# The Open Provenance Model (v1.01)


| | |
|---|---|
| Luc Moreau (Editor) | (U. of Southampton) |
| Beth Plale | (Indiana U.) |
| Simon Miles | (King's College) |
| Carole Goble, Paolo Missier | (Manchester U.) |
| Roger Barga, Yogesh Simmhan | (Microsoft) |
| Joe Futrelle, Robert E. McGrath, Jim Myers | (NCSA) |
| Patrick Paulson | (PNNL) |
| Shawn Bowers, Bertram Ludaescher | (U. Davis) |
| Natalia Kwasnikowska, Jan Van den Bussche | (U. Hasselt) |
| Tommy Ellkvist, Juliana Freire | (U. Utah) |
| Paul Groth | (USC) |


July 17, 2008


**Abstract**

In this paper, we introduce the Open Provenance Model , a model for provenance that is designed to meet the following requirements: (1) To allow provenance information to be exchanged between systems, by means of a compatibility layer based on a shared provenance model. (2) To allow developers to build and share tools that operate on such a provenance model. (3) To define the model in a precise, technology-agnostic manner. (4) To support a digital representation of provenance for any "thing", whether produced by computer systems or not. (5) To define a core set of rules that identify the valid inferences that can be made on provenance graphs.

# 1 Introduction

Provenance is well understood in the context of art or digital libaries, where it respectively refers to the documented history of an art object, or the documentation of processes in a digital object's life cycle [5]. Interest for provenance in the "e-science community" [13] is also growing, since provenance is perceived as a crucial component of workflow systems [2] that can help scientists ensure reproducibility of their scientific analyses and processes.

Against this background, the *International Provenance and Annotation Workshop* (IPAW'06), held on May 3-5, 2006 in Chicago, involved some 50 participants interested in the issues of data provenance, process documentation, data derivation, and data annotation [8, 1]. During a session on provenance standardization, 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 Provenance Challenge was born, and from the outset, the challenge was set up to be *informative* rather than *competitive*. The first Provenance Challenge was 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. Participants simulated or ran a Functional Magnetic Resonance Imaging workflow, from which they implemented and executed a pre-identified set of "provenance queries". Sixteen teams responded to the challenge, and reported their experience in a journal special issue [10].

The first Provenance Challenge was followed by the second Provenance Challenge, aiming at establishing inter-operability of systems, by exchanging provenance information. Thirteen teams [12] responded to this second challenge. Discussions indicated that there was substantial agreement on a core representation of provenance. As a result, following a workshop in August 2007, in Salt Lake City, a data model was crafted and released as the *Open Provenance Model* (v1.00) [9].

The starting point of this work is the community agreement summarized by Miles [7]. We assume that provenance of objects (whether digital or not) is represented by an annotated causality graph, which is a directed acyclic graph, enriched with annotations capturing further information pertaining to execution. For the purpose of this paper, a provenance graph is defined to be *a record of a past execution* (or current execution), and not a description of something that could happen in the future.

The *Open Provenance Model* (OPM) is a model for provenance that is designed to meet the following requirements:

- To allow provenance information to be exchanged between systems, by means of a compatibility layer based on a shared provenance model.

- To allow developers to build and share tools that operate on such provenance model.
- To define the model in a precise, technology-agnostic manner.
- To support a digital representation of provenance for any "thing", whether produced by computer systems or not.
- To define a core set of rules that identify the valid inferences that can be made on provenance graphs.

While specifying this model, we also have some *non*-requirements:

- It is not the purpose of this document to specify the internal representations that systems have to adopt to store and manipulate provenance internally; systems remain free to adopt internal representations that are fit for their purpose.
- It is not the purpose of this document to define a computer-parsable syntax for this model; model implementations in XML, RDF or others will be specified in separate documents, in the future.
- We do not specify protocols to store such provenance information in provenance repositories.
- We do not specify protocols to query provenance repositories.

On June 19th 2008, twenty participants attended the first OPM workshop [3] to discuss the version of the specification. Minutes of the workshop and recommendations [4] were published, and led to the current version (v1.01) of the Open Provenance Model.

# 2 Basics

## 2.1 Entities

Our primary concern is to be able to represent how "things", whether digital data such as simulation results, physical objects such as cars, or immaterial entities such as decisions, came out to be in a given state, with a given set of characteristics, at a given moment. It is recognised that many of such "things" can be stateful: a car may be at various locations, it can contain different passengers, and it can have a tank full or empty; likewise, a file can contain different data at different moments of its existence. Hence, from the perspective of provenance,

we introduce the concept of an *artifact* as an immutable[1] piece of state; likewise, we introduce the concept of a *process* as actions resulting in new artifacts.

A process usually takes place in some context, which enables or facilitates its execution: examples of such contexts are varied and include a place where the process executes, an individual controlling the process, or an institution sponsoring the process. These entities are being referred to as *Agents*. Agents, as we shall see when we discuss causality dependencies, are a cause (like a catalyst) of a process taking place.

The Open Provenance Model is based on these three primary entities, which we define now.

**Definition 1 (Artifact)** *Immutable piece of state, which may have a physical embodiment in a physical object, or a digital representation in a computer system.*

**Definition 2 (Process)** *Action or series of actions performed on or caused by artifacts, and resulting in new artifacts.*

**Definition 3 (Agent)** *Contextual entity acting as a catalyst of a process, enabling, facilitating, controlling, affecting its execution.*

The Open Provenance Model is a model of artifacts *in the past*, explaining how they *were* derived. Likewise, as far as processes are concerned, they may also be in the past, i.e. they may have already completed their execution; in addition, processes can still be currently running (i.e., they have not completed their execution yet). In no case is OPM intended to describe the state of future artifacts and the activities of future processes.

We introduce a graphical notation and a formal definition for provenance graphs. Specifically, artifacts are represented by circles, and are denoted by elements of the set $Artifact$. Likewise, processes are represented graphically by rectangles and denoted by elements of the set $Process$. Finally, agents are represented by octogons and are elements of the set $Agent$ in the formal notation.

## 2.2 Dependencies

A provenance graph aims to capture the causal dependencies between the abovementioned entities. Therefore, a provenance graph is defined as a directed graph, whose nodes are artifacts, processes and agents, and whose edges belong to one of following categories depicted in Figure 1. An edge represents a causal dependency, between its source, denoting the effect, and its destination, denoting the cause.

[1] In the presence of streams, we consider an artifact to be a slice of stream in time, i.e. the stream content at a specific instant in the computation. A future version of OPM will refine the model to accomodate streams fully as they are recognized to be crucial in many applications.

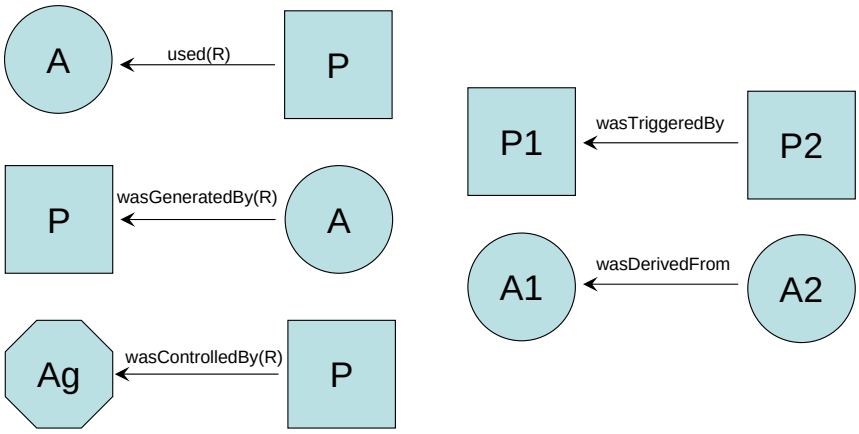


Figure 1: Edges in the Provenance Model

The first two edges express that a process *used* an artifact and that an artifact *was generated by* a process. Since a process may have used several artifacts, it is important to identify the *roles* under which these artifacts were used. (Roles are denoted by the letter $R$ in Figure 1.) Likewise, a process may have generated many artifacts, and each would have a specific *role*. For instance, the division process uses two numbers, with roles dividend and divisor, and produces two numbers, with roles quotient and remainder. Roles are meaningful only in the context of the process where they are defined. The meaning of roles is not defined by OPM but by application domains; OPM only uses roles syntactically (as "tags") to distinguish the involvement of artifacts in processes.

A process is caused by an agent, essentially acting as a catalyst or controller: this causal dependency is expressed by the *was controlled by* edge. Given that a process may have been catalyzed by several agents, we also identify their roles as catalysts. We note that the dependency between an agent and a process represents a control relationship, and not a data derivation relationship. It is introduced in the model to more easily express how a user (or institution) controlled a process.

It is also recognized that we may not be aware of the process that generated some artifact $A_2$, but that artifact $A_2$ was *derived from* another artifact $A_1$. Likewise, we may not be aware of the exact artifact that a process $P_2$ used, but that there was some artifact generated by another process $P_1$. Process $P_2$ is then said to have been *triggered by* $P_1$. Both edges *wasDerivedFrom* and *wasTriggeredBy* are introduced, because they allow a dataflow or process oriented views of past executions to be adopted, according to the preference of system designers. (Since *wasDerivedFrom* and *wasTriggeredBy* are edges that summarize some activities for which all details are not being exposed, it was felt that it was

not necessary to associate a role with such edges.)

As far as conventions are concerned, we note that causality edges use past tense to indicate that they refer to past execution. Causal relationships are defined as follows.

**Definition 4 (Causal Relationship)** *A causal relationship is represented by an arc and denotes the presence of a causal dependency between the source of the arc (the effect) and the destination of the arc (the cause). Five causal relationships are recognized: a process used an artifact, an artifact was generated by a process, a process was triggered by a process, an artifact was derived from an artifact, and a process was controlled by an agent.*

Multiple notions of causal dependencies were considered for OPM. A very strong notion of causal dependency would express that a set of entities was necessary and sufficient to explain the existence of another entity. It was felt that such a notion was not practical, since, with an open world assumption, one could always argue that additional factors may have influenced an outcome (e.g. electricity was used, temperature range allowed computer to work, etc). It was felt that weaker notions, only expressing *necessary dependencies*, were more appropriate. However, even then, one can distinghish data dependencies (e.g. where a quotient is clearly dependent on the dividend and divisor) from a control dependency where the mere presence of some artifact or the beginning of a process can explain the presence of another entity. A number of factors have influenced us to adopt a weak notion of causal dependency for OPM.

- *Expressibility.* It is anticipated that systems will produce descriptions of what their components are doing, without having intimate knowledge of the exact internal data and control dependencies. Weak notions of dependency are necessary for such systems to be able to use OPM in practice.
- *Composability.* We shall see how OPM supports multi-level descriptions (Section 3). In a system consisting of the parallel composition of two subcomponents, the high level summary of the system requires a weaker notion of dependency than the low level descriptions of its subcomponents.

Hence, we adopt the following causal dependencies in OPM. We anticipate that subclasses of these dependencies, capturing stronger notions of causality, may be defined in specific systems, and over time, may be incorporated in OPM.

**Definition 5 (Artifact Used by a Process)** *In a graph, connecting a process to an artifact by a* used *edge is intended to indicate that the process required the availability of the artifact to complete its execution. When several artifacts are connected to a same process by multiple* used *edges, all of them were required for the process to complete.*

Alternatively, a stronger interpretation of the *used* edge would have required the artifact to be available for the process to be able to start. It is believed that such a notion may be useful in some circumstances, and it may be defined as a subtype of *used*. We note that both interpretations of *used* coincide, when processes are modelled as instantaneous.

**Definition 6 (Artifacts Generated by Processes)** *In a graph, connecting an artifact to a process by an edge* wasGeneratedBy *is intended to mean that the process was required to initiate its execution for the artifact to be generated. When several artifacts are connected to a same process by multiple* wasGeneratedBy *edges, the process had to have begun, for all of them to be generated.*

**Definition 7 (Process Triggered by Process)** *A connection of a process $P_2$ to a process $P_1$ by a "was triggered by" edge indicates that the start of process $P_1$ was required for $P_2$ to be able to complete.*

We note that the relationship $P_2\, WasTriggeredBy\, P_1$ (like the other causality relationships we describe in this section) only expresses a *necessary* condition: $P_1$ was required to have started for $P_2$ to be able to complete. This interpretation is weaker than the common sense definition of "trigger", which tends to express a sufficient condition for an event to take place.

**Definition 8 (Artifact Derived from Artifact)** *An edge "was derived from" between two artifacts $A_1$ and $A_2$ indicates that artifact $A_1$ needs to have been generated for $A_2$ to be generated.*

**Definition 9 (Process Controlled by Agent)** *The assertion of an edge "was controlled by" between a process $P$ and an agent $Ag$ indicates that a start and end of process $P$ was controlled by agent $Ag$.*

## 2.3 Roles

A role is an annotation on *used*, *wasGeneratedBy* and *wasControlledBy*.

**Definition 10 (Role)** *A role designates an artifact's or agent's function in a process.*

A role is used to differentiate among several use, generation, or controlling relations.

1. A process may use (generate) more than one artifact. Each *used* (*wasGeneratedBy*) relation may be distinguished by a role with respect to that process. For example, a process may use several files, reading parameters from one, and reading data from another. The used relations would be labeled with distinct roles.

2. An artifact might be used by more than one process, possibly for different purposes. In this case, the *used* relations can be distinguished or said to be the same by the roles associated with the *used* relations. For example, a dictionary might be used by one process to look up the spelling of "provenance", (role = "look up provenance"), while another process uses the same dictionary to hold open the door ( role = "doorstop").

3. An agent may control more than one process. In this case, the different processes may be distinguished by the role associated with the *wasControlledBy* relation. For example, a gardener may control the digging process (role = "dig the bed"), as well as planting a rose bush (role = "plant") and watering the bush (role = "irrigating")

4. A process may be controlled by more than one agent. In this case, each agent might have a distinct control function, which would be distinguished by roles associated with the *wasControlledBy* relations. For example, boarding the train may be controlled by the ticket agent (role = "sell ticket"), the gate agent (role = "take ticket") and the steward (role = "guide to seat").

A role has meaning only within the context of a given process (and/or agent). For a given process, each *used*, *wasGeneratedBy* or *wasControlledBy* relation has a role specific to the process, though the roles may have no meaning outside that process. In general, for a given process (agent) with several arcs, each role should be distinct for that process. However, it is possible, though not recommended, for roles to be the same within a context. For example, baking a cake with two eggs, may define each egg as a separate artifact, and the two used edges might have the identical role, say, egg.

The role is recommended but may be unspecified when not known. It is recommended to give roles whenever possible. For interoperability, communities should define standard sets of roles with agreed meanings. In addition, a reserved value will be defined for "undefined", which should be used when the role is not known or omitted.

## 2.4 Examples

An example illustrating all the concepts and a few of the causal dependencies is displayed in Figure 2. This provenance graph expresses that John baked a cake with ingredients butter, eggs, sugar and flour.

A computational example is displayed in Figure 3. The final data product is a scientific-grade mosaic of the sky, which was produced by a process that used scientific images in FITS format (such as the Sloan Digital Sky Survey data set) and a parameter indicating the size of the mosaic to be produced. The process was caused by the Pegasus/Condor Dagman agent.

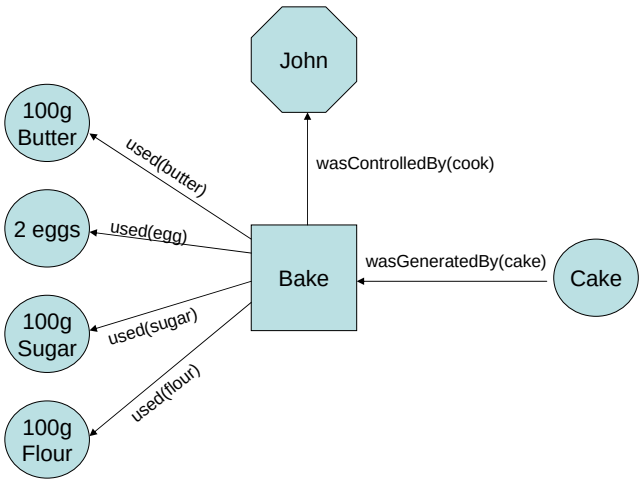


Figure 2: Victoria Sponge Cake Provenance

While graphs can be constructed by incrementally connecting artifacts, processes, and agents with individual edges, the meaning of the causality relations can be understood in the context of all the *used* (or *wasGeneratedBy*) edges, for each process. By connecting a process to several artifacts by *used* edges, we are not just stating the individual inputs to the process. We are asserting a causal dependency expressing that the process could take place and complete only because *all* these artifacts were available. Likewise, when we express that several artifacts were generated by a process, we mean that these artifacts would not have existed if the process had not begun its execution; furthermore, *all* of them were generated by the process; one could not have been generated without the others. The implication is that any single generated artifact is caused by the process, which itself is caused by the presence of all the artifacts it used. We will use such a property to derive transitive closures of causality relations in Section 6.

We can see here the crucial difference between artifacts and the data they represent. For instance, the *data* may have existed, but the particular *artifact* did not. For example, a BLAST search can be given a DNA sequence and return a set of "similar" DNA sequences; however, these returned sequences all existed prior to the process (BLAST) invocation, but the artifacts are novel.

As illustrated by the two examples above, the entities and edges introduced in Figure 1 allow us to capture many of the use cases we have come across in the provenance literature. However, they do not allow us to provide descriptions at multiple level of abstractions, or from different view points. To support these, we allow multiple descriptions of a same execution to coexist.

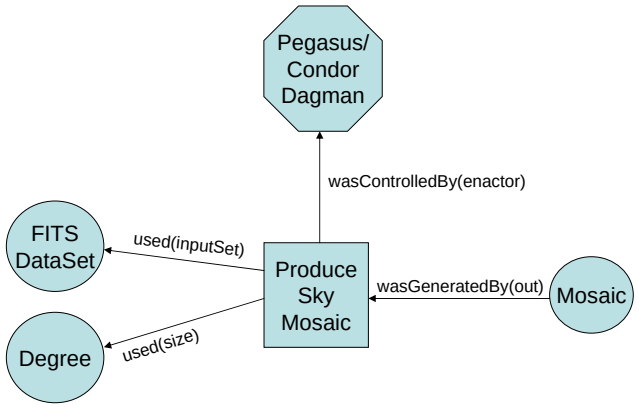


Figure 3: Montage Provenance

# 3 Overlapping and Hierarchichal Descriptions

Figure 4 shows two examples of provenance graphs describing what led the list (3,7) to being as it is. According to the left-hand graph, the list was generated by a process that added one to all constituents of the list (2,6). According to the right-hand graph, the derivation process of (3,7) required the list to be created from values 3 and 7, respectively obtained by adding one to 2 and 6, themselves being the data products obtained by accessing the contents of the original list (2,6).

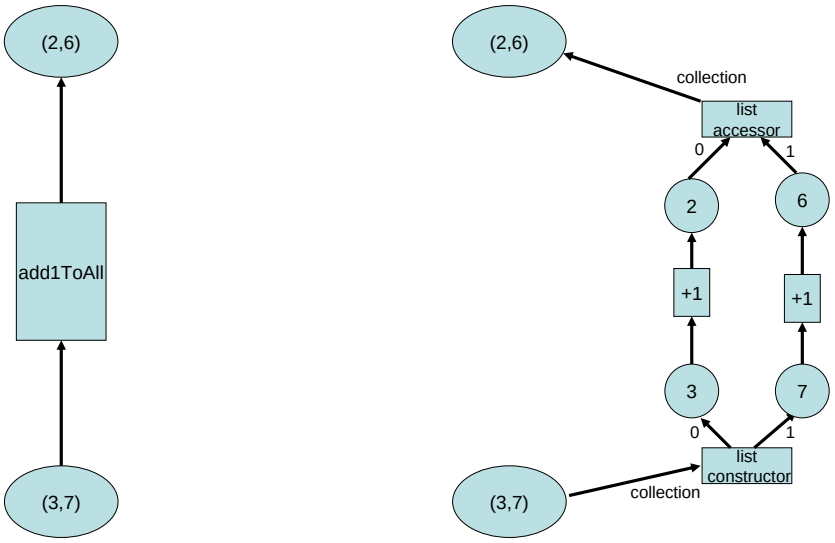


Figure 4: Examples Provenance Graph

Assuming these two graphs refer to the same lists (2,6) and (3,7), they provide two different explanations of how (3,7) was derived from (2,6): these explanations would offer different levels of details about the same derivation. The requirement of providing details at different levels of abstraction or from different viewpoints is common for provenance systems, and hence, we would expect both accounts to be integrated in a single graph. In Figure 5, we see how the two provenance graphs of Figure 4 were integrated, by selecting different colors for nodes and edges. The darker (green) part belonged to the left graph of Figure 4, whereas the lighter (orange) part is the alternate description from the right graph of Figure 4. (Graphs in this paper are better viewed in color.) The darker and lighter subgraphs are two different overlapping *accounts* of the same past execution, offering different levels of explanation for such execution. Such subgraphs are said to be *overlapping accounts* because they share some common nodes (2,6) and (3,7). Furthermore, the lighter part (orange) provides more details than the darker subgraph (green): the lighter part is said to be a *refinement* of the darker grapher.

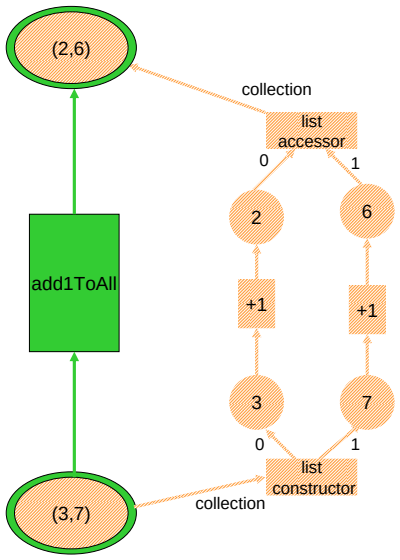


Figure 5: Example of Overlapping and Hierarchical Accounts in a Provenance Graph

Observing Figure 5, it becomes crucial to contrast the edges originating from artifact (3,7) with those originating from the list constructor process. Indeed, the *used* edges out of the list constructor process mean that *both* artifacts 3 and 7 were required for the process to take place. On the contrary, since the edges out of artifact (3,7) are colored differently, they indicate that alternate explanations exist for the process that led to such artifact being as it is. Using the analogy

of AND/OR graphs, a process with *used* edges corresponds to an AND-node, whereas an artifact with *wasGeneratedBy* edges from different accounts represent an OR-node.

It is possible to use refinements repeatedly to create a hierarchy of accounts, as illustrated in Figure 6. We see that a third account (blue) is introduced, to explain how one of the +1 processes was performed.

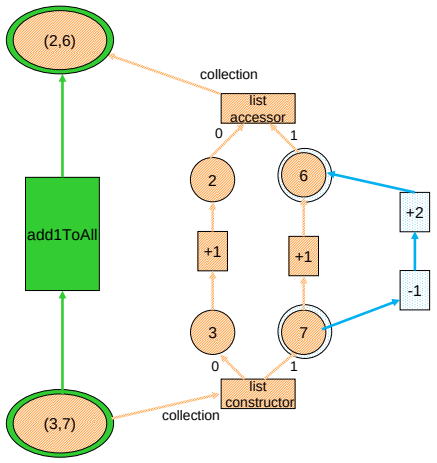


Figure 6: Hierarchy of Accounts in a Provenance Graph

By combining several accounts, we can obtain cycles, as illustrated by Figure 7. Here, in the first view (darker, orange account), a description of two processes P1a and P1b is presented, and their dependencies on artifacts A1, A2 and A3. In the second view (lighter, blue account), it is stated that the two processes P1a and P1b are in fact a single process operating on input A2 and producing A1 and A3. If we combine the two views, a circle has been created: $A_2 \rightarrow P_2 \rightarrow A_1 \rightarrow P_1 \rightarrow A_2$.

While overlapping accounts are intended to allow various descriptions of a same execution, it is recognized that these accounts may differ in their description's semantics. In general, such semantic differences may not be expressed by structural properties we can set constraints on in the model (beyond the constraints identified in this document).

# 4 Provenance Graph Definition

The open provenance model is defined according to the following rules, which we formalise in Section 5.

1. *Accounts* are entities that we assume can can be compared.

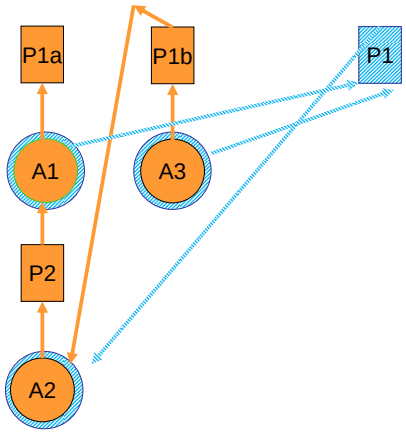


Figure 7: Multiple Accounts Creating Cycle

2. *Artifacts* are identified by unique identifiers. Artifacts contain a placeholder for a domain specific value or reference to a piece of state. Two artifacts are equal if and only if they have the same identifier (irrespective of their placeholder contents[2]). Artifacts can optionally belong to accounts: account membership is declared by listing the accounts an artifact belongs to.

3. *Processes* are identified by unique identifiers. Processes contain a placeholder for domain specific values or references. Two processes are equal if and only if they have the same identifier (irrespective of their placeholder contents). Processes can optionally belong to accounts: account membership is declared by listing the accounts a process belongs to.

4. *Agents* are identified by unique identifiers. Agents contain a placeholder for domain specific values or references. Two agents are equal if and only if they have the same identifier (irrespective of their placeholder contents). Agents can optionally belong to accounts: account membership is declared by listing the accounts an agent belongs to.

5. *Edges* are identified by their source, destination, and role (for those that include a role). The source and destination consist of identifiers for artifacts, processes, or agents, according to Figure 1. Edges can also op-

[2]In the Open Provenance Model, artifact identifiers are the only way to distinguish artifacts in the graph structure. Two artifacts differ if they have different ids, even though they may refer to a same application data product. Two different artifacts are therefore separate nodes in a provenance graph: they have two different computational histories. Given that an artifact represents an instantaneous state of an object, one expect the actural data for a given artifact to remain constant over time.

tionally belong to accounts: account membership is defined by listing the accounts an edge belongs to. Structural equality applies to edges: two edges *used*, *wasGeneratedBy* and *wasControlledBy* are equal if they have the same source, the same destination, the same role, and the same accounts; two edges *wasTriggeredBy* and *wasDerivedFrom* are equal if they have the same source, the same destination, and the same accounts. The meaning of roles is not defined by OPM but by application domains; OPM only uses roles syntactically (as "tags") to distinguish the involvement of artifacts in processes.

6. Roles are mandatory in edges *used*, *wasGeneratedBy* and *wasControlledBy*. The meaning of a role is defined by the semantics of the process they relate to. Role semantics is beyond the scope of OPM.

7. To ensure that edges establish a causal connection between actual causes and effects, the model assumes that if an edge belongs to an account, then its source and destination also belong to this account. In other words, the *effective account membership* of an artifact/process/agent is its declared account membership and the account membership of the edges it is souce and destination of.

8. An *OPM graph* is a set of artifacts, processes, agents, edges, and accounts, as specified above. OPM graphs may be disconnected. The empty set is an OPM graph. A singleton containing an artifact, a process or an agent is an OPM graph. The set of OPM graphs is closed under the intersection and union operations, i.e. the intersection of two OPM graphs is an OPM graph (and likewise for union). We note at this stage that syntactically valid OPM graphs may not necessarily make sense from a provenance viewpoint. Rules below refine the OPM graph concept.

9. A view of an OPM graph according to *one* account, referred to as *account view*, is the set of elements whose effective account membership for artifacts, processes, and agents, and account membership for edges contain the account.

10. While cycles can be expressed in the syntax of OPM, a *legal account view* is defined as an acyclic account view, which contains at most one *wasGeneratedBy* edge per artifact. This ensures that within one account, an OPM graph captures proper causal dependencies, and that a single explanation of the origin of an artifact is given.

11. Hence, a *legal OPM graph* is one for which all account views are legal.

12. Legal account views are OPM graphs. The union of two legal account views is an OPM graph (it is not necessarily a legal view since it may contain cycles). The intersection of two legal account views is a legal account view.

13. Two account views can be declared to be *overlapping* to express the fact that they represent different descriptions of an execution.

14. A declaration that two views are overlapping is *legal* if the views have some artifact, process or agent in common. Whilst one could infer whether two graphs actually overlap, this would typically require the graphs to be parsed fully in order to make such an inference; instead, we rely on explicit declarations of such overlapping properties to facilitate the processing and traversal of graphs.

15. An account view $v_1$ can be declared to be a *refinement* of another account view $v_2$ to express the fact $v_1$ provides further details about an execution than $v_2$.

16. A declaration that a view is a refinement of another *legal* if the views have some common "input" artifact and "output" artifact. (Definition needs to be refined!)

17. A *provenance graph* is a legal OPM graph where overlapping and refined views are legal.

18. *Edges* can optionally be annotated with time information. This aspect is discussed in Section 7.

19. A provenance graph does not need to contain time annotations.

Having defined the concept of a provenance graph, we now study its formal specification.

# 5 Timeless Formal Model

Figure 8 provides a set-theoretic definition [11, 6] of the open provenance model, based on the concepts introduced so far. The model of causality we propose is timeless since time precedence does not imply causality: if a process $P_1$ occurs before a process $P_2$, in general, we cannot infer that $P_1$ caused $P_2$ to happen. However, the converse implication holds assuming time is measured according to a single clock.

Even though the provenance model is timeless, we recognize the importance of time, since time is easily observable by computer systems or users. Hence, in Section 7, we examine how the causality graph can be annotated with time. We will also specify constraints that one would expect time annotations to satisfy (in terms of monotonicity with respect to time) in sound causality graphs.

We assume the existence of a few primitive sets: identifiers for processes, artifacts and agents, roles, and accounts. These sets of identifiers provide indentifies to the corresponding entities within the scope of a given provenance

graph. A given serialization will standardize on these sets, and provide concrete representations for them.

It is important to stress that the purpose of these identifiers is to define the structure of graphs: they are not meant to define identities that are persistent and reliably resolvable over time.

In the model, processes, artifacts and agents are identified by their IDs, and are associated with a value and zero or more accounts — noted $\mathbb{P}(Account)$, the powerset notation. In the set-theoretic notation, identifiers map to the corresponding value and account membership. In other words, with a database perspective, elements of $ProcessId$, $ArtifactId$ and $AgentId$ are keys to processes, artifacts and agents, respectively.

The five causality edges can be easily specified by sets *Used*, *WasGeneratedBy*, *WasTriggeredBy*, *WasDerivedFrom*, and *WasControlledBy* making use of identifiers for artifacts, processes or agents, roles, and the associated accounts.

Finally, an OPM graph needs to identify explicitly which accounts are overlapping or refinements. For this, we use a set $Overlaps$ enumerating lists of overlapping accounts, and a set $Refines$ enumerating lists of refined accounts.

$$
\begin{array}{rcll}
ProcessId & : & \textit{primitive set} & \text{(Process Identifiers)} \\
ArtifactId & : & \textit{primitive set} & \text{(Artifact Identifiers)} \\
AgentId & : & \textit{primitive set} & \text{(Agent Identifiers)} \\
Role & : & \textit{primitive set} & \text{(Roles)} \\
Account & : & \textit{primitive set} & \text{(Accounts)} \\
Value & : & \textit{application specific set} & \text{(Values)} \\
Process & = & ProcessId \rightarrow Value \times \mathbb{P}(Account) & \\
Artifact & = & ArtifactId \rightarrow Value \times \mathbb{P}(Account) & \\
Agent & = & AgentId \rightarrow Value \times \mathbb{P}(Account) & \\
Used & = & ProcessId \times Role \times ArtifactId \times \mathbb{P}(Account) & \\
WasGeneratedBy & = & ArtifactId \times Role \times ProcessId \times \mathbb{P}(Account) & \\
WasTriggeredBy & = & ProcessId \times ProcessId \times \mathbb{P}(Account) & \\
WasDerivedFrom & = & ArtifactId \times ArtifactId \times \mathbb{P}(Account) & \\
WasControlledBy & = & ProcessId \times Role \times AgentId \times \mathbb{P}(Account) & \\
Overlaps & = & Account \times Account & \\
Refines & = & Account \times Account & \\
OPMGraph & = & Artifact \times Process & \\
 & & \times Agent \times \mathbb{P}(Used) & \\
 & & \times \mathbb{P}(WasGeneratedBy) \times \mathbb{P}(WasTriggeredBy) & \\
 & & \times \mathbb{P}(WasDerivedFrom) \times \mathbb{P}(WasControlledBy) & \\
 & & \times \mathbb{P}(Overlaps) \times \mathbb{P}(Refines) &
\end{array}
$$

Figure 8: Timeless Causality Graph Data Model

The model of Figure 8 specifies all the necessary building blocks for creating OPM graphs. We now revisit the definition provided by Section 4, re-examining

each item, and explaining it in terms of the formal model.

1. Accounts are elements of the set $Account$.

2. All artifacts of a graph must have identifiers belonging to the set $ArtifactId$. A function $A$ of type $Artifact$ is total on the set $ArtifactId$. For an artifact id $a$, account membership is $A(a).acc$. In OPM, the artifact entity contains a placeholder, $A(a).value$, for application specific values or references to the actual piece of state.

3. All processes of a graph must have identifiers belonging to the set $ProcessId$. A function $P$ of type $Process$ is total on the set of $ProcessId$. For a process id $p$, account memberhsip is $P(p).acc$. A process contains a placeholder $P(p).value$ for application specific valuers or references to the actual process.

4. All agents of a graph must have identifiers belonging to the set $AgentId$. For the total function $AG$, and for an agent id $ag$, account memberhsip is $AG(ag).acc$. Placeholder for the actual agent value is $AG(ag).value$.

5. Equality on edges is defined as follows:

   For any *used* edges $u_1 = \langle p_1, r_1, a_1, acc_1 \rangle \in Used$ and $u_2 = \langle p_2, r_2, a_2, acc_2 \rangle \in Used$, $u_1 = u_2$ if $p_1 = p_2$, $a_1 = a_2$, $r_1 = r_2$, $acc_1 = acc_2$.

   For any *wasGeneratedBy* edges $g_1 = \langle a_1, r_1, p_1, acc_1 \rangle \in WasGeneratedBy$ and $g_2 = \langle a_2, r_2, acc_2 \rangle \in Used$, $g_1 = g_2$ if $p_1 = p_2$, $a_1 = a_2$, $r_1 = r_2$, $acc_1 = acc_2$.

   For any *wasControlledBy* edges $c_1 = \langle p_1, r_1, ag_1, acc_1 \rangle \in WasControlledBy$ and $ag_2 = \langle p_2, r_2, ag_2, acc_2 \rangle \in WasControlledBy$, $c_1 = c_2$ if $p_1 = p_2$, $ag_1 = ag_2$, $r_1 = r_2$, $acc_1 = acc_2$.

   For any *wasDerivedFrom* edges $d_1 = \langle a_1, a_1', acc_1 \rangle \in WasDerivedFrom$ and $d_2 = \langle a_2, a_2', acc_2 \rangle \in DerivedFrom$, $d_1 = d_2$ if $a_1 = a_2$, $a_1' = a_2'$, $acc_1 = acc_2$.

   For any *wasTriggeredBy* edges $t_1 = \langle p_1, p_1', acc_1 \rangle \in WasTriggeredBy$ and $t_2 = \langle p_2, p_2', acc_2 \rangle \in WasTriggeredBy$, $t_1 = t_2$ if $p_1 = p_2$, $p_1' = p_2'$, $acc_1 = acc_2$.

6. The model does not place any constraints on roles, beyond their membership to the set $Role$.

7. We introduce a convenience function $accountOf^{gr}$ operating on entities of a graph $gr$. For a given OPM graph $gr = \langle A, P, AG, U, G, T, D, C, Ov, Re \rangle$, where $A \in Artifact, P \in Process, AG \in Agent$, and $U \subseteq Used, G \subseteq$

$WasGeneratedBy, T \subseteq WasTriggeredBy, D \subseteq WasDerivedFrom, C \subseteq WasControlledBy, Ov \subseteq Overlapping, Re \subseteq Refinement$

$$
\begin{aligned}
accountOf^{gr}(p) &= P(p).acc \\
accountOf^{gr}(a) &= A(a).acc \\
accountOf^{gr}(ag) &= AG(ag).acc \\
accountOf^{gr}(u) &= acc \text{ where } u = \langle p, r, a, acc\rangle \in U \\
accountOf^{gr}(g) &= acc \text{ where } g = \langle a, r, p, acc\rangle \in G \\
accountOf^{gr}(t) &= acc \text{ where } t = \langle p_1, p_2, acc\rangle \in T \\
accountOf^{gr}(d) &= acc \text{ where } d = \langle a_1, a_2, acc\rangle \in D \\
accountOf^{gr}(c) &= acc \text{ where } c = \langle p, r, ag, acc\rangle \in C
\end{aligned}
$$

We then introduce *effectiveAccountOf*:

$$
\begin{aligned}
&effectiveAccountOf^{gr}(p) \\
&= accountOf^{gr}(p) \\
&\quad \cup_{i,j,k} accountOf^{gr}(u_{i,j,k}) \text{ where } u_{i,j,k} = \langle p, r_i, a_j, acc_k\rangle \in U \\
&\quad \cup_{i,j,k} accountOf^{gr}(d_{i,j,k}) \text{ where } d_{i,j,k}\langle a_i, r_j, p, acc_k\rangle \in G \\
&\quad \cup_{i,j} accountOf^{gr}(t_{i,j}) \text{ where } t_{i,j} = \langle p, p_i, acc_j\rangle \in T \\
&\quad \cup_{i,j} accountOf^{gr}(t_{i,j}) \text{ where } t_{i,j} = \langle p_i, p, acc_j\rangle \in T \\
&\quad \cup_{i,j,k} accountOf^{gr}(c_{i,j,k}) \text{ where } c_{i,j,k} = \langle p, r_i, ag_j, acc_k\rangle \in C
\end{aligned}
$$

(It is defined similarly for artifacts and agents.)

8. No topological restriction is placed on OPM graphs. For instance, $\langle p, r_1, a, \emptyset\rangle \in U$ and $\langle a, r_2, p, \emptyset\rangle \in G$ are two acceptable edges of an OPM graph, which would create a circularity.

   If $gr_1 = \langle A_1, P_1, AG_1, U_1, G_1, T_1, D_1, C_1, Ov_1, Re_1\rangle$ and $gr_2 = \langle A_2, P_2, AG_2, U_2, G_2, T_2, D_2, C_2, Ov_2, Re_2\rangle$, then $gr_1 \cup gr_2 = \langle A_1 \sqcup A_2, P_1 \sqcup P_2, AG_1 \sqcup AG_2, U_1 \cup U_2, G_1 \cup G_2, T_1 \cup T_2, D_1 \cup D_2, C_1 \cup C_2, Ov_1 \cup Ov_2, Re_1 \cup Re_2\rangle$, where the $\sqcup$ operator is define as: $A_1 \sqcup A_2(x) = \langle v, a_1 \cup a_2\rangle$ with $A_1(x) = \langle v, a_1\rangle$ and $A_2(x) = \langle v, a_2\rangle$.

   If $gr_1 = \langle A_1, P_1, AG_1, U_1, G_1, T_1, D_1, C_1, Ov_1, Re_1\rangle$ and $gr_2 = \langle A_2, P_2, AG_2, U_2, G_2, T_2, D_2, C_2, Ov_2, Re_2\rangle$, then $gr_1 \cap gr_2 = \langle A_1 \sqcap A_2, P_1 \sqcap P_2, AG_1 \sqcap AG_2, U_1 \cap U_2, G_1 \cap G_2, T_1 \cap T_2, D_1 \cap D_2, C_1 \cap C_2, Ov_1 \cap Ov_2, Re_1 \cap Re_2\rangle$, where the $\sqcap$ operator is define as: $A_1 \sqcap A_2(x) = \langle v, a_1 \cap a_2\rangle$ with $A_1(x) = \langle v, a_1\rangle$ and $A_2(x) = \langle v, a_2\rangle$.

   If $gr_1, gr_2 \in OPMGraph$, then

   $$gr_1 \cup gr_2 \in OPMGraph$$

   and

   $$gr_1 \cap gr_2 \in OPMGraph.$$

9. For an OPMGraph $gr = \langle A, P, AG, U, G, T, D, C, Ov, Re \rangle$, for an account $\alpha$, $view(\alpha, gr)$ is $\langle A_\alpha, P_\alpha, AG_\alpha, U_\alpha, G_\alpha, T_\alpha, D_\alpha, C_\alpha, Ov, Re \rangle$, where:

$$\begin{aligned}
A_\alpha \subseteq A \quad &\text{with} \quad A_\alpha = \{(a, acc) \in A \text{ such that } \alpha \in \mathit{effectiveAccountOf}^{gr}(a)\} \\
P_\alpha \subseteq P \quad &\text{with} \quad P_\alpha = \{(p, acc) \in P \text{ such that } \alpha \in \mathit{effectiveAccountOf}^{gr}(p)\} \\
AG_\alpha \subseteq AG \quad &\text{with} \quad AG_\alpha = \{(ag, acc) \in AG \text{ such that } \alpha \in \mathit{effectiveAccountOf}^{gr}(ag)\} \\
U_\alpha \subseteq U \quad &\text{with} \quad U_\alpha = \{\langle p, r, a, acc \rangle \in U \text{ such that } \alpha \in acc\} \\
G_\alpha \subseteq G \quad &\text{with} \quad G_\alpha = \{\langle a, r, p, acc \rangle \in G \text{ such that } \alpha \in acc\} \\
T_\alpha \subseteq T \quad &\text{with} \quad T_\alpha = \{\langle p_1, p_2, acc \rangle \in T \text{ such that } \alpha \in acc\} \\
D_\alpha \subseteq D \quad &\text{with} \quad D_\alpha = \{\langle a_1, a_2, acc \rangle \in D \text{ such that } \alpha \in acc\} \\
C_\alpha \subseteq C \quad &\text{with} \quad C_\alpha = \{\langle p, ag, acc \rangle \in C \text{ such that } \alpha \in acc\}
\end{aligned}$$

10. A legal account view $gr = \langle A, P, AG, U, G, T, D, C, Ov, Re \rangle$ is such that there is no cycle in $U, G, T, D$ and if $\langle a_1, r_1, p_1, acc_1 \rangle \in G$ and $\langle a_1, r_2, p_2, acc_1 \rangle \in G$, then $\langle a_1, r_1, p_1, acc_1 \rangle = \langle a_1, r_2, p_2, acc_1 \rangle$, where $acc_1$ is a singleton.

11. Two accounts $\alpha_1, \alpha_2$ are declared to be overlapping in an OPMgraph $gr = \langle A, P, AG, U, G, T, D, C, Ov, Re \rangle$, if $\langle \alpha_1, \alpha_2 \rangle \in Ov$ or $\langle \alpha_2, \alpha_1 \rangle \in Ov$.

12. Two accounts $\alpha_1, \alpha_2$ are declared to be legally overlapping in an OPMgraph if they are overlapping and if their respective account views $\langle A_1, P_1, AG_1, U_1, G_1, T_1, D_1, C_1, Ov_1, Re_1 \rangle$ and $\langle A_2, P_2, AG_2, U_2, G_2, T_2, D_2, C_2, Ov_2, Re_2 \rangle$ are such that

$$\begin{aligned}
&\mathit{Domain}(A_1) \cap \mathit{Domain}(A_2) \neq \emptyset \\
\mathit{or} \quad &\mathit{Domain}(P_1) \cap \mathit{Domain}(P_2) \neq \emptyset \\
\mathit{or} \quad &\mathit{Domain}(AG_1) \cap \mathit{Domain}(AG_2) \neq \emptyset.
\end{aligned}$$

Hence, the overlapping relationship is reflexive, symmetric but *not* transitive.

13. An account $\alpha_1$ is declared to refine account $\alpha_2$ in an OPMgraph $gr = \langle A, P, AG, U, G, T, D, C, Ov, Re \rangle$, if $\langle \alpha_1, \alpha_2 \rangle \in Re$.

14. An account $\alpha_1$ is declared to be legally refining account $\alpha_2$ in an OPMgraph if they are overlapping and if their respective account views $gr_1 = \langle A_1, P_1, AG_1, U_1, G_1, T_1, D_1, C_1, Ov_1, Re_1 \rangle$ and $gr_2 = \langle A_2, P_2, AG_2, U_2, G_2, T_2, D_2, C_2, Ov_2, Re_2 \rangle$ are such that

$$\begin{aligned}
&\mathit{source}(gr_2) \subseteq \mathit{source}(gr_1) \\
\mathit{and} \quad &\mathit{sink}(gr_2) \subseteq \mathit{sink}(gr_1)
\end{aligned}$$

***Concept is currently ill-defined. Definition remaining to be finalised. Can we define refinement just on syntactic properties of the graphs?*** Hence, the refinement relationship is reflexive, asymmetric and transitive.

# 6 Inferences

The Open Provenance Model has defined the notion of *OPM graph* based on a set of syntactic rules and the notion of *Provenance Graph* adding a set of topological constraints. Provenance graphs are aimed at representing causality graphs explaining how processes and artifacts came out to be. It is expected that a variety of reasoning algorithms will exploit this data model, in order to provide novel and powerful functionality to users. It is beyond the scope of this document to include an extensive coverage of relevant reasoning algorithms. However, provenance graphs, by means of edges, capture causal dependencies, which can be summarised by means of transitive closure that we describe in this section.

## 6.1 One Step Inferences

In Section 2, we have introduced the two causal dependencies *wasTriggeredBy* and *wasDerivedFrom* acting as abbreviation for causal dependencies *used* and *wasGeneratedBy*. Figure 9 shows their exact meaning.

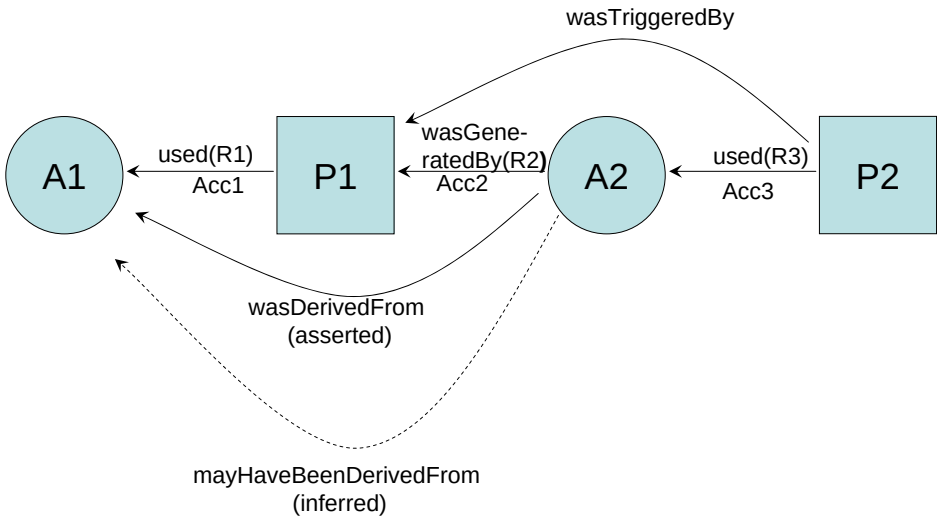


Figure 9: One Step Inference in the Provenance Model

Figures 10 and 11 formalize Figure 9 by introducing rules for each inference that can be performed in the Open Provenance Model. A rule consists of two expressions separated by a horizontal line. The expression above the line is a hypothesis, whereas the expression below the line is a conclusion that can be inferred from the hypothesis.

In Equation (1), a *wasTriggeredBy* edge is inferred from the existence of a *used* and *wasGeneratedBy* edges, as per described in Figure 9. We note that the

inferred *wasTriggeredBy* edge relies on both accounts $acc_2$ and $acc_3$, hence, it is given $acc_2 \cup acc_3$ as account.

$$\frac{\langle p_2, r_3, a_2, acc_3\rangle \in \mathit{Used} \ \wedge \ \langle a_2, r_2, p_1, acc_2\rangle \in \mathit{WasGeneratedBy}}{\langle p_2, p_1, acc_2 \cup acc_3\rangle \in \mathit{WasTriggeredBy}} \tag{1}$$

$$\frac{\langle p_2, p_1, acc\rangle \in \mathit{WasTriggeredBy}}{\begin{array}{ll}\exists a_2, r_2, r_3, acc_2, acc_3, & \langle p_2, r_3, a_2, acc_3\rangle \in \mathit{Used} \\ & \wedge \langle a_2, r_2, p_1, acc_2\rangle \in \mathit{WasGeneratedBy} \\ & \wedge \ acc_2 \cup acc_3 = acc\end{array}} \tag{2}$$

Figure 10: One Step Inference Rules (1)

Equation (2) is the reverse of Equation (1): it allows us to establish that the edge $\langle p_2, p_1, acc\rangle \in \mathit{WasTriggeredBy}$ is hiding the existence of some artifact $a_2$, used by $p_2$ and generated by $p_1$. The inferred edges *used* and *wasGeneratedBy* are asserted in the context of some account $acc_2$ and $acc_3$, whose union is the original account *acc*. We note that Equation (2) allows us to establish the existence of some artifact $a_2$ (and $r_1, r_2, acc_1, acc_2$) but it does not tell us what their ids and values are. This is the consequence of using *wasTriggeredBy*, which is a lossy summary of the composition of *used* and *wasGeneratedBy*.

The kind of inferences that can be made about *wasDerivedFrom* is of a different nature. Indeed, without any internal knowledge of $P_1$ in Figure 9, it is impossible to ascertain there is an actual data dependency between $A_1$ and $A_2$.

> **Remark.** Concretely, a rule such as the following would lead to incorrect inferences since it allows arbitrary outputs to a process to be inferred to be dependent on arbitrary inputs to the same process.
>
> $$\frac{\langle a_2, r_2, p_1, acc_2\rangle \in \mathit{WasGeneratedBy} \wedge \langle p_1, r_1, a_1, acc_1\rangle \in \mathit{Used}}{\langle a_2, a_1, acc_1 \cup acc_2\rangle \in \mathit{WasDerivedFrom}}$$

While it is unreasonable to infer an exact dependency by means of *wasDerivedFrom*, it is useful to be able to infer that a dependency *may* exist. To this end, we introduce the edge *mayHaveBeenDerivedFrom* that marks such a potential dependency. Hence, if $\langle a_1, a_2\rangle \in \mathit{WasDerivedFrom}$, then $\langle a_1, a_2\rangle \in \mathit{MayHaveBeenDerivedFrom}$, but not vice-versa. Hence, Equation (3) states that a *mayHaveBeenDerivedFrom* edge can be derived from the existence of a succession of *wasGeneratedBy* and *used* edges. Equation (4) is to (2) what *wasDerivedFrom* is to *wasTriggeredBy*.

In rules 1 and 3, the inferred edges have accounts $acc_2 \cup acc_3$ and $acc_1 \cup acc_2$, respectively. Hence, the artifacts and processes connected by these edges will have an effective account membership modified accordingly. We note that rules 1 and 3 effectively creates relationships in the union of multiple account views.

$$\frac{\langle a_2, r_2, p_1, acc_2\rangle \in \mathit{WasGeneratedBy} \wedge \langle p_1, r_1, a_1, acc_1\rangle \in \mathit{Used}}{\langle a_2, a_1, acc_1 \cup acc_2\rangle \in \mathit{MayHaveBeenDerivedFrom}} \tag{3}$$

$$\frac{\langle a_2, a_1, acc\rangle \in \mathit{WasDerivedFrom}}{\begin{array}{ll}\exists p_1, r_1, r_2, acc_1, acc_2, & \langle a_2, r_2, p_1, acc_2\rangle \in \mathit{WasGeneratedBy} \\ & \wedge \langle p_1, r_1, a_1, acc_1\rangle \in \mathit{Used} \\ & \wedge\ acc_1 \cup acc_2 = acc\end{array}} \tag{4}$$

Figure 11: One Step Inference Rules (2)

## 6.2 Transitive Closure

Users want to find out the causes of an artifact, not due to one process, but potentially, due to an unknown number of them.

Hence, for the purpose of expressing queries or expressing inferences about provenance graphs, we introduce *four new relationships*, which are transitive versions of existing relationships, namely *Used*$^*$, *WasGeneratedBy*$^*$, *WasDerivedFrom*$^*$ and *WasTriggeredBy*$^*$. Their definitions are displayed in Figure 12. We note that Figure 12 contains definitions (as opposed to inference rules of Figures 10 and 11, which specify which edges can be inferred from which edges). For convenience, we have also introduced a generic causal dependency *wasDependentOn*$^*$ (see equations (9) to (12)). Note that similar inference rules can be defined for *MayHaveBeenDerivedFrom*.

Equations (7) and (8) are one of the multiple possible ways of defining edges *used*$^*$ and *wasGeneratedBy*$^*$. Other definitions could be expressed and proved equivalent (such as *used*$^*$ can be derived from a single *used* and *wasDerivedFrom*$^*$).

$$\langle a_2, a_1, acc\rangle \in \mathit{WasDerivedFrom}^* \quad (5)$$
$$\text{if } a_2 = a_1 \vee \exists a_3,\ \langle a_2, a_3, acc_2\rangle \in \mathit{WasDerivedFrom} \wedge \langle a_3, a_1, acc_1\rangle \in \mathit{WasDerivedFrom}^* \wedge acc = acc_1 \cup acc_2$$

$$\langle p_2, p_1, acc\rangle \in \mathit{WasTriggeredBy}^* \quad (6)$$
$$\text{if } p_2 = p_1 \vee \exists p_3,\ \langle p_2, p_3, acc_2\rangle \in \mathit{WasTriggeredBy} \wedge \langle p_3, p_1, acc_1\rangle \in \mathit{WasTriggeredBy}^* \wedge acc = acc_1 \cup acc_2$$

$$\langle p, a, acc\rangle \in \mathit{Used}^* \quad (7)$$
$$\text{if } \exists p_2, r, acc_1, acc_2,\ \langle p, p_2, acc_2\rangle \in \mathit{WasTriggeredBy}^* \wedge \langle p_2, r, a, acc_1\rangle \in \mathit{Used} \wedge acc = acc_1 \cup acc_2$$

$$\langle A, P, acc\rangle \in \mathit{WasGeneratedBy}^* \quad (8)$$
$$\text{if } \exists p_2, R, acc_1, acc_2,\ \langle A, R, p_2, acc_2\rangle \in \mathit{WasGeneratedBy} \wedge \langle p_2, P, acc_1\rangle \in \mathit{WasTriggeredBy}^* \wedge acc = acc_1 \cup acc_2$$

$$\langle A, P, acc\rangle \in \mathit{WasDependentOn}^* \quad \text{if} \quad \langle A, P, acc\rangle \in \mathit{WasGeneratedBy}^* \quad (9)$$
$$\langle a_1, a_2, acc\rangle \in \mathit{WasDependentOn}^* \quad \text{if} \quad \langle a_1, a_2, acc\rangle \in \mathit{WasDerivedFrom}^* \quad (10)$$
$$\langle p_1, p_2, acc\rangle \in \mathit{WasDependentOn}^* \quad \text{if} \quad \langle p_1, p_2, acc\rangle \in \mathit{WasTriggeredBy}^* \quad (11)$$
$$\langle P, A, acc\rangle \in \mathit{WasDependentOn}^* \quad \text{if} \quad \langle P, A, acc\rangle \in \mathit{Used}^* \quad (12)$$

Figure 12: Transitive Closures

# 7 Formal Model and Time Annotations

The Open Provenance Model allows for causality graphs to be annotated with time annotations. In this model, time is *not* intended to be used for deriving causality: if causal dependencies exist, they need to be made explicit with the appropriate edges. However, time may have been observed during the course of a process, and we would expect such time information to be compatible with causal dependencies: the time of an effect should be greater than the time of its cause (for a same clock). Hence, time is useful in validating causality claims.

In the Open Provenance Model, time may be associated to *instantaneous occurrences* in a process. We currently recognize four instantaneous occurrences, which have a reasonable shared understanding in real life and computer systems. Two of them pertain to artifacts, whereas the other two relate to processes. For artifacts, we consider the occurrences of *creation* and *use*, whereas for processes, we consider their *starting* and *ending*.

The rationale for choosing instant time for the OPM model is the same as for adopting artifacts as immutable pieces of state. At a specific time, an object we consider will be in a specific state, which we refer to as artifact, and for which we can express the causality path that led to the object being in such a state.

In some scenarios, occurrences of use or creation of objects and occurrences of starting or ending of processes may not be instantenous. To capture such scenarios, detailed processes and artifacts, and their respective causal dependencies, need to be made explicit, in order to be expressible in the OPM model. For instance, the starting of a nuclear power plant is not usefully modelled as an instantatenous occurrence, when one tries to understand failures that occurred during this activity; hence, this whole starting occurrence must be modelled by one process (or possibly several), which in turn have instanenous beginnings and endings.

In the Open Provenance Model, time information is expected to be obtained by *observing* a clock when an occurrence occurs. Given that time is observed, time accuracy is limited by the granularity of the clock and the granularity of the observer's activities. Hence, while the notion of time we consider is instantaneous, the model allows for an interval of accuracy to support granularity of clocks and observers. In the OPM model, an instantaneous occurrence happening at time $t$ is annotated by two observation times $t^m, t^M$, such that the occurrence is known to have occurred *no later* than $t^M$ and no earlier than $t^m$. Hence, $t \in [t^m, t^M]$.

Concretely, for an artifact, we will be able to state that it was used (or generated by) no earlier than time $t_1$ or no later than time $t_2$. For a process, we will be able to state that it was started (or terminated), no earlier than time $t_1$ or no later than time $t_2$.

In Figure 13, we revisit our formal model, examining where time annotations are permitted. We first introduce a new primitive set *Time*, for which a given serialization will specify a format (such as the standard coordinated universal

$$
\begin{array}{rcll}
ProcessId & : & primitive\ set & \text{(Process Identifiers)} \\
ArtifactId & : & primitive\ set & \text{(Artifact Identifiers)} \\
AgentId & : & primitive\ set & \text{(Agent Identifiers)} \\
Role & : & primitive\ set & \text{(Roles)} \\
Account & : & primitive\ set & \text{(Accounts)} \\
Value & : & application\ specific\ set & \text{(Values)} \\
Time & : & primitive\ set & \text{(Time)} \\
Process & = & ProcessId \rightarrow Value \times \mathbb{P}(Account) & \\
Artifact & = & ArtifactId \rightarrow Value \times \mathbb{P}(Account) & \\
Agent & = & AgentId \rightarrow Value \times \mathbb{P}(Account) & \\
OTime & = & Time \times Time & \text{(Observed Time)} \\
Used & = & ProcessId \times Role \times ArtifactId \times \mathbb{P}(Account) \ \times OTime^0 & \\
WasGeneratedBy & = & ArtifactId \times Role \times ProcessId \times \mathbb{P}(Account) \ \times OTime^0 & \\
WasTriggeredBy & = & ProcessId \times ProcessId \times \mathbb{P}(Account) \ \times OTime^0 & \\
WasDerivedFrom & = & ArtifactId \times ArtifactId \times \mathbb{P}(Account) \ \times OTime^0 & \\
WasControlledBy & = & ProcessId \times Role \times AgentId \times \mathbb{P}(Account) \ \times OTime^0 \times OTime^0 & \\
Overlaps & = & Account \times Account & \\
Refines & = & Account \times Account & \\
OPMGraph & = & Artifact \times Process & \\
 & & \times Agent \times \mathbb{P}(Used) & \\
 & & \times \mathbb{P}(WasGeneratedBy) \times \mathbb{P}(WasTriggeredBy) & \\
 & & \times \mathbb{P}(WasDerivedFrom) \times \mathbb{P}(WasControlledBy) & \\
 & & \times \mathbb{P}(Overlaps) \times \mathbb{P}(Refines) &
\end{array}
$$

Figure 13: Causality Graph Data Model and Time Annotations

time, UTC). We then introduce *Observed Time* as a pair of time values (whose set is $OTime$). All time annotations are optional, which we note by $OTime^0$ in the definitions.

Edges involve $OTime$ in their cartesian product. Edges from *WasGeneratedBy* and *Used* can be annotated by an *optional* timestamp, marking the associated artifact was known to be generated or used, at a given time (expressed as an observation interval).

For *WasControlledBy*, we allow two *optional* timestamps marking when the process was known to be started or terminated, respectively.

For *WasDerivedFrom*, we also allow one *optional* timestamp. Given Figure 9 and associated inferences, for a given edge $\langle a_1, a_2, acc\rangle \in WasDerivedFrom$, there is an implicit process that generated $a_1$ and that consumed $a_2$. The time annotation indicates when the artifact was generated.

Likewise, for *WasTriggeredBy*, we also allow one *optional* timestamp. Given Figure 9 and associated inferences, for a given edge $\langle p_1, p_2, acc\rangle \in WasTriggeredBy$, there is an implicit artifact that was used by $p_1$ and generated by $p_2$. The time annotations indicates the time when the artifact was used by $p_1$.

# 8 Time Constraints and Inferences

The model of causality in OPM is essential timeless since time precedence does not imply causality: if a process $P_1$ occurs before a process $P_2$, in general, we cannot infer that $P_1$ caused $P_2$ to happen. However, the converse implication holds assuming time is measured according to a single clock.

We therefore expect time annotations to be consistent with causality. To this end, we extend the definition of legal account view, defined as: an acyclic account view, which contains at most one *wasGeneratedBy* edge per artifact, and in which *causation is time-monotonic*, as displayed in Figure 15, and discussed below.

We remind the reader that all observed times are pairs of instanteous time values. For $T_1 = (t_1^m, t_1^M)$, with $t_1^m \leq t_1^M$, and $T_2 = (t_2^m, t_2^M)$, with $t_2^m \leq t_2^M$ inequality is defined as follows:

$$T_1 < T_2 \quad if \quad t_1^m \leq t_1^M < t_2^m \leq t_2^M$$
$$T_1 \leq T_2 \quad if \quad t_1^m \leq t_1^M \leq t_2^m \leq t_2^M$$

According to Figure 14, an artifact must exist before it is being used ($T_1 < T_3$ and $T_4 < T_6$). If an artifact is used by a process, it will actually be used after the start of the process ($T_2 < T_3$). A process generates artifacts before its end ($T_4 < T_5$), and a process starts precedes its generation of artifacts ($T_2 < T_4$) and its end ($T_2 < T_5$).

Equipped with these definitions, Figure 15 formally states the time constraints illustrated by Figure 14.

Equation (13) states that generation of an artifact precedes its use. Equation (14) requires a process to start before it uses artifacts, but after the artifact that caused it was generated; the use of the artifact taking place before the end of the process.

Equation (15) states that generation of an artifact by a process is preceded by the start of the process and takes place before the end of the process.

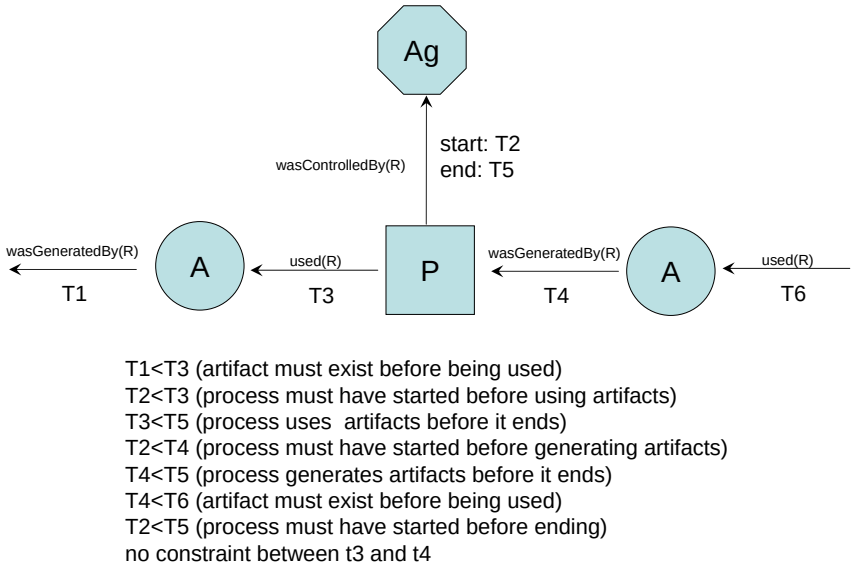


Figure 14: Time Constraints in the Open Provenance Model

$$\frac{used(p_1, r_1, a, acc_1, T_3) \ \wedge \ wasGeneratedBy(a, r_2, p_2, acc_1, T_1)}{T_1 < T_3} \tag{13}$$

$$\frac{used(p, r_1, a, acc_1, T_3) \ \wedge \ wasControlledBy(p, r_3, ag, acc_1, T_2, T_5)}{T_2 < T_3, T_3 < T_5} \tag{14}$$

$$\frac{wasGeneratedBy(a, r_2, p, acc_1, T_4) \ \wedge wasControlledBy(p, r_3, ag, acc_1, T_2, T_5)}{T_2 < T_4, T_4 < T_5} \tag{15}$$

Figure 15: Causation is Time-Monotonic

# 9 Support for Collections

Collections represent groups of objects. Computer programs in general, and workflows in particular, usually offer primitives to manipulate such collections. It is therefore important that OPM offers the means to represent collections and their provenance. Specifically, it is crucial to be able to distinguish the provenance of collections from the provenance of the items contained in them.

Collections are represented by artifacts, and an OPM graph can express that a collection was used or was generated by a process. (Likewise, a summary edge can also express that a collection was derived from another.)

At any point in a computation, a collection consists of a group of member artifacts, which can be enumerated by means of a *collection accessor*, and individually used by processes. Symmetrically, a group of artifacts generated by processes can be grouped into a collection by means of a *collection constructor*.

Collection types are defined by means of collection accessors and constructors; such operations are expressed by OPM processes, and the algebraic properties of these operations define the properties of collections: e.g. ordered or unordered collections, bags or sets, indexable collections or not.

> *Over time, in order to promote inter-operability, OPM needs to define accessors and constructors for common collections.*

Figure 16 illustrates an example of collection, whose provenance consists of two overlapping views (refinements). In the high level view, the collection `[b1, b2, b3, ...]` is described as resulting from mapping a function `f` over a collection `[a1, a2, a3, ...]`.

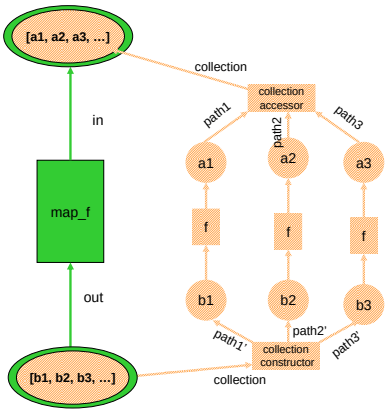


Figure 16: Provenance of a Collection

The individual members of collection `[b1, b2, b3, ...]` were generated by application of process `f` to the members of collection `[a1, a2, a3, ...]`. The

convention is that the role associated with each individual of a collection is the path that allows us to access that individual artifact in the collection. It could be a simple index (0, 1, ...) when the collection is an ordered list, or it can be an XPath expression when the collection is an XML document. The 'collection' role is used to mark the used edge in the accessor and the generated edge in the constructor. Algebraic definitions of constructors and accessors must also define the roles that are permitted.

# 10 Example of Representation

In this Section, we construct an explicit representation of the model for Figure 4. It appears in Figure 17, where we see:

- an explicit enumeration of artifact and process ids (no agent in this graph);
- the symbols O and G to denote orange and green accounts, respectively;
- explicit mappings for processes and artifacts from their ids to their values and accounts;
- in this representation, the "values" of processes is a URI to concepts in an ontology, whereas the "values" of artifacts are immediate;
- the list of edges;
- explicit declaration of refinement.

# 11 Conclusion

The document has introduced the open provenance model, consisting of a technology-independent specification and a graphical notation, to express causality graphs representing *past* executions. In the future, we will define a serialization format for this model. We will also specify protocols by which provenance of artifacts can be determined, and protocols for applications to record descriptions of their execution. We invite teams that have defined their own provenance model to establish whether their representations can be converted into this model and vice-versa.

$$
\begin{array}{rcl}
ProcessID & = & \{p_1, p_2, p_3, p_4, p_5\} \\
ArtifactID & = & \{a_1, a_2, a_3, a_4, a_5, a_6\} \\
Account & = & \{\mathsf{G}, \mathsf{O}\} \\
P \subseteq Process & = & \\
\{\ p_1 & \rightarrow & \langle \texttt{http://process.org/add1ToAll}, \{\mathsf{G}\}\rangle, \\
p_2 & \rightarrow & \langle \texttt{http://process.org/split}, \{\mathsf{O}\}\rangle, \\
p_3 & \rightarrow & \langle \texttt{http://process.org/plus1}, \{\mathsf{O}\}\rangle, \\
p_4 & \rightarrow & \langle \texttt{http://process.org/plus1}, \{\mathsf{O}\}\rangle, \\
p_5 & \rightarrow & \langle \texttt{http://process.org/cons}, \{\mathsf{O}\}\rangle\ \} \\
A \subseteq Artifact & = & \\
\{\ a_1 & \rightarrow & \langle (\mathsf{2}, \mathsf{6}), \{\mathsf{G}, \mathsf{O}\}\rangle, \\
a_2 & \rightarrow & \langle (\mathsf{3}, \mathsf{7}), \{\mathsf{G}, \mathsf{O}\}\rangle, \\
a_3 & \rightarrow & \langle \mathsf{2}, \{\mathsf{O}\}\rangle, \\
a_4 & \rightarrow & \langle \mathsf{6}, \{\mathsf{O}\}\rangle, \\
a_5 & \rightarrow & \langle \mathsf{3}, \{\mathsf{O}\}\rangle, \\
a_6 & \rightarrow & \langle \mathsf{7}, \{\mathsf{O}\}\rangle\ \} \\
u \subseteq Used & = & \\
& \{ & used(p_1, in, a_1, \{\mathsf{G}\}), \\
& & used(p_2, pair, a_1, \{\mathsf{O}\}), \\
& & used(p_3, in, a_3, \{\mathsf{O}\}), \\
& & used(p_4, in, a_4, \{\mathsf{O}\}), \\
& & used(p_5, left, a_5, \{\mathsf{O}\}), \\
& & used(p_5, right, a_6, \{\mathsf{O}\})\ \} \\
g \subseteq WasGeneratedBy & = & \\
& \{ & wasGeneratedBy(a_2, out, p_1, \{\mathsf{G}\}) \\
& & wasGeneratedBy(a_3, left, p_2, \{\mathsf{O}\}), \\
& & wasGeneratedBy(a_4, right, p_2, \{\mathsf{O}\}), \\
& & wasGeneratedBy(a_5, out, p_3, \{\mathsf{O}\}), \\
& & wasGeneratedBy(a_6, out, p_4, \{\mathsf{O}\}), \\
& & wasGeneratedBy(a_2, pair, p_5, \{\mathsf{O}\})\ \} \\
r \subseteq Refines & = & \\
& \{ & refines(\mathsf{G}, \mathsf{O})\ \}
\end{array}
$$

Figure 17: Representation of Figure 4

# A Best Practice on the Use of Agents

With the defined notion of account, we now revisit the sky mosaic example. Instead of Figure 3, a different description could encompass the steps the operating system (or the grid) goes through in order to execute a program (as in the PASS and ES3 approaches). Figure 18 illustrates *some* possible causal dependencies for a system-level description. Here, we see an explicit reference to the workflow script used by the enactor.

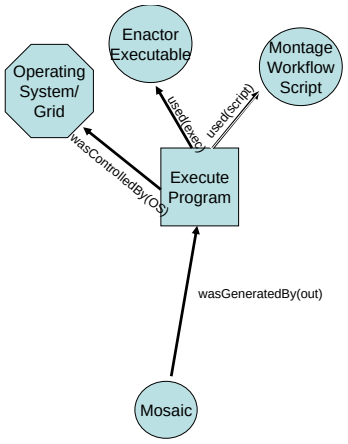


Figure 18: Overlapping Montage Provenance

Naturally, both descriptions can coexist in a same provenance graph, using the concept of overlapping descriptions, as depicted by Figure 19. While such a description is perfectly acceptable, it fails to tell us that the agent Pegasus/Condor Dagman is this executable, which itself was activated under the control of the operating system (or Grid).

In other circumstances, it is necessary to explain that multiple agents were all controlling a same process, but from different perspective. For the case present, the researcher who controlled the experiment, the enactment engine, and the funding institution are all potential causes of the experiment. We then obtain Figure 20, where we see three processes triggering the production of a mosaic. Further experience will the model will allow us to identify guidelines to promote inter-operability of systems.

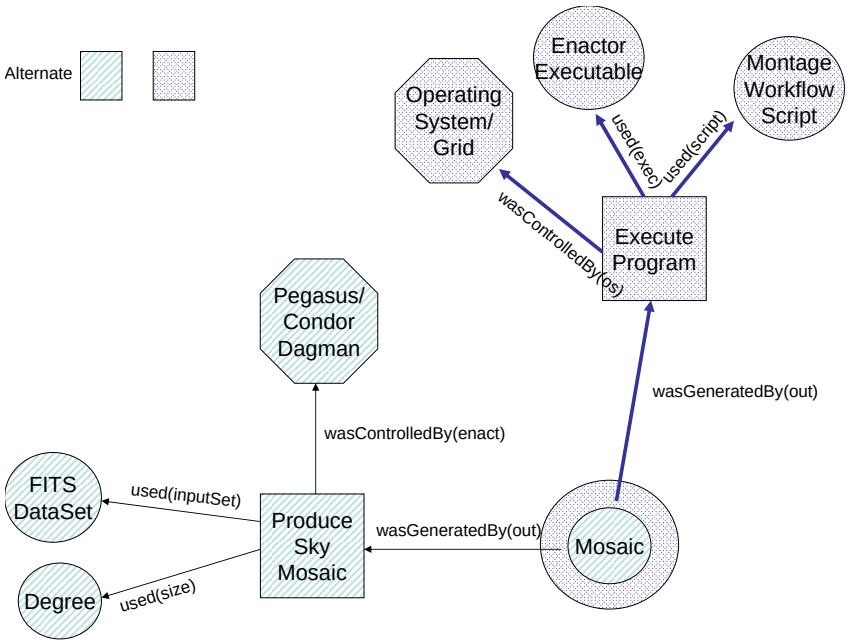


Figure 19: Montage Provenance

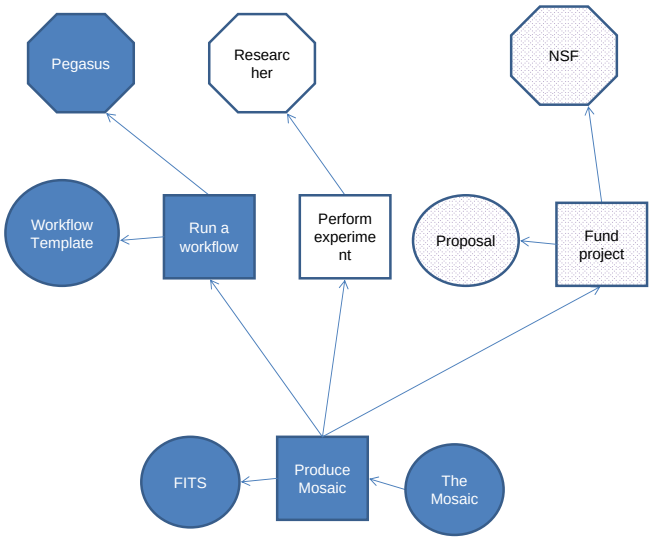


Figure 20: Multiple Agents Controlling a Process

# References


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

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

[3] Open provenance model workshop: Towards provenance challenge 3. http://twiki.ipaw.info/bin/view/Challenge/OpenProvenanceModelWorkshop, June 2008.

[4] Paul Groth. First opm workshop minutes. http://twiki.ipaw.info/bin/view/challenge/firstopmworkshopminutes, Information Science Institute, USC, July 2008.

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

[6] G. Huet, Gilles Kahn, and Christine Paulin-Mohring. The Coq Proof Assistant — Tutorial. Technical report, INRIA, 1999. Available from `coq.inria.fr`.

[7] Simon Miles. Technical summary of the second provenance challenge workshop. http://twiki.ipaw.info/bin/view/challenge/secondworkshopminutes, King's College, July 2007.

[8] 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.

[9] 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.

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

[11] David A. Schmidt. *Denotational Semantics. A Methodology for Language Development*. Brown Publishers, 1986.

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

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