Towards Query Interoperability: PASSing PLUS[*†]

Uri J. Braun, Margo I. Seltzer
Harvard School of Engineering and Applied Sciences

Adriane Chapman, Barbara Blaustein, M. David Allen, Len Seligman
The MITRE Corporation

Abstract

We describe our experiences importing PASS [16] provenance into PLUS [7]. Although both systems import and export provenance that conforms to the Open Provenance Model (OPM) [14], the two systems vary greatly with respect to the granularity of provenance captured, how much semantic knowledge the system contributes, and the completeness of provenance capture. We encountered several problems reconciling provenance between the two systems and use that experience to specify a Common Provenance Framework, that provides a higher degree of interoperability between provenance systems. In each case, the problems stem from the fact that OPM interoperability is a weaker requirement than query interoperability. Our goal in presenting this work is to generate discussion about differing degrees of interoperability and the requirements thereof.

1 Introduction

The Provenance Challenges [18, 19, 20] were a set of community exercises designed to help refine and shape the Open Provenance Model [14]. Each of the Challenges was centered on a simple workflow. The First Provenance Challenge [18] focused on collection and querying of provenance information within individual provenance systems. The Second Provenance Challenge [19] began looking at interoperability. Each participant ran a subsection of the designated workflow and then attempted to run designated queries over the entire workflow, using provenance captured by many different systems. The results of this Challenge informed the creation of the Open Provenance Model, designed to standardize core provenance concepts. The Third Provenance Challenge [20] again focused on interoperability. Each participant ran the entire workflow and published the results representing an OPM provenance graph. Other participants then imported these provenance graphs and executed queries over it.

The Provenance Challenges have been successful in that they have enabled the production of a largely accepted model for representing provenance. Additionally, they have honed that model to address the needs of many diverse systems. Moreover, they have accomplished all of this in an incredibly short time period while involving dozens of groups with different agendas. However, the nature of the experiments within the Provenance Challenges has directed the spotlight in particular areas, omitting other possible issues. This work highlights a few of those areas illustrating the problems we encountered and leading to our recommendation of a higher level standard to increase provenance interoperability.

We investigate the challenges involved in integrating two systems that both export and import OPM graphs. Our goal was to import data from the Harvard Provenance Aware Storage System (PASS) [16] system into MITRE's PLUS system [7]. PLUS provides the ability to visually represent and query provenance, as well as perform basic queries over graphs, such as "return all Word documents within four steps of a 'string search' process". We wanted to go the next step and be able to query information captured in a completely different system using the visualization and query tools of the other system.

Although both systems can represent their provenance information in an OPM representation, and PASS had successfully integrated data during the Second Provenance Challenge, we found that we needed more commonality to perform the query and visualization tasks we wanted, using our own workflows. Our work illuminates the areas that fall outside of the scope of the OPM, but are still critical to achieve semantic integration between provenance systems. In particular, we identify the need

[*]Approved for Public Release; Distribution Unlimited (09-5309)

[†]This work was supported by the National Science Foundation under grants CNS-0614784 and IIS-0849392 (Seltzer and Braun) and by the MITRE Innovation Program (Chapman, Blaustein, Allen, and Seligman).

for systems to make common assumptions about reporting order, object referencing and object identification to facilitate meaningful query across multiple provenance systems. We call this desired goal *query interoperability*. Query interoperability is the ability to import data from another system and extract sufficient semantics to allow queries to resolve entities, understand all relationships among entities, and reason over activities and arguments. Given fundamentally different provenance systems and their diverse semantics, we have found it impossible to do this for workflows that are not previously seen and well understood, as discussed below in Section 4.

We use our experience integrating PASS and PLUS provenance data to begin defining a richer semantic model on top of OPM; our goal is to allow provenance systems to share provenance and data at a sufficient level that we can use a set of common utilities to query the data and make both semantic and structural inferences. OPM lays the foundation by providing a data representation interchange layer. Using it, it is possible to write a parser between systems for simple, well defined workflows and make certain inferences about a provenance graph. Our Common Provenance Framework dictates aspects of the semantics and representation necessary to query provenance across different systems and previously unknown workflows. In particular, we propose a set of concepts, constraints and tools that are essential to true query interoperability between systems.

In this paper, we:

- identify the challenges encountered in importing PASS data into PLUS, and
- recommend a Common Provenance Framework that addresses these challenges, providing an increased degree of interoperability for use with the OPM.

The remainder of this paper is organized as follows. We begin by describing the PASS and PLUS implementations and reviewing the OPM in Section 2, while Section 3 describes the manner in which we attempted to integrate the two systems. Section 4 discusses challenges we encountered in trying to import PASS data into PLUS. In Section 5 we describe our recommendations in the form of the Common Provenance Framework that can be utilized with the OPM model and schema. Finally, we discuss related work and conclude in Sections 6–7.

2 Background

In this section, we review the OPM and then highlight those aspects of the PASS and PLUS systems needed for later discussion.

2.1 Open Provenance Model

The Open Provenance Model (OPM) is a proposed standard for describing provenance data [14]. The standardization process used a series of community challenges [13, 18, 19, 20] to identify provenance concepts common across disparate systems. This integration involved each team constructing tools to import and export data to and from other systems. The goals of the OPM are:

- Facilitate provenance information exchange between systems,
- Allow developers to build and share tools that operate on conforming provenance
- Define a precise, technology-agnostic model of provenance,
- Support a digital representation of provenance for any artifact, digital or otherwise, and
- Define a core set of rules that identify valid inferences over provenance graphs.

The OPM currently consists of a base model and a schema encoding a representation of this model in XML. OPM divides the world into entities and edges. There are three types of entities and five types of edges. Values are additional attributes attached to entities providing additional information about the entities they describe.

Table 1 provides the mapping between the OPM terms for the three different entity types and their counterparts in PASS and PLUS. The classic example of OPM entities is that a flour, water and eggs (Artifacts) are brought together by a chef (Agent) and baked (Process) into a cake (Artifact). The OPM also contains edges, which represent relationships amongst entities; these edges can have an edge type, such as “generated” or “usedBy”. In addition to edge type, an edge may have annotations specifying roles that describe the relationship between the entities it connects. The OPM Schema is a direct reflection of the model described above. The main set of entities are the same as in the OPM model: graph, agent, process, artifact, account, etc. We will discuss the PLUS and PASS entities described in Table 1 in more detail below.

Table 1: Object types as defined in OPM, PLUS and PASS.

OPM	PLUS	PASS
Artifact	Data Object	File
Agent	Activity	File (see note)
Process	Invocation	Process

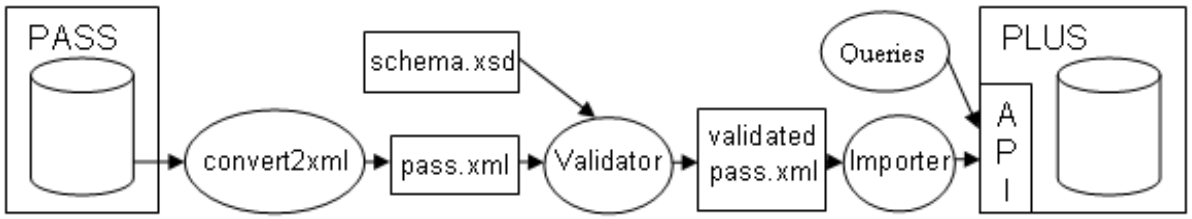

Figure 1: The PASS-PLUS architecture.

2.2 PLUS

MITRE's PLUS system is a Java application with a MySQL database back-end that supports the reporting, storage, and query of provenance data. Applications record lineage information (provenance) by explicitly reporting it using PLUS standard APIs. Alternately, applications can use a set of pluggable PLUS lineage reporting modules, allowing them to log lineage data without application modification. Looking at Table 1, a PLUS artifact can be any object the user wishes to register, for example, strings, files, XML messages, relational data items of arbitrary granularity, etc. Similarly, "activities" can be logged and can take any form, be it a web service, a representation of a human activity like "produce analysis report", or a program run on an operating system. "Invocations" are concrete instantiations of those activities, e.g. "produce analysis report on Dec 22nd". "Invocations" are closely aligned with the OPM concept of "Process". PLUS also explicitly supports the five different edge types of the OPM.

PLUS can operate in two modes; either as a passive recipient of objects that are reported by another system or as an active observer, in the case where an application uses one of the lineage reporting modules provided by PLUS. Because PLUS focuses on logging lineage in distributed systems, situations can arise where capturing a complete lineage graph is not possible. As such, the lineage graphs in PLUS can be sparse, depending on the information that can be feasibly reported or observed given the technical infrastructure.

PLUS assigns each object a unique id, which is generated by computing a time dependent random value. PLUS identifies edges by a combination of the incident objects and the graph in which they participate.

2.3 PASS

Harvard's PASS system is a modified Linux kernel that intercepts system calls, translating them into provenance events as appropriate. Like PLUS, PASS also provides a standard API through which applications can disclose provenance objects and relationships. Referring back to Table 1, PASS represents all OPM artifacts, agents, and processes as objects. Objects can be either active or passive. OPM artifacts are passive objects corresponding to Linux files, construed broadly to include pipes, sockets, etc. Activities are also passive objects corresponding to Linux binaries. OPM Processes are active objects that correspond to Linux processes, which are effectively invocations of an executable file. For example, an invocation that uses the `copy` command to transform `src` to `dst` is an invocation with two inputs: the source file, `src`, and the executable file `copy`. The fact that the role of the `copy` file is executable and thus represents the activity is not represented explicitly. PASS places virtually no restrictions on edge types; that is PASS edges are named, but names can be arbitrarily created.

PASS assigns each object — both artifacts and processes — a *pnode* or provenance node. As the objects represented by these nodes change, PASS increases the version number of the corresponding pnode to differentiate the states of the object at different points in time. Thus, at any given point of time, an object is represented by a $< pnode, version >$ pair. Pnodes inherit all attributes from prior versions, and there is an implied relationship between two consecutive versions.

PASS collects metadata for each pnode it records. For artifacts this includes: the class of object — file, pipe, socket; name and path; times for creation and versioning; and creator. For processes this includes: name, input files, command line arguments, timestamps for creation and versioning, and parent process. Entities inherit the attributes of prior versions of the pnode. Thus the provenance explicitly associated with a particular $< pnode, version >$ pair is a subset of the complete provenance for the pnode. Although an activity might be represented as an executable file, PASS does not recognize the executable file as different from any other input to a process, so there is no easy way to identify activities.

3 System Integration

Our goal in integrating data from PASS and PLUS was to demonstrate that it is possible for two rather different systems embodying different philosophies of recording vs. observation, collection mechanisms and focus to provide provenance query interoperability. This demonstration did not require any kind of tight coupling between the systems, and we intentionally avoided such a coupling. Figure 1 depicts our initial integration of the two systems. Our intended integration model was to simply have PASS export its data in a standardized XML format (using the `convert2xml` utility developed for the Second Provenance Challenge) and have PLUS import standard XML via its `importer`.

However, we discovered that the standard OPM concepts and XML schema were not sufficiently well specified to facilitate query interoperability. Query interoperability is the ability to import data from another system and extract sufficient semantics to allow queries to:

- match on process arguments
- include all documented relationships among entities
- resolve entity names
- identify all activities

Neither OPM nor XML provide sufficient detail with regard to both encoding and identification. Encoding rules do not ensure that: references in the data refer back to existing entities, specify where the entity type is provided, provide an encoding for arguments to an activity, or specify how to identify an activity. Identification is not sufficiently addressed as there is no consideration for several versions of a single object, and naming of entities does not specify how names are resolved especially if the provenance from system A refers to an object on system B. These issues will be discussed further in Section 4.

We used the provenance from several different workflows to test the correctness of the integrated systems. Our smallest workflow, *tsx*, was a test program that simply writes a fixed string to one file, reads that file and writes what it read to a second file. The next smallest workflow, *challenge*, consisted of the provenance we gathered for the First Provenance Challenge. Our largest workflow was `gawkpkg`. It was created by untarring and compiling the `gawk` package. Table 2 contains the breakdown of each of the three workflows in terms of numbers of entities and edges and overall file size. We count entities in two ways, either considering each version as a distinct entity — under column "Versioned" — or treating all the versions collectively as a single entity — under column "Not".

Table 2: Workflows imported from PASS into PLUS

Workflow	Entities		Edges	Size
	Versioned	Not		
tsx	114	25	36	27KB
challenge	42,252	5,479	24,657	5.9MB
gawkpkg	152,541	21,895	83,282	50MB

There are two important characteristics to note. First, according to PASS, all the entities are versioned. Second, our `gawkpkg` workflow is significantly larger than the challenge workflow. These two characteristics made provenance integration significantly more difficult than previously experienced in the provenance challenges. In the next section, we discuss those challenges, leading into Section 5 where we propose a framework for moving forward in provenance interoperability.

4 Challenges

The challenges we encountered fell into three general categories: provenance encoding, object identity including issues of naming and versioning, and verification of provenance across systems. In each case, the problems stem from the fact that OPM interoperability is a weaker requirement than query interoperability. Query interoperability requires a deeper semantic understanding of the nodes comprising a provenance graph.

This deeper semantic understanding is similar to schema integration, it goes beyond that. Schema integration is focused on finding the overlaps and commonalities between two schemas [22]. The problem in this case goes beyond schema matching since the underlying semantics are different in different provenance capture systems. The expectations of what granularity an object is captured at, the relationship types encapsulated, etc. are all built into each individual provenance system. When provenance is shared across systems, we are faced with issues deeper than finding that "employee" == "person"; we are forced to interpret the fundamental assumptions of the underlying semantics of provenance in the sharing systems. We explore these issues here by considering the challenges of importing data in the PASS XML format into PLUS.

The PASS XML format contains lineage records arranged in a way that is convenient for PASS to export (see Figure 2). We found that ordering the XML output in this way created several implementation difficulties when it came time to reading those XML files and use them to create lineage records in PLUS. This discussion details what those issues were, and how they were resolved.

```
01 <pass-data>
02 <provenance pnode="2" version="0">
03   <record type="TYPE">
04     <data>FILE</data></record>
05   <record type="NAME">
06     <data>myscript</data></record>
07 </provenance>
08 <provenance pnode="13" version="0">
09   <record type="TYPE">
10     <data>PROC</data></record>
11   <record type="ARGV">
12     <data>myscript -v
--ignore-case -v --case-sensitive
mycfg data.in data.out
13     </data></record>
14 </provenance>
15 <provenance pnode="13" version="2">
16   <record type="INPUT">
17     <xref pnode="2" version="1">
18   </record>
19   <record type="INPUT">
20     <xref pnode="14" version="1">
21   </record>
22   <record type="INPUT">
23     <xref pnode="15" version="1">
24   </record>
25 <provenance pnode="14" version="0">
26   <record type="TYPE">
27     <data>FILE</data></record>
28   <record type="NAME">
29     <data>mycfg.in</data></record>
30 </provenance>
31 <provenance pnode="15" version="0">
32   <record type="TYPE">
33     <data>FILE</data></record>
34   <record type="NAME">
35     <data>data.in</data></record>
36 </provenance>
37 <provenance pnode="15" version="1">
38     ...</provenance>
39 <provenance pnode="16" version="0">
40   <record type="TYPE">
41     <data>FILE</data></record>
42   <record type="NAME">
43     <data>data.out</data></record>
44 </provenance>
45 <provenance pnode="16" version="1">
46   <record type=INPUT>
47     <xref pnode="13" version="2">
48   </record></provenance></pass-data>
```

Figure 2: Contrived PASS XML data corresponding to the execution of `myscript ... mycfg data.in data.out`.

4.1 Encoding

We found that unresolved and forward references, both of which are allowed by OPM, proved difficult to handle in the large provenance graphs we were manipulating. Additionally, while OPM defines specific object types, it intentionally avoids defining representations of those objects or even how to specify object types. However, query interoperability requires understanding object types and details of process activation such as command-line parameters and their order.

4.1.1 References

When the XML representations of provenance graphs exceed memory sizes, addressing forward and unresolved references becomes challenging.

Forward References OPM places no restrictions on the order in which nodes are defined in an XML representation of the provenance graph. PASS orders nodes by pnode and version, since this is the order it uses internally. Many systems, including PLUS, view history and therefore provenance as static. In such systems, provenance records are naturally immutable. In PLUS, an object must be defined before it can be used, so forward references pose a problem. Consider the XML file from PASS in Figure 2. The record $<15, 1>$ is utilized in the relationship on Line 23 before it is actually defined within the XML file. Because PLUS considers provenance immutable, it is impossible to create a dummy object using only the information found in the relationship and then fill in the complete object when it is finally encountered. Instead, all relationships utilizing an unseen object must be saved until all objects have been declared. While this is possible in small files, it will not scale to large systems, and is the result of philosophical differences. OPM's XML schema does not disallow this behavior nor do xsds provide sufficient to express such a restriction. Forward references can be addressed by: outlawing them, performing a topological sort or – our approach – creating placeholders for the not-yet-defined nodes awaiting their resolution. As the number of nodes grows, this becomes a resource consumption issue.

Unresolved References OPM specifies that objects have unique identifiers, and does not require that all identifiers referenced in an OPM graph also be defined in that graph. PASS takes advantage of this fact and produces OPM graphs containing references to nodes that are not defined in the XML representations of them. Line 20 of Figure 2 is an example of an unresolved reference as it refers to version 1 of an object but only version 0 appears. PASS does not reify versions that do not differ

thus the reference to version 1 is equivalent to a reference to version 0. Not only do these unresolved references pose a challenge for interoperability, simply identifying which references are unresolved is non-trivial in large provenance files. Validation against an xsd is insufficient to express this restriction; we need a richer constraint mechanism which becomes computationally expensive. At present, we cannot distinguish an unresolved reference from a forward reference until we get to the end of the XML file. Therefore we end up treating them just like forward references, creating placeholders for them until we get to the end of the file and realize that they are unresolved. We would support a requirement that data files not have any unresolved references. Another option would be to resolve those references as part of a topological sort.

4.1.2 Entities

In addition to challenges posed by references, the targets of those references, entities, also introduce interesting problems. It is not clear whether all OPM documentation agrees as to how to encode entity types or identify entities of being of a particular type. For example, it is unclear whether activities are widely accepted as they do not appear in the XML schema [3, 15]. Furthermore, although OPM provides for the representation of process arguments, we need additional specifications of those arguments to facilitate query interoperability.

Entity type PASS versions objects to represent those objects at different points in time. As mentioned in Section 2.3, PASS versions are implicitly related. However, PASS identifies entities as being either artifacts or processes when they are first created, thus the type is associated only with the first version of an entity. Therefore, identifying the type of an entity requires tracing the version relationship back to the initial version. Because there is no requirement that all versions of an entity appear in an OPM graph it may be impossible to identify an entity's type.

Arguments While the OPM can deal with arguments to processes and even offers the ability, through roles, to describe how those arguments are used, the schema allows for too much flexibility in encoding arguments. Roles allow participation to be described. The `myscript` command might note that its first argument is the configuration file and the second is the output data. This may work well if the system knows the semantics for each command but does not apply more generally. For instance, it is not clear how to specify command line arguments and environment variables. Command line arguments are a vector; order and repetition matter. Some processes allow `-v` to be passed multiple times with each specification raising the verbosity of the output. Other commands support flags with opposite meanings with the last specified taking precedence – such as `--ignore-case` and `--case-sensitive`. We know of no agreed upon ways in the provenance community to record vectors in a way that preserves the semantics we have just described. For true query interoperability, it is imperative that arguments behave the same way across systems. While order is an attribute that can be added within a schema, the required repetitions, order and semantics of understanding certain arguments cannot be well defined purely through schema validation.

Activities In PASS, an activity store could be specified either as the name of the executable or the pnode and version corresponding to that executable. Given the OPM model and schema, it is not clear which of the two is correct. While the difference may seem irrelevant, next consider a binary file that supports several execution modes. By reporting just the name of the file, the exact execution mode is lost. For example, in many systems `grep`, `egrep` and `fgrep` invoke the same executable – the name by which the executable is invoked dictates its behavior. Is `grep` best modeled as one common activity or or three separate activities? The existence of a model and schema state that an activity exists, but gives no indication as to the semantics of how that activity should be recorded and queried.

4.2 Identification

Entity identification is a fundamental requirement of provenance. However, the notion of identity becomes blurred in the presence of mutation. Systems such as PLUS that consider only immutable entities handle mutation by creating new entities; systems such as PASS represent mutation through versioning. OPM happily supports both representations, but query interoperability requires that we be able to translate from one to the other.

Naming PASS identifies objects by a $<pnode, version>$ pair whereas PLUS uses a randomly generated unique id. This means that references to other nodes must be converted from pnode version pairs to PLUS identifiers. This requires that an importer maintain a mapping, which consumes considerable overhead when dealing with tens of thousands of records.

Versioning PASS uses versioning to represent mutation. It is not clear how to meld this with the OPM. Folding all the versions together reduces the semantic content, one symptom of which includes the possibility of

introducing cycles [9]. Another option is to create OPM edges to connect the various versions explicitly. However, these edges have important semantic meaning that is not captured by existing edge types. This poses a problem as the PLUS visualization tools had no idea how to deal with these implicit relationships. The engineering solutions, building extra edges to represent version relationships or merging versions, creates only one-off solutions. For query interoperability across systems, the underlying meaning must be well defined and approach agreed upon.

4.3 Verification

Given an XML representation of an OPM graph, the only validation possible is schema validation. Schema validation ensures only that the XML file being imported conforms to the given schema in terms of elements and structure. This guarantees that an edge or artifact has the appropriate attributes populated and that edges are of appropriate types. However, it is impossible to require and check higher level constraints, such as those discussed in the previous sections. For instance, schema validation cannot determine that the nodes adjacent to an edge are all unique, i.e., that there are no self loops. Currently, within a valid OPM XML document, it is possible to state that an edge exists between a node and itself, despite the fact that this is in clear violation of the OPM model. An imported file that validates against the OPM XML schema could still violate the OPM model. Even if such violation did not occur, we lack agreement on specifying references, encoding arguments, identifying activities as well as other issues discussed above.

5 Common Provenance Framework

Our experience merging data from two systems suggests that mere conformance to OPM is insufficient to facilitate the integration that we needed. We suggest that there are a collection of concepts, constraints, and tools that are crucial to manipulating provenance from heterogeneous systems.

We begin by introducing dictionaries and collections as abstract concepts essential to reasoning about provenance across systems. Then we introduce a number of constraints that we want to impose on the provenance documents we exchange. Finally, we introduce the Schematron [4] to validate that a document conforms to both the OPM and our constraints.

5.1 Concepts

Naming is repeatedly identified as one of the challenging aspects of provenance integration. We introduce dictionaries to address that problem. A second challenge arises out of the fact that different provenance systems capture data at different semantic levels and different granularities. We propose collections to manage this complexity.

Dictionaries While we might be able to impose constraints to create a consistent naming system among different systems, for example, we can require that all artifacts begin with "Artifact", the same object on two different systems may still be referenced using different names [19], which creates problems. We propose source identifiers and explicit naming dictionaries to address this problem. Source identifiers specify the system from which the data is being imported; naming dictionaries map names on one system to names on another system. Existing literatures describes approaches that provide the capabilities for such dictionaries [12].

Let's consider the use of dictionaries in the PASS/PLUS scenario. Imagine that we have an object represented in a PASS file by the pnode 42 and the version 7. We will refer to the PASS name by the tuple $<42, 7>$. PLUS wants to use the unique ID 123456789 to reference the object. Let's assume that the PASS source identifier is PASS-v2.0-140.247.60.118, which identifies that this is a PASS export from version 2.0 of the PASS system, generated by the machine with the IP address 140.247.60.118. The PASS exported file must contain a dictionary entry that maps $<42, 7>$ to PASS-v2.0-140.247.60.118$<42, 7>$. When PLUS imports this object, it creates a dictionary entry mapping 123456789 to PASS-v2.0-140.247.60.118$<42, 7>$. Now, one can construct tools that translate between the different names of the object.

We can take this example one step further. Let's imagine that PLUS now wants to export this object to a third system. The PLUS export dictionary must contain a mapping for the object, but it has two ways of expressing that mapping. It can either export it as an object with its own source ID or it can export the originating reference – correct behavior is a function of the access controls on the object. That is, it can export a dictionary mapping: 123456789 to either PLUS-123456789 (the local name) or PASS-v2.0-140.247.60.118$<42, 7>$ (the originator's mapping).

Together source identifiers and naming dictionaries provide the ability to translate names in one provenance system to names in another system and transfer such references among arbitrary systems.

Collections Dictionaries solve a naming problem when objects map 1:1 between systems. However, when mappings are many:1 or 1:many, a simple dictionary is insufficient. Consider the interoperability that arises between two systems one of which tracks provenance on

lines (records) in a file, while the other tracks provenance on the file in its entirety. The file is composed of a *collection* of lines; it is useful to explicitly represent that relationship in a way that facilitates assigning provenance attributes to the correct object (a particular line or the entire file).

We propose an encoding for collections that is flexible enough to support multiple uses. *Combining* collections encode multiple objects (e.g., lines of a file) as a single object (e.g., the file), including any edges relating the members of the collection. This provides a way to assign a unique identity to the collection itself. A *Version* collection is a special form of a combining collection that encodes object mutability as expressed via versioning, as is done in PASS. In a version collection, the relation expressed by the edges of the members of the collection is the versioning relationship. *Splitting* collections handle the inverse of combining collections. In this case, we encode a single object as multiple constituent objects. It splits the one object into several component objects and edges, creating explicit edges in the provenance graph expressing how those components comprise the original object.

In the PASS to PLUS conversion, we use Version combinations to coalesce all versions of an entity to a single entity for manipulation in PLUS. While the split collection is not used in this effort, if we were going from PLUS to PASS, it could prove useful.

5.2 Constraints

Dictionaries and collections provide key concepts to facilitate provenance exchange, but we can go further if we are willing to constrain how we represent provenance. The constraints serve two purposes: model-integrity and query interoperability. Model-integrity ensures that the graph represented in an XML document is conformant to the OPM Model. Query interoperability formally defines the semantics of entities and activities in a manner that ensures the ability to execute provenance queries on provenance data assimilated from more than one system.

Reference Requirements As discussed in Section 4, forward and dangling references posed problems for PLUS when ingesting PASS data. We address dangling references via the dictionary and forward references through ordering.

It is infeasible to remove dangling references entirely. Consider a computational scientific experiment where years of research and provenance describe an experimental result. We would like to be able to exchange the output data with colleagues and provide provenance so they can continue working with the data and produce future results. However, we do not wish to require that the provenance captured explicitly contain the entire several-year history of the data. Thus, at some point, we want the provenance transmitted to be sufficient permit queries on the original system when necessary. In lieu of removing dangling references, we require that all entities not explicitly defined in the exported provenance contain dictionary references that can be used to query about that entity.

In the case of our research result above, it may be sufficient to simply transmit the data object itself with nothing more than a single dictionary entry that tells the receiving entry the source identifier and local name of the object. Thus the OPM graph would contain a single node representing the entity. In PASS this node would be named with a pnode,version tuple and the dictionary entry would be: $<pnode, version>$ maps to PASS-v2.0-140.247.60.118$<pnode, version>$. The exporting system might choose to export more data such as the graph describing the last transformation that produced the result with dictionary entries for every input to that result.

Forward references are, in some sense, simpler – we just require that we define nodes before we reference them. This suggests that we need to either perform a topological sort on all the nodes in the exported provenance graph, or that we output ancestors before their descendants. In practice, this proves potentially challenging. It is significantly easier to maintain ancestry links from children to parents, because at the time a child is created, it knows the identity of its ancestors, while the set of children of a node may be constantly growing. Nonetheless, we believe that the burden of sorting should rest with the exporting system, not the importing system, so we require that entities be topologically sorted in the export file.

Acyclic Constraints While the OPM and every other formal model of provenance of which we are aware describe provenance as forming a directed acyclic graph (DAG), it is possible to create schema-conforming XML representations of an OPM graph that are cyclic. Clearly, we need to make this invalid, thus we impose the constraint that the XML not represent cyclic graphs.

Entity Constraints Entity types are fundamental to OPM and to other provenance models and representations. Regardless as to whether an entity is an artifact, process or even a collection we argue that its type must be known. As we mentioned earlier, it is not clear whether or not this is a hard requirement considering the differences between the various OPM documents and the XML schema. We argue for keeping the type information outside the dictionary because we are transferring the type information with the individual nodes and maintaining this binding should simplify collections and as-

sure there is one location for type information — therefore no disagreement. Note that a collection that represents a process, might contain artifacts. This mapping and its complexities should be encoded in the system not a dictionary. Without types, it is difficult, if not impossible, to make inferences over the graph, thus we further constrain provenance graphs to contain only typed references. Since it is not clear how a receiving system should handle a reference to an invalid or untyped reference, we propose only allowing subclassing of an agreed upon set of entity types.

Constraint Summary We have outlined four constraints we wish to impose on provenance to better facilitate interoperability. We believe there are other worthwhile constraints and encourage the community to agree upon a common set. We summarize our constraints here.

- All references not defined in an exported file must appear in the dictionary.
- Entities should appear in topological order in the export file.
- The exported graph must not contain cycles.
- Every entity must specify its type.

5.3 Verification

Many of the constraints described above are implicit in the OPM model. The trouble arises because schema validation is insufficient to verify that all the constraints are observed.

XML schema validation makes guarantees about the structure of an XML document and the values it can contain. While schema validation can check the schema to ensure that the appropriate element types are used, schema validation can not enforce ancestor-descendent relationships. Schema validation can check to ensure that a subset of values are used in the XML document for an attribute, as defined in the schema. Schema verification can go a step further, restricting values across attributes based on the values of other attributes [3]. The Schematron [4] is a post-Schema "rule-based verification for making assertions about the presence or absence of patterns in XML trees," [5]. We propose the use of a Schematron to ensure that documents conform to the OPM and strictly adhere both to the OPM and additional constraints of the Common Provenance Framework.

Schematrons provide the power to create rules across elements and attributes (and their values). However, Schematron rules must be crafted carefully to be sure that the appropriate rules fire in the appropriate order. For instance, consider the constraint that all *red* cars must have *grey* interiors. If an XML document contains a *brick red* car, the rule does not fire, and the document will pass verification even if the interior is not grey. Thus, in adopting a Schematron, we must take care in specifying rules and constraints in sufficient precision and generality to facilitate the development of the other tools.

6 Related Work

The First Provenance Challenge [18] compared query results on a common workload, whereas the second and third challenges sought to test interoperability by forcing groups to import and query data from other groups. Over the course of the first two challenges the workload and queries remained the same, while the third challenge introduced a more complex workflow and set of queries [20]. Very diverse groups participated in the challenge, each with their own ideology on what provenance information to capture, from the grid [17], to workflow executions [8] to higher-level workflow modifications [21]; and unique storage mechanisms, from relational [10], to RDF triples [6]. The diversity of systems and experiences produced agreement on a few noteworthy aspects of provenance. Everyone now agrees that provenance forms a directed acyclic graph (DAG). There is also agreement that objects can be artifacts or processes. Some also distinguish activities, where processes are instances of activities[15].

To cope with a higher degree of interoperability, we propose a set of constraints, concepts, and tools. These concepts and tools are not unique to provenance. For example, dictionaries are widely utilized to map concepts or objects between systems. MiMI [11] utilizes a set of dictionaries provided by the National Center for Biotechnology Information (NCBI) to relate and integrate proteins that are reported in disparate systems under disparate names. Additionally, the idea of having constraints over an XML schema is not new; nor is the use of a Schematron [4]. In fact, production systems exist that actively rely upon the use of the Schematron to further constrain and check XML documents between systems. One of the most famous systems is in use by Health Information Technology Standards (HITSP) to allow sharing of medical records between medical institutions. US law requires that all electronic health records are exchanged in the C32 format, which has an XML representation. NIST's Schematron [2] and Laika [1] perform XML verification of valid C32 documents before they are exchanged between medical institutions.

7 Conclusions

In this work, we utilize the OPM model [14] and schema to exchange provenance information between the

PASS [16] and PLUS [7] provenance systems. Our goal was to achieve *query interoperability*. In other words, we wished to use the visualization and graph query capabilities of PLUS over provenance collected and stored by a completely different system, PASS. While the OPM model and schema provided a good starting point, OPM interoperability is not query interoperability. After encountering and working through a series of challenges during the integration, we arrived at set of recommendations for a Common Provenance Framework that includes additional concepts, constraints and tools The Common Provenance Framework builds on the OPM for tighter and easier query interoperability.

We have recommended a series of constraints to facilitate the transfer of richer semantics when transferring provenance between systems. Enforcing a topological order, defining unresolved references and forcing all entities to make their entity-type explicit greatly simplify importing provenance into other systems. We also suggest other specific constraints to enrich the semantics. Dictionaries provide a mechanism for bridging identity between the exporting and importing systems. Collections bridge the granularity divides that occur when an entity on the exporting system maps to either several or part of an entity on the receiving system. Version collections provide a way of handling versioning constraints. We recommend the use of vectors to encode entities — such as command line arguments — where the number and order of the arguments is semantically meaningful. We recommend the use of Schematron to enforce these constraints.

A Common Provenance Framework built on top of OPM should allow systems to share a common sense of object identity and provide the ability to share query and visualization tools. The scheme should allow for maintaining identity in distributed environments. It should also allow visualizations to realize more than just a common sense of entity-type and relationship-type. Given a richer encoding for parameters, it should be possible to cross domains and still preserve the semantic meaning of the parameters from other systems. Collections should enable systems to relate low level observations with coarser views. Similarly granularity over time should also be bridgeable. While we have not slain all semantic dragons possible, we believe that this approach is a good and extensible approach that takes a step in the right direction.

References

[1] LAIKA. http://laika.sourceforge.net/.

[2] NIST Schematron. http://xreg2.nist.gov/cda-validation/archives.html/.

[3] Open Provenance Model XML Schema.

[4] Schematron. http://www.schematron.com.

[5] Schematron. http://en.wikipedia.org/wiki/Schematron.

[6] Tupelo. http://tupeloproject.ncsa.uiuc.edu/.

[7] BLAUSTEIN, B. T., SELIGMAN, L., MORSE, M., ALLEN, M. D., AND ROSENTHAL, A. Plus: Synthesizing privacy, lineage, uncertainty and security. In *ICDE Workshops* (2008), pp. 242–245.

[8] BOWERS, S., MCPHILLIPS, T., WU, M., AND LUDSCHER, B. Project histories: Managing data provenance across collection-oriented scientific workflow runs. In *DILS* (2007), pp. 27–29.

[9] BRAUN, U., HOLLAND, D. A., MUNISWAMY-REDDY, K.-K., AND SELTZER, M. Coping with cycles in provenance. http://www.eecs.harvard.edu/~syrah/pass/pubs/cycles.pdf, 2006.

[10] COHEN-BOULAKIA, S., BITON, O., COHEN, S., AND DAVIDSON, S. Addressing the provenance challenge using zoom. *Concurr. Comput. : Pract. Exper. 20*, 5 (2008), 497–506.

[11] JAYAPANDIAN, M., CHAPMAN, A., ET AL. Michigan Molecular Interactions (MiMI): Putting the jigsaw puzzle together. *Nucleic Acid Research* (Jan 2007), D566–D571.

[12] KEMENTSIETSIDIS, A., ARENAS, M., AND MILLER, R. J. Mapping data in peer-to-peer systems: semantics and algorithmic issues. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 2003), ACM, pp. 325–336.

[13] MOREAU, L., ET AL. The First Provenance Challenge. *Concurrency and Computation: Practice and Experience*. Published online. DOI 10.1002/cpe.1233, April 2008.

[14] MOREAU, L., FREIRE, J., FUTRELLE, J., MCGRATH, R., MYERS, J., AND PAULSON, P. The open provenance model, December 2007.

[15] MOREAU, L., KWASNIKOWSKA, N., AND DEN BUSSCHE, J. V. The foundations of the open provenance model. April 2009.

[16] MUNISWAMY-REDDY, K.-K., HOLLAND, D. A., BRAUN, U., AND SELTZER, M. Provenance-aware storage systems. In *Proceedings of the 2006 USENIX Annual Technical Conference* (June 2006).

[17] Provenance aware service oriented architecture. http://twiki.pasoa.ecs.soton.ac.uk/bin/view/PASOA/WebHome.

[18] The First Provenance Challenge. http://twiki.ipaw.info/bin/view/Challenge/FirstProvenanceChallenge.

[19] The Second Provenance Challenge. http://twiki.ipaw.info/bin/view/Challenge/SecondProvenanceChallenge.

[20] The Third Provenance Challenge. http://twiki.ipaw.info/bin/view/Challenge/ThirdProvenanceChallenge.

[21] SCHEIDEGGER, C., KOOP, D., SANTOS, E., VO, H., CALLAHAN, S., FREIRE, J., AND SILVA, C. Tackling the Provenance Challenge one layer at a time. *Concurrency and Computation: Practice and Experience 20* (April 2008), 473–483.

[22] SMITH, K., MORSE, M., MORK, P., LI, M., ROSENTHAL, A., ALLEN, D., AND SELIGMAN, L. The role of schema matching in large enterprises. In *CIDR* (2009).