The Open Provenance Model (v1.1)

Luc Moreau (Editor) (U. of Southampton)

Current contributors to date:
Simon Miles (King’s College)
Paolo Missier (Manchester U.)
Yogesh Simmhan (Microsoft)
Joe Futrelle, Jim Myers, Eric Stephan (NCSA)
Natalia Kwasnikowska, Jan Van den Bussche (U. Hasselt)
Juliana Freire (U. Utah)
Paul Groth (USC)
Ben Clifford

November 13, 2009

Abstract

The Open Provenance Model is 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. This document contains the specification of the Open Provenance Model (v1.1) resulting from a community-effort to achieve inter-operability in the Third Provenance Challenge.
1 Introduction

Provenance is well understood in the context of art or digital libraries, 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” [14] 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 [7, 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 [13] 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) [8].

The starting point of this work is the community agreement summarized by Miles [6]. 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.

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 version v1.01 of the Open Provenance Model [11], which was actively used during the Third Provenance Challenge, which aimed at exchanging provenance information encoded in OPM and an-
answered precise provenance queries. Some 15 teams participated in this third challenge, and decided to adopt an open-source model for the governance of OPM. A series of proposals were put forward, publically reviewed, and put to vote. The result of which is version 1.1 of the Open Provenance Model, which we present in this paper.

2 Requirements

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.

3 Basics

The Open Provenance Model allows us to characterize what caused “things” to be, i.e., how “things” depended on others and resulted in specific states. In essence, it consists of a directed graph expressing such dependencies. We introduce here the constituents of such a graph.
3.1 Nodes

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\textsuperscript{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 kinds of nodes, which we define now.

\begin{definition}{Artifacts} Immutable piece of state, which may have a physical embodiment in a physical object, or a digital representation in a computer system. \end{definition}

\begin{definition}{Process} Action or series of actions performed on or caused by artifacts, and resulting in new artifacts. \end{definition}

\begin{definition}{Agent} Contextual entity acting as a catalyst of a process, enabling, facilitating, controlling, affecting its execution. \end{definition}

The Open Provenance Model is a model of artifacts in the past, explaining how they were derived. Likewise, processes also occurred in the past, i.e. they 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.

To facilitate understanding and promote a shared visual representation, we introduce a graphical notation for provenance graphs. Specifically, artifacts are represented by circles. Likewise, processes are represented graphically by rectangles. Finally, agents are represented by octagons.

\textsuperscript{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 accommodate streams fully as they are recognized to be crucial in many applications.
3.2 Dependencies

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

![Diagram of edges in the Open Provenance Model](image)

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 rest. 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 \textit{was controlled by} edge. Given that a process may have been controlled by several agents, we also identify their roles as controllers. 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 express easily 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 \textit{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 \textit{triggered by} \(P_1\). Edges \textit{was derived from} and \textit{was triggered by} are introduced, because they allow a dataflow and process oriented views of past executions to be adopted, respectively, according to the preference of system designers. (Since these edges summarize some activities for which all details are not being exposed, it was felt that it was not necessary to associate a role with them.)

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 \textit{necessary dependencies}, were more appropriate. However, even then, one can distinguish 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 4). 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) A “used” edge from process to an artifact is a causal relationship intended to indicate that the process required the availability of the artifact to be able 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 could have required the artifact to be available for the process to be able to start. (Such an interpretation corresponds to a call-by-value procedure invocation where the arguments are required for the procedure to be invoked.) 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. However, such a stronger notion is not compositionally: an artifact $A$ may have been required to begin execution of $P_1$, but it does not mean that $A$ was required by a to begin $P_2$, a super-process of $P_1$.

Definition 6 (Artifacts Generated by Processes) A “was generated by” edge from an artifact to a process is a causal relationship 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 “was generated by” edges, the process had to have begun, for all of them to be generated.

A stronger interpretation is that the process had to complete for the artifact to be generated. This alternative interpretation was rejected because it made it difficult to model pipelined processes exchanging artifacts.

Definition 7 (Process Triggered by Process) An edge “was triggered by” from a process $P_2$ to a process $P_1$ is a causal dependency that indicates that the start of process $P_1$ was required for $P_2$ to be able to complete.

We note that the relationship $P_2$ was triggered by $P_1$ (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” from artifact $A_2$ to artifact $A_1$ is a causal relationship that indicates that artifact $A_1$ needs to have been generated for $A_2$ to be generated.

Definition 9 (Process Controlled by Agent) An edge “was controlled by” from a process $P$ to an agent $Ag$ is a causal dependency that indicates that the start and end of process $P$ was controlled by agent $Ag$.

3.3 Roles

Roles are constituents of “used”, “was generated by”, and “was controlled by” edges, aimed at distinguishing the nature of the dependency when multiple such edges are connected to a same process.

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 (resp, generate) more than one artifact. Each “used” (resp, “was generated by”) relation may be distinguished by a role with respect to that process. For example, a process may use several files, reading parameters from one (role = “parameters”), and reading data from another (role = “data”).

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 their associated roles. 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 “was controlled by” 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 controlling function, which would be distinguished by roles associated with the “was controlled by” 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”).
From an OPM’s perspective, roles have a syntactic nature and are scoped by the process, by the process to which they are related to. A role has meaning only within the context of a given process (and/or agent). For a given process, each “used”, “was generated by” or “was controlled by” 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.

It is recommended that roles be specified, but they 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 (by means of profiles, defined in Section 9). In addition, a reserved value will be defined for “undefined”, which should be used when the role is not known or omitted.

3.4 Examples

TODO: [[make data derivation explicit]]

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.

![Figure 2: Victoria Sponge Cake Provenance](image)

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 3: Montage 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 7.

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.
4 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](image)

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 lighter (red) part belonged to the left graph of Figure 4, whereas the darker (black) 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 darker part (black) provides more details than the lighter subgraph (red): the darker part is said to be a *refinement* of the lighter graph.

![Diagram](image)

**Figure 5: Overlapping and Hierarchical Accounts in a Provenance Graph**

Observing Figure 5, it becomes crucial to contrast the edges “was generated by” originating from artifact (3,7) with the edges “used” originating from the constructor process. Indeed, the edges “used” out of the constructor process mean that both artifacts 3 and 7 were required for the process to take place. On the contrary, since the edges “was generated by” from 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 edges “used” corresponds to an AND-node, whereas an artifact with edges “was generated by” 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 were performed.

By combining several accounts, we can obtain cycles, as illustrated by Figure 7 (left). Here, in the first view (darker, black account), a description of two processes p1a and p1b is presented, and their dependencies on artifacts a0, a1, a2 and a3. In the second view (lighter, red account), it is stated that the two processes p1a and p1b are in fact a single process operating on inputs a0 and a2,
Figure 6: Hierarchy of Accounts in a Provenance Graph

and producing a1 and a3. If we combine the two views, a cycle of “used” and “was generated by” edges has been created: a2 → p2 → a1 → p1 → a2. In the right-hand side of Figure 7, we make data derivations explicit: in this example, we observe that no cycle of “was derived from” is created, since the two accounts are compatible (since one provides more details than the other). In the most general case, where accounts may be conflicting, we can anticipate cycle of “was derived from” edges to be resulting from the union of several accounts.

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).

5 Observation Time and Time Constraints

The Open Provenance Model allows for causality graphs to be decorated with time information. 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
Figure 7: Multiple Accounts Creating Cycle
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 instantaneous. 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 instantaneous 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 instantaneous 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 specified in term of 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 8, we revisit OPM entities indicating how time information may be expressed in the model. We note again that time information is optional in OPM and is expressed as an observation time interval.

Edges “used” and “was generated by” can be extended with an *optional* timestamp, indicating that the associated artifact was known to be generated or used, at a given time.

For a “was controlled by” edge, we allow two *optional* timestamps marking when the process was known to be started or terminated, respectively. For a process that is not source of a “was controlled by” edge, we allow the process to
be decorated by two timestamps directly.

For a “was derived from” edge, one optional timestamp is permitted, which indicates when the artifact was generated. Likewise, for “was triggered by” edge, we also allow one optional timestamp that marks the time when the communicated artifact was used by the edge source.

![Diagram of Provenance Model](image)

Figure 8: Time in the Provenance Model

The model of causality in OPM is essentially 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.

According to Figure 9, 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 ($T_2 < T_5$).

### 6 Provenance Graph Definition

The open provenance model is defined according to the following rules.

1. An OPM entity can be an node, an edge, a role, an account, a graph, or an annotation.

2. Accounts are identified by unique identifier. Two accounts are equal if and only they have the same identifier.
3. **Artifacts** are identified by unique identifiers. Artifacts are entities that represent an application instantaneous piece of state. Two artifacts are equal if and only if they have the same identifier (irrespective of the state they represent\(^2\)). Artifacts can optionally belong to accounts: account membership is declared by listing the accounts an artifact belongs to.

4. **Processes** are identified by unique identifiers. Processes represent applications activities. 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.

5. **Agents** are identified by unique identifiers. 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.

6. **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 optionally belong to accounts: account membership is defined by listing the accounts an edge belongs to. Structural equality applies to edges: two edges of type “used”

---

\(^2\)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.
(resp. “was generated by”, or “was controlled by”) are equal if they have the same source, the same destination, the same role, and the same accounts; two edges of type “was derived from” (resp. “was triggered by”) 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 and agents in processes.

7. Roles are mandatory in edges “used”, “was generated by”, and “was controlled by”. The meaning of a role is defined by the semantics of the process they relate to. Role semantics is beyond the scope of OPM.

8. 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 adjacent to (i.e., it is source and destination of).

9. 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.

10. 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.

11. While cycles can be expressed in the syntax of OPM, an account view is legal if it is free of cycle of “was derived from” edges and if it contains at most one “was generated by” 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.

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

13. 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.
14. *Edges* can optionally be decorated with time information. *Processes* without “was controlled by” edge can also optionally be decorated with time information.

15. A provenance graph is not not required to contain time information.

16. Within an account, time information must be consistent with causality. To this end, the definition of legality of an account view is extended with an extra condition requiring that *causation is time-monotonic*, as displayed in Figure 9.

All observed times are pairs of instanteous time values. For $T_1 = (t^m_1, t^M_1)$, with $t^m_1 \leq t^M_1$, and $T_2 = (t^m_2, t^M_2)$, with $t^m_2 \leq t^M_2$ inequality is defined as follows:

$$
T_1 < T_2 \text{ if } t^m_1 \leq t^M_1 < t^m_2 \leq t^M_2 \\
T_1 \leq T_2 \text{ if } t^m_1 \leq t^M_1 \leq t^m_2 \leq t^M_2
$$

17. Two account views are said to be *overlapping* if the views have some artifact, process or agent in common\(^3\).

18. An account view $v_1$ is a *refinement* of another account view $v_2$ if any inferred dependency in $v_1$ can also be inferred in $v_2$. **TODO: [[[Check this is consistent with formal model.]]]**

19. In an OPM graph, relations between account (overlap, refinement, and any other) may be asserted. Account relation assertions are legal if two account asserted to be in relationship satisfy this relationship’s definition.

We assume the existence of a few primitive sets: identifiers for processes, artifacts and agents, roles, and accounts. These sets of identifiers provide identities 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 identifiers is to define the structure of graphs: they are not meant to define identities that are persistent and reliably resolvable over time.

\(^3\)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, explicit declarations of such overlapping properties can be considered to facilitate the processing and traversal of graphs.
7 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.

7.1 Completion Rules

In Section 3, we have introduced the two causal dependencies “was triggered by” and “was derived from” as summary edges for a process view (where an intermediary artifact was unknown) and a data view (where an intermediary process was unknown), respectively. Figures 10 and 11 describe *completion rules*, i.e. one-step inferences that can be performed in the Open Provenance Model. A rule explains how a subgraph can be converted into another subgraph.

![Figure 10: Inference: Artifact Introduction and Elimination](image)

Figure 10 displays a bidirectional transformation, i.e. an equivalence. According to the forward transformation, a “was triggered by” edge is inferred from the existence of “used” and “was generated by” edges. We note that the inferred “was triggered by” edge belongs to the set of accounts given by the intersection of accounts of the “used” and “was generated by” edges.

\[\text{TODO: } [[\text{Justify better rationale for intersection: By taking the intersection, we ensure that multiple usage of these completion rules do not result in new edges adjacent to an existing asserted artifact in a specific account. A theoretical inves-}\]
Figure 10 shows completion rule is bidirectional: it allows us to establish that the “was triggered by” edge is hiding the existence of some artifact used by $P_2$ and generated by $P_1$. The inferred edges “used” and “was generated by” are asserted in the same account context as the original “was triggered by” edge. The completion rule allows us to establish the existence of some artifact but it does not tell us what their id is. This is the consequence of using “was triggered by”, which is a lossy summary of the composition of “used” and “was generated by”.

![Diagram](image)

Figure 11: Inference: Process Introduction

In Figure 11, there is only one completion rule permitted: a “was derived from” edge hides the presence of an intermediary process. Inferred edges are asserted with accounts as the original edge. The converse rule does not hold however, since, without any internal knowledge of $P$, it is impossible\(^5\) to ascertain there is an actual data dependency between $A_1$ and $A_2$.

**TODO:** Check consistency with formal model: In rules reftrigger:consequence and refderivedFrom:consequence, 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 reftrigger:consequence and refderivedFrom:consequence effectively creates relationships in the union of multiple account views. ]]

### 7.2 Multi Step Inferences

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

\(^5\)It is suggested that a profile could offer an annotation indicating that all outputs of a process are dependent on all its inputs. For processes annotated in this way, the converse inference, i.e. process elimination, would hold.
Hence, for the purpose of expressing queries or expressing inferences about provenance graphs, we introduce four new relationships, which are multi-step versions of existing relationships, namely Used\(^\ast\), WasGeneratedBy\(^\ast\), WasDerivedFrom\(^\ast\) and WasTriggeredBy\(^\ast\). They are illustrated in Figures 12 and 13.

TODO: [[explain multi-step transitions. Say they are obtained by artifact elimination. never by process elimination!]]

\[
\text{GeneratedBy}^\ast : A_2 \rightarrow^\ast P_1, A_2 \rightarrow^\ast P_1, A_2 \rightarrow^\ast P_1
\]
\[
\text{Used}^\ast : P_2 \rightarrow^\ast A_3, P_2 \rightarrow^\ast A_2, P_2 \rightarrow^\ast A_1
\]

Figure 12: Inference: Multi Step Edges (a)

\[
\text{DerivedFrom}^\ast : A_3 \rightarrow^\ast A_2, A_3 \rightarrow^\ast A_1, A_2 \rightarrow^\ast A_1
\]
\[
\text{TriggeredBy}^\ast : P_2 \rightarrow^\ast P_1
\]

Figure 13: Inference: Multi Step Edges

8 Annotations

Practical experience with the third Provenance Challenge has shown the need for “extra information” to be added to OPM entities. Such extra information is typically required for inter-operability purpose, to allow meaningful exchange of provenance information. Examples include subtyping of edges, descriptions of processes, and reference to values of artifacts. To accommodate “extra information”
in an extensible manner, the Open Provenance Model allows for all its entities to be annotated, by means of the annotation framework, which we describe below.

8.1 The OPM Annotation Framework

The OPM annotation framework is defined according to the following rules.

1. An *OPM annotation* is a class of objects distincts from the other OPM entities.

2. An *annotable entity* can be an OPM Graph\(^6\), an OPM node, an OPM edge, an OPM account, an OPM Role, or an OPM annotation.

3. An *annotated entity* is an annotable entity associated with one or more instances of annotations.

4. Every annotated entity must be uniquely identifiable in the context of an OPM graph by means of an identifier.

5. An annotation instance is an object of the class OPM Annotation and consists of the following:
   - a subject: an annotable entity (identified by its identifier) to which the annotation is attached;
   - a non-empty set of property-value pairs:
     - the property includes a namespace to represent its scope,
     - the value must be typed;
   - a list of accounts.

The intended meaning of a property-value pair is that the annotated entity (i.e. the subject) is provided with additional descriptions, each consisting of a property of the subject and the value of this property for the subject, in the context of some accounts.

Multiple property-value pairs are allowed within an annotation instance. It is legal for a same property to occur multiple times with different values.

6. Annotations can be annotated and subtyped.

\(^6\)OPM is intended to be technology agnostic. However, there is an acknowledgement that annotating a graph may present challenges with some technologies such as RDF. The implications are such capability are currently under investigation.
8.2 Common OPM Properties

For inter-operability purpose, OPM defines a set of common properties. We identify each property by a unique URI; we define the expected type of subjects and values associated with such property. Finally, we state the intended meaning of the property.

<table>
<thead>
<tr>
<th>type</th>
<th>subject:</th>
<th>an annotable entity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>property:</td>
<td><a href="http://openprovenance.org/property#type">http://openprovenance.org/property#type</a></td>
</tr>
<tr>
<td></td>
<td>value:</td>
<td>a URI</td>
</tr>
<tr>
<td></td>
<td>meaning:</td>
<td>Denotes the type of an OPM entity. Types are represented by a URI.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pname</th>
<th>subject:</th>
<th>an annotable entity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>property:</td>
<td><a href="http://openprovenance.org/property#pname">http://openprovenance.org/property#pname</a></td>
</tr>
<tr>
<td></td>
<td>value:</td>
<td>a URI</td>
</tr>
<tr>
<td></td>
<td>meaning:</td>
<td>Denotes a persistent name that can be used by OPM graph queriers to compare OPM entities. The scope of this name is intended to be global.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>label</th>
<th>subject:</th>
<th>an annotable entity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>property:</td>
<td><a href="http://openprovenance.org/property#label">http://openprovenance.org/property#label</a></td>
</tr>
<tr>
<td></td>
<td>value:</td>
<td>a String</td>
</tr>
<tr>
<td></td>
<td>meaning:</td>
<td>This property provides a human-readable version of an OPM entity.</td>
</tr>
</tbody>
</table>
9 OPM Profiles

OPM is a toplevel representation framework for provenance, and we recognize that some communities will develop their own best practice and usage guideline. To encourage such a notion of best practice or usage guideline, we formalise it by means of the concept of an OPM profile. For instance, a set of conventions is currently emerging to represent “collections” in OPM; it is suggested that all these conventions can expressed in a “collection profile”. Whenever an OPM graph adopts these conventions, it can be annotated with this profile so that queriers may exploit this declaration in order to process graph.

An OPM profile is intended to define a specialisation of OPM, and therefore must remain compatible with the semantics of OPM described in this document. Concretely, this means that a profile-compliant OPM graph is an OPM graph, whose semantics is described in this document. This implies that all inferences specified by this document remain valid in a profile-compliant OPM graph. For the avoidance of doubt, any extension of OPM that does not preserve the OPM semantics must not be defined as a profile, and must not be referred to as OPM. Profiles are specified in separate documents that are independent of this core specification.
An OPM profile consists of the following elements:

1. A mandatory unique global identifier for the profile.
   
   Such profile identifier must be used as the value of the profile property in an annotation to the OPM graph that supports such a profile.

2. An optional controlled vocabulary for annotations.
   
   In this context, a controlled vocabulary for annotations is a specification of the properties, its permitted subjects, and its permitted values (such as types or enumerated values). Such controlled vocabulary may be used for some of the following:
   
   (a) Subtyping edges and nodes in OPM graphs by means of the type property;
   
   (b) Defining application-specific properties: for instance, a position property attached to nodes can be exploited by a visualisation tool to render OPM graphs.

3. Optional general guidance to express OPM graphs.
   
   There are typically many different ways in which OPM can be used to describe an execution. For inter-operability purpose, it is therefore good to provide some guidance on how to structure OPM graphs. For instance, it may be useful to identify several types of accounts (e.g., for high-level and low-level descriptions) and to mandate that each account contains edges of specific sub-types.

   Likewise, common software engineering patterns involved in the design and implementation of an application may also be reflected in OPM graphs; for instance, the publish/subscribe pattern of an application can result in a set of OPM conventions to express publisher and consumer processes and the flow of information between them.

4. Optional profile expansion rules.
   
   In some specific circumstances, it may not be necessary to express all edges or nodes related to an execution because they can be derived. Hence, profiles may contain rules, referred to as expansion rules, which convert an profile-compliant OPM graph into another OPM graph. The process of applying profile expansion rules to generate an OPM graph is called profile expansion, and the resulting graph is said to be profile-expanded. We draw the reader’s attention to the terminology adopted here. Profile expansion should be distinguished from the inferences defined in Section 7 (which consisted of completion rule and multi-step inferences).

   Profile expansion constructs a profile-expanded OPM graph by adding new elements (and possibly removing some), satisfying the following constraints:
(a) A profile-compliant graph is an OPM graph;
(b) A profile-expanded graph is an OPM graph,
(c) The semantics of the profile-compliant graph and of the profile-expanded graph are solely defined by this document;
(d) Any multi-step edge inferred between two nodes in a profile-compliant graph can also be inferred in the profile-expanded graph (but not vice-versa)\footnote{In fact, the profile expansion rules generate an OPM graph that is a refinement of the original graph. Any node of the profile-compliant graph is also a node of the profile-expanded graph (but the latter may contain extra nodes). Any multi-step edge that can be inferred in the profile-compliant graph can also be inferred in the profile-expanded graph.}
(e) Profile expansion is node-preserving, it may be edge-lossy, and it may be annotation-lossy provided that condition (4d) holds.

As a result, there is not need of knowing about a profile to be able to analyse a profile-expanded graph.

From a reasoning perspective, an OPM reasoning engine is only required to implement the inference rules described in this document. Profile-compliant OPM graphs can be translated into OPM graphs by the profile expansion process. Alternatively, a reasoning engine may be profile-aware, and may be able to reason on profile-compliant OPM graphs without requiring profile expansion to take place. The inferences that result from both approaches must be the same.

5. Optional serialisation specific syntax.

A profile may introduce syntactic short-cuts for specific serialisations. The serialisation needs to explain how such short-cuts can be translated into core OPM, and vice-versa.

\textbf{TODO: [[Make better distinction with inference, and clarify lossy nature]]}

We can envisage that controlled vocabularies, patterns and inference rules may all be expressed in some declarative language, which could be used to automatically check whether an OPM graph is compliant with a profile, and to perform profile expansion automatically. There is however no off-the-shelf solution that we can reuse for this purpose. Hence, our assumptions is that profiles will be mostly specified in natural language, and that profile-compliance and profile-expansion routines will have to be implemented by hand. We welcome solutions to make these steps as automatic as possible.
10 Discussion

TODO: [[Revisit requirements.]]
 TODO: [[Discuss evaluation as part of PC3.]]
 TODO: [[Discuss agents briefly, [see v.101] need guidance, profile]]
 TODO: [[Mention collections]]

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.
References


