The Foundations of the Open Provenance Model

Luc Moreau	(U. of Southampton, UK)
Natalia Kwasnikowska	(U. Hasselt, Belgium)
Jan Van den Bussche	(U. Hasselt, Belgium)

April 20, 2009

Abstract

The Open Provenance Model (OPM) is a community-driven data model for Provenance that is designed to support inter-operability of provenance technology. Underpinning OPM, is a notion of directed acyclic graph, used to represent data products and processes involved in past computations, and causal dependencies between these. The Open Provenance Model was derived following two "Provenance Challenges", international, multi-disciplinary activities trying to investigate how to exchange information between multiple systems supporting provenance and how to query it. The OPM design was mostly driven by practical and pragmatic considerations, and is being tested in a third Provenance Challenge, which has just started. The purpose of this paper is to investigate the theoretical foundations of this data model. The formalisation consists of a set-theoretic definition of the data model, a definition of the inferences by transitive closure that are permitted, a formal description of how the model can be used to express dependencies in past computations, and finally, a description of the kind of time-based inferences that are supported. A novel element that OPM introduces is the concept of an account, by which multiple descriptions of a same execution are allowed to co-exist in a same graph. Our formalisation gives a precise meaning to such accounts and associated notions of alternate and refinement.

Warning *It was decided that this paper should be released as early as possible since it brings useful clarifications on the Open Provenance Model, and therefore can benefit the Provenance Challenge 3 community. The reader should recognise that this paper is however an early draft, and several sections are incomplete. Additionally, figures rely on colours but these may be difficult to read when printed in a black and white. It is advisable to print the paper in colour.*

1 Introduction

In the fine arts and in digital libraries, provenance respectively refers to the documented history of an art object, or the documentation of processes in a digital object's life cycle [27]. The "e-science community" [45] also shows a growing interest in provenance since it is crucial to ensure reproducibility of scientific experiments [18].

Over the years, a series of systems have been developed to track and exploit provenance in many different ways [30]. Following a discussion session on standardisation at the *International Provenance and Annotation Workshop* (IPAW'06) [33, 5], a consensus began to emerge, whereby the provenance research community needed to understand better the capabilities of the different systems, the representations they used for provenance, their similarities, their differences, and the rationale that motivated their designs.

Hence, the first and second Provenance Challenges [44, 10] were set up in order to provide a forum for the community to understand the capabilities of different provenance systems and the expressiveness of their provenance representations. The participating teams [36] ran an agreed Functional Magnetic Resonance Imaging workflow, exported provenance information, and implemented pre-identified "provenance queries" asking typical questions about past execution of the workflow. As discussions indicated that there was substantial agreement on a core representation of provenance, the *Open Provenance Model* (OPM) [34] (subsequently revised by a broader committee [38]) was put forward as a data model by which systems can exchange provenance information. Such agreed model is being the focus of a Third Provenance Challenge [48], where its suitability is practically evaluated by using it as the agreed format for provenance information exchange.

From the outset, because precision matters when systems have to inter-operate, OPM was described in a technology-agnostic manner, both in natural language and using a formal notation. The key structure defined in the Open Provenance Model is an *OPM graph*, a directed acyclic graph aimed at representing causal, data and control dependencies of past computations. The specification also outlined the kind of inferences that would be permitted over such graphs, but it also identified various challenges in maintaining consistency. These uncertainties are not desirable because for systems to be able to inter-operate really, implementers would have to agree on separate sets of conventions.

Furthermore, a novel element that OPM introduces is the concept of an account, a description of a past execution; multiple accounts are allowed to co-exist in a same graph, hereby representing different observations or explanations at different levels of abstraction of a same execution. The foundational underpinning of accounts was mostly absent from the original specification. In this paper, we provide a precise meaning for such accounts and associated notions of alternate and refinement.

Hence, the contribution of this paper is fivefold:

1. A full formalisation of the Open Provenance Model that clarifies a number of issues in the first version of the model.

2. The introduction of the concept of APath underpinning the permitted inferences over OPM graphs.

3. A formal definition of the notions of account alternate and refinement.

4. A characterisation of the suitability of OPM to express a description of past execution.

5. Finally, a characterisation of the kind of time inference allowed by the model.

2 OPM Overview

The purpose of this Section is to overview the Open Provenance Model [38] and provide intuition about its key components. The rest of the paper is then dedicated to its formalisation. This overview relies upon the OPM graphical representation, which we now introduce. The core data structure that OPM supports is an OPM graph, consisting of nodes and edges; we first focus on the former. Physical objects or data products, referred to as *artifacts*, are represented as ellipsis, whereas activities that produce and consume artifacts, also known as *processes*, are represented by rectangles. These entitities constitute the nodes of OPM graphs, and are annotated with their respective identifiers. Edges in OPM graphs are directional, from effect to cause, explaining how an effect resulted from a cause. Graphs should be read from bottom to top, explaining how effects are repeatedly derived from causes. An example of OPM graph is displayed in Figure 1, illustrating how a cake resulted from a baking process and was made of several ingredients.

In Figure 1, we distinguish three types of edges. An artifact–process edge indicates that the artifact *was generated by* the process, e.g. cake–bake. A process–artifact edge indicates that the process *used* the artifact, e.g. bake–2 eggs. An artifact–artifact edge states that the former artifact *was derived from* the latter, e.g. cake–2 eggs. There is a fourth kind of edge, illustrated in Figure 2: a process–process edge states that the former process *was informed* by the latter. We note that all edges are characterised by the past tense, to denote that they refer to a past execution. This point is quite crucial: the aim of OPM graphs is to represent executions that took place in the past (up to the present); OPM is not seeking to express how future computations may run. A point to note is that artifacts are *instantaneous* pieces of data or state, whereas processes have a duration.

In fact, OPM edges have a more precise definition, based on event ordering.

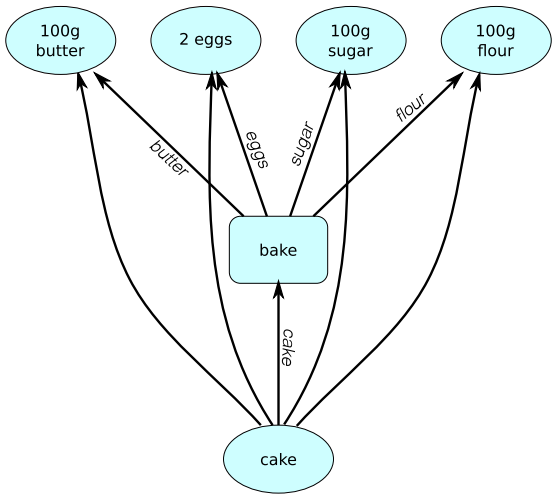


Figure 1: Baking Cake

- An artifact was generated by a process, if the process had to initiate its execution, for the artifact to be produced.
- A process used an artifact, if the artifact had to be present for the process to be able to complete.
- An artifact was generated from another artifact if the latter had to exist, for the former to be generated.
- Finally, a process was informed by another process if the latter had to initiate its execution for the former to complete.

The event orderings explain why edges 'used' and 'generated by' are not enough to explain dependencies between outpus and inputs. To express that 'flour' was actually involved in the production of 'cake', we use an explicit artifact to artifact edge.

In OPM, a process may be connected to several artifacts by 'used' edges. To distinguish them, such edges are annotated by a *role* indicating the functions of the artifacts in that process. When several artifacts are connected to a same process by multiple used edges, all of them were required for the process to complete. Similar conventions apply for edges between artifacts and processes. Illustrative roles are displayed in Figure 1 only; roles are not displayed in the following figures to avoid cluttering them.

OPM graphs are intended to be used to describe past executions. Such descriptions may be provided at different levels of abstraction, for instance, higher-level business-logic oriented description for end-users, whereas lowel-level one for systems people. Descriptions may even be produced by different observers.

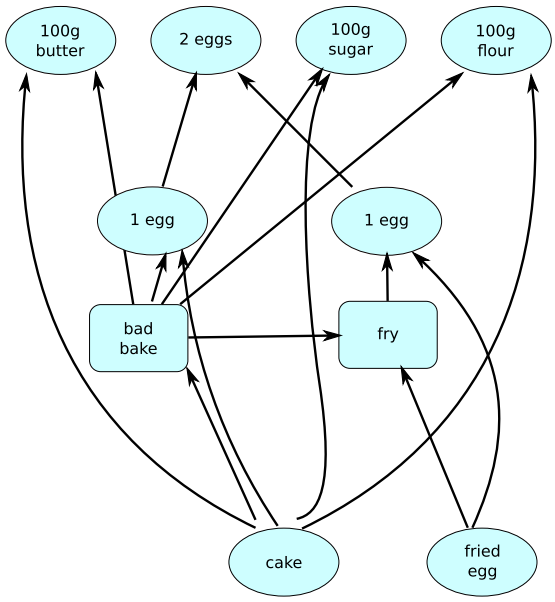


Figure 2: Baking a Bad Cake

While Figure 1 captures the provenance of a cake, as described by a waiter to a customer, the baker can provide a different interpretation of what occurred, as illustrated in Figure 2. The latter description indicates that one of the eggs was actually fried, while the baked cake actually used a single egg. Both descriptions pertain to a same execution, but they simply provide different information about it. Each description is referred to as an *account*. Hence, OPM allows for both accounts to be combined in a single graph, as illustrated by Figure 3.

We use colouring to encode account membership. The original baking activity is described with orange edges, whereas the baking/frying activity is encoded with black edges. Blue is used for edges belonging to both activities. Account membership of nodes is "inherited" from the edges they are incident to.

Interestingly, properties about accounts can be expressed. The orange and black accounts are said to *overlap* because they share common nodes (the original input ingredients and the final cake). More precisely, these accounts are *alternate* versions of a same story, namely that the cake was derived from '2 eggs' (and the other ingredients). In the black account, this dependency is inferred by transitivity of (cake, 1 egg) and (1 egg, 2 eggs). Furthermore, the black account is a *refinement* of the orange accout, because all data derivation of the orange account also hold in the black account.

Finally, there are further constraints over OPM graphs to make them legal. Within an account, we require an artifact to be generated by a single process, to ensure that we do not have conflicts about the artifact's origin. Also, within an account, we require the graph to be acyclic, so that it accurately expresses causal

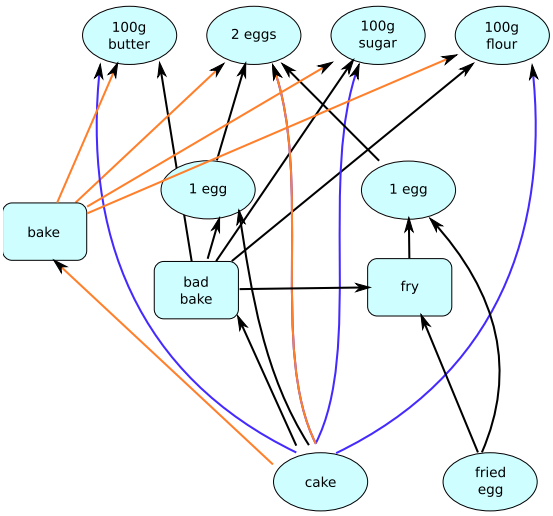


Figure 3: Original and Refined Account of Baking Process

dependencies between processes and artifacts. Having overviewed the key aspects of the Open Provenance Model, we are now ready to undertake its formalisation.

3 Formalisation

3.1 Definition

Figure 4 provides a definition of the Open Provenance Model, which we now paraphrase. The set *OPMGraph*, the set of all possible OPM graphs, is expressed as a dependent type[1]. In addition to sets of process identifiers, artifact identifiers, roles, account, domain values, and Edges an OPM graph also contains an account mapping function, a value mapping function and account relation declarations. Nodes are identified by labels, and can be of two[2] sorts: processes or artifacts. Edges can connect any two nodes, and hence can be of four sorts as per the dependencies introduced in Section 2: *Used*, *WasGeneratedBy*, *WasDerivedFrom*, *WasInformedBy*. Edges of the first two sorts are annotated with a role characterising the nature of the dependency.

[1]We use a dependent type [40, 3] to characterise the constraints that exist between the components of an OPM graph. Specifically, we use a dependent sum type, which extends the usual Cartesian product to model pairs where the type of the second component depends on the first. In our notation, all components upon which types depend are made explicit as parameters as the type OPMGraph. Here, for instance, the type of *ValueOf* depends on *Node* (defined as $ArtifactId \cup ProcessId$ and *Account*).

[2]The original OPM specification [38] introduces a notion of agent that we have not formalised in this document.

An OPMGraph is a dependent type defined as follows

$$\begin{aligned} & OPMGraph(ProcessId, ArtifactId, Role, Account, Value, Edge) \\ = \quad & AccountOf \times ValueOf \times AccountRelation \end{aligned}$$

where

$$\begin{array}{rcl} ProcessId & : & \textit{a primitive set of process identifiers} \\ ArtifactId & : & \textit{a primitive set of artifact identifiers} \\ & & ProcessId \cap ArtifactId = \emptyset \\ Role & : & \textit{a primitive set of roles} \\ Account & : & \textit{a primitive set of accounts} \\ Value & : & \textit{an application-specific set of values} \\ Node & = & ProcessId \cup ArtifactId \\ Used & = & ProcessId \times Role \times ArtifactId \\ WasGeneratedBy & = & ArtifactId \times Role \times ProcessId \\ WasDerivedFrom & = & ArtifactId \times ArtifactId \\ WasInformedBy & = & ProcessId \times ProcessId \\ AccountOf & = & Node \cup Edge \rightarrow \mathbb{P}(Account) \\ ValueOf & = & Node \times Account \rightarrow \mathbb{P}(Value) \\ Overlap & = & \mathbb{P}(Account) \\ Alternate & = & \mathbb{P}(Account) \\ Refinement & = & Account \times Account \\ AccountRelation & = & \mathbb{P}(Overlap) \times \mathbb{P}(Alternate) \times \mathbb{P}(Refinement) \\ \text{with constraint} & & \\ Edge & \subseteq & Used \cup WasGeneratedBy \cup WasDerivedFrom \\ & & \cup\ WasInformedBy \end{array}$$

Concretely, an instance of an OPM graph is therefore a tuple of 9 elements

$$gr = \langle ProcessId, ArtifactId, Role, Account, Value, Edge, accountOf, valueOf, aRel \rangle$$

where

$$\begin{array}{rcl} accountOf & \text{is of type} & AccountOf \\ valueOf & \text{is of type} & ValueOf \\ aRel & \text{is of type} & AccountRelation \end{array}$$

Figure 4: Timeless Causality Graph Data Model

In OPM, every node and edge can be associated with 0, 1, or more accounts, which is captured by the *AccountOf* function type. We note that the domain of a function of type *AccountOf* is exactly the set of nodes and edges defined for the current graph. Likewise, nodes can be associated with domain specific information, which is beyond the scope of this formalisation, but this association is expressed by the *ValueOf* function type. (We note that different accounts can associate a same node to different values.) Finally, a set of declarations can assert the accounts that overlap, are alternate or are refinements of each other.

Given instance of an OPM graph represented by the tuple of 9 elements

$$gr = \langle PId, AId, R, Acc, Val, E, aOf, vOf, aRel \rangle \in OPMGraph,$$

we introduce a convenient notation for accessing the components of gr:

$$\begin{aligned}
ArtifactId^{gr} &= AId \\
ProcessId^{gr} &= PId \\
Node^{gr} &= ArtifactId^{gr} \cup ProcessId^{gr} \\
Edge^{gr} &= E \\
Role^{gr} &= R \\
Account^{gr} &= Acc \\
accountOf^{gr} &= aOf \\
valueOf^{gr} &= vOf \\
accountRelation^{gr} &= aRel
\end{aligned}$$

Likewise, we introduce notations $Used^{gr}$, $WasGeneratedBy^{gr}$, $WasDerivedFrom^{gr}$, $WasInformedBy^{gr}$ to denote the various subsets of edges of a graph gr. For instance, $Used^{gr} = \{e | e \in Edge^{gr} \ \wedge \ e \in Used(ProcessId^{gr}, Role^{gr}, ArtifactId^{gr})\}$. Hence, for any graph gr, the following equality holds:

$$Edge^{gr} = Used^{gr} \cup WasGeneratedBy^{gr} \cup WasDerivedFrom^{gr} \cup WasInformedBy^{gr}$$

The above notations will prove convenient when referring to several OPM graphs at the same time.

For a given OPM graph gr, *artifacts* and *processes* of gr are respectively elements of the sets $ArtifactId^{gr}$ and $ProcessId^{gr}$. For any $a \in ArtifactId^{gr}$, the account membership of a is $accountOf^{gr}(a)$. For any $p \in ProcessId^{gr}$, the account membership of p is $accountOf^{gr}(p)$.

Note that any edge $e \in Edges^{gr}$ for an OPM Graph gr, either belongs to

$$Used^{gr} \cup WasGeneratedBy^{gr}$$

and is of the form $e = \langle x_1, r, x_2 \rangle$ with r some role, or belongs to

$$WasDerivedFrom^{gr} or \, WasInformedBy^{gr}$$

and is then of the form $e = \langle x_1, x_2 \rangle$. In both cases, we introduce the notation *effect*(e) to denote the effect node of e, i.e. x_1, and *cause*(e) to denote the cause node of e, i.e. x_2. We also say that x_1 and x_2 are *incident* to e, and denote this by *isIncident*(x_i, e) for $i = 1, 2$. Note that edges are considered to be equal simply if they are equal in the mathematical sense, i.e. they are the same tuple.

To ensure that cause and effect of an edge belong to the same account as the edge, a node "inherits" the accounts defined by the edges it is cause or effect of. Formally, this is expressed as follows.

Definition 1 (Effective Account) *For a given OPM graph gr, we define the function*

$$\mathit{effectiveAccountOf}^{gr} : \mathit{Node}^{gr} \cup \mathit{Edge}^{gr} \to \mathbb{P}(\mathit{Account})$$

as follows:

- *If $x \in \mathit{Node}^{gr}$, then*

$$\mathit{effectiveAccountOf}^{gr}(x) = \mathit{accountOf}^{gr}(x) \cup \bigcup \{\mathit{accountOf}^{gr}(e) \mid e \in \mathit{Edge}^{gr} \text{ and } \mathit{isIncident}(x, e)\}$$

- *If $x \in \mathit{Edge}^{gr}$, then*

$$\mathit{effectiveAccountOf}^{gr}(x) = \mathit{accountOf}^{gr}(x)$$

As suggested by Section 2, we can consider an account as a colouring of a graph. An *account view* is the subgraph obtained by selecting nodes and edges with a given colour. Formally, this operation is defined as follows.

Definition 2 (Account View) *For a given OPM graph gr and an account α, we define the* account view *of gr according α, denoted by $\mathit{view}(gr, \alpha)$, as the OPM graph with the following constituents:*

- $\mathit{Node}^{view(gr,\alpha)} = \{n \in \mathit{Node}^{gr} \mid \alpha \in \mathit{effectiveAccountOf}^{gr}(n)\}$
- $\mathit{Edge}^{view(gr,\alpha)} = \{e \in \mathit{Edge}^{gr} \mid \alpha \in \mathit{effectiveAccountOf}^{gr}(e)\}$
- $\mathit{accountOf}^{view(gr,\alpha)} = \lambda x.\{\alpha\}$
- $\mathit{valueOf}^{view(gr,\alpha)} = \lambda x.\lambda \alpha'.$ *if* $(x \in \mathit{Node}^{view(gr,\alpha)} \wedge \alpha' = \alpha)$ *then* $\mathit{valueOf}^{gr}(x, \alpha')$ *else* $\emptyset$
- $\mathit{accountRelation}^{view(gr,\alpha)} = \{\emptyset, \emptyset, \emptyset\}$.

In this definition, *accountOf* for a view is the constant function that maps each node to α; we can see from this definition that we simply ignore any information pertaining to other accounts. The *valueOf* function in the view is the restriction of $valueOf^{gr}$ on $Node^{view(gr,\alpha)} \times \{\alpha\}$. This definition can be generalised to a view over *several* accounts by using the union operator defined in Section 3.6.

An account view is considered legal if it does not contain any cycle, and if it contains at most one $\langle a, r, p\rangle$ edge (of sort *WasGeneratedBy*) for any artifact a.

Definition 3 (Legal Account View) *For an OPM graph gr and an account $\alpha \in Account$, the account view $view(gr, \alpha)$ is considered to be* legal *if*

1. *it is acyclic,*
2. *if for any edges $\langle a_1, r_1, p_1\rangle$, $\langle a_1, r_2, p_2\rangle \in Edge^{gr}$, then $\langle a_1, r_1, p_1\rangle = \langle a_1, r_2, p_2\rangle$.*
3. *for any $n \in Node^{gr}$, $valueOf^{gr}(n, \alpha)$ is not empty, if α is in $effectiveAccountOf^{gr}(n)$.*

By extension, a graph is said to be legal if all its account views are legal.

Semantically, a legal account view is important since it provides a valid causality graph (without cycle) and where at most one edge explains how each artifact is derived from a process.

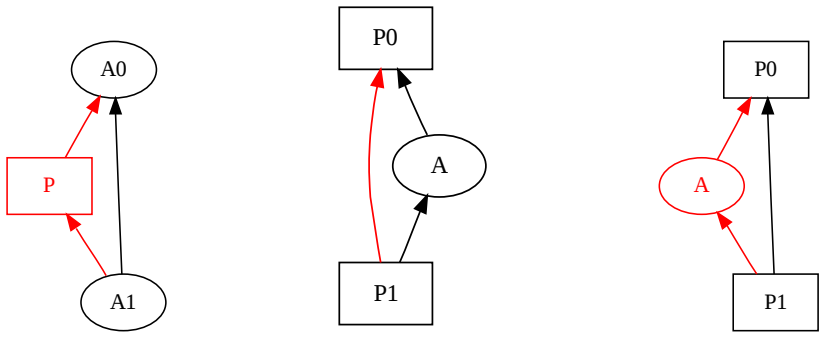
3.2 Definitional Constraints

Whilst Section 3.1 introduces a data model for OPM graphs, this syntactic framework does not enforce the meaning of edges intuitively introduced in Section 2. To remedy this problem, at least partially, in this section, we introduce a set of rules providing an *explanation* for some of the OPM edges. Such explanation takes the form of a subgraph expansion. First, we present the rules graphically, before formalising them.

Figure 5 graphically presents three rules, using the following colour convention, for the sake of conciseness. In black, we find patterns of OPM graphs. In red, we represent the graph elements that can be added to a graph, by application of the rule. We now describe these rules.

Rule (Completion) states that if an artifact A_1 was derived from an artifact A_0, then there exists a process P such that A_1 was generated by P and P used A_0. The rule does not specify which process was involved, nor the respective roles of these artifacts with regard to this process, but it states that such a process P existed.

Rule (Equivalence 1) states that if there is an artifact that was generated by a process P_0 and used by a process P_1, then P_1 was informed by P_0. Rule (Equivalence 2) is the opposite to (Equivalence 1). If a process P_1 was informed by P_0, then there exists an artifact that was used by the former and generated



(Completion) (Equivalence 1) (Equivalence 2)

Figure 5: Definitional Constraints

by the latter; (Equivalence 2) however does not specify which artifact A was involved, nor its roles with regard to the two processes.

We note that rules (Equivalence 1) and (Equivalence 2) really state an equivalence: a *WasInformedBy* edge contains as much information as the presence of an Artifact and associated *Used* and *WasGeneratedBy* edges. There is however no opposite to (Completion), since replacing a *WasDerivedFrom* edge by a process is a *lossy* transformation. Indeed, let us consider the hypothetical opposite transformation: the presence of a process does not imply the existence of a *WasDerivedFrom* edge: for instance, when artifact A_0 is used after artifact A_1 was generated, no such *WasDerivedFrom* dependency can possibly exist.

Definitional constraints should be considered in conjunction with the legal notion of account view (see Definition 2). For instance, the left hand of Figure 6 depicts an artifact A_2 derived from two artifacts A_0 and A_1. Rule (Completion) can be applied twice here, but the legality constraint requires an artifact to be generated from a single process in a given view. Hence, the right hand side of Figure 6 displays the *only possible* completion, where introduced process P used both A_0 and A_1, and generated A_2.

Likewise application of rule (Completion) to the left graph of Figure 7 entails that there is a process that used A_0 and that generated A_1, but the legality constraint implies that this process is P. Hence, we can derive that P used A_0. This is a derived (Completion) rule, which is a consequence of the (Completion) rule and the legality definition of account views (Definition 3).

(Completion) rule and (Equivalence 2), which both introduce a new node in a graph, can result in some uncertainty as illustrated by Figures 8 and 9. In Figure 8, it is unknown whether the two processes P_0 and P_1 introduced by the (Completion) rule are identical. Likewise, in Figure 9, it is not known whether

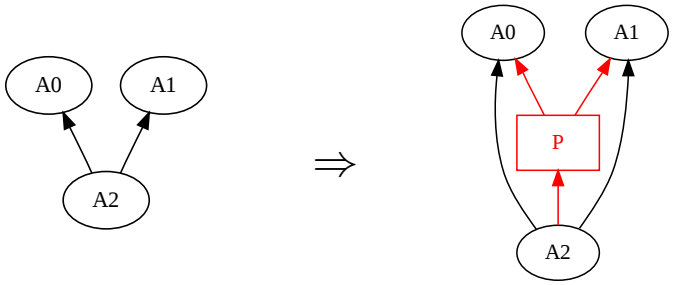


Figure 6: Process insertion

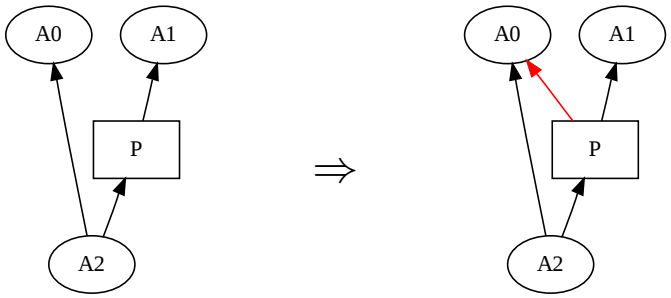


Figure 7: Derived (Completion) Rule

the two artifacts A_0 and A_1 introduced by Equivalence 2 are the same.

Hence, repeated application of definitional constraints can result in multiple graphs. While this kind of uncertainty may be regarded by some as problematic, we will establish that the uncertainty due to definitional constraints does not affect inferences that OPM graphs allow us to perform. Specifically and intuitively, the uncertainty in the topology does not affect the chains of artifacts underpinning OPM graphs. We establish such a property formally in the paper, after we introduce the inferences that are permitted by OPM graphs.

Beforehand, in Figure 10, we summarise the definitional constraints introduced in this section. The constraints are expressed with judgement, where premises are expressed above a bar, and consequents below it. Importantly, the rules make it explicit how accounts can be assigned to the created edges. (They are implicitly assigned to nodes with the *effectiveAccountOf* function.) To facilitate their reading, equations are given the same name as the corresponding rules

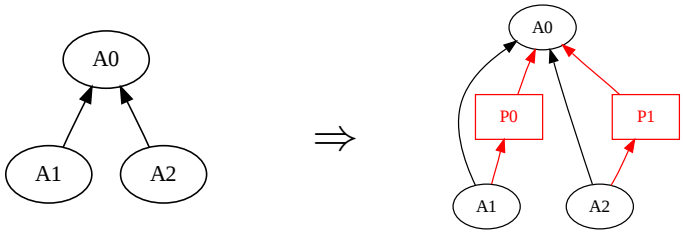


Figure 8: Uncertainty: Is $P_0 = P_1$ or $P_0 \neq P_1$?

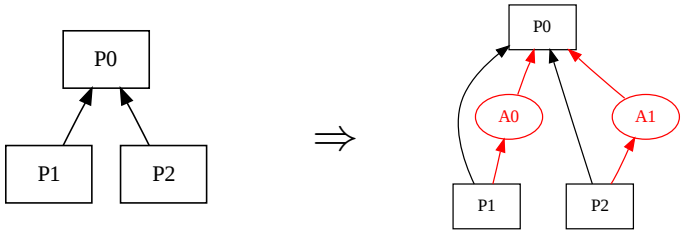


Figure 9: Uncertainty: Is $A_0 = A_1$ or $A_0 \neq A_1$?

in Figure 5.

For equations (Equivalence 2) and (Completion), we note that the new edges belong to the same account as the edge that is "expanded". For equation (Equivalence 1), the two existing edges in the premise of the judgement can be member of different accounts. There are two possibilities for the account membership of the resulting *WasInformedBy* edge. In the first case, a conservative inference is that the edge belongs to the intersection of both sets of accounts: its interpretation is that if both accounts state it, then the inference is valid. In the second case, a more permissive inference is that the edge belongs to the union of the account sets, meaning that it was inferred because we relied on several accounts to reach this knowledge. There is no right or wrong solution: application domain designers will have to decide how (Equivalence 1) should proceed according to their needs. It should be noted that the conservative and permissive interpretations of (Equivalence 1) only apply to *inferred* edges. In the permissive interpretation, if an inferred edge belongs to accounts α_1 and α_2, it does not mean that such an inference could have been carried out in α_1 or α_2 alone. Instead, it means that both the knowledge of α_1 and α_2 was necessary to infer this edge. We note that such an interpretation of inferred edges, differs from the interpretation of

For any OPM graph gr, the application of a definition constraint results in a new graph gr' that extends gr according to the following rules. The resulting graph gr' is said to be an *expansion* of gr.

$$\frac{\langle p_2, r_3, a_2\rangle \in \mathit{Used}^{gr} \ \wedge \ \langle a_2, r_2, p_1\rangle \in \mathit{WasGeneratedBy}^{gr}}{\langle p_2, p_1\rangle \in \mathit{WasInformedBy}^{gr'}} \qquad \text{(Equivalence 1)}$$

$$\begin{aligned} &\text{with } \mathit{accountOf}^{gr'}(\langle p_2, p_1\rangle) \\ &\quad = \ \mathit{accountOf}^{gr}(\langle p_2, r_3, a_2\rangle) \ \bigcirc \ \mathit{accountOf}^{gr}(\langle a_2, r_2, p_1\rangle) \\ &\qquad \text{where symbol } \bigcirc \text{ denotes } \cap \text{ or } \cup. \end{aligned}$$

$$\frac{\langle p_2, p_1\rangle \in \mathit{WasInformedBy}^{gr}}{\begin{array}{ll}\exists a_2, r_2, r_3, & \langle p_2, r_3, a_2\rangle \in \mathit{Used}^{gr'} \\ & \wedge \ \langle a_2, r_2, p_1\rangle \in \mathit{WasGeneratedBy}^{gr'}\end{array}} \qquad \text{(Equivalence 2)}$$

$$\begin{aligned} \text{with } \mathit{accountOf}^{gr'}(\langle p_2, r_3, a_2\rangle) &= \mathit{accountOf}^{gr}(\langle p_2, p_1\rangle) \\ \mathit{accountOf}^{gr'}(\langle a_2, r_2, p_1\rangle) &= \mathit{accountOf}^{gr}(\langle p_2, p_1\rangle) \\ \mathit{ArtifactId}^{gr'} &= \mathit{ArtifactId}^{gr} \cup \{a_2\} \\ \mathit{valueOf}^{gr'}(a_2) &= v \ \text{ for some value } v \end{aligned}$$

$$\frac{\langle a_2, a_1\rangle \in \mathit{WasDerivedFrom}^{gr}}{\begin{array}{ll}\exists p_1, r_1, r_2, & \langle a_2, r_2, p_1\rangle \in \mathit{WasGeneratedBy}^{gr'} \\ & \wedge \ \langle p_1, r_1, a_1\rangle \in \mathit{Used}^{gr'}\end{array}} \qquad \text{(Completion)}$$

$$\begin{aligned} \text{with } \mathit{accountOf}^{gr'}(\langle a_2, r_2, p_1\rangle) &= \mathit{accountOf}^{gr}(\langle a_2, a_1\rangle) \\ \mathit{accountOf}^{gr'}(\langle p_1, r_1, a_1\rangle) &= \mathit{accountOf}^{gr}(\langle a_2, a_1\rangle) \\ \mathit{ProcessId}^{gr'} &= \mathit{ProcessId}^{gr} \cup \{p_1\} \\ \mathit{valueOf}^{gr'}(p_1) &= v \ \text{ for some value } v \end{aligned}$$

Figure 10: Expansion of Graph by Definitional Constraints

asserted edges discussed in Section 3.6.

Definition 4 (Maximally Expanded Graph) *For any legal OPM graph gr, gr is maximally expanded if gr is closed under the application of definitional constraints of Figure 10.*

In principle, we coud apply (Equivalence 2) multiple times to an edge. However, already after the first time, it would be "closed" for that edge, so we do not gain anything by applying the inference to the same edge again.

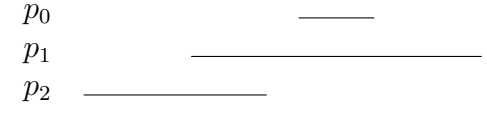
Definition 5 (Legal Expansions of Graph) *For any legal OPM graph gr, the set of all possible expansions of a graph, noted as expansions(gr), is defined as the set of legal, maximally expanded graphs derived from gr, by successive application of the definitional constraints of Figure 10.*

Whilst the expansion rules introduced in this Section provide a useful clarification about the meaning of OPM graphs, they do not fully capture the time dependencies introduced in Section 2. This aspect of the model is the focus of Section 5.

3.3 Inference Rules

The purpose of a provenance graph is to explain how an artifact is causally dependent on other artifacts and processes, by means of one or more one-step dependencies introduced in the previous section. To formalise this notion, we define three multistep relations *WasDerivedFrom**, *Used**, *WasGeneratedBy**. We note that relations *Used**, *WasGeneratedBy** are intented to be inferred, whereas *WasDerivedFrom** can be both inferred and asserted. In the latter case, someone can use an edge belonging to *WasDerivedFrom** to assert that an artifact was derived from another, using one or more derivation steps.

We note that *WasInformedBy* is not transitive. Indeed, let us consider the OPM graph gr, such that $\langle p_2, p_1\rangle \in WasInformedBy^{gr}$ and $\langle p_1, p_0\rangle \in WasInformedBy^{gr}$, where processes p_0, p_1, p_2 are scheduled as follows.



The edge $\langle p_2, p_1\rangle \in WasInformedBy^{gr}$ states that p_2 completes after p_1 starts; likewise $\langle p_1, p_0\rangle \in WasInformedBy^{gr}$ states that p_1 completes after p_0 starts. However, there is no overlap between p_2 and p_0 and p_2 completes before p_0 even started. So, in this case, the relation $\langle p_2, p_o\rangle$ does not hold. So, we conclude that the relationship *WasInformedBy* is not transitive.

Therefore, to distinguish asserted edges from inferred edges, we distinguish the set *Edge*, denoting all asserted edges and expanded edges by definitional

constraints, from $Edge^*$ denoting the set of all inferred multistep edges. We then introduce $OPMGraph^*$ as the set of all OPM Graphs with multistep edges, in which the *accountOf* function is also extended to take elements of $Edge^*$ in its domain.

An OPMGraph* is a dependent type defined as follows

$$\begin{aligned} & OPMGraph^*(ProcessId, ArtifactId, Role, Account, Value, Edge, Inferred) \\ = \;& AccountOf \times ValueOf \times AccountRelation \end{aligned}$$

where

$$\begin{aligned} WasDerivedFrom^* &= ArtifactId \times ArtifactId \\ Used^* &= ProcessId \times ArtifactId \\ WasGeneratedBy^* &= ArtifactId \times ProcessId \\ AccountOf &= Edge \cup Inferred \rightarrow \mathbb{P}(Account) \\ \text{with constraint} & \\ Edge &\subseteq Used \cup WasGeneratedBy \cup WasDerivedFrom \\ &\quad \cup WasInformedBy \\ Inferred &\subseteq WasDerivedFrom^* \cup Used^* \cup WasGeneratedBy^* \end{aligned}$$

Similarly to previously, we define convenience notations. For any graph gr, $WasDerivedFrom^{*gr}$ denotes the subset of asserted edges $\{e | e \in Edge^{gr} \wedge e \in WasDerivedFrom^*\}$. Likewise, $Inferred^{gr}$ denotes[3] the set of inferred edges of gr. All other notations and definitions remain the same as in *OPMGraph*.

Figure 11 summarises the inferences allowed on OPM graphs. In all three inferences, we observe that the middle artifact is "eliminated", by creating a new edge that bypasses the artifact, by connecting an effect derived from the artifact to a cause that resulted in the artifact. (We note that such an inference is not allowed when an artifact appears in the context of two process, as described above.). The notation $(*)$ is used in the figures to denote a one step dependency or a multistep dependency. The inferred edge is always a multistep dependency.

While Figure 11 presents all the valid inference rules that prevail for OPM graphs without asserted multistep edges, it is necessary to cater for asserted $WasDerivedFrom^*$ edges, by the multistep (Completion*) of Figure 12. (We note that such a rule is superfluous for edges $WasDerivedFrom^*$ that are inferred, since successive application of the (Completion) Rule and artifact elimination rule infer the same graph). We note that this inference rule leads to the same potential ambiguities as its one-step version.

Figure 13 formalises all permitted transitions. Equations (AssertedD), (AssertedU) and (AssertedG) deal with asserted edges: any asserted edge also belongs to the

[3]We have not found the need to refer to subsets of inferred edges, so we did not define accessors for these. We note that $WasDerivedFrom^{*gr}$ allows access to *asserted* edges of that kind.

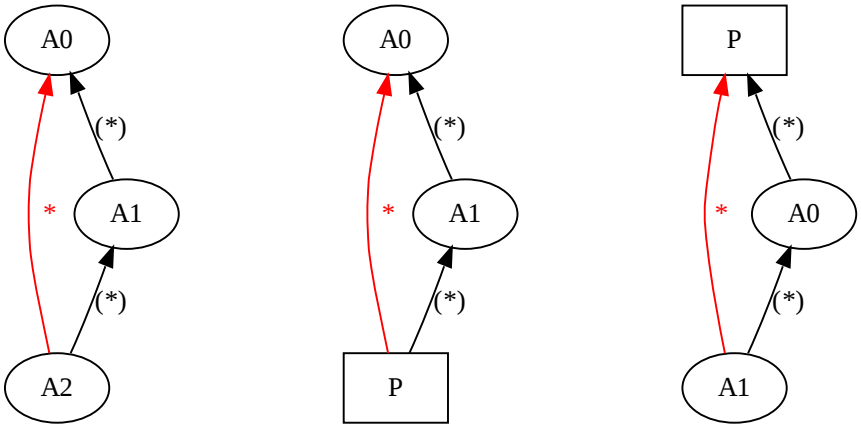


Figure 11: Inference Rules: Artifact "Elimination"

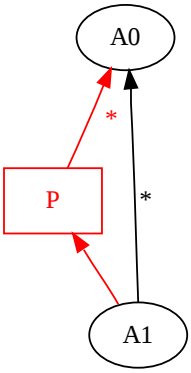


Figure 12: Inference Rules: Multistep Completion*

set of multistep edges. In particular, we note that equation (AssertedD) takes care of both one-step and multistep asserted edges. Equations (Transitivity AAA), (Transitivity PAA) and (Transitivity AAP) specify which multistep relations are transitive. Equation (Completion*) is the multistep completion rule.

3.4 Notion of an A-Path

The inference rules indicate that we can infer new edges by bypassing artifacts (occurring in a suitable context). This means that, as far as causal dependencies are concerned, a node is dependent on another if there is a path from the former to the latter involving artifacts as intermediary nodes. Such a notion is captured by the notion of an A-Path, i.e. a path involving artifacts as intermediary nodes.

Definition 6 (A-Path) *For any OPM Graph gr, an A-Path between two nodes $x, y, \in Node^{gr}$, which we note $isAPath^{gr}(x, y)$, is defined as a sequence of nodes $n_0, \ldots, n_m \in Node^{gr}$, such that*

1. *$n_0 = x$, $n_m = y$, $n_i \in ArtifactId^{gr}$ for any $0 < i < m$, and*
2. *there is a sequence of edges $e_0, \ldots, e_{m-1} \in Edge^{gr}$ such that $effect(e_i) = n_i$ and $cause(e_i) = n_{i+1}$ for $0 \leq i < m$.*

In some situations, it is convenient to be able to state that among a set of nodes N (typically a strict subset of $Node^{gr}$), there exists an A-Path.

Definition 7 (A-Path on Node Subset) *For any OPM Graph gr, there exists an A-Path between two nodes x, y restricted to set N, which we note $isAPath^{gr}(x, y, N)$, if $isAPath^{gr}(x, y)$ holds and is a sequence of nodes $n_0, \ldots, n_m \in N$.*

In Section 3.2, we indicated that the uncertainty produced by the introduction rules (Completion) and (Equivalence 2) did not affect the kind of inferences that are permitted over OPM graphs. We formalise this property by the following theorem, establishing that whatever the uncertainty introduced, the set of A-Paths is preserved.

Theorem 1 *For any OPM Graph gr, and for any $gr' \in expansions(gr)$, the set of all A-Paths for gr is equal the set of all A-paths for gr' restricted to $Node^{gr}$:*

$$isAPath^{gr}(x, y) \text{ iff } isAPath^{gr'}(x, y, Node^{gr}).$$

Proof 1 *For any two nodes x and y in gr, we will actually show that if there is an A-path from x to y in gr', then that path already existed in gr. (The converse implication is trivial since gr is a subgraph of gr'.) Thereto, observe that the definitional constraint rules do not introduce any new edges between artifacts. So, we only have to worry about the first edge on the given path, if x is a process*

For any OPM graph gr, the application of the inference rules results in a new $OPMgraph^*$ gr' that extends gr according to the following rules. The resulting graph gr' is said to be inferred from gr.

$$\frac{\langle a_2, a_1\rangle \in \mathit{WasDerivedFrom}^{gr} \text{ or } \langle a_2, a_1\rangle \in \mathit{WasDerivedFrom}^{*gr}}{\langle a_2, a_1\rangle \in \mathit{Inferred}^{gr'}} \quad \text{(AssertedD)}$$

$$\frac{\langle p, a\rangle \in \mathit{Used}^{gr}}{\langle p, a\rangle \in \mathit{Inferred}^{gr'}} \quad \text{(AssertedU)}$$

$$\frac{\langle a, p\rangle \in \mathit{WasGeneratedBy}^{gr}}{\langle a, p\rangle \in \mathit{Inferred}^{gr'}} \quad \text{(AssertedG)}$$

$$\frac{\langle a_2, a_3\rangle \in \mathit{Inferred}^{gr'} \wedge \langle a_3, a_1\rangle \in \mathit{Inferred}^{gr'}}{\langle a_2, a_1\rangle \in \mathit{Inferred}^{gr'}} \quad \text{(Transitivity AAA)}$$

$$\begin{aligned} &\text{with } \mathit{accountOf}^{gr'}(\langle a_2, a_1\rangle) \\ &\quad = \quad \mathit{accountOf}^{gr}(\langle a_2, a_3\rangle) \bigcirc \mathit{accountOf}^{gr}(\langle a_3, a_1\rangle) \end{aligned}$$

$$\frac{\langle p, a_2\rangle \in \mathit{Inferred}^{gr'} \wedge \langle a_2, a_1\rangle \in \mathit{Inferred}^{gr'}}{\langle p, a_1\rangle \in \mathit{Inferred}^{gr'}} \quad \text{(Transitivity PAA)}$$

$$\begin{aligned} &\text{with } \mathit{accountOf}^{gr'}(\langle p, a_1\rangle) \\ &\quad = \quad \mathit{accountOf}^{gr'}(\langle p, a_2\rangle) \bigcirc \mathit{accountOf}^{gr'}(\langle a_2, a_1\rangle) \end{aligned}$$

$$\frac{\langle a_1, p\rangle \in \mathit{Inferred}^{gr'} \wedge \langle a_2, a_1\rangle \in \mathit{Inferred}^{gr'}}{\langle a_2, p\rangle \in \mathit{Inferred}^{gr'}} \quad \text{(Transitivity AAP)}$$

$$\begin{aligned} &\text{with } \mathit{accountOf}^{gr'}(\langle a_2, p\rangle) \\ &\quad = \quad \mathit{accountOf}^{gr'}(\langle a_1, p\rangle) \bigcirc \mathit{accountOf}^{gr'}(\langle a_2, a_1\rangle) \end{aligned}$$

$$\frac{\langle a_2, a_1\rangle \in \mathit{WasDerivedFrom}^{*gr}}{\begin{array}{l}\exists p_1, r_1, r_2, \quad \langle a_2, r_2, p_1\rangle \in \mathit{WasGeneratedBy}^{gr'} \\ \qquad\qquad \wedge \langle p_1, r_1, a_1\rangle \in \mathit{Inferred}^{gr'}\end{array}} \quad \text{(Completion}^*\text{)}$$

$$\begin{aligned} \text{with } \mathit{accountOf}^{gr'}(\langle a_2, r_2, p_1\rangle) &= \mathit{accountOf}^{gr}(\langle a_2, a_1\rangle) = \mathit{accountOf}^{gr'}(\langle p_1, r_1, a_1\rangle) \\ \mathit{ProcessId}^{gr'} &= \mathit{ProcessId}^{gr} \cup \{p_1\} \\ \mathit{valueOf}^{gr'}(p_1) &= v \text{ for some value } v \end{aligned}$$

Figure 13: Transitive Closures

node, and about the last edge on the given path, if y is a process node. We will now argue that in both cases, there is no problem.

If x is a process node and the first edge on the path was not already in gr, that edge must have been created by Equivalence rule 2. But then the edge ends in an added artifact node that has no further edge to another artifact node. So that edge cannot lie on the A-path.

If y is a process node and the last edge on the path was not already in gr, again that edge must have been created by Equivalence rule 2. But then the edge starts in an added artifact node that has no incoming edge from another artifact node. So again that edge cannot lie on the A-path.

The above theorem can be rephrased by saying that for any OPM Graph gr, and for any $gr' \in expansions(gr)$, there are no nodes $x, y \in Node^{gr}$ such that $isAPath^{gr'}(x, y)$ but $\neg isAPath^{gr}(x, y)$.

Theorem 2 *For any OPM Graph gr, and for any $gr^* \in OPMGraph^*$ inferred from gr, the set of all A-Paths for gr is equal to the set of all A-paths for gr^* restricted to $Node^{gr}$:*

$$isAPath^{gr}(x, y) \text{ iff } isAPath^{gr^*}(x, y, Node^{gr})$$

Proof 2 *This is obvious. Indeed, the three transitivity inference rules only shortcut A-paths. Furthermore, the Completion* rule introduces a process node and hence does not influence the A-paths.*

Theorem 3 *For any OPM Graph gr, and for any OPM Graph* gr^* maximally inferred from gr, $isAPath^{gr}(x, y)$ if and only if there exists $e \in Inferred^{gr^*}$ such that $x = effect(e)$ and $y = cause(e)$.*

Proof 3 *The "if" implication follows from the previous theorem. The "only-if" implication follows because any A-path can be shortcut to a single *-edge by repeated application of the transitivity inference rules.*

3.5 Account Relations

Having defined the concept of A-Path and associated transitive inferences over OPM graphs, we are now in position to characterise the remaining notion of "account relation" introduced in Figure 4. We first discuss the three different kinds of relations over accounts that can be declared in OPM graphs, namely overlap, alternate and refinement.

We note that such account relations are explicitly declared in an OPM graph, so that the processor of an OPM Graph does not need to read the graph fully, and reason in order to establish such relations. Hence, the declarations are introduced for efficiency reasons.

Overlap Two accounts are overlapping if they have at least one node in common, i.e. they refer to some common artifact or process. We formalise this notion and generalise it to m accounts.

Definition 8 (Overlapping Declaration) *For any OPM graph gr, m accounts $\alpha_0, \ldots, \alpha_{m-1}$ (with $m > 1$) are said to be overlapping, if $\{\alpha_0, \ldots, \alpha_{m-1}\} \in Overlap^{gr}$.*

Definition 9 (Legal Overlapping Declaration) *For any OPM graph gr, a declaration of m overlapping accounts $\{\alpha_0, \ldots, \alpha_{m-1}\} \in Overlap^{gr}$ is legal, if there is a node $n \in\subseteq Node^{gr}$ such that $\alpha_i \in effectiveAccountOf^{gr}(n)$ for $0 \leq i < m$.*

We note that if $\{\alpha_0, \alpha_1\} \in Overlap^{gr}$ and $\{\alpha_1, \alpha_2\} \in Overlap^{gr}$, it does not imply that $\{\alpha_0, \alpha_2\} \in Overlap^{gr}$, since α_0, α_1 may overlap over a set of nodes different than the set of nodes over which α_1, α_2 overlap. Hence, *Overlap* is not transitive. Furthermore, we made this relation n-ary to be able to express the overlapping property over more than two accounts, since it could not be expressed in terms of a binary relation. We also note that no common edge is required for overlapping accounts, but dependencies are the subject of alternates and refinements.

Alternate Two accounts can overlap over some artifacts and processes but may be descriptions of two totally different activities. For instance, X used a dictionary to lookup the definition of a word and Y used the same dictionary as a door stopper. These two accounts are unrelated, except for the dictionary over which they overlap. On the other hand, the accounts corresponding to the diaries of the three authors of this paper on 11/03/09, have in common the fact that they met at the University of Hasselt, brainstormed OPM, defined the notion A-Path, and drafted this paper, while also containing non-overlapping descriptions of their respective activities in the morning and evening. Such accounts, with a common story, possibly expressed with different levels of details are said to be alternate. We formalise this notion as follows.

Definition 10 (Alternate Declaration) *For any OPM graph gr, m accounts $\alpha_0, \ldots, \alpha_{m-1}$ (with $m > 1$) are said to be alternate, if $\{\alpha_0, \ldots, \alpha_{m-1}\} \in Alternate^{gr}$.*

Definition 11 (Legal Alternate Declaration) *For any OPM graph gr, a declaration of m alternate accounts $\{\alpha_0, \ldots, \alpha_{m-1}\} \in Alternate^{gr}$ is legal, if there is a non-empty set $P \subseteq Node^{gr} \times Node^{gr}$, such that for any $\langle x, y\rangle \in P$, $isAPath^{gr_i}(x, y)$ for all $gr_i = view(gr, \alpha_i)$ with $0 \leq i < m$. The set P is said to be the set of A-Paths over which the accounts provide alternate descriptions. (We note that the actual path connecting x and y does not matter, provided that some connection exists beween x and y).*

Refinement A specific and very common case of alternate description occurs when one account provides more details than another over a *same* execution. Such a notion is being referred to as refinement. Previously introduced Figures 1 and 2 are an example of account refinement. While the relations *Overlap* and *Alternate* were n-ary and symmetric, a refinement is a binary and asymmetric relation. We formalise this notion as follows.

Definition 12 (Refinement Declaration) *For any OPM graph gr, α_0 is said to be a refinement of α_1, if $\langle \alpha_0, \alpha_1\rangle \in Refinement^{gr}$.*

Definition 13 (Legal Refinement Declaration) *For any OPM graph gr, a declaration of a refinement $\langle \alpha_0, \alpha_1\rangle \in Refinement^{gr}$ is legal, if the set*

$$P = \{\langle x, y\rangle \mid \langle x, y\rangle \in Node^{gr_1} \times Node^{gr_1} \text{ and } isAPath^{gr_1}(x, y)\},$$

is not empty and such that for any $\langle x, y\rangle \in P$, $isAPath^{gr_0}(x, y)$, where $gr_i = view(gr, \alpha_i)$ with $i = 0, 1$. The set P is said to be the set of A-Paths over which α_0 provides a refined description of α_1.

In other words, a refinement of α_1 into α_0 is such that any A-Path of α_1 is also an A-Path of α_0. Hence, such A-Paths constitute the alternate descriptions of these two accounts. Hence, every refinement is also a legal alternate description. We summarise such property as follow. For any OPM graph gr,

$$Overlap^{gr} \supseteq Alternate^{gr} \supseteq Refinement^{gr}.$$

3.6 Graph Operations

To do:*Provide more rationale for these operations, justify disjunctive and conjunctive interpretations. Should we introduce graph renaming, where graph ids are being renamed, everything else being equal.*

In this section, we define the OPM graph operations, union and intersection. Before considering these, we need to understand what it means to have more

than one accounts on a asserted edge. According to a disjunctive interpretation it means that the asserted edge is there according to at least one of the accounts. According to a conjunctive interpretation, it means means that the asserted edge is there according to each of the account.

Both interpretations are reasonable, and therefore are reflected in the union and intersection operations. Both rely on a union operator $\bigcirc$ over sets of accounts (and sets of values). In the conjunctive case, the intersection is used to only consider consider the accounts that are common among the common elements of both graphs. In the disjunctive case, the union is chosen on the assumption that when two OPM graphs are joined, we still would like to keep all possible accounts from the common elements of both graph.

We note that such interpretations of accounts on asserted edges differ from the permissive and conservative interpretations of inferred edges.

Definition 14 (OPM Graph union) *Let gr_1 and gr_2 be two OPM graphs. We define the union of gr_1 and gr_2, denoted $gr_1 \sqcup gr_2$, as follows:*

$$\begin{aligned} Node^{gr_1 \sqcup gr_2} &= Node^{gr_1} \cup Node^{gr_2} \\ Edge^{gr_1 \sqcup gr_2} &= Edge^{gr_1} \cup Edge^{gr_2} \\ accountRelation^{gr_1 \sqcup gr_2} &= accountRelation^{gr_1} \cup accountRelation^{gr_2} \end{aligned}$$

where symbol $\bigcirc$ denotes $\cap$ or $\cup$. Furthermore, $accountOf^{gr_1 \sqcup gr_2}$ is the point-wise union of $accountOf^{gr_1}$ and $accountOf^{gr_2}$ defined as:

$$\begin{aligned} & accountOf^{gr_1 \sqcup gr_2}(x) \\ &= accountOf^{gr_1}(x) \text{ if } x \in DOM(accountOf^{gr_1}(x)) \setminus DOM(accountOf^{gr_2}(x)) \\ &= accountOf^{gr_2}(x) \text{ if } x \in DOM(accountOf^{gr_2}(x)) \setminus DOM(accountOf^{gr_1}(x)) \\ &= accountOf^{gr_1}(x) \bigcirc accountOf^{gr_2}(x) \\ &\quad \text{if } x \in DOM(accountOf^{gr_1}(x)) \cap DOM(accountOf^{gr_2}(x)) \end{aligned}$$

and likewise $valueOf^{gr_1 \sqcup gr_2}$ is the point-wise union of $valueOf^{gr_1}$ and $valueOf^{gr_2}$ defined as:

$$\begin{aligned} & valueOf^{gr_1 \sqcup gr_2}(x) \\ &= valueOf^{gr_1}(x) \text{ if } x \in DOM(valueOf^{gr_1}(x)) \setminus DOM(valueOf^{gr_2}(x)) \\ &= valueOf^{gr_2}(x) \text{ if } x \in DOM(valueOf^{gr_2}(x)) \setminus DOM(valueOf^{gr_1}(x)) \\ &= valueOf^{gr_1}(x) \bigcirc valueOf^{gr_2}(x) \\ &\quad \text{if } x \in DOM(valueOf^{gr_1}(x)) \cap DOM(valueOf^{gr_2}(x)) \end{aligned}$$

Definition 15 (OPM Graph Intersection) *Let gr_1 and gr_2 be two OPM graphs. We define the intersection of gr_1 and gr_2, denoted $gr_1 \sqcap gr_2$, as follows:*

$$\begin{aligned} Node^{gr_1 \sqcap gr_2} &= Node^{gr_1} \cap Node^{gr_2} \\ Edge^{gr_1 \sqcap gr_2} &= Edge^{gr_1} \cap Edge^{gr_2} \\ accountRelation^{gr_1 \sqcap gr_2} &= accountRelation^{gr_1} \cap accountRelation^{gr_2} \end{aligned}$$

Furthermore, $accountOf^{gr_1 \sqcap gr_2}$ is the point-wise union (or intersection) of $accountOf^{gr_1}$ and $accountOf^{gr_2}$ defined as:

$$
\begin{aligned}
&accountOf^{gr_1 \sqcap gr_2}(x) \\
&\quad = \; accountOf^{gr_1}(x) \text{ if } x \in DOM(accountOf^{gr_1}(x)) \setminus DOM(accountOf^{gr_2}(x)) \\
&\quad = \; accountOf^{gr_2}(x) \text{ if } x \in DOM(accountOf^{gr_2}(x)) \setminus DOM(accountOf^{gr_1}(x)) \\
&\quad = \; accountOf^{gr_1}(x) \bigcirc accountOf^{gr_2}(x) \\
&\qquad\quad \text{if } x \in DOM(accountOf^{gr_1}(x)) \cap DOM(accountOf^{gr_2}(x))
\end{aligned}
$$

where symbol $\bigcirc$ denotes $\cap$ or $\cup$. Likewise $valueOf^{gr_1 \sqcap gr_2}$ is the point-wise union (or intersection) of $valueOf^{gr_1}$ and $valueOf^{gr_2}$ defined as:

$$
\begin{aligned}
&valueOf^{gr_1 \sqcap gr_2}(x) \\
&\quad = \; valueOf^{gr_1}(x) \text{ if } x \in DOM(valueOf^{gr_1}(x)) \setminus DOM(valueOf^{gr_2}(x)) \\
&\quad = \; valueOf^{gr_2}(x) \text{ if } x \in DOM(valueOf^{gr_2}(x)) \setminus DOM(valueOf^{gr_1}(x)) \\
&\quad = \; valueOf^{gr_1}(x) \bigcirc valueOf^{gr_2}(x) \\
&\qquad\quad \text{if } x \in DOM(valueOf^{gr_1}(x)) \cap DOM(valueOf^{gr_2}(x))
\end{aligned}
$$

The results of the $\sqcup$ and $\sqcap$ operations over OPM graphs are themselves OPM graphs. We note that the union operation over two graphs does not necessarily preserve the legality of accounts since a given account in the union may contain cycles or multiple *WasGeneratedBy* edges for a given artifact.

To do:*The union/intersection operations may not preserve the legality of refinement and alternate declarations. Investigate circumstances in which refinement/alternate are preserved.*

4 OPM and Past Executions

The motivation for designing the Open Provenance Model is that such a notion of directed acyclic graph is suitable for representing causal dependencies resulting from past executions. In this section, we show *how* the proposed model can be used to express such execution-related dependencies, and establish that any inference over the encoded OPM graph allows us to derive any transitive dependency that existed over an execution.

We use the following set-theoretic definition for a trace, describing a past execution of a program. A trace consists of an ordered sequence of statements (we use $\mathbb{L}(S)$ to denote the set of order sequences over elements of S). All statements denote the assignment of a value to a variable. In the simplest case, a constant is assigned to a variable. In the more complex case, the result of a function call is assigned to a variable. Function calls identify the function called and its arguments (all in the form of variables). The involvement of an argument

in a function call is specified by its role (and several arguments may potentially share a same role). Given that a same function call is allowed to return multiple results, we identify each of them by their respective role.

$$\begin{aligned} \mathit{Function} &: \mathit{primitive\ set\ of\ functions} \\ \mathit{Variable} &: \mathit{primitive\ set\ of\ variables} \\ \mathit{Constant} &: \mathit{primitive\ set\ of\ constants\ over\ some\ domain\ of\ values} \\ \mathit{Trace} &= \mathbb{L}(\mathit{Statement}) \\ \mathit{Call} &= \mathit{Function} \times \mathbb{P}(\mathit{Role} \times \mathit{Variable}) \times \mathit{Role} \\ \mathit{Statement} &= \mathit{Variable} \times \mathit{Constant} \ \cup \ \mathit{Variable} \times \mathit{Call} \end{aligned}$$

Some further constraints are added to traces: we assume that all statements assign a new variable, and that all references to a variable in a statement must be preceded by an assignment of that variable earlier in the trace. We note that with these constraints, our traces have a single-assignment property.

To illustrate this definition, we consider the following trace, where v_2 is assigned to the quotient of v_0 and v_1, whereas v_3 is assigned to the remainder of the division of v_0 and v_1. And finally v_4, is the minimum of $v_0, \ldots, v_3$.

$$\begin{aligned} v_0 &= 14 \\ v_1 &= 3 \\ v_2 &= \mathit{division}(\mathsf{dividend} : \{v_0\}, \mathsf{divisor} : \{v_1\}).\mathsf{quotient} \\ v_3 &= \mathit{division}(\mathsf{dividend} : \{v_0\}, \mathsf{divisor} : \{v_1\}).\mathsf{remainder} \\ v_4 &= \mathit{minimum}(\mathsf{input} : \{v_0, v_1, v_2, v_3\}).\mathsf{min} \end{aligned}$$

Many forms of dependencies can be considered in such a trace language. For instance, a *control dependency* expresses that a variable is read by a statement; for instance, v_2 is control-dependent on v_0 since the assignment of v_2 cannot proceed before v_0 has been assigned. (Such notion is usually considered in compilers for instruction-level parallelism [1].) A *where-from dependency* [7] or *alias equivalence* identifies that the output of a function is the same (e.g. at the same location) as the content of variable. For instance, in this example, the content of v_4 is the same as the content of v_3. (We note here that this is an accurate dependency established by computing the minimum function.). A *functional dependency* expresses that variable values were used to compute the result of a function. For instance, both values of v_0 and v_1 where used in order to determine v_2 (and also v_3).

In this paper, we abstract away from such dependencies as follows. An element of the set *Dependency* expresses that a variable has some form of dependency on a set of other variables.

$$\begin{aligned} \mathit{Dependency} &= \mathit{Variable} \times \mathbb{P}(\mathit{Variable}) \\ \mathit{Dependencies} &= \mathbb{P}(\mathit{Dependency}) \end{aligned}$$

Any execution trace and associated set of dependencies can be converted into an OPM graph gr, by means of the following translation function.

$$T[\![\ ]\!] \ : \ \mathit{Trace} \times \mathit{Dependencies} \rightarrow \mathit{OPMGraph} \times \mathit{Mapping}$$

The translation of a trace tr and dependencies dep, results in a graph gr and mapping m constructed as follows:

1. If $T[\![tr, dep]\!] = \langle gr, m \rangle$, then:

$$\begin{array}{lcl} gr & \in & \mathit{OPMGraph} \\ m & : & \mathit{Variable}^{tr} \rightarrow \mathit{ArtifactId}^{gr} \cup \ \mathit{Call}^{tr} \rightarrow \mathit{ProcessId}^{gr} \end{array}$$

2. For each variable v_i and associated stored value c, add a new artifact a to $\mathit{ArtifactId}^{gr}$, with $\mathit{valueOf}^{gr}(a) = c$ and $m(v_i) = a$.

3. For any function call $fc = f(r_0 : \{v_{0,0}, v_{0,1}, \ldots\}, \ldots)$, add a new process p to $\mathit{ProcessId}^{gr}$, with $\mathit{valueOf}^{gr}(p) = f$; for any variable $v_{i,j}$, add an edge $\langle p, r_i, a \rangle$ to Used^{gr}, where $a = m(v_{i,j})$ for some role r_i. In addition, $m(fc) = p$

4. For any assignment $v = fc.r$, where fc is a function call $f(r_0 : \{v_{0,0}, v_{0,1}, \ldots\}, \ldots)$ with $m(fc) = p$, add a new edge $\langle v, r, p \rangle$ to $\mathit{WasGeneratedBy}^{gr}$.

5. For any dependency $\langle v, (v_0, \ldots, v_n) \rangle \in dep$, add new edges $\langle a, a_i \rangle$ to $\mathit{WasDerivedFrom}^{gr}$, where $a = m(v)$ and $a_i = m(v_i)$.

Should we have a role here?

By construction, the translation of a trace maps every variable to a unique artifact in an OPM graph, and every data dependency is expressed by a *WasDerivedFrom* edge. As a result, every transitive data dependency is expressed by an A-Path in the OPM graph, and can be computed by inferences over the graph. Hence, using this translation, we can claim that OPM graphs are suitable to represent past executions and associated dependencies. The translation also helps developer decide how to generate OPM graphs for their programs.

5 Time Annotations

Section 2 informally introduces edges supported by OPM by defining them in terms of time ordering of events (e.g., *Used* edge means that process coudl complete only after artifact was used). So far, our discussion of OPM has been purely timeless. We address the problem time annotations on OPM graphs in this section.

As OPM graphs can be annotated by time information, we introduce an annotation function, which, for an OPM graph gr, has the following type.

$$\begin{array}{rcl} \mathit{Time} & : & \textit{primitive set of time} \\ \mathit{Time}_{\perp} & : & = \mathit{Time} \cup \{\perp\} \\ \mathit{TimeAnnotation}^{gr} & = & \mathit{EdgeAnnotation}^{gr} \cup \mathit{ProcessAnnotation}^{gr} \\ \mathit{EdgeAnnotation}^{gr} & = & (\mathit{Used}^{gr} \cup \ \mathit{WasGeneratedBy}^{gr} \cup \ \mathit{WasDerivedFrom}^{gr}) \\ & & \times \mathit{Account}^{gr} \rightarrow \mathit{Time}_{\perp} \\ \mathit{ProcessAnnotation}^{gr} & = & \mathit{ProcessId}^{gr} \times \mathit{Account}^{gr} \rightarrow (\mathit{Time}_{\perp} \times \mathit{Time}_{\perp}) \end{array}$$

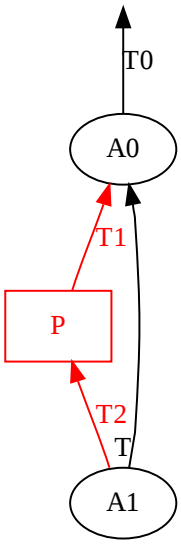
Time annotations are optional and hence are formalised as the set $\mathit{Time}_{\perp}$ containing the distinguished element $\perp$ denoting absence of time annotation. Time information can be added as an annotation to some edges and to some nodes. Specifically, *Used*, *WasGeneratedBy*, *WasDerivedFrom* edges may be annotated with the time at which an artifact was used, generated, and generated respectively. Processes may be annotated with their start time and/or their end time. Additionally, time annotations may differ for different accounts, hence the parametrisation with accounts.

While causal dependency implies time ordering, the converse does not necessarily hold. Therefore, an annotated OPM graph allows us to derive time constraints. However, the mere presence of time annotations permits us to revisit the (Completion) rule, as illustrated in Figure 14.

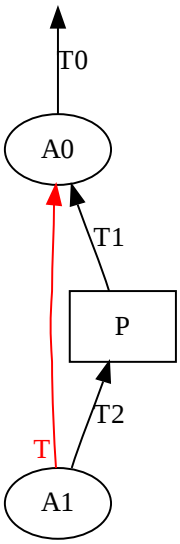
The left hand side contains a revised (Completion) rule, where a *WasDerivedFrom* edge implies the existence of an intermediary process. Constraints over time annotations on those edges are expressed at the right of the figure: artifact A_0 must be generated (T_0) before it is used (T_1); the time at which A_1 was derived (T) is the time at which it was generated by P (T_2). In addition, for the process P to have helped derive A_1 from A_0, at the minimum, A_0 must have been used (T_1) before A_1 was generated (T_2), otherwise the *WasDerivedFrom* would simply not hold.

Interestingly, when all these time constraints are satisfied, rule (Completion) has now an opposite, and we have a full equivalence, hence the names (Equivalence 3) and (Equivalence 4) (A similar equivalence was introduced for *WasInformedBy* edges in Figure 5 in a timeless context). Whilst (Equivalence 4) establishes that there is a dependency between A_1 and A_0, we note that it may not necessarily be a *data* dependency, since the actual content of A_0 may not have been actually exploited to generate A_1. However, at the minimum, there is a control dependency, since A_1 would not have been generated if A_0 had not been produced (everything else being equal).

Following the introduction of (Equivalence 4), we are now in presence of a single step *WasDerivedFrom* edge implied by the definitional constraints. Hence, our notion of A-Path has to be extended to accommodate intermediary processes, with specific constraints on their incident edges.



(Equivalence 3)
Revised completion



(Equivalence 4)

$T_2 = T$
$T_0 < T_1 < T_2$

Figure 14: Completion Rule and Time

Definition 16 (APT-Path) *For any OPM Graph gr, and time annotation* $ta \in TimeAnnotation^{gr}$ *an APT-Path between two nodes* $x, y, \in Node^{gr}$*, which we note* $isAPTPath^{gr}(x, y)$*, is defined as a sequence of nodes* $n_0, \ldots, n_m \in Node^{gr}$*, such that:*

1. *there is a sequence of edges* $e_0, \ldots, e_{m-1} \in Edge^{gr}$ *such that* $effect(e_i) = n_i$ *and* $cause(e_i) = n_{i+1}$ *for* $0 \leq i < m$*.*
2. $n_0 = x$ *and* $n_m = y$
3. *for any* $0 < i < m$*, either:*
   - $n_i \in ArtifactId^{gr}$ *, or*
   - $n_i \in ProcessId^{gr}$*, with* $n_{i-1} = effect(e_{i-1}) \in ArtifactId^{gr}$ *and* $n_{i+1} = cause(e_i) \in ArtifactId^{gr}$
     and $ta(e_{i-1}, \alpha) > ta(e_i, \alpha)$ *for any* $\alpha \in accountOf^{gr}(e_{i-1}) \cap accountOf^{gr}(e_i)$*.*

From an annotated OPM graph gr, one can derive a set of time constraints using the inference rules of Figure 15. Rule 1 requires that the beginning time

of a process precedes its ending time. Rule 2 states that the derived time and generated time of an artifact are identical. Rule 3 states that the time at which an artifact was generated precedes the time at which it was used. Rule 4 states that any a process used an artifact after its start and before its end. Likewise, Rule 5 states that any process generated an artifact after its start and before its end.

The (Completion) rule over asserted multistep *WasDerivedFrom* edges may lead to different inferences. To explain this, let us consider the left hand of Figure 16, where the (Completion) rule applied to edge A_2 to A_0 and equation (6) allow us to derive the following constraints:

$$\begin{array}{ll} T_0 < T_1, T_0 < T_1 & \\ T_2 < T_3 & \\ T_1 < T_6 & \text{by (Completion) rule of single step edge } \langle A_2, A_0 \rangle \\ T_6 = T_7 & \end{array}$$

On the other hand, if we consider the right-hand of Figure 16, we can derive the following constraints, where there exists a choice because of the presence of the multistep edge $\langle A_2, A_0 \rangle$.

$$\begin{array}{ll} T_0 < T_1, T_0 < T_1 & \\ T_2 < T_3 & \\ T_1 < T_6 \vee T_3 < T_6 & \text{by (Completion) rule of multistep edge } \langle A_2, A_0 \rangle \\ T_6 = T_7 & \end{array}$$

The choice results from the fact that the multistep edge $\langle P, A_0 \rangle$ implied by the multistep (Completion) rule can either be realised by the single step $\langle P, A_0 \rangle$ or the two steps $\langle P, A_1 \rangle, \langle A_1, A_0 \rangle$. At least one of this dependency was effective before A_2 was generated. While there is uncertainty about when an artifact was being used, there is however no uncertainty about the generated artifact A_2. Whatever option we consider, we can infer that $T_0 < T_6$.

To do:*Discuss the semantic property according to which all valid inferences over time constraints (by transitivity of $<$) are the ones that would be derived by transitive closure over OPM edges.*

For any OPM graph gr and time annotation $ta \in \mathit{TimeAnnotation}^{gr}$, we can derive the following time constraints.

$$\frac{p \in \mathit{ProcessId}^{gr} \wedge \alpha \in \mathit{effectiveAccountOf}^{gr}(p) \wedge \mathit{ta}^{gr}(p, \alpha) = \langle T_1, T_2 \rangle}{T_1 < T_2} \quad (1)$$

$$\frac{\begin{array}{lll} g = \langle a_1, r, p \rangle \wedge\ g \in \mathit{WasGeneratedBy}^{gr} & \wedge\ \alpha_1 \in \mathit{accountOf}^{gr}(g) & \wedge\ \mathit{ta}^{gr}(g, \alpha_1) = T_1 \\ d = \langle a_1, a_2 \rangle \wedge\ d \in \mathit{WasDerivedFrom}^{(*)gr} & \wedge\ \alpha_2 \in \mathit{accountOf}^{gr}(d) & \wedge\ \mathit{ta}^{gr}(d, \alpha_2) = T_2 \\ & & \alpha_1 = \alpha_2 \end{array}}{T_1 = T_2} \quad (2)$$

$$\frac{\begin{array}{llll} u = \langle p_1, r_1, a \rangle & \wedge\ u \in \mathit{Used}^{gr} & \wedge\ \alpha_1 \in \mathit{accountOf}^{gr}(u) & \wedge\ \mathit{ta}^{gr}(u, \alpha_1) = T_2 \\ g = \langle a, r_2, p_2 \rangle & \wedge\ g \in \mathit{WasGeneratedBy}^{gr} & \wedge\ \alpha_2 \in \mathit{accountOf}^{gr}(g) & \wedge\ \mathit{ta}^{gr}(g, \alpha_2) = T_1 \\ & & \alpha_1 = \alpha_2 & \end{array}}{T_1 < T_2} \quad (3)$$

$$\frac{\begin{array}{lll} u = \langle p, r_1, a \rangle \wedge\ u \in \mathit{Used}^{gr} & \wedge\ \alpha_1 \in \mathit{accountOf}^{gr}(u) & \wedge\ \mathit{ta}^{gr}(u, \alpha_1) = T_2 \\ \alpha_2 \in \mathit{effectiveAccountOf}^{gr}(p) & \wedge\ \mathit{ta}^{gr}(p, \alpha_2) = \langle T_1, T_3 \rangle & \\ & \alpha_1 = \alpha_2 & \end{array}}{T_1 < T_2 < T_3} \quad (4)$$

$$\frac{\begin{array}{lll} g = \langle a, r_2, p \rangle \wedge\ g \in \mathit{WasGeneratedBy}^{gr} & \wedge\ \alpha_1 \in \mathit{accountOf}^{gr}(g) & \wedge\ \mathit{ta}^{gr}(g, \alpha_1) = T_2 \\ \alpha_2 \in \mathit{effectiveAccountOf}^{gr}(p) & \wedge\ \mathit{ta}^{gr}(p, \alpha_2) = \langle T_1, T_3 \rangle & \\ & \alpha_1 = \alpha_2 & \end{array}}{T_1 < T_2 < T_3} \quad (5)$$

$$\frac{d = \langle a_1, a_2 \rangle \wedge\ d \in \mathit{WasDerivedFrom}^{(*)gr} \quad \wedge\ \alpha_1 \in \mathit{accountOf}^{gr}(d) \quad \wedge\ \mathit{ta}^{gr}(d, \alpha_1) = T}{\begin{array}{lll} u = \langle p, r_1, a_2 \rangle \wedge\ u \in \mathit{Used}^{(*)gr} & \wedge\ \alpha_1 \in \mathit{accountOf}^{gr}(u) & \wedge\ \mathit{ta}^{gr}(u, \alpha_1) = T_1 \\ g = \langle a_1, r_2, p \rangle \wedge\ g \in \mathit{WasGeneratedBy}^{gr} & \wedge\ \alpha_2 \in \mathit{accountOf}^{gr}(g) & \wedge\ \mathit{ta}^{gr}(g, \alpha_2) = T_2 \end{array}} \quad T_1 < T_2 \quad (6)$$

Figure 15: Causation is Time-Monotonic

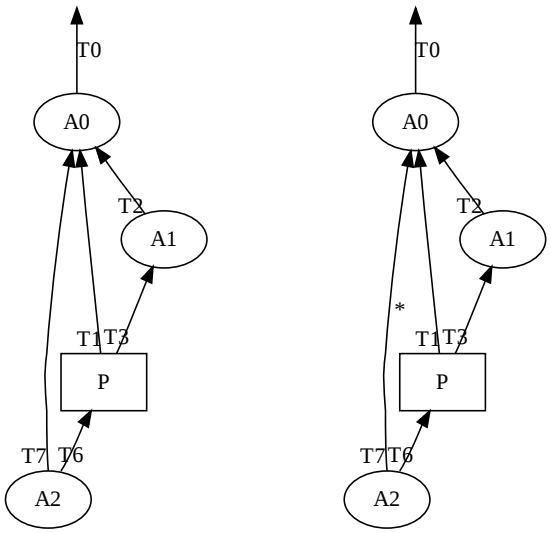


Figure 16: Uncertainty over Time Inference due to multistep *WasDerivedFrom*

6 Related Work

Provenance, also referred to as lineage [17], has been extensively but not exclusively studied in the context of scientific data [45, 6, 19, 33], since it is considered as crucial to ensure reproducibility of scientific experiments [18]. Traditionally, this field of research is divided according to workflow and data provenance. Workflow provenance (also referred to as coarse-grained provenance) records the complete history (or workflow) of the derivation of some dataset [9], whereas data provenance (also known as fine-grained provenance) is an account of the derivation of a piece of data in a dataset. The latter is been extensively investigated by the database community, whereas the former has been the focus of the e-science and grid communities. However, this distinction becomes artificial when dealing with applications that involve both workflow and database technology, as illustrated by the application adopted by the third Provenance Challenge [48].

Instead, given that OPM is a data model, we structure our related work in four categories *(i)* representing provenance, *(ii)* creating such representation, *(iii)* querying it, *(iv)* and storing it. We refer to surveys and proceedings for usage and requirements of provenance [45, 6, 19, 33, 30].

6.1 Provenance Representation

Following the first two Provenance Challenges [36, 44], the Open Provenance Model was designed as a data model to promote inter-operability of systems with provenance capabilities. Its features are inspired by a consensus that began to emerge after the second Provenance Challenge; as a result, it embeds many characteristics of systems produced by the teams involved in the challenges.

OPM edges *Used*/*WasGeneratedBy* correspond to traditional inputs and outputs of processes. But existence of an input for a given output does not necessarily imply data dependency. Such dependencies are explicitly represented by *WasDerivedFrom* edges in OPM. A similar distinction exists in the PASOA model [25], where so-called interaction p-assertions are concerned with I/Os, whereas relation p-assertions with data dependencies. Many systems recognise the importance of such data dependencies and incorporate them explicitly in their model, as summarised in the first Challenge editorial [37].

OPM introduces notions of account, account view and refinement. Abstraction capabilites are also a near-universally recognised requirement for provenance. Multiple systems allow for layering of provenance representation: this helps users focus on the right information for the task at hand. For instance, in myGrid, four different views (process, data, organisational, and knolwedge) are defined [49]. In VisTrails, the provenance information space is divided into workflow evolution layer, workflow layer and execution layer [43]. Likewise, abstract service descriptions, service instantiation descriptions, data instantiation and runtime execution constitute the four layers of the Redux model [4]. PASS (Provenance Aware Storage System) also advocates multiple layers to deal with the complexity of data and processes [39]. Views and layering can be captured by the OPM notion of account, and associated account views and refinements. However, provenance information does not have always to be structured hierarchically. The Zoom system [16] allows for user views to be expressed: such views help hide complexity of multi-step operations, offering the user a higher-level of abstraction. Formally, user views are defined as a partition of steps of a workflow specification. In PASOA [25], coexisting views of multiple asserters allow for potentially conflicting interpretations of a same execution, and non-hierachichal tracers cater for the kind of data diffusion encountered in peer-to-peer computing. Again, accounts can be used to represent such kind of a information grouping. If OPM is to be considered as a data model for inter-operability, extant work seems to suggest that some layers need to be agreed, in which specific types of nodes and edges are allowed to be specified.

Sahoo *et al.* [42] present a provenance algebra based on OWL, with a lot of similarity to OPM (without accounts), and argue for a Semantic Web approach to OPM. Whilst OPM can be encoded in RDF [21], we conjecture that RDF and OWL-based reasoning cannot express the notion of legal refinement introduced in this paper.

This paper explains how the OPM model can be used to express dependencies that existed during a past execution. Our approach however does not assume a specific execution language, but instead assumes an execution trace that would be produced by a typical programming language runtime. A number of other approaches have undertaken formal studies of provenance in the context of programming languages, defining provenance-based interpretation of such languages. Doing so, they derived alternate representations of provenance which we now discuss in turn.

While investigating the expressiveness of query and update languages for databases, Buneman *et al.* [7] defined a provenance-aware semantics for variants of the Nested Relational Calculus (NRC). In this context, domain values are extended with a tag, which indicates values origin, i.e., their provenance (The approach is akin to color propagation in databases.) Provenance of a value (in a query result) is represented as a unique color that is associated with another value in an input cell. This representation of provenance is suitable to answer where-queries (see Section 6.2). It cannot in its current form express the provenance of new values created by a query.

Cheney *et al* [13] define a traced evaluation of NRC queries, which results in an explicit trace, considered to be an explanation of the dynamic execution history of an expression. They then develop a metatheory of trace evaluation, introducing properties of consistency and fidelity that characterise how well traces describe execution. The grammar of traces is directly derived from the syntax of expressions being evaluated. Such traces are not provenance per se. Instead, provenance is extracted by means of queries (to which, we come back in Section 6.2). This approach is very similar to the one adopted in the PASOA architecture [24], where the information stored about execution is referred to as process documentation, and provenance is retrieved by means of provenance queries [32]. Those approaches are architecturally compatible with OPM since OPM consists of a representation model for provenance.

Green *et al.* [23] state that semirings of polynomials yield a form of provenance for relation calculus queries. Such polynomials describe *how* a tuple in the output was derived from the input. Polynomial exponents are used to indicate the number of times a given value is being used in a computation. The authors demonstrate that this representation is richer than others such as where and why provenance. OPM with its dependencies can also rerepsent such information: in addition, it introduces roles to some of its edges, to characterise how these values contributed to a final result.

Souilah *et al.* [47] propose the provenance calculus, which enriches the asynchronous π-calculus with information describing the channels values are communicated over. Within the calculus, provenance is represented as a sequence of input and output events, intended to describe the path over which a data item was communicated over. This representation naturally maps to sequences of *WasDerivedFrom* edges in OPM. In addition, the authors define a denotation

of provenance, which consists as a set of assertions about the past of the value it relates to.

Ludaescher *et al.* [29] consider a workflow-based model of computation and a set of observables resulting from their execution. They define a provenance model as a set of provenance assertions corresponding to execution observables and runtime observables. They apply their approach to Kahn process networks, with observables consisting of read and write operations (similarly to [47]).

Two of the authors [28] have investigated how to map the NRC dataflow to OPM. An interesting aspect of this mapping is concerned with data collections, for which an encoding in terms of OPM constructs is proposed. It is also noted that NRC-dataflow specific data dependencies can be inferred with NRC-dataflow inference rules, but cannot be expressed as inference in OPM. This suggests that OPM should cater for domain specific inferences as well.

6.2 Provenance Queries

The database community has identified several ways of characterising the origin of a value in a database, and has given them mono-syllabic names: where [7], why [8], how [23], and route [14] provenance. In their original form, these notions of provenance are typically investigated in the context of some query/update language. However, they can also be regarded as queries over a provenance graph representation such as OPM. In Section 4, we suggest that where-from or alias dependency can be expressed in an OPM graph; where-provenance can then be extracted from the OPM graph by following such types of dependencies.

Some form of graph traversal is generally (see comparison table in [37]) seen as essential to retrieve provenance. Miles [32] proposes a scoping mechanism that allows queriers to identify the subset of past activities that are relevant to their interest. Cheney *et al* [13] similarly extract where and how provenance from their provenance traces.

Common patterns of provenance queries begin to emerge, and APIs are being designed to support them directly, e.g., Karma [46] and Vistrails [43]. For instance, common querying tasks in Vistrails return all direct or indirect components preceding an activity in a past workflow.

Sahoo *et al.* [42] propose a systematic classification of provenance queries along three broad categories: querying for provenance metadata, querying for data values, and modifying provenance metadata. The first essentially consists of a transitive closure over provenance graphs, similar to the A-Path concept in OPM. The second consists of joins across provenance and application data: these are not expressible in OPM since it deliberately does not model application data to keep its focus on the essence of provenance. The third include merging and comparing provenance from multiple sources, and corresponds to the operations over graphs, including union and intersection, legal alternate and refinement. Similarly, Futrelle *et al.* [22] distinguish two primary classes of conditions for

queries: transitive closures and attributed queries.

6.3 Provenance Recording

OPM is a data model for provenance aiming at inter-operability of systems. It is envisaged that OPM graphs could be the result of queries over repositories, often called provenance stores [35], containing provenance, process documentation [32] or traces [13]. Nothing however in the model dictates how such a kind of information is to be recorded in provenance stores.

Groth *et al.* [26] address the problem of distributed recording of process documentation, where autonomous, distributed services composed in a workflow have to record process documentation independently. The challenge is twofold: make sure that process documentation produced by a service "connects" with process documentation generated by another; avoid unnessary synchronisation to ensure scalable and efficient recording. Their solution consists of a data model for process documentation that allows asynchronous recording, and a requirement put on applications to pass message identifiers.

ES3 [20] intoduces the concept of a Probulator, which transparently captures application envents and transmits that to a persistence storage akin to a Provenance Store. PASS (Provenance Aware Storage System) [39] proposes a modified kernel that intercepts OS-level events and records them in a scalable manner for future querying.

6.4 Provenance Storage

Since workflow execution time can be long and runs numerous, the amount of provenance information being recorded can be very large. OPM is concerned with inter-operable provenance data exchange, and hence does not offer a solution to the problem of compact storage. However, it could benefit from existing techniques in this field.

In the case where the workflow script is known, more compact representations could be devised, where the provenance trace only explictly represents dependencies that are not already present in a static workflow [15, 4]. From an inter-operability viewpoint, this approach presents some challenges because queriers not aware of the syntax and semantics of the workflow language used, would not be able to understand the retrieved provenance.

The database community also considers efficient representation of provenance for the type of provenance they consider. For instance, where-provenance [7] can be efficiently implemented with coloring technique. The downside of this representation is that it is tied to the query language and it cannot support other forms of queries.

Chapman *et al.* [11] propose a set of techniques, including factorisation and inheritance, to decrease the amount of storage required for provenance. It would

be interesting to investigate these in the context of OPM by specifying the formal properties of OPM graphs that these compact representations preserve.

6.5 Other Related Work

It should be pointed out that there is a large tradition of graph-based data models, for which we reder the reader to the excellent survey [2]. We have used the term "causal dependency" with a liberal interpretation in this paper. There is a vast amount of literature on causality. Pearl [41] discusses the possibility of learning causal relationship from raw data, relying on formal techniques such as graphs and probabilitic dependencies. Such techniques are extensively used in Artificial Intelligence and social sciences. The OPM approach differs since raw observations are "atomic" causal dependencies assumed to be observed by participants in a computation. It is beyond the scope of OPM to define how such atomic causal dependencies can be observed. In practice, techniques such as program analysis [12], Probulator [20], operating system calls interceptions [39] can be used to derive and observe such dependencies. Also, application designers have a role in specifying the dependencies that their applications produce [31]. Once the full expressiveness of account is exploited, some OPM graphs may contain conflicting information in different accounts, reasoning techniques about such conflicts will have to be devised, and probabilitic methods may be suitable to that end.

7 Conclusion

In this paper, we have proposed a formal definition of the Open Provenance Model, a data model for provenance aiming at information exchange inter-operability between systems with provenance capabilities. The formalisation introduces the notion of A-Path to characterise the kind of inferences that are allowed over OPM graphs. Such notion is then used to in the definition of alternate and refinement, which were inexistent in the original OPM specification.

We envisage a vast amount of potential activity around this OPM specification. First, from a practical viewpoint, OPM is the focus of the third challenge, where pragmatic considerations of data exchange and provenance queries will help evaluate its expressivity; bindings to XML and RDF are being proposed and tools operating over OPM representations begin to emerge. From a more theroretical perspective, issues pertaining to recording and querying need to be formalised and investigated, so that systems where OPM takes a more prominent role can be built. One of the novel elements of OPM is its notion of account. One needs to devise methods for reasoning in the presence of accounts in order to exploit this data model fully.

References

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey Ullman. *Compilers: Principles, Techniques and Tools*. Addison Wesley, 1986.

[2] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1–39, 2008.

[3] Lennart Augustsson. Cayenne—a language with dependent types. In *ICFP '98: Proceedings of the third ACM SIGPLAN international conference on Functional programming*, pages 239–250, New York, NY, USA, 1998. ACM.

[4] Roger S. Barga and Luciano A. Digiampietri. Automatic capture and efficient storage of escience experiment provenance. *Concurrency and Computation: Practice and Experience*, 20(5), 2008.

[5] Raj Bose, Ian Foster, and Luc Moreau. Report on the International Provenance and Annotation Workshop (IPAW06). *Sigmod Records*, 35(3):51–53, September 2006.

[6] Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: A survey. *ACM Computing Surveys*, 37(1):1–28, March 2005.

[7] Peter Buneman, James Cheney, and Stijn Vansummeren. On the expressiveness of implicit provenance in query and update languages. *ACM Trans. Database Syst.*, 33(4):1–47, 2008.

[8] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Why and Where: A Characterization of Data Provenance. In *Proceedings of 8th International Conference on Database Theory (ICDT'01)*, volume 1973 of *Lecture Notes in Computer Science*, pages 316–330, London, UK, 2001. Springer.

[9] Peter Buneman and Wang-Chiew Tan. Provenance in databases. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1171–1173, New York, NY, USA, 2007. ACM.

[10] http://twiki.ipaw.info/bin/view/Challenge/FirstProvenanceChallenge, June 2006.

[11] Adriane P. Chapman, H. V. Jagadish, and Prakash Ramanan. Efficient provenance storage. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 993–1006, New York, NY, USA, 2008. ACM.

[12] J. Cheney, A. Ahmed, and U. Acar. Provenance as Dependency Analysis. *Under consideration for publication in Math. Struct. in Comp. Science*, March 2008.

[13] James Cheney, Umut A. Acar, and Amal Ahmed. Provenance traces (extgended report). Technical Report http://arxiv.org/abs/0812.0564v1, University of Edinburgh, December 2008.

[14] Laura Chiticariu and Wang-Chiew Tan. Debugging schema mappings with routes. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 79–90. VLDB Endowment, 2006.

[15] Ben Clifford, Ian Foster, Mihael Hategan, Tiberiu Stef-Praun, Michael Wilde, and Yong Zhao. Tracking provenance in a virtual data grid. *Concurrency and Computation: Practice and Experience*, 20(5), 2008.

[16] Sarah Cohen-Boulakia, Olivier Biton, Shirley Cohen, and Susan Davidson. Addressing the provenance challenge using zoom. *Concurrency and Computation: Practice and Experience*, 20(5), 2008.

[17] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2):179–227, 2000.

[18] Ewa Deelman and Yolanda Gil (Eds.). Workshop on the challenges of scientific workflows. Technical report, Information Sciences Institute, University of Southern California, May 2006.

[19] Juliana Freire, David Koop, and Luc Moreau, editors. *Provenance and Annotation of Data — International Provenance and Annotation Workshop, IPAW 2008*, volume 5272 of *Lecture Notes in Computer Science*. Springer-Verlag, 2008.

[20] James Frew, Dominic Metzger, and Peter Slaughter. Automatic capture and reconstruction of computational provenance. *Concurrency and Computation: Practice and Experience*, 20(5), 2008.

[21] Joe Futrelle. Tupelo semantic content repository – tutorial on provenance. http://tupeloproject.ncsa.uiuc.edu/node/2, 2008.

[22] Joe Futrelle and Jim Myers. Tracking provenance semantics in heterogeneous execution systems. *Concurrency and Computation: Practice and Experience*, 20(5), 2008.

[23] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS '07: Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 31–40, New York, NY, USA, 2007. ACM.

[24] Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, Victor Tan, Sofia Tsasakou, and Luc Moreau. D3.1.1: An Architecture for Provenance Systems. Technical report, University of Southampton, November 2006.

[25] Paul Groth, Simon Miles, and Luc Moreau. A Model of Process Documentation to Determine Provenance in Mash-ups. *Transactions on Internet Technology (TOIT)*, In publication, 2008.

[26] Paul Groth and Luc Moreau. Recording process documentation for provenance. *IEEE Transactions on Parallel and Distributed Systems*, In publication, September 2009.

[27] PREMIS Working Group. Data dictionary for preservation metadata — final report of the premis working group. Technical report, Preservation Metadata: Implementation Strategies (PREMIS), 2005.

[28] Natalia Kwasnikowska and Jan Van den Bussche. Mapping the nrc dataflow model to the open provenance model. In *Provenance and Annotation of Data and Processes: Second International Provenance and Annotation Workshop, IPAW 2008, Salt Lake City, UT, USA, June 17-18, 2008. Revised Selected Papers*, pages 3–16, Berlin, Heidelberg, 2008. Springer-Verlag.

[29] Bertram Ludäscher, Norbert Podhorszki, Ilkay Altintas, Shawn Bowers, and Timothy M. McPhillips. From computation models to models of provenance: the rws approach. *Concurrency and Computation: Practice and Experience*, 20(5), 2008.

[30] Simon Miles, Paul Groth, Miguel Branco, and Luc Moreau. The requirements of recording and using provenance in e-science experiments. *Journal of Grid Computing*, 5(1):1–25, 2007.

[31] Simon Miles, Paul Groth, Steve Munroe, and Luc Moreau. Prime: A methodology for developing provenance-aware applications. *ACM Transactions on Software Engineering and Methodology, under review*, 2009.

[32] Simon Miles and Luc Moreau. Determining provenance through scoped queries over causal graphs. *To Be Submitted*, 2007.

[33] Luc Moreau and Ian Foster, editors. *Provenance and Annotation of Data — International Provenance and Annotation Workshop, IPAW 2006*, volume 4145 of *Lecture Notes in Computer Science*. Springer-Verlag, May 2006.

[34] Luc Moreau, Juliana Freire, Joe Futrelle, Robert E. McGrath, Jim Myers, and Patrick Paulson. The open provenance model (v1.00). Technical report, University of Southampton, December 2007.

[35] Luc Moreau, Paul Groth, Simon Miles, Javier Vazquez, John Ibbotson, Sheng Jiang, Steve Munroe, Omer Rana, Andreas Schreiber, Victor Tan, and Laszlo Varga. The Provenance of Electronic Data. *Communications of the ACM*, 51(4):52–58, April 2008.

[36] Luc Moreau and Bertram Ludaescher, editors. *Special Issue on the First Provenance Challenge*, volume 20. Wiley, April 2007.

[37] Luc Moreau, Bertram Ludäscher, Ilkay Altintas, Roger S. Barga, Shawn Bowers, Steven Callahan, George Chin Jr., Ben Clifford, Shirley Cohen, Sarah Cohen-Boulakia, Susan Davidson, Ewa Deelman, Luciano Digiampietri, Ian Foster, Juliana Freire, James Frew, Joe Futrelle, Tara Gibson, Yolanda Gil, Carole Goble, Jennifer Golbeck, Paul Groth, David A. Holland, Sheng Jiang, Jihie Kim, David Koop, Ales Krenek, Timothy McPhillips, Gaurang Mehta, Simon Miles, Dominic Metzger, Steve Munroe, Jim Myers, Beth Plale, Norbert Podhorszki, Varun Ratnakar, Emanuele Santos, Carlos Scheidegger, Karen Schuchardt, Margo Seltzer, Yogesh L. Simmhan, Claudio Silva, Peter Slaughter, Eric Stephan, Robert Stevens, Daniele Turi, Huy Vo, Mike Wilde, Jun Zhao, and Yong Zhao. The First Provenance Challenge. *Concurrency and Computation: Practice and Experience*, 20(5):409–418, April 2007.

[38] Luc Moreau (Editor), Beth Plale, Simon Miles, Carole Goble, Paolo Missier, Roger Barga, Yogesh Simmhan, Joe Futrelle, Robert McGrath, Jim Myers, Patrick Paulson, Shawn Bowers, Bertram Ludaescher, Natalia Kwasnikowska, Jan Van den Bussche, Tommy Ellkvist, Juliana Freire, and Paul Groth. The open provenance model (v1.01). Technical report, University of Southampton, July 2008.

[39] Kiran-Kumar Muniswamy-Reddy, Uri Braun, David A. Holland, Peter Macko, Diana Maclean, Daniel Margo, Margo Seltzer, and Robin Smogor. Layering in provenance-aware storage systems. In *Proceedings of 2009 USENIX Annual Technical Conference*, San Diego, CA, June 2009.

[40] Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's type theory: an introduction*. Clarendon Press, New York, NY, USA, 1990.

[41] Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge, 2000.

[42] Satya S. Sahoo, Roger S. Barga, Jonathan Goldstein, and Amit P. Sheth. Provenance algebra and materialized view-based provenance management. Technical report, Microsoft Research, 2008.

[43] Carlos Scheidegger, David Koop, Emanuele Santos, Huy Vo, Steven Callahan, Juliana Freire, and Claudio Silva. Tackling the provenance challenge one layer at a time. *Concurrency and Computation: Practice and Experience*, 20(5), 2008.

[44] Second challenge team contributions. http://twiki.ipaw.info/bin/view/Challenge/ParticipatingTeam June 2007.

[45] Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, September 2005.

[46] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. Querying capabilities of the karma provenance framework. *Concurrency and Computation: Practice and Experience*, 20(5):441–451, 2008.

[47] Issam Souilah, Adrian Francalanza, and Vladimiro Sassone. Provenance in distributed systems. In *First Workshop on the Theory and Practice of Provenance*, 2009.

[48] The third provenance challenge. http://twiki.ipaw.info/bin/view/Challenge/ThirdProvenanceChall February 2009.

[49] Jun Zhao, Carole Goble, Robert Stevens, and Daniele Turi. Mining taverna's semantic web of provenance. *Concurrency and Computation: Practice and Experience*, 20(5), 2008.