Representing distributed systems using the Open Provenance Model

Paul Groth\textsuperscript{a,}\textsuperscript{*}, Luc Moreau\textsuperscript{b}

\textsuperscript{a} VU University Amsterdam, De Boelelaan 1081-1087 1081 HV Amsterdam, Netherlands
\textsuperscript{b} University of Southampton, Southampton SO17 1BJ, United Kingdom

\section*{ARTICLE INFO}

\textbf{Article history:}
Received 22 December 2009
Received in revised form 4 August 2010
Accepted 12 October 2010
Available online 20 November 2010

\textbf{Keywords:}
Distributed systems
OPM profile
Provenance

\section*{ABSTRACT}

From the World Wide Web to supply chains and scientific simulations, distributed systems are a widely used and important approach to building computational systems. Tracking provenance within these systems is crucial for determining the trustworthiness of data they produce, troubleshooting problems, assigning responsibility for decisions, and improving performance. To facilitate such tracking, the Open Provenance Model (OPM) has been created to enable the interchange of provenance between a distributed system's components. However, to date, the ability of OPM to represent distributed systems has not been verified. In this work, we show how OPM can be used to represent a set of distributed systems' patterns. We present a profile that shows that these patterns are a specialisation of OPM. Finally, we define a contract that enables participants in a distributed system to ensure that their provenance can be integrated cohesively.

\section*{1. Introduction}

The rise of distributed systems both in computational and non-computational environments is a key driving force for provenance systems. For example, products are now assembled from thousands of parts built by hundreds of distributed suppliers in global supply chains, scientific discoveries are based on aggregating datasets produced by research labs throughout the world, and web sites are increasingly created through the use of hierarchies of web services. In each case, there is a need for provenance for a range of activities from quality assurance to performance optimization. Thus, provenance needs to be gathered, collated, and understood across heterogeneous systems that are physically and logically distributed.

To address this need, the Open Provenance Model (OPM)\textsuperscript{[1]} has been developed to facilitate provenance interoperability between systems by providing a common model for provenance. So far, OPM has been primarily used to interchange provenance information between separate monolithic applications. However, today, even when an application uses distributed technologies, such as in workflow systems, provenance is very often produced as if there were an omniscient entity that could view all parts of the application. For instance, workflow enactors such as Kepler\textsuperscript{[2]} or Taverna\textsuperscript{[3]} centrally record provenance, making assumptions about the actual functionality of services they invoke. They assume that a particular service actually performs its functionality as advertised. Such an assumption is not realistic from a distributed systems perspective. It also raises the question of whether OPM is truly adapted to a distributed setting.

Given that distributed systems are such a prominent driver for provenance, in this paper, we investigate the suitability of OPM to represent provenance from a distributed perspective. Specifically, the question can be posed as follows: is OPM expressive enough to describe the provenance of artifacts produced by distributed systems?

To address this question, we rely on changes to the Open Provenance Model\textsuperscript{[1]} stemming from the Third Provenance Challenge. First, using \textit{subtyping} of nodes and edges (by means of annotations), we demonstrate that some distributed systems' communication pattern can be expressed in OPM. Second, using \textit{attribution} (also expressed as an annotation), we identify the different entities that contributed to provenance graph elements. Third, using the \textit{concept of profile}, we demonstrate that such a communication pattern can be defined as a specialisation of OPM, and hence, is compatible with its semantics.

Finally, for documentation of a distributed activity to be coherently captured, participants in a distributed system need to adhere to a set of conventions, derived from the pattern we introduce. Hence, the pattern allows us to define the contract that a participant has to meet for its provenance to be integrated with the rest of an application's provenance.

In summary, the contributions of this paper are as follows:

1. An approach for modeling distributed processes in OPM.
2. An OPM profile for representing distributed systems.
3. A participant’s contract for producing provenance in a distributed system.

The paper is organised as follows. We enumerate the requirements that a distributed system has for provenance (Section 2). We show how these requirements can be met using OPM (Section 3). We further verify our approach by presenting an OPM profile designed for distributed systems (Section 4). This is followed in Section 5 by the definition of the contract that is entailed by the profile for the participants in a distributed system. Our paper evaluates the design against the set of requirements and some compactness metrics. We discuss related work in Section 6.

2. Requirements

The aim of this work is to establish that a model of what has happened in a distributed system is compatible with OPM. This model should be generic enough to encompass a variety of distributed systems whether those systems are synchronous, asynchronous, failure prone or failure free. Thus, we adopt the following definition.

A distributed system is a set of processes that communicate via message-passing, i.e., a process can send or receive messages. No constraints are made about the reliability of communication channels or the order of messages received in those communication channels.

This definition results in a number of requirements on the model.

1. The model must be able to represent messages explicitly.
2. Because provenance is captured in a distributed fashion, the model must be able to represent the dependency between received messages and sent messages.
3. The model must be able to represent lost messages or messages that were manipulated during communication.
4. In order to represent the content of messages, the model must be able to represent their construction.
5. Locality is a central tenet in distributed systems [4], therefore, processes should be modeled independently.
6. Both because processes are ephemeral or the possibility for manipulation, a persistent notion of identity needs to be captured.
7. Because the scope of an application may contain many message exchanges, the model must be able to distinguish and group communications into activities.

These requirements are also reflected in the provenance queries that can be asked in a distributed system. While the Third Provenance Challenge’s workflow [5] is not modeled as a distributed system, it does have cases where components are distributed. A brief overview of the workflow is as follows: a series of CSV files are loaded, the contents of these files are verified by an engine and then the records contained within those files are stored into a database. With a distributed perspective, we can ask the following provenance queries.

1. Were any records bundled together in a message when storing those records into the database (Requirements 1, 2 and 4)?
2. Were any messages sent by the verification engine and not received by the database (Requirement 3)?
3. Does the database have records that were not verified independently by the verification engine (Requirement 5)?
4. Was a single database loaded using multiple loading and verification processes (Requirements 6 and 7)?

The questions apply in a troubleshooting context, where a user wants to understand the order in which operations are performed and which data elements are flowing in the system. Alternatively, they are also important to understand the timing of actions performed by, or in a broader context, of decisions made by the system.

3. Distributed systems in OPM

Here, we define how to represent distributed systems using OPM. This takes the form of patterns that show how to represent critical parts of these systems. We use the terminology as described in OPM 1.1 [1] that is in the same issue.

OPM, rightly, does not define the mechanism by which documentation describing provenance is captured. It only specifies how it is represented. In a previous work [6], the authors have noted that in distributed systems, it is vital to be able to ascertain from this documentation who is responsible for that documentation. Following along another work [7], we introduce the notion of an asserter, which is the entity responsible for a piece of documentation. Additionally, to preserve the principles of locality, documentation of an event should be produced by the entities that have observed it. For example, a client should not document the operation of a server, but should document its exchanging of messages with the server. Thus, when specifying this representation, we distinguish the entities that are asserter for some portion of the provenance graph.

Given the central role of communication in the definition of distributed systems, we first address how to model such communications.

3.1. Representing communication

Fig. 1 shows a representation to model communication between processes. This model makes explicit the construction, sending, receiving, and usage of a message. For each message passed between two processes, we introduce a sender and a receiver process. The sender documents how a message is constructed from some set of artifacts. For example, this could be how an XML web services message was built from the contents of two files. Correspondingly, the receiver documents how a message is
received and separate artifacts are extracted from it. Thus, the view on messages is that they are containers for several data items. We introduce WasDerivedFrom edges (shown as dotted lines in Fig. 1) between the message that was generated by the sender process and the artifacts that it consumed, which explicitly state that the message contains those artifacts. These novel edges are defined to belong to WasConstructedFrom, a subclass of WasDerivedFrom. We follow the same approach for the receiver process, but symmetrically adopting the subclass WasExtractedFrom.

In order to model the possible presence of manipulation, loss of message, or other communication irregularities, we introduce separate message artifacts for the sender and receiver. This pair of artifacts is then joined by an edge WasSameMessageAs, a subclass of the WasDerivedFrom that denotes that they represent two separate instances of the message at different moments of its lifetime (when sent and received). This edge could be asserted for example because they share the same message id or have the same message content. Who asserts this edge is discussed in the next section (Section 3.2). Interestingly, by splitting up the sending and receiving of message, documentation can be created separately and then be connected at a later date, which is a benefit in systems where documentation may be produced asynchronously.

With the above described edges, from the semantics of OPM, one can infer that there is a multi-step edge WasDerivedFrom" between the artifacts a1r and a1s. However, there is also an edge of the same type between a1r and a2s, simply because the presence of both a1s and a2s was required for the message to be composed, sent, and a1r to be extracted from it. This inference however ignores the properties of message constructors and accessor, and unfortunately does not allow us to infer that a1r is in fact the copy of a1s extracted from the message. Therefore, such a relationship needs to be asserted explicitly by means of a WasDerivedFrom edge between these two artifacts, subtyped as WasCopyOf.

Using the above pattern, one can describe message-passing between processes using OPM. We now discuss how to represent attribution using OPM.

3.2. Representing attribution

OPM provides the concept of account to allow various descriptions of the same execution to coexist in a provenance graph. We make use of this mechanism for expressing attribution. In particular, for each message communication, we introduce three different accounts corresponding to the sender, receiver and a shared view of communication. In Fig. 1, the sender's view of communication is in blue and the receiver's view is in red. The shared view is shown in black.

We view the sender and receiver accounts as identifying documentation that was asserted by processes belonging to the application. Therefore, we introduce an annotation, attributedTo that labels the account with the identity of the entity responsible for that description of execution. The value of this property is some form of identity, for example, a URL pointing to the particular entity or more strongly a cryptographic certificate.

The shared account differs somewhat from the sender and receiver accounts, since it can only be created once all the documentation regarding an interaction between two processes has been gathered. For example, the notion that a1r was a copy of a1s can only be ascertained by comparing the artifacts through comparison of their content or position within a message. Indeed, this shared view is typically constructed by parties other than those communicating, e.g, a provenance store or a monitoring application. Such an entity is referred to as an artifact/message mapper, or mapper for short. Thus, it is crucial to denote that this is another account of the communication, because the information it contains is in fact inferred from the information asserted by the two application processes. This account is also given an attributedTo annotation, with value mapper, denoting the entity that inferred this information.

3.3. Representing global activities

An important notion in distributed systems is the ability to view a group of interactions between entities as part of a larger activity. For example, in a database context, a series of database commits can viewed as a larger transaction; likewise, for Web Services, separate messages are paired together to form a request–response pattern [8].

In many systems, such global activities can be delimited by the propagation of a contextual object by the processes involved. In OPM, we can expose these activities through accounts. Therefore, we introduce another annotation, tracer, that labels accounts as representing such system wide activities. The value of the annotation is the set of contextual object identifiers that are propagated by the involved processes. For example, in a remote procedure call the client and server would embed a session id within the messages exchanged. This session id is an example of the contextual object that would be in the tracer annotation. We illustrate such a request–response in Fig. 2 with a green dashed circle around the set of documentations describing this activity.

3.4. Revisiting requirements

We now discuss how the patterns described above fulfill the requirements for modeling a distributed systems execution. Because of the central role of communications in distributed systems, Requirements 1–4 enumerate the need to represent accurately how messages are sent and received. We satisfy these requirements by introducing a pattern that explicitly describes the communication process in terms of sending and receiving...
of messages constructed from artifacts. By modeling the sending and receiving of messages separately, we allow for possible manipulation, loss, or errors in transit to be represented. For example, if a message is sent by a process but never received, the model only needs to describe the sending side of the pattern. This separation of sending and receiving of messages allows for both synchronous and asynchronous communication to be represented.

The separation of sender and receiver also ensures that the documentation of execution from different processes stays independent (Requirement 5). Furthermore, through the use accounts, attribution of documentation can be explicitly represented (Requirement 6). Finally, the accounts mechanism allows us to identify global activities across documentation provided by many different processes (Requirement 7). This is important for being able to query and analyze the provenance of results produced by a distributed system.

We note that, in order to meet these requirements, it was necessary to introduce new constructs outside of what is currently present in OPM. In particular, the tracer and attributedTo annotations were introduced. These constructs go beyond the causal definition of most of the current constructs within OPM. However, these constructs are critical for making sense of a distributed system provenance in order to determine responsibility and demarcate provenance.

Looking back at the queries from the challenge that motivated this work, we can see how the proposed solutions can address these queries.

1. Were any records bundled together in a message when storing those records into the database?
   Using the communication pattern, we can represent how records were aggregated by the sender process into a single message going to the database.

2. Were any messages sent by the verification engine and not received by the database?
   Using the communication pattern, a system can check whether any sent message artifact generated by the verification engine does not have a WasSameMessage edge connecting it to a received message artifact used by the database.

3. Does the database have records that were not verified independently by the verification engine?
   Using attribution, we can check who is responsible for all the records in the database.

4. Was a single database loaded using multiple loading and verification processes?
   Using tracers, multiple loading and verification processes can be demarcated.

3.5. Summary

In summary, message communications are modeled by means of explicit message artifacts, a sender and a receiver view for all artifacts, explicit dependencies between messages and artifacts (in the form of constructor and accessor relations), and explicit dependencies between sent and received artifacts. All accounts (and therefore artifacts and edges they contain) are attributed to the sender or receiver processes, or to the third party mapper. Finally, tracers delimiting the scope of activities can be expressed as accounts.

4. A distributed system profile for OPM

While a distributed system’s execution can be represented using the fundamental OPM constructs, the resulting representation is not compact. For example, Fig. 2 requires eight nodes to represent the exchange of just two messages. OPM does not pre-define specific subclasses of edges and annotations that are useful for understanding a distributed system’s execution. OPM 1.1 provides a notion of profiles which allow for the expression of conventions for the use of OPM. Here, we describe a profile, referred to as the D-profile, for the representation of distributed systems in OPM that is both rich and compact.

An OPM profile consists of four elements: a profile identifier, an optional controlled vocabulary, optional guidance on how to express OPM graphs, and optional profile expansion rules [1]. One can see Section 3 as describing the controlled vocabulary and general guidance of our profile. In this section, we focus on how the compact representation relates to OPM (i.e., profile expansion rules). In particular, to be OPM 1.1 compatible, a profile needs to define a set of rules that specify how a profile-compliant OPM graph can be expanded into an OPM graph that can be reasoned over by reasoners who are aware of the profile. Below, we define such expansion rules for the D-profile. The description in this section borrows some of the notation from [9], which we introduce when needed.

4.1. D-profile representation

Our D-profile representation is based on the notion of embedding much of the information described in Section 3 within annotations. Precisely, we consider an OPM graph, $gr$, where each artifact $a \in Artifact^{gr}$ has the following annotations:

- A message id (mid) identifying the message that artifact $a$ was a part of.\(^1\)
- A sender payload (pls), with a typed value expressed as an application specific serialization as in the OPM 1.1 value annotation.
- A receiver payload (plr), similarly defined for the receiver.
- A set of tracers (tr).

We term an artifact with these annotations, a D-Artifact. The D-Artifact encapsulates the exchange of a message between two processes. In addition to these annotations on artifacts, we introduce the following annotations for processes:

- An attributedTo (at) annotation identifying the source of this documentation.

Obtaining the value of an annotation of an OPM annotatable entity $x$ is denoted as $x_{\text{prop}}$, where prop is a property name. The D-profile relies on the following novel properties: at, mid, pls, plr, tr, pa, and OPM properties value, type. For example, $q_{\text{mid}}$ denotes the value of the mid annotation for artifact $a$.

Using these annotations, we define nine profile expansion rules, namely, two communication expansion rules, two message encapsulation rules, a WasDerivedFrom maintenance rule, two mapper rules, a tracer rule, and an attribution rule. Each rule consists of a set of premises and a set of consequents separated by a horizontal bar. A rule is triggered when its premises are all satisfied, and results in consequents being asserted. Rules may consider both the OPM graph, $gr$, and the new expanded graph $gr^*$. The expansion rules also create two mappings:

$\text{sent}^{gr^*} : Artifact^{gr} \times Process^{gr} \rightarrow Artifact^{gr^*}$

$\text{received}^{gr^*} : Artifact^{gr} \times Process^{gr} \rightarrow Artifact^{gr^*}$

Given an artifact $a \in Artifact^{gr}$, the first mapping tells us which artifact in the expanded graph $gr^*$ is the sent version of $a$ for a process $p \in Process^{gr}$ that generated $a$. The second tells us which

\(^1\) For simplicity, we assume the use of a message id, however, this could be replaced by a function in the expansion process that determined the identity between messages.
artifact is the received artifact, for a process \( p \in \text{Process}^{gr} \) that used \( a \).

We also use the following notation. \( \text{accountOf}^{gr}(x) \) is used to denote the set of accounts for an entity \( x \) in the OPM graph \( gr \). We use \( \text{effectiveAccountOf}^{gr}(x) \) from [9], which denotes the accounts of an entity but for a node also includes the accounts of any edge where it is either a cause or an effect. Subscript \( r \) is used to denote entities associated with the reception of a message. Subscript \( s \) is used to denote entities associated with sending a message. If a subscript \( m \) follows an artifact then it denotes that it represents a message.

### 4.1.1. Communication expansion rules

These rules introduce novel artifacts for the sender side and the receiver side, and suitable dependencies between them. The pre-condition of the rules indicates that there were no previous expansions pertaining to artifacts with a given message id (property:mid) in a given account. The first rule depicted in Fig. 3 expands \( \text{WasGeneratedBy} \) edges between a process and a D-Artifact. The second expands \( \text{Used} \) edges [Fig. 4]. For clarity, we do not depict accounts in any figure that depicts a rule.

\[
\begin{align*}
\exists a_m, p_s, a_m & : \text{new in } gr^* \\
\langle a_s, p, a_m \rangle & \in \text{WasGeneratedBy}^{gr} \\
\langle a, p \rangle & \in \text{WasGeneratedBy}^{gr} \\
\langle a_m, a_r \rangle & \in \text{WasGeneratedBy}^{gr} \\
a_r[\text{value}] & = a[\text{mid}] \\
p_r[a_r] & \in \text{Used}^{gr} \\
\langle a_m, a_r \rangle & \in \text{WasExtractedFrom}^{gr} \\
\langle a_m, a_r \rangle & \in \text{WasExtractedFrom}^{gr} \\
a_m[\text{type}] & = \text{msg} \\
a_m[\text{mid}] & = a[\text{mid}] \\
\end{align*}
\]

with \( \text{accountOf}^{gr}(x) := \text{accountOf}^{gr}(a) \cup \text{accountOf}^{gr}(\langle a, p \rangle) \) for any \( x = \{a_s, p_s, a_m, \langle a, p \rangle, \langle p_r, a_r \rangle, \langle a_m, a_r \rangle\} \).

### 4.1.2. Message encapsulation rules

These rules ensure that messages are not duplicated in the expanded graph. That is, if an artifact message identifier (property:mid) already exists in the expanded graph for a particular account, the actual message artifacts need to be reused. The pre-condition of the rule indicates that there was a previous expansion pertaining to mid. Note, \( A_m, P_2 \) are used in the figures to illustrate already expanded processes and artifacts.

The following rule addresses message encapsulation for \( \text{WasGeneratedBy} \) edges and is depicted in Fig. 5.

\[
\begin{align*}
\exists a_m, p_s, a_m & : \text{new in } gr^* \\
\langle a_s, p, a_m \rangle & \in \text{WasGeneratedBy}^{gr} \\
\langle a, p \rangle & \in \text{WasGeneratedBy}^{gr} \\
\langle a_m, a_r \rangle & \in \text{WasGeneratedBy}^{gr} \\
a_r[\text{value}] & = a[\text{mid}] \\
p_r[a_r] & \in \text{Used}^{gr} \\
\langle a_m, a_r \rangle & \in \text{WasExtractedFrom}^{gr} \\
\langle a_m, a_r \rangle & \in \text{WasExtractedFrom}^{gr} \\
a_m[\text{type}] & = \text{msg} \\
a_m[\text{mid}] & = a[\text{mid}] \\
\end{align*}
\]

with \( \text{accountOf}^{gr}(x) := \text{accountOf}^{gr}(a) \cup \text{accountOf}^{gr}(\langle a, p \rangle) \) for any \( x = \{a_m, p_s, a_m, \langle p_r, a_r \rangle, \langle a, p \rangle, \langle a_m, a_r \rangle\} \).
This rule can only be applied under the assumption that a single process does not use the same artifact multiple times.

4.1.3. Mapper rules

These rules connect the artifacts generated by the communication and encapsulation rules. The rule below connects sent and received messages together and is depicted in Fig. 7.

\[
\exists a_{ms}, a_{mr} \in gr^*, \quad a_{ms}[mid] = a_{mr}[mid] \\
\text{and } a_{ms}[type] = msg \\
\text{and } a_{mr}[type] = msg \\
\langle a_{ms}, a_{mr} \rangle \in \text{WasSameMessageAs}^g
\]

Additionally, sent and received artifacts stemming from the same D-Artifact are connected together by the WasCopyOf edge using the rule shown in Fig. 8 and defined as follows:

\[
\begin{align*}
\langle a_1, a_2 \rangle & \in \text{WasCopyOf}^g \\
p, p_2 & \in \text{Process}^g \\
a_1, a_2 & \in \text{Artifact}^g \\
\text{received}^g(a_1, p) & = a_2, \quad \langle a_1, a_2 \rangle \notin \text{WasCopyOf}^g
\end{align*}
\]

The WasCopyOf edge can be asserted because the artifacts are extracted directly from the same D-Artifact and are associated within the mapping tables kept during rule application.

4.1.4. WasDerivedFrom maintenance rule

If two D-Artifacts \( a_1, a_2 \) are linked by a WasDerivedFrom relation, i.e. \( \langle a_1, a_2 \rangle \), a similar relation needs to be asserted in the expanded graph. Given that \( a_2 \) is a cause and that \( a_1 \) is an effect, the relation will be asserted between \( \text{received}^g(a_2, p) \) and \( \langle a_1, a_2 \rangle \) as follows (see Fig. 9):

\[
\langle a_1, a_2 \rangle \in \text{WasDerivedFrom}^g \\
p, p_2 \in \text{Process}^g \\
\langle a_1, a_2 \rangle, \langle a_1, p \rangle \in \text{Artifact}^g \\
\text{received}^g(a_2, p) \in \text{WasGeneratedBy}^g
\]

\[
\langle \text{sent}^g(a_1, p), \text{received}^g(a_2, p) \rangle \in \text{WasDerivedFrom}^g
\]

4.1.5. Tracer rule

Intuitively, the rule states that if an artifact in a D-profile graph is associated with a set of tracers, then all entities expanded from that artifact should have an account that refers to those tracers, as well. Formally, the rule is defined as follows:

\[
\begin{align*}
a \in \text{Artifact}^g \land a[tr] & = T \text{ for some set } T \text{ of tracers} \\
\text{account}^g(x) & := \text{account}^g(x) \cup \{\text{acc}(T)\} \\
\text{for any } x = \{a_1, a_2, a_{ms}, a_{mr}, p, a_1, a_2, \langle a_1, a_2 \rangle, \langle p_2, a_1 \rangle, \langle a_{ms}, a_{mr} \rangle, \langle a_{mr}, a_{ms} \rangle, \langle a_1, a_2 \rangle, \langle p_1, a_1 \rangle, \langle a_1, p \rangle \}
\end{align*}
\]

where acc is an account constructor with reference to a set of tracers.
4.1.6. Attribution rule

We assign attribution based on the processes that are documented as generating and producing D-Artifacts. Here, \( \nu \) denotes an account. We note that attribution is a property on each account. The rule goes through each account associated with existing processes in \( gr \) and assigns the attribution of that account to the new processes and artifacts within \( gr^* \). Essentially, it transfers the attribution from the unexpanded graph to the expanded graph. Furthermore, for those edges that are derived from the result of having aggregated documentation, attribution is assigned to the mapper, i.e. the entity performing the expansion rules.

\[
(a, p) \in \text{WasGeneratedBy}^{gr} \land (p_2, a) \in \text{Used}^{gr}
\]

\[
v^{gr}[at] = v^{gr}[at]
\]

\[
\forall v \in \text{accountOf}^{gr} (p) \rightarrow v^{gr^*} \in \text{accountOf}^{gr^*} (x)
\]

\[
\forall x \in \{a, p, a_m, \langle a, p \rangle, \langle a_s, p \rangle, \langle a_m, p \rangle \}
\]

\[
v^{gr^*}[at] = v^{gr}[at]
\]

\[
\forall v \in \text{accountOf}^{gr^*} (p_2) \rightarrow v^{gr^*} \in \text{accountOf}^{gr^*^*} (x)
\]

\[
\forall x \in \{a, p, a_m, \langle a, p \rangle, \langle p_r, a_r \rangle, \langle p_r, a_m \rangle \}
\]

\[
v^{gr^*^*}[at] = \text{mapper}
\]

\[
\forall v^{gr^*^*} \in \text{accountOf}^{gr^*^*} (x) \rightarrow v^{gr^*^*} \in \{a_m, a_m\} \}
\]

4.2. Properties of the expanded graph

There are a number of properties of the expanded graph that are entailed from the above expansion rules.

- The OPM edge “was triggered by” between two processes cannot be used to describe fully attributed external communication, since this edge connects two processes, and therefore breaks the locality principle that an assertion can only refer to information from a single locality.
- In the expanded graph, the cause of a WasDerivedFrom edge is always a received artifact.
- In the expanded graph, the effect of a WasDerivedFrom is always a sent artifact.

We can calculate an upper bound on the resulting number of nodes, \(|n|\) and of edges \(|e|\) in the expanded graph using the following formulas.

\[
|n| = 3|\text{Used}| + 3|\text{WasGeneratedBy}| + |p|\quad (1)
\]

\[
|e| = 4|\text{Used}| + 4|\text{WasGeneratedBy}|
+ |\text{WasDerivedFrom}| + |a|
\]

\[
(2)
\]

where \(|\text{Used}|\) is the number of Used edges in the D-OPM graph, \(|a|\) is the number of D-Artifacts, \(|p|\) is the number of processes in the D-OPM graph, \(|\text{WasGeneratedBy}|\) is the number of WasGeneratedBy edges and \(|\text{WasDerivedFrom}|\) is the number of WasDerivedFrom edges. These equations provide an upper bound on the number of nodes and edges because they assume that every D-Artifact models the exchange of only one data item within a message. It is possible for a D-Artifact to represent the exchange of many data items all bundled into a single message. Under this assumption, the number of nodes and edges is derived from the fact that for every Used and WasGeneratedBy edge there will be a new process to handle communication and new artifacts to represent the data as well as the message being exchanged as well as edges to connect these new nodes within the graph.

4.3. Supporting abstraction

We note that these rules can be reversed, an OPM graph can be compacted into a D-OPM graph. This is an interesting point, because it allows for the support of multiple levels of abstraction. One can ignore the communication level and look at the dependency level by “rolling” up the communication level into annotations. Likewise, in the same graph both levels can exist side by side through the use of differing accounts.

4.4. Implementation and evaluation

A prototype of the above mapping was generated for the Third Provenance Challenge. Using a series of in-house tools, the process documentation for the challenge’s workflow [5] was captured. This captured documentation was then exported as XML serialized OPM graphs. Two graphs were generated: one capturing what is roughly equivalent to the D-profile graph, whereas the other capturing what is roughly equivalent to the expanded OPM graph after application of the expansion rules. The implementation of the D-profile graph differs in that it does not use annotations directly and instead embeds information in the value field. The expanded OPM graph’s differences are detailed below. Fig. 10 shows a comparison between these two graphs. Using these measures, the equations for determining the upper bound of expanded graphs can be verified empirically. Using Eq. (1) the upper bound on number of nodes in the expanded graph is 517. Likewise, the upper bound should be 859 edges as calculated by Eq. (2). The gap between the upper bound and the empirical measures is for a number of reasons. First, in the implementation communication is modeled as one artifact and not two as specified by the D-profile. Secondly, data is bundled into more than one message and not separated into individual messages. Thirdly, the implementation did not produce WasConstructedBy and WasExtractedFrom edges.

5. Contract

In order to use the D-profile for interoperability, participants in a distributed system need to adhere to certain obligations and expectations stemming from the patterns previously defined. These obligations and expectations form a contract that these participants must fulfill in order to claim that they are D-profile compliant. The contract defines the assumptions that the D-profile makes about the execution of participants and how these participants document that execution. The expectations and obligations of this contract on a participant are defined as follows:

- All messages sent by a participant contain a message id.

---

• A participant must maintain an appropriate identity.
• A participant appropriately propagates contextual objects representing system wide activities.
• A participant only documents provenance of artifacts within its locality.
• When creating D-Artifacts with the same message id, the participant ensures that information documented therein was part of the same message.
• All documentation created by a participant should be in an account that is attributed with the identity of that participant.
• For all messages sent or received by a participant, a representation of the value of that message should be stored in the D-profile.

Through compatibility with this contract, participants in a distributed system can ensure that mappers such as provenance stores can correctly collate their provenance with provenance provided by other participants. Furthermore, they can ensure that the provenance they provide using the D-profile can be correctly expanded, thus, ensuring interoperability with those systems that only make use of core OPM.

6. Related work

Given that today’s applications tend to be composed by assembling services and components together, possibly dynamically, capturing provenance of data products produced by such applications is not only critically important, but is also challenging. The work on provenance is extensive, ranging from operating systems to scientific workflow, here, we focus on related work directly related to modeling provenance in distributed systems. For an in-depth review of the literature, we refer the readers to Moreau’s extensive survey [10].

OPM 1.1 introduces temporal constraints, expressed according to the relation “has happened before”, that must be satisfied by OPM graphs. Such constraints can be derived from the D-profile introduced in this paper, and by Lamport’s constraint that a message is sent before it is received [11]. For instance, the OPM temporal constraint that an artifact must have been generated before it is being used is explained as follows: the artifact was generated, was embedded in a message, sent by the process that generated the artifact itself, and received and extracted by the process that consumed the artifact.

Souilah et al. [12] present a formal provenance in distributed systems. Their approach consists of a provenance semantics for a variant of the π-calculus in which exchanged data are enriched with a provenance field that accumulates the send/receive operations it underwent. Given that provenance information is carried along with data, sharing of provenance information is not possible for data products that have a common provenance. On the contrary, OPM, with its directed acyclic graph nature, can express such a sharing.

The D-profile expansion described in this paper makes it explicit that multiple asserters may contribute elements of provenance for a given D-Artifact: the sender contributes information related to the sending of the artifact, likewise the receiver contributes information pertaining to its receiving, whereas the mapper links up sender and receiver’s contributions. The proposed D-expansion makes the respective contributors explicit. Several authors [13–15] have proposed the use of cryptographic techniques not only to ensure the authenticity of provenance claims, but also to ensure that they are non-reputable. Furthermore, Hasan et al. [14] ensure that chains of claims are protected, ensuring that an attacker cannot claim to be part of a chain without being detected. Their model of provenance is restricted to read and write access to files, and should be extended to support the D-expansion described in this paper.

This paper builds on our prior work on recording [6] and representing provenance [7] in distributed systems adopting such notions as sender and receiver views of messages, attribution, and tracers. It extends this work by presenting a compact representation for distributed system provenance and addressing how OPM can be used to model such systems. Additionally, it provides a more flexible representation allowing duplicate messages, multiple accounts and broadcast messages to be represented. Karma [16] is an alternate approach to provenance, which relies on an event model, to avoid being tightly coupled to a particular workflow system, hereby allowing the provenance of multiple systems to be captured.

The D-profile allows for a compact representation of distributed system’s provenance in the OPM context, where provenance is represented by directed acyclic graph. Alternative compact representations exist. For instance, Heinis and Alonso [17] show that workflows with a tree structure produce lineage dependencies that can be efficiently stored and queried using interval encoding. They define a provenance query as the transitive closure over a DAG. By a series of benchmarks, they show that recursive queries require little space but can be slow, whereas storing all paths leads to faster queries but increases storage requirement significantly. By using intervals to represent trees, provenance of a node can be determined by finding all the intervals that enclose the interval of this node. They explore how arbitrary DAGs can be transformed into equivalent DAGs that can be encoded with one-dimensional intervals.

7. Conclusion

Distributed systems are key motivators for making provenance systems interoperable. In this paper, we have shown how the emerging Open Provenance Model can be used to represent distributed systems. We presented a profile for OPM that describes a controlled vocabulary for describing provenance in distributed systems. Additionally, this profile provides for a compact representation of such provenance that, using well defined expansions rules, can be converted to OPM. The profile addresses important properties such as locality, the presence of failures and attribution. Finally, a contract for the those wishing to use the D-profile is defined. This work, for the first time, shows that OPM is indeed suitable for representing distributed systems.

References


