] T. Hey, S. Tansley, K. Tolle (Eds.), The Fourth Paradigm: Data-Intensive Scientific Discovery, Microsoft Research, Redmond, Washington, 2009. URL http://research.microsoft.com/en-us/collaboration/fourthparadigm/

[5] J. P. Mesirov, Accessible reproducible research, Science 327 (5964) (2010) 415–416. doi:10.1126/science.1179653. URL http://dx.doi.org/10.1126/science.1179653

[6] L. Philip, A. Chorley, J. Farrington, P. Edwards, Data provenance, evidence-based policy assessment, and e-social science, in: Third International Conference on e-Social Science, 2007. URL http://www.scientificcommons.org/40739576

[7] Nature Editorial, Illuminating the black box, Nature 6. URL http://dx.doi.org/10.1038/442001a

[8] J. Rees, Recommendations for independent scholarly publication of data sets, Tech. rep., Creative Commons Working Paper (Mar. 2010). URL http://neurocommons.org/report/data-publication.pdf

[9] L. Moreau, P. Groth, S. Miles, J. Vazquez, J. Ibbotson, S. Jiang, S. Munroe, O. Rana, A. Schreiber, V. Tan, L. Varga, The Provenance of Electronic Data, Communications of the ACM 51 (4) (2008) 52–58. URL http://www.ecs.soton.ac.uk/~lavm/papers/cacm08.pdf

[10] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, J. Myers, Examining the challenges of scientific workflows, IEEE Computer 40 (12) (2007) 26–34. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/MC.2007.421 doi:http://doi.ieeecomputersociety.org/10.1109/MC.2007.421.

[11] L. Moreau, The foundations for provenance on the web, Foundations and Trends in Web Science. URL http://eprints.ecs.soton.ac.uk/18176/

[12] P. Buneman, S. Khanna, W.-C. Tan, Why and Where: A Characterization of Data Provenance, in: Proceedings of 8th International Conference on Database Theory (ICDT'01), Vol. 1973 of Lecture Notes in Computer Science, Springer, London, UK, 2001, pp. 316–330. doi:http://dx.doi.org/10.1007/3-540-44503-X_20.

[13] Y. Cui, J. Widom, Practical lineage tracing in data warehouses, in: Proceedings of the 16th International Conference on Data Engineering (ICDE'00), San Diego, California, 2000, pp. 367–378. doi:http://dx.doi.org/10.1109/ICDE.2000.839437.

[14] I. Souilah, A. Francalanza, V. Sassone, A formal model of provenance in distributed systems, in: J. Cheney (Ed.), TAPP'09: First workshop on on Theory and practice of provenance, USENIX Association, San Francisco, CA, 2009. URL http://www.usenix.org/event/tapp09/tech/full_papers/souilah/souilah.pdf

[15] J. Cheney, U. A. Acar, A. Ahmed, Provenance traces (extended report), Tech. Rep. http://arxiv.org/abs/0812.0564v1, University of Edinburgh (Dec. 2008).

[16] Y. Zhao, M. Wilde, I. Foster, Applying the virtual data provenance model, in: L. Moreau, I. Foster (Eds.), Proceedings of the International Provenance and Annotation Workshop 2006 (IPAW'2006), Vol. 4145 of Lecture Notes in Computer Science, Springer, 2006, pp. 148–161. doi:http://dx.doi.org/10.1007/11890850_16.

[17] P. Missier, C. Goble, Workflows to open provenance graphs, round-trip, Future Generation Computer Systems.

[18] S. S. Sahoo, R. S. Barga, J. Goldstein, A. P. Sheth, Provenance algebra and materialized view-based provenance management, Tech. Rep. 76523/tr-2008-170, Microsoft Research (2008).

[19] S. S. Sahoo, A. Sheth, C. Henson, Semantic provenance for escience: Managing the deluge of scientific data, Internet Computing, IEEE 12 (4) (2008) 46–54. doi:http://dx.doi.org/10.1109/MIC.2008.86.

[20] O. Hartig, Provenance information in the web of data, in: Proceedings of the Linked Data on the Web Workshop (LDOW'09), Madrid, Spain, 2009. URL http://events.linkeddata.org/ldow2009/papers/ldow2009_paper18.pdf

[21] P. Groth, S. Miles, L. Moreau, A Model of Process Documentation to Determine Provenance in Mash-ups, Transactions on Internet Technology (TOIT) 9 (1) (2009) 1–31. doi:http://doi.acm.org/10.1145/1462159.1462162.

[22] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, J. V. den Bussche, The open provenance model core specification (v1.1), Future Generation Computer Systemshttp://dx.doi.org/http://dx.doi.org/10.1016/j.future.2010.07.005 doi:http://dx.doi.org/10.1016/j.future.2010.07.005.

[23] L. Moreau, B. Ludaescher, I. Altintas, R. S. Barga, S. Bowers, S. Callahan, G. Chin Jr., B. Clifford, S. Cohen, S. Cohen-Boulakia, S. Davidson, E. Deelman, L. Digiampietri, I. Foster, J. Freire, J. Frew, J. Futrelle, T. Gibson, Y. Gil, C. Goble, J. Golbeck, P. Groth, D. A. Holland, S. Jiang, J. Kim, D. Koop, A. Krenek, T. McPhillips, G. Mehta, S. Miles, D. Metzger, S. Munroe, J. Myers, B. Plale, N. Podhorszki, V. Ratnakar, E. Santos, C. Scheidegger, K. Schuchardt, M. Seltzer, Y. L. Simmhan, C. Silva, P. Slaughter, E. Stephan, R. Stevens, D. Turi, H. Vo, M. Wilde, J. Zhao, Y. Zhao, The first provenance challenge, Concurrency and Computation: Practice and Experience 20 (5) (2008) 409–418. doi:http://dx.doi.org/10.1002/cpe.1233.

[24] S. Bechhofer, D. D. Roure, M. Gamble, C. Goble, I. Buchan, Research objects: Towards exchange and reuse of digital knowledge, in: The Future of the Web for Collaborative Science, 2010. doi:http://dx.doi.org/10.1038/npre.2010.4626.1.

[25] S. Miles, P. Groth, M. Branco, L. Moreau, The requirements of using provenance in e-science experiments, Journal of Grid Computing 5 (1) (2007) 1–25. doi:http://dx.doi.org/10.1007/s10723-006-9055-3.

[26] J. B. Buckheit, D. L. Donoho, Wavelap and reproducibile research, Tech. rep., Stanford University (1995).

[27] M. Schwab, M. Karrenbach, J. Claerbout, Making scientific computations reproducible, Computing in Science and Engineering 2 (6) (2000) 61–67. doi:http://dx.doi.org/10.1109/5992.881708.

[28] L. Pan, L. M. Batten, Reproducibility of digital evidence in forensic investigations, in: Digital Forensic Research Workshop (DFRWS'05), 2005. URL http://www.dfrws.org/2005/proceedings/pan_reproducibility.pdf

[29] B. N. Levine, M. Liberatore, Dex: Digital evidence provenance supporting reproducibility and comparison, in: Proceedings of the Digital Forensic Research workshop(DFRWS'09), 2009. http://dx.doi.org/http://dx.doi.org/10.1016/j.diin.2009.06.011 doi:http://dx.doi.org/10.1016/j.diin.2009.06.011.

[30] V. Stodden, Scientific reproducibility and software (Feb. 2010).
URL http://www.stanford.edu/~vcs/talks/VictoriaStoddenICES-Austin-Feb2010.pdf

[31] S. B. Davidson, J. Freire, Provenance and scientific workflows: challenges and opportunities, in: SIGMOD Conference, 2008, pp. 1345–1350. doi:http://doi.acm.org/10.1145/1376616.1376772.

[32] O. Biton, S. Cohen-Boulakia, S. B. Davidson, C. S. Hara, Querying and managing provenance through user views in scientific workflows, in: International Conference Data Engineering (ICDE'08), IEEE Computer Society, Los Alamitos, CA, USA, 2008, pp. 1072–1081. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/ICDE.2008.4497516 doi:http://doi.ieeecomputersociety.org/10.1109/ICDE.2008.4497516.

[33] C. Silva, J. Freire, S. P. Callahan, Provenance for visualizations: Reproducibility and beyond, Computing in Science and Engineering 9 (5) (2007) 82–89. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/MCSE.2007.106 doi:http://doi.ieeecomputersociety.org/10.1109/MCSE.2007.106.

[34] D. Koop, C. E. Scheidegger, J. Freire, , C. T. Silva, The provenance of workflow upgrades, in: International Provenance and Annotation Workshop (IPAW '10), 2010.

[35] L. Moreau, N. Kwasnikowska, J. V. den Bussche, The foundations of the open provenance model, Tech. rep., University of Southampton (Apr. 2009).
URL http://eprints.ecs.soton.ac.uk/17282/

[36] N. Kwasnikowska, L. M. J. V. den Bussche, A formal account of the open provenance model, Tech. rep., University of Southampton (Apr. (to appear 2010)).

[37] J. Cheney, Causality and the semantics of provenance, CoRR - Computing Research Repository abs/1006.1429.
URL http://arxiv.org/abs/1006.1429

[38] J. Y. Halpern, J. Pearl, Causes and Explanations: A Structural-Model Approach. Part I: Causes, Br J Philos Sci 56 (4) (2005) 843–887. http://arxiv.org/abs/http://bjps.oxfordjournals.org/cgi/reprint/56/4/843.pdf arXiv:http://bjps.oxfordjournals.org/cgi/reprint/56/4/843.pdf, doi:10.1093/bjps/axi147.

[39] J. Y. Halpern, J. Pearl, Causes and Explanations: A Structural-Model Approach. Part II: Explanations, Br J Philos Sci 56 (4) (2005) 889–911. http://arxiv.org/abs/http://bjps.oxfordjournals.org/cgi/reprint/56/4/889.pdf arXiv:http://bjps.oxfordjournals.org/cgi/reprint/56/4/889.pdf, doi:10.1093/bjps/axi148.

[40] P. Missier, D. Turi, C. Goble, T. Oinn, D. D. Roure, Taverna workflows: Syntax and semantics, in: IEEE International Conference on e-Science and Grid Computing, IEEE Press, 2007, pp. 441–448.
URL http://eprints.ecs.soton.ac.uk/14700/

[41] L. Moreau, S. Miles, P. Missier, P. Groth. [link].
URL http://openprovenance.org/model/opm.owl

[42] R. E. McGrath, J. Futrelle, Reasoning about provenance with owl and swrl rules, in: AAAI Spring Symposium 2008 "AI Meets Business Rules and Process Management", 2008.
URL http://cet.ncsa.uiuc.edu/publications/mcgrath-futrelle-rules.pdf

[43] W3C Incubator Activity, Provenance incubator group charter (Sep. 2009).
URL http://www.w3.org/2005/Incubator/prov/charter

[44] S. Sahoo, P. Groth, O. Hartig, S. Miles, S. Coppens, J. Myers, Y. Gil, L. Moreau, J. Zhao, M. Panzer, D. Garijo, Provenance vocabulary mappings, Tech. rep., W3C (2010).

[45] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, M. Wilde, Swift: Fast, reliable, loosely coupled parallel computation, in: IEEE Congress on Services, 2007, pp. 199 –206. doi:http://dx.doi.org/10.1109/SERVICES.2007.63.

[46] E. Prud'hommeaux, A. Seaborne (Eds.), SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/ (Jan. 2008).

[47] J. Golbeck, J. Hendler, A semantic web approach to the provenance challenge, Concurrency and Computation: Practice and Experience 20 (5) (2008) 431–439. doi:http://dx.doi.org/10.1002/cpe.1238.

[48] J. Kim, E. Deelman, Y. Gil, G. Mehta, V. Ratnakar, Provenance trails in the wings/pegasus system, Concurrency and Computation: Practice and Experience 20 (5) (2008) 587–597. doi:http://dx.doi.org/10.1002/cpe.1228.

[49] L. Ding, J. Michaelis, J. McCusker, D. L. McGuinness, Linked provenance data: A semantic web-based approach to interoperable workflow traces, Future Generation Computer Systems.

[50] J. J. Carroll, C. Bizer, P. Hayes, P. Stickler, Named graphs, provenance and trust, in: WWW '05: Proceedings of the 14th international conference on World Wide Web, ACM Press, New York, NY, USA, 2005, pp. 613–622. doi:http://doi.acm.org/10.1145/1060745.1060835. URL http://www2005.org/cdrom/docs/p613.pdf

[51] T. Gibson, K. Schuchardt, E. Stephan, Application of named graphs towards custom provenance views, in: J. Cheney (Ed.), TAPP'09: First workshop on on Theory and practice of provenance, USENIX Association, San Francisco, CA, 2009.

[52] S. Miles, S. C. Wong, W. Fang, P. Groth, K.-P. Zauner, L. Moreau, Provenance-based validation of e-science experiments, Web Semantics: Science, Services and Agents on the World Wide Web 5 (1) (2007) 28–38. doi:doi:10.1016/j.websem.2006.11.003.

[53] P. Groth, L. Moreau, Representing distributed systems using opm, Future Generation Computer Systems.

[54] J. C. Reynolds, The Discoveries of Continuations, Lisp and Symbolic and Computation, Special Issue on Continuations 6 (3/4) (1993) 233–248.

[55] D. Hull, R. Stevens, P. Lord, C. Wroe, C. Goble, Treating shimantic web syndrome with ontologies, in: University, Milton Keynes, UK, 2004.

[56] Y. Zhao, J. Dobson, I. Foster, L. Moreau, M. Wilde, A Notation and System for Expressing and Executing Cleanly Typed Workflows on Messy Scientific Data, Sigmod Record 34 (3) (2005) 37–43.
URL http://www.ecs.soton.ac.uk/~lavm/papers/sigmod05.pdf

[57] N. Kwasnikowska, J. Van Den Bussche, Multiple and hierarchical refinement in opm (Aug. 2009).
URL http://twiki.ipaw.info/bin/view/OPM/ChangeProposalMultipleHierarchicalRefinement

[58] J. Zhao, Open provenance model vocabulary specification.
URL http://open-biomed.sourceforge.net/opmv/ns.html

[59] O. Hartig, J. Zhao, Guide to the provenance vocabulary (Jul. 2010).
URL http://sourceforge.net/apps/mediawiki/trdf/index.php?title=Guide_to_the_Provenance_Vocabulary

[60] I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows for e-Science, Vol. XXII, Springer, 2007.

[61] Y. Gil, P. Groth, V. Ratnakar, C. Fritz, Expressive reusable workflow templates, 2009, pp. 344 –351. doi:http://dx.doi.org/10.1109/e-Science.2009.55.

[62] S. Miles, The poem format for opm (Sep. 2010).
URL http://twiki.ipaw.info/bin/view/OPM/POEMFormat

[63] S. B. Davidson, S. Khanna, D. Panigrahi, S. Roy, Preserving Module Privacy in Workflow Provenance, ArXiv e-printshttp://arxiv.org/abs/1005.5543 arXiv:1005.5543.
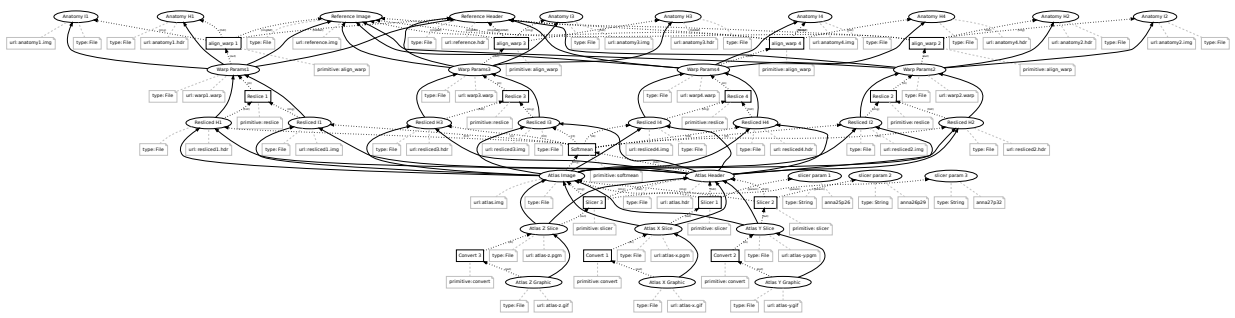
Appendix A. Figure

Figure A.17: PC1 OPM graph (for illustrative purpose only)