A Use Case-Guided Comparison of OPM and PML

Paulo Pinheiro da Silva, Steve Roach

Department of Computer Science,
University of Texas at El Paso, El Paso, TX

Abstract

The World Wide Web Consortium (W3C) Provenance Incubator Group has the goal of providing "a state-of-the art understanding ... in the area of provenance for Semantic Web technologies." To enhance the mutual understanding of language capabilities, the group is attempting to map several of the existing provenance languages such as Provenir, PREMIS, and the Proof Markup Language (PML) into the provenance language the Open Provenance Model (OPM). OPM is intended to be a precise inter-lingua for exchanging provenance information. This article contributes to the understanding of the capabilities of OPM and PML by establishing a set of six common and relatively simple provenance use cases and comparing the OPM and PML models and implications of those models. A provenance use case consists of a scenario and a provenance question associated with the scenario. Some of the use cases are taken from the OPM specification document. The use cases in this article may be considered essential for any provenance encoding intended to be used for provenance interoperability. The modeling of the use cases exposes a number of substantial difficulties in creating and interpreting OPM specifications for use by machine reasoning systems.

Keywords:
Provenance, PML, OPM, Semantic Web

1. Motivation

The World Wide Web Consortium (W3C) Provenance Incubator Group (PROV-XG[1]) has been established with the goals of providing a state-of-the art understanding of, developing technologies in, and possibly standardizing the area of provenance for the Semantic Web (http://www.w3.org/2005/Incubator/prov/charter). Many are the candidates for a common provenance language including Open Provenance Model (OPM) [1], Proof Markup Language (PML) [2], Provenance Vocabulary [3], and PREMIS [4]. In an effort to create a mutual understanding of the capabilities of these languages, PROV-XG selected one of them to be a target language and asked the group to map the other languages into the target language. From the candidate languages, OPM has the larger number of users within PROV-XG, and OPM was selected to be the target language.

In order to explore the mapping of PML into OPM, we first identified provenance use cases that we consider to be essential for encoding, recording, and querying provenance information. We then compared the the handling of these use cases by both OPM and PML. During the effort of mapping PML into OPM, we identified serious limitations of OPM. The quality of the mapping from PML to OPM raises questions about the usefulness of these maps to achieve effective provenance interoperability. To further explore the impact of these modeling limitations, we discuss how the use cases affect the support that provenance provides to applications based on an artifact's attribution, understanding, reproducibility and trust.

This article does not advocate for the use of any specific notation. Our expectation is that the issues discussed in this article will create a common understanding of both OPM and PML, foster a better understanding of how other provenance notations can be mapped into OPM and PML, and eventually contribute towards a development of a standard provenance language.

The rest of this article is structured as follows. A brief review and an initial mapping of OPM and PML is presented in Section 2. A collection of use cases used describing challenges on the use of languages to encode provenance information is presented in Section 3. A description of how OPM and PML are used to encode

[1] PROV-XG is the tag name of the group within W3C

use case-related provenance information is presented in Section 4. A summarized discussion about key differences between OPM and PML is presented in Section 5. Conclusions are presented in Section 6.

2. OPM and PML Background

An inter-lingua for provenance is minimally expected to have language constructs, syntax for putting these constructs together, and a notation. Both OPM and PML have those language components. OPM constructs are mainly derived from scientific workflow concepts, while PML constructs are mainly derived from proof theory as described in [5]. The most noticeable difference between these languages is with their notations. OPM has a graphical notation designed for human consumption, while PML uses RDF/XML aimed for machine consumption. Moreover, we see that OPM developers are becoming more concerned about the machine use of provenance: there are two ongoing proposals for an XML serialization of OPM models. It is also true that PML users would benefit from a graphical notation for the language, and some PML developers would like OPM to be its graphical notation. A major issue for the PML adoption of OPM's graphical notation, however, is that there is no straightforward mapping between PML and OPM constructs and syntaxes, which is the discussion topic of this article. An additional benefit of this PML and OPM mapping would be that of formalizing a semantics for OPM and to ground this semantics into well-established proof-theory.

In the rest of this section, we briefly review the main constructs, syntax and notation of OPM and some specific components of PML. PML will be further described along with use cases used to discuss the OPM and PML encodings of provenance.

2.1. OPM Basic Scenario

This section provides a brief description of the Open Provenance Model through an example provenance scenario. This section is strictly based on definitions from the OPM core specification V.1.1 [1], which is also the source of the quotes used in this section.

A provenance model in OPM is a graph composed of nodes connected through direct arcs. A node can be an oval representing an OPM *Artifact*, a box representing an OPM *Process* or an octagon representing an OPM *Agent*. For example, Figure 1 shows the provenance of an artifact `cake` where: `bake` is a Process; `100g butter`, `two eggs`, `100g flour`, `100g sugar` and `cake` are Artifacts; and `John` is an Agent.

- An *Artifact* is an "immutable piece of state, which may have a physical embodiment in a physical object, or a digital representation in a computer system" (OPM's Definition 1);
- A *Process* in an "action or series of actions performed on or caused by artifacts, and resulting in new artifacts" (OPM's Definition 2);
- An *Agent* is a "contextual entity acting as a catalyst of a process, enabling, facilitating, controlling, or affecting its execution" (OPM's Definition 3).

An arc in an OPM model is "a causal dependency between the source of the arc (the effect) and the destination of the arc (the cause)." For example, the model in Figure 1 shows ten causal dependencies (four solid arcs and six dotted arcs) representing four kinds of causal dependencies between nodes. The kinds of the causal dependencies (and thus the meaning of the arcs) are specified by the types of the nodes that are the end-points of the arcs, i.e., the sources and destinations of the arcs. These types of causal dependencies are defined as follows:

- An edge of type *used* connects a source Process to a destination Artifact. It indicates "that the process required the avilability of the artifact to be able to complete its execution" (OPM's Defininition 5);
- An edge of type *wasGeneratedBy* connects a source Artifact to a destination Process. It indicates "that the process was required to initiate its execution for the artifact to have been generated" (OPM's Definition 6);
- An edge of type *wasDerivedFrom* connects a source Artifact to a destination Artifact. It indicates that the destination artifact needs to have been generated for the source artifact to be generated (OPM's Definition 8);
- An edge of type *wasControlledBy* connects a source Process to a destination Agent. It indicates that "the start and end of process P was controlled by the agent" (OPM's Definition 9).

In terms of the example in Figure 1, we can say that `John` started and ended the process `bake`. That `bake` could only occur because `100g butter`, `two eggs`, `100g flour`, `100g sugar` were available before the end of the execution of `bake`. That `cake` was the output of `bake` and that it was derived from `100g butter`, `two eggs`, `100g flour` and `100g sugar`.

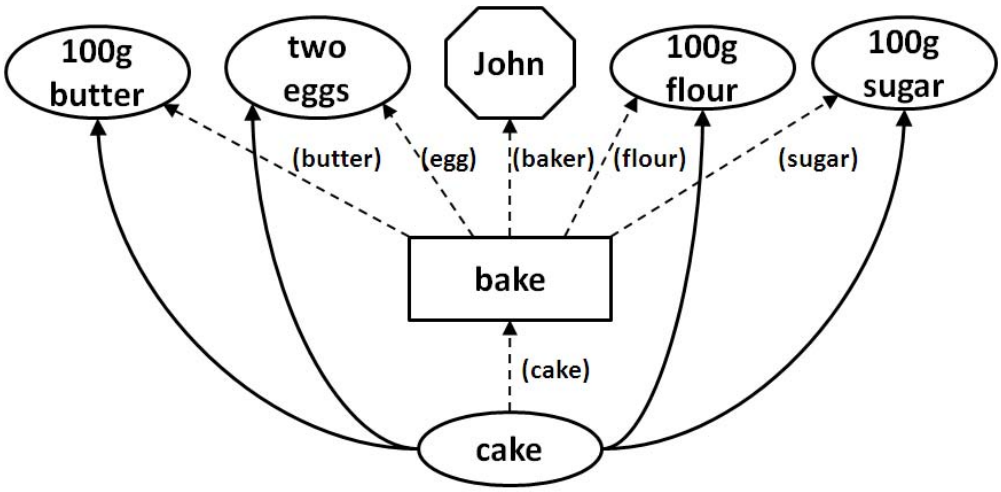

Figure 1: Bake cake use case (Figure 3) from the OPM specification V1.1.

2.2. PML Concepts and Relations

The PML primitive concepts and relations for representing knowledge provenance are formally specified in OWL and compiled into explanation ontologies known as *PML ontologies* or just *PML*. PML provides vocabulary for *justification metadata* (also called PML-P) and *justification data* (also called PML-J). Justification metadata focuses on representational primitives used for describing properties of identified-things such as information, language and sources (including organization, person, agent, services). Justification data focuses on representational primitives used for explaining dependencies among identified-things. This includes constructs for representing how conclusions are derived.

The goal of PML-P is to provide a set of extensible representational primitives that may be used to annotate the provenance of information. This includes, for example, representing which sources were used and who encoded the information.

- The foundational concept in PML-P is *IdentifiedThing*. An instance of IdentifiedThing refers to an entity in the real world, and its properties annotate the entitys properties such as name, description, creation date-time, authors, and owner. PML-P includes two key subclasses of IdentifiedThing motivated by knowledge provenance representational concerns: *Information* and *Source*.
  - The concept *Information* supports references to information at various levels of granularity and structure. It can be used to encode for example a formula in a logical language, a natural language fragment, or a scientific dataset. Line 3 in Figure 2 says that the content of Information is a string ‘‘`cake`’’.
  - The concept *Source* refers to an information container, and it is often used to refer to all the information from the container. A source could be a document, an agent, or a web page. PML-P provides a simple but extensible taxonomy of sources.

The goal of PML-J is to provide the concepts and relations used to encode traces of process executions used to both assert and derive a conclusion. A justification requires concepts for representing conclusions, zero or more sets of conclusion antecedents, and the information manipulation steps used to transform/derive conclusions from sets of antecedents. Note that antecedents may also be conclusions derived from other antecedents. The justification vocabulary has two main concepts:

- A *NodeSet* includes structure for representing a conclusion and a set of alternative steps, each of which can provide an alternative justification for a conclusion. The term NodeSet is chosen because it captures the notion of a set of nodes (with steps) from one or many justification trees deriving the same conclusion. Figure 2 shows the node set for `cake`. As stated in line 2 (*hasConclusion*), the Information `cake` is the is the conclusion of the node set. Because the node set has the property *isConse-*

```
<pmlj:NodeSet rdf:about="http://ex/Cake.owl#Cake">
  <pmlj:hasConclusion>
    <pmlp:Information>
      <pmlp:hasRawString rdf:datatype="http://www.w3.org/2001/XMLSchema#string">cake</
          pmlp:hasRawString>
    </pmlp:Information>
  </pmlj:hasConclusion>
  <pmlj:isConsequentOf>
    <pmlj:InferenceStep>
      <pmlj:hasIndex rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</pmlj:hasIndex>
      <pmlj:hasInferenceRule
           rdf:resource="http://ex/registry/DPR/BAKE.owl#BAKE"/>
      <pmlj:hasInferenceEngine
           rdf:resource="http://ex/registry/IE/JOHN.owl#JOHN"/>
      <pmlj:hasAntecedentList>
        <pmlj:NodeSetList>
          <ds:first rdf:resource="http://ex/Butter.owl#Butter"/>
          <ds:rest>
            <pmlj:NodeSetList>
              <ds:first rdf:resource="http://ex/Eggs.owl#Eggs"/>
              <ds:rest>
                <pmlj:NodeSetList>
                  <ds:first rdf:resource="http://ex/Flour.owl#Flour"/>
                  <ds:rest>
                    <pmlj:NodeSetList>
                      <ds:first rdf:resource="http://ex/Sugar.owl#Sugar"/>
                    </pmlj:NodeSetList>
                  </ds:rest>
                </pmlj:NodeSetList>
              </ds:rest>
            </pmlj:NodeSetList>
          </ds:rest>
        </pmlj:NodeSetList>
      </pmlj:hasAntecedentList>
    </pmlj:InferenceStep>
  </pmlj:isConsequentOf>
</pmlj:NodeSet>
```

Figure 2: PML encoding of Bake Cake example.

quenceOf (line 7), we can infer that the conclusion of the node set has at least one justification.

- An *InferenceStep* represents a justification for the conclusion of the corresponding NodeSet. The term *inference* refers to generalized *information manipulation step* (or just *step*). It could be a standard logical step of inference, an information extraction step, any computation process step, or an assertion of a fact or assumption. It can also be a complex process that cannot be described in terms of more atomic processes such a web service or application functionality. The node set in Figure 2 has one inference step (lines 8–34) and thus one justification. The justification is that *InferenceEngine* `JOHN.owl` (line 12) performed the inference rule `BAKE.owl` (line 11) using the conclusions of the following node sets as antecedents: `Butter.owl` (line 16), `Eggs.owl` (line 19), `Flour.owl` (line 22) and `Sugar.owl` (line 25).

The provenance for `cake` is distributed and is not entirely encoded in Figure 2. For example, `Butter.owl` in line 16 is another node set for Information *butter*, and as such it may have its own justification (or justifications). Further, we know that `JOHN.owl` in line 12 is a PML Agent playing the role of an inference engine, but one needs to visit `JOHN.owl`'s PML encoding to learn more provenance information about John such as full name and affiliations.

2.3. An Naiv̈e PML and OPM Mapping

The following is one naiv̈e mapping between PML and OPM:

- PML *Information* maps into OPM *Artifact*;
- PML *InferenceStep* maps into OPM *Process*;
- PML *InferenceEngine* maps into OPM *Agent*;
- PML *Source* maps into OPM *Observer* (not formally defined in the OPM specification); and
- A combination of a PML *isConsequenceOf* property and PML *hasAntecedentList* property map into OPM *wasDerivedFrom*.

The mappings above, although fairly intuitive and acceptable for those with a reasonable understanding of both PML and OPM, is far more complex than it appears to be. Adopting these mappings hides important limitations on the use of both languages, which needs to be discussed by a broader community. In the rest of this article, we discuss how just the first mapping between information and artifact holds.

3. Provenance Use Cases

In this article, use cases are used to discuss challenges to the use of both PML and OPM to model provenance situations. While the set of use cases described here is not complete, i.e., it does not cover every possible situation, the use cases do describe common and useful provenance questions. Each use case has a statement and a question and is identified by a number. The use case question is one that may be asked in the future about the provenance situation and is answerable from information in the use case statement. In an ideal provenance-supported environment, a provenance encoding is satisfactory if the encoding has knowledge enough to answer the provenance question.

3.1. Provenance of Assertions

According to Dictionary.com [6], an assertion is "a positive statement or declaration, often without support or reason." Assertion is a basic attribution mechanism, and without such mechanism one cannot properly identify the sources of artifacts.

Use Case 1. Statement*: John said that the sky is blue.*
Question*: Which entity said that "the sky is blue?"*

In Use Case 1, there is an information source, `John`, who is capable of *creating* and *providing* some information, specifically, that `the sky is blue`. The source in Use Case 1 is a human agent. More generally, an agent could be anything capable of creating data, for example, a thermometer (a sensor), which is capable of creating (sensing) the current temperature.

Use Case 2. Statement*: It is written in the NY Times that the sky is blue.*
Question*: Where was written that "the sky is blue?"*

In Use Case 2, "the NY Times" is an information source like "John" in Use Case 1, although it is a different kind of information source since it may not be the creator of the information it is providing. For provenance purposes, one may need to know that the news was the *provider* of the information.

3.2. Provenance of Derivations

Derivation occurs when new information (or artifact) is formed from available information (or artifact) through some transformation. Information available for a given derivation was asserted or created through other derivations. The Bake Cake example used to introduce OPM in Figure 1 is used here as a use case.

Use Case 3. Statement*: John is a cook who baked a cake using 100g of butter, two eggs, 100g of flour and 100g of sugar as ingredients. Recipe R was used to guide the process of baking the cake.*
Question*: What was the recipe used to bake the cake?*

The recipe of a cake is probably the best piece of information one may retrieve from a cake's provenance if the intention is to bake a similar cake. Knowing the right quantities of each ingredient is also important for a cook to bake a similar cake.

In terms of provenance, we stress three derivation aspects that need to be encoded: the dependency between the derivation's final product and the information (or artifacts) required or desired for performing the derivation; the exact role of each provided artifact during the derivation; and a description of the derivation in case one wants to reproduce the derivation. The question in Use Case 3 stresses the need to describe at some level the derivation.

3.3. Provenance of Derivation Roles

Use Case 4. Statement*: In addition to the statement in Use Case 3, it is also known that a heating device such as oven is required to bake the cake.*
Question*: Why do we use butter in the cake?*

The derivation dependency between `100g butter` and `cake` stated in Use Case 3 is different than the derivation dependency between `oven` and `cake` in Use Case 4. Butter plays a role of being a ingredient of cake, while oven plays a role of being a heating device of the baking process. Use Case 4 raises a number of important provenance challenges. A fundamental challenge is to know whether any artifact used as in input for a derivation constitutes part of the final product of the derivation or if it plays another role such as a supporting device or a control mechanim. Many are the possible answers for the question in Use Case 4. A relevant answer for provenance is that the cake is made, in part, of butter.

3.4. Provenance of Accounts

Credible information is often accredited to multiple sources. When information is derived through complex processes, it is often the case that accounts are not exactly the same. When distinct accounts are not identical, it is necessary to capture these multiple accounts and to combine common evidences as much as possible.

Use Case 5. Statement*: John stated that he saw a yellow Honda Civic hitting a post. John further mentioned that according to Joseph, another witness of the event, the same yellow Honda Civic ran a red light 30 seconds earlier, further down the street.*
Question*: Who saw the yellow Honda Civic running a red light and crashing into the post?*

In Use Case 5, it is unknown whether Joseph saw the car crashing into the post. It is known, however, that Joseph saw the car running the red light. In this case, we expect an answer to the question in Use Case 5 to be derived from two accounts: one from John and another from Joseph. Further, we expect the answer to Use Case 4's question to be that Joseph saw the car running the red light and that John saw the car crashing into the post.

3.5. Provenance Identifiers

Identifiers have been required throughout most of the use cases in this section. Here we stress the provenance need of an appropriate strategy to identify provenance elements.

Use Case 6. Statement*: Both John and Tania observed and reported a single event of a yellow Honda Civic crashing a post. John identified the car as 'the yellow car' while Tania identified the same car as 'the Honda Civic'.*
Question*: Are the two reports about the same event, i.e., the same car crash?*

The question in Use Case 6 requires provenance languages to provide information enough to compare and decide whether provenance entities in distinct accounts are the same. For example, for the car crashing in each report to be the same event, it is required for the car that crashed into the post in each report to be the same.

4. OPM and PML Support for Provenance Use Cases

In this section, we will exercise the use of both OPM and PML to encode the use case statements in Section 3.

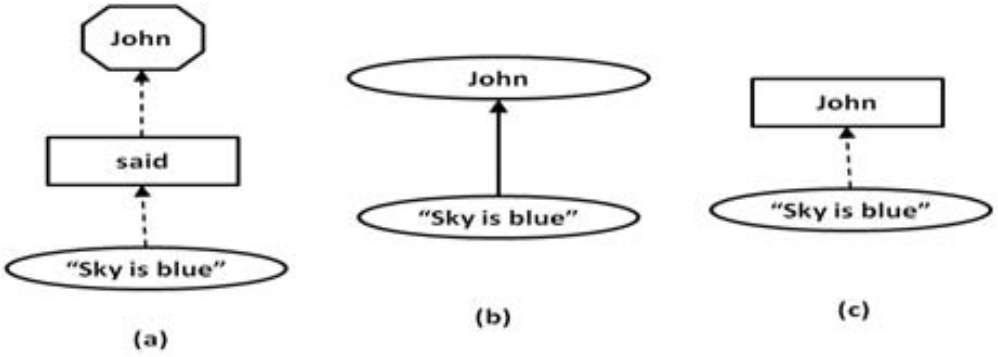

Figure 3: Encoding assertions in OPM.

4.1. Encoding Assertions

4.1.1. OPM Handling of Assertions

One way to encode Use Case 1 in OPM is to say that Agent is `John`, Process is `said`, and Artifact is `the sky is blue` as presented in diagram (a) in Figure 3. In this encoding, `John` is not explicitly credited as the entity asserting that `the sky is blue`: according to the OPM specification, agent `John` is controlling process `said`, which is the creator of the artifact `the sky is blue`.

An alternative OPM encoding for Use Case 1 is for the Artifact `the sky is blue` to be derived from another Artifact `John` as presented in diagram (b) in Figure 3. In this encoding, an issue is with the fact that `John` would not be a person but the state of the person at a given time. This creates a number of new challenges due to the potential need to identify other assertions from John without consideration of the state of John, e.g., assertions made by John when he was tired.

A third alternative OPM encoding for Use Case 1 is for the Artifact `the sky is blue` to be generated by a Process `John` as presented in diagram (c) in Figure 3. This third alternative is even more confusing than the previous alternatives: a process[2] is, according to the OPM specification, an action that ceases to exist immediately before the artifact `the sky is blue` is created. It is unlikely that John ceased to exist just because he asserted that `the sky is blue`.

For Use Case 2, one could use OPM to say that the Agent is `the NY Times` and that `is written` is the process of realizing that the Artifact `the sky is blue` is stated somewhere. OPM, however, does not have a way of saying *where* the artifact is written (i.e., where is the container of the artifact). Furthermore, OPM does not offer a straightforward way of encoding the fact that the artifact was asserted in a document such as `the NY Times`. Fundamentally, OPM lacks a mechanism to encode Artifacts as assertions, whether these are assertions from an agent as required in Use Case 1 or a document as required in Use Case 2.

4.1.2. PML Handling of Assertions

In PML, we say that an Artifact (that we map into PML *Information*) is asserted by an information source (i.e., PML *Source*). A PML Source is either a PML *Agent* or a PML *Document*, with the distinction that a PML *Agent* is capable of asserting PML *Documents* and that PML *Documents* are asserted by PML *Agents*. To illustrate this, we use the example in Figure 4. The node set in this figure is referred to in Figure 2. The node set, however, is the PML encoding that shows us the conclusion of the node set, `100g butter` as presented in line 4 of Figure 4, and the justification for conclusion as presented by the inference step in lines 8–17. This justification is an assertion since the rule `Told.owl` in line 10 is a *direct assertion*. Moreover, it tells us that the butter came from `Walmart.owl` (line 13) on the 17th of October 2010 at 10:30am (line 14). In this case, `Walmart.owl` is a PML *Agent* (to be more specific, this agent is of type PML *Organization* also defined in PML).

PML shares OPMs notion that OPM *Artifacts* are states of objects. PML *Source*, e.g., `Walmart.org`, however, is not a state of an object, but a proper representation of the object with a life span matching the life span of the real object, during which the object can transition through multiple states. The life span of a PML *Source* is a time interval, while *Information* is

[2]We understand that an OPM Process is in fact a process execution, an interpretation that we believe is shared with some of the OPM developers.

```
1  <pmlj:NodeSet rdf:about="http://ex/Butter.owl#Butter">
2    <pmlj:hasConclusion>
3      <pmlp:Information>
4        <pmlp:hasRawString rdf:datatype="http://www.w3.org/2001/XMLSchema#string">100g butter</
             pmlp:hasRawString>
5      </pmlp:Information>
6    </pmlj:hasConclusion>
7    <pmlj:ConsequentOf>
8      <pmlj:InferenceStep>
9        <pmlj:hasIndex rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</hasIndex>
10       <pmlj:hasInferenceRule rdf:resource="http://inference-web.org/registry/DPR/Told.owl#Told"/>
11       <pmlj:hasSourceUsage>
12         <pmlp:SourceUsage>
13           <pmlp:hasSource rdf:resource="http://ex/registry/ORG/Walmart.owl#Walmart"/>
14           <pmlp:hasUsageDateTime rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
                 2005-10-17T10:30:00Z</pmlp:hasUsageDateTime>
15         </pmlp:SourceUsage>
16       </pmlj:hasSourceUsage>
17     </pmlj:InferenceStep>
18   </pmlj:isConsequentOf>
19 </pmlj:NodeSet>
```

Figure 4: PML encoding of an assertion for `100g butter`.

a statement asserted at a time point. To associate Information to Source, PML has the *SourceUsage* concept (lines 12–15 in Figure 4), which assigns the *Source* as the entity asserting the *Information* at a given date and time encoded as a Source Usages timestamp (line 14). This timestamp is when *Information* associated with *SourceUsage* was retrieved from *Source*. PML *SourceUsage* construct has a number of other properties used to characterize how information was retrieved from information sources, as further described in [7].

4.2. Encoding of Derivations

4.2.1. OPM Handling of Derivations

OPM Approach for Causality. One approach to encode Use Case 3 is to say that the ingredients `100g butter`, `two eggs`, `100g flour`, and `100g sugar` plus the recipe *Recipe R* are the artifacts used by the process `bake` to derive an artifact `the cake` as shown in Figure 5. In this figure, according to OPM, all the ingredients and the recipe are the cause of the effect `cake`, since `bake` was caused by the presence of the ingredients and the recipe. In fact, [1] says that "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 claim that the encoding of the ingredients and recipe in Figure 5 are not sufficient for baking the `case` since none of these artifacts has the intent of baking the cake, i.e., the true cause of one baking the case. A reasonable cause for baking the cake would be something like "John is hungry" or "John was trying a new cake recipe he saw in the Food Channel". Even more explicitly, the intention for baking the cake could be a question such as "what can John cook for me?" None of these ways of expressing the intention of baking a cake are described in Figure 5.

Our understanding from the OPM Specification is that arcs are temporal dependencies between entities, but not necessarily causal dependencies. The OPM specification confirms that arcs are temporal dependencies when it states the following: "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."

OPM Approach for Artifact Dependency. For answering the question in Use Case 3 (what was the recipe used), it is unclear whether a "was derived from" relationship should be used between `Recipe R` and `cake`. Figure 5 shows that `bake` depends on `Recipe R` since "all of [edges connected to a process] were required for the process to complete" and `Recipe R` was required for the process bake to complete. The creation of the cake is also dependent on `bake` due to the "was generated by" edge connecting `bake` and the `cake`. We assume that one cannot infer that `cake` depends on `Recipe R` because of the following:

- Definition 4 in [1] is silent regarding the transitivity of causal relationships; and
- [1] says that "the fact that a process used an artifact and generated another does not imply the latter was

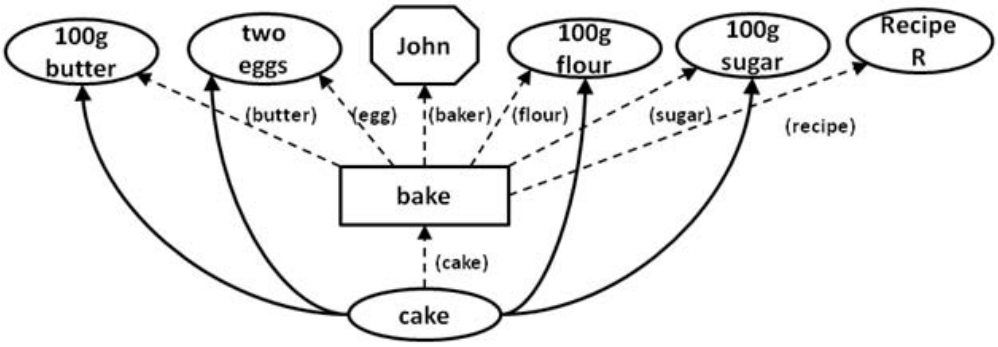

Figure 5: Using recipe R to bake a cake.

derived from the former; such relationship needs to be asserted explicitly."

If we assume OPM Causal Relationships to be transitive, then there is no need for one to use "was derived from" and the restriction above should be dropped. Otherwise, we question the purpose of using the relationships "used" since they are not part of the provenance of the cake, the object of discussion.

OPM Approach for Derivation Description – OPM Processes. The recipe of a cake is probably the best piece of information one may retrieve from a cake's provenance if the intention is to bake a similar cake. This is the motivation for the question in Use Case 3, and we ask if the OPM encoding in Figure 5 is enough to answer the use case's question.

A role, `recipe`, may be used to identify that `Recipe R` is the recipe for the cake. Since neither machines nor humans know the meaning of the general-purpose role, we view this solution as hard-coded for answering the use case question. For instance, one could have used `instruction` instead of `recipe` in Figure 5, which would break any effort of leveraging the role `recipe`.

Assuming that a recipe `Recipe R` can be retrieved from the OPM encoding as an answer to the use case question, what is the real meaning of an OPM Process, e.g., the meaning of the `bake` process? Is `Recipe R` a specification of a process and `bake` an execution of such process? We understand that these are all fundamental questions for effectively using OPM encodings.

4.2.2. PML Handling of Derivations

PML Approach for Causality. PML has explicit and implicit constructs to encode causality. Explicitly, PML defines a *Query* as a mechanism for annotating the derivation or assertion purpose of one or more artifacts. These are the artifacts that play the role of being answers for the query. Figure 6 shows how PML *Query* is used to explain the cause for John to bake a cake. In line 4, the PML Query has "What ?X can John cook tonight?"[3] as the query's content. `?X` is a query variable and `cake` is one of the possible answers for the query (i.e., a value for `?X`). `cake` was already defined and justified in 2. In other words, in the example above, `cake` is one of the things that `John can cook tonight`.

By inspecting the provenance of `cake` one can verify whether all the ingredients were available for baking the cake. The OPM encoding indicates that the ingredients were available at the time John baked the cake. The PML encoding also indicates the time the ingredient were available through the timestamps of the source usages of the ingredients and for the timestamp of the derivation.

Implicitly, PML defines the *Assumption* inference rule used to differentiate non-factual assertions from factual assertions. In this case, for example, the non-factual assertions are used to encode *hypothesis refinements*, *assumption testing* and *refutation statements* used to motivate artifact generation. This approach is further discussed in [8].

PML Approach for Artifact Dependency. In PML, temporal dependency between two artifacts is implemented through the use of a pair of PML properties: *isConsequenceOf* connecting an artifact playing the role of a conclusion of a node set to an inference step justifying the conclusion; and *hasAntecedentList* connecting the inference step to another node set. For example, there is a temporal dependency between `cake` in Figure 2 and

[3]The 'tonight' in the query's content corresponds to the night of the day that the query was asked.

```
1 <Query rdf:about="http://ex/Cooking.owl#tonight">
2   <pmlp:hasContent>
3     <pmlp:Information>
4       <pmlp:hasRawString rdf:datatype="http://www.w3.org/2001/XMLSchema#string">What ?X can John
            cook tonight?</pmlp:hasRawString>
5         <pmlp:hasLanguage rdf:resource="http://inference-web.org/registry/LG/English.owl#English"
              />
6     </pmlp:Information>
7   </pmlp:hasContent>
8   <hasAnswer rdf:resource="http://ex/Cake.owl#Cake"/>
9 </Query>
```

Figure 6: PML encoding of a query.

`100g butter` in Figure 4. This dependency is established by the *isConsequenceOf* property of the cake's node set in line 7 of Figure 2 and the *hasAntecedentList* property of the inference step in line 14 of the same figure.

This two-step connection has been criticized for not being user-friendly; however, this is the key mechanism for PML to support multiple accounts as further discussed in Section 4.4.2.

PML Approach for Derivation Description. The PML solution for encoding the statement in Use Case 3 is fundamentally distinct than the approach taken by OPM. As an action or collection of actions, an OPM Process would need to be triggered by the presence of some data (a data flow approach), or by another process (a control flow approach), or by a combination of both.

A relevant question is whether it is important for provenance to know the mechanism that was used to trigger a process or to know the conditions for triggering the process. We claim that the recipe of the action is more important than the invocation of the action because whatever method is used to invoke the action, it can rely on provenance information to derive an explanation for an action execution. Moreover, by having access to the recipe, one may be able to reproduce the experience of baking a cake. However, simply having access to all the ingredients as specified in OPM may not enable the cook to bake a cake. In PML, the recipe is modeled as an *InferenceRule* (we will further discuss these rules when we revisit OPM Roles in Section 3.3).

The recipe alone is not enough to tell us when and where the cake was baked, and such properties are important. In PML, an inference rule is attached to a derivation through an inference step that has the properties one would expect to know about a process execution [9]. For example, the fact that the inference rule `Bake.owl` in lines 10-11 of Figure 2 is not a direct assertion (otherwise it would be named `Told.owl`) tells us that the conclusion `cake` in line 4 was derived. In fact, a PML Inference Step is an "execution" of an inference rule, the `Bake.owl` rule in case of Figure 2, that identifies how information is derive from other information and thus how information depends on information.

4.3. Encoding of Derivation Roles

4.3.1. OPM Handling of Derivation Roles

Use Case 7 asks for an explanation for using butter in the process of baking a cake. Figure 7 includes an artifact `oven` that like `100g butter` is also required to bake the cake. The explanation for having `oven` as a required artifact for the process `bake`, however, should be different than the other artifacts required by the `bake` process that are ingredients of the cake. Moreover, although it is modeled that the butter is required to bake the cake, it is unclear why the butter is required.

In OPM, we see that roles are intuitive for human users of OPM models who understand the domain. The roles are meaningless for both machines and human users who do not understand the domain. The motivation for OPM to have roles may be one of the reasons why OPM lacks a better semantics for roles. For instance, roles may have been added to OPM in order to support, for instance, the restart of a workflow execution. Roles are used to bind values to ports of workflow actors at workflow restart time. Provenance, however, is also about explaining why such bindings are necessary. By not having proper descriptions of OPM processes, OPM is limited in its ability to specify the semantics of roles as promised in the OPM specification.

4.3.2. PML Handling of Derivation Roles

An inference step is an application (or execution) of an inference rule to the antecedents of the inference step. In PML, the *hasMetaBinding* property of an inference step is used to bind the inputs of an inference step, i.e., the antecedents, to the rule associated with the

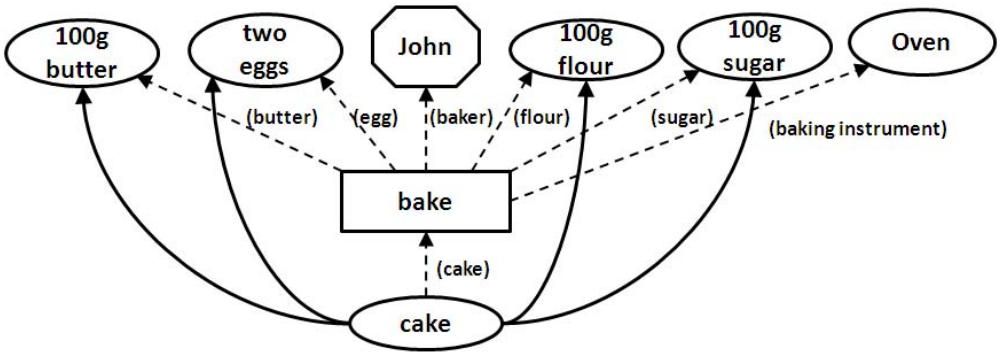

Figure 7: Using an oven to bake a cake.

inference step. The use of this binding mechanism is only required if the inference rule is formally specified and if there is ambiguity on how antecedents are bound to rule specifications. For example, let assume that the inference rule `bake` has the following specification in InferenceML:

```
Bake ::== (heat (mix a b c d) o);;
          (HeatingDevice o,
           Ingredient a b c d)
```

Without getting into the details of how the rule above is specified, we can say that to bake a cake (or to apply a rule `bake` to the antecedents of the inference step), one needs to mix **a**, **b**, **c** and **d** together and then use **o** to heat the mixture. In this case, **o** is a heating device and **a**, **b**, **c** and **d** are ingredients. If the type of **o** is already specified as a heating device and the `bake` rule requires just one heating device then there is no need for one to specify a role for **o**. If the order of **a**, **b**, **c** and **d** is irrelevant for the execution of the rule `bake`, it is also correct to omit the roles for the ingredients. However, if the order of the ingredients is relevant, one can use the *hasMetaBinding* property of inference step to say, for example, that `100g butter` is bound to **a**, `two eggs` is bound to **b**, `100g flour` is bound to **c** and `100g sugar` is bound to **d**.

In OPM, roles are used to specify how artifacts are related to processes. Using the naïve mapping between PML *InferenceStep* and OPM *Process*, we see an indirect but meaningful connection between artifacts to inference steps through inference rules: inference rules are associated with inference steps and the *hasMetaBinding* property of inference steps can be used for artifact disambiguation if two or more artifacts of the same type play distinct roles in a rule specification.

4.4. Encoding of Accounts

4.4.1. OPM Handling of Accounts

In OPM, accounts are represented by the merging of two or more graphs where each original graph is represented by a different color. OPM knows the observer responsible for each account. The question in Use Case 5, however, is to state that one observer is endorsing another observer by referring to a statement of the other observer as a true description of some part of the provenance encoding. In other words, to say that `John` is leveraging `Joseph`'s description of a related event, i.e., that the same car ran the red light a minute before crashing into the post.

4.4.2. PML Handling of Accounts

We can consider an OPM *Account* to be a materialization of a PML justification. However, it is important to note that a PML justification is not a PML construct, but a PML concept. This means that PML does not have an element named justification in the PML ontology. Instead, in [2], *proof* is formally defined, which in current PML terms is called a *justification*[4]. A justification is a selection of a path in a provenance graph composed of inter-connected inference steps. In term of graphs, the provenance of an artifact is a graph of inference steps encoding multiple justifications, each defined as a tree in this graph of inference steps. PML justifications are derived from the provenance already encoded in more primitive concepts such as PML *Inference Steps* and *Node Sets*.

[4]PML used the term *justification* as a generalization of the term *proof*: every formally described justification is a proof but if the description of the justification is not complete enough to be, for example, formally verified, then that is a justification may not deserves the official label of being a proof.

In PML, each conclusion of a node set can have multiple accounts, and each distinct account is encoded as a new combination of inter-connected inference steps. This combination of inference steps allows PML to encode complex justifications for the car crash in Use Case 5. For example, compare the following two alternative justifications:

1. Justification 1: the driver of the car lost control of the car since the road signs at the place of the crashing were inappropriate since a tree was obstructing the view of the signs;
2. Justification 2: the driver lost control of the car for no reason.

In this case, there are justifications for the same car crashing. The *isConsequenctOf* property of the node set concluding the car crashing would have two *Inference Steps*, one for each justification. Justification 1 would probably be encoded as a new justification on its own, while justification 2 would be restricted to a single assertion of who said that the car lost control for no reason. It is possible that justification 1 would also have further variations implemented by other node sets with multiple inference steps.

The fact that PML justifications are derived from PML provenance encodings lead us to ask why OPM needs an *Account* construct and whether account information can be derived from provenance information already encoded in OPM concepts such as *Artifacts* and *Processes*.

4.5. Encoding of Provenance Identifiers

4.5.1. OPM Identifiers

In Use Case 6, Tania and John observe the same car crash. According to OPM, entity identifiers need to be the same for two entities to be the same. In this case, the crashes described by Tania and John cannot be the same since the identifier `the yellow car` is different than the identifier `the Honda Civic`. In general, it is likely that `John` and `Tania` mentioned in Use Case 6 would use distinct identifiers for the car that crashed. In practical terms, OPM's idealized vision of the world where provenance entities have one and only one unique identifier is unfeasible for a provenance language.

4.5.2. PML Identifiers

PML is a Semantic Web-based notation and as such relies of the use of URIrefs to identify its elements. In addition to providing representational primitives for use in encoding information content as a string, the PML ontology includes primitives supporting access to externally referenced content via *hasUrl*, which links to an online document, or *hasInfoSourceUsage*, which records when, where and by whom the information was obtained. This concept allows users to assign an URI reference to information. At the justification level, the PML ontology specifies that every Node Set has exactly one URI and is its unique identifier.

PML does not require provenance entities, e.g., an instance of PML *Person*, to have a unique identifier. For example, each person may have multiple URIs. In Use Case 6, PML accepts the fact that `the yellow car` and `the Honda Civic` can be distinct identifiers of the same entity. Eventually, the Ontology Web Language (OWL)'s [10] owl:sameAs construct can be used to encode the knowledge that `the yellow car` and `the Honda Civic` are the same entity. This capability of an entity to have more than one identifier is essential, for instance, to allow the gathering of multiple accounts about the car crash, e.g., in this case, one account from `John` and another from `Tania`.

5. Discussion

The use cases presented in Section 3 have the goal of supporting applications that rely on provenance information. In this section we describe how limitations on how to model provenance information affects the provenance support to those applications. Later, we compare the interoperability and simplicity of the OPM and PML encodings.

5.1. Provenance Applications

5.1.1. Attribution

Provenance expects to identify known sources used to assert an artifact. In an ideal world, if an artifact has been derived from other artifacts, provenance should allow applications to inspect, e.g., traverse, the derivation traces of artifacts until the provenance of every artifact is grounded on assertions attributed to known information sources. For the derivation traces, it is also relevant to know the authors/developers of the functionalities used to derive artifacts. Use Cases 1 shows that OPM does not have a notion of an entity capable of asserting information, e.g., a PML *Agent*, and thus cannot establish a relation between an artifact and the entity asserting this artifact. These entities are often called *sources*. Use Case 2 shows that OPM is not able to refer to information repositories or containers such as publications and databases. Thus, OPM cannot assign associate artifacts to those repositories. In other words, Use Cases 1 and 2 show that OPM does not have a way

of characterizing the information providers that deserve credit and are liable for their assertions.

Regarding derivations, Use Case 3 shows that OPM can assign a process execution (and thus the process itself) to OPM artifacts derived from the process execution. However, OPM does not have a mechanism to identify the developers of such processes or even more important, what are the key mechanisms (e.g., algorithms) executed by these processes. In PML, process mechanisms are formalized as inference rules and PML can store provenance information about the inference rules. For example, if a given process is a Fast Fourier Transformation, PML can identify the program as an inference engine, it can identify the authors of the inference engine, it can say that the engine implements a fast Fourier Transformation, and it can point to the publication (or publications) where the Fast Fourier Transformation algorithm is defined.

Finally, PMLs Source Usage concept is used to establish a proper authorship between artifacts and information containers, i.e., sources and agents. Further, Inference Web infrastructure is used to encode, discover and reuse source information [11].

5.1.2. Understanding

Artifacts (and information) are sometimes unexpected and surprising to their users. For example, a map may be unexpected when a person sees a mountain in the map in a place that the person believes there is no mountain. Other times, artifacts are presented as a result of a request, e.g., as a query, and the artifact result of the query may be considered unexpected, incomplete or inconsistent when compared against the query. For example, a cake may be a surprising answer for the question "what are the available drinks?" The provenance of the artifact is a key piece of information for understanding whether the artifact was correctly derived or that the artifact is an assertion and that the assertion was properly attributed to an information source. In general terms, provenance is the data used to create understanding about how artifacts came to be.

To support the generation of explanations from provenance, provenance traces need to explain entailment, assertions and both, i.e., multiple accounts. Entailment support in OPM is very limited since, as shown in the analysis of Use Case 3, OPM *Process* does not include a description of the process itself. In PML, process execution is encoded by *Inference Steps* that are associated with *Inference Rules*. The inference rules can be used in combination with specification languages and thus be used to provide rich entailment explanations. Furthermore, Use Case 4 shows that OPM does not provide a mechanism to explain how roles in causal dependencies can be used to explain the exact contribution of an artifact to a process and that for machine consumptions, just the labels of the roles may not be satisfactory as it may be for human consumption.

In the case of assertions, in addition to the attribution issues as discussed in Section 5.1.1, the full disclosure of how sources are used including the exact time and location are important pieces of provenance information. For example, it may be strange for one to say that "it is raining" when it is dry and sunny. In this case, the fact that "it is raining" was said yesterday when it was wet can help better understand artifact context. According to Use Cases 1 and 2, OPM does not have a way to encode the provenance encoded in PML Source Usage.

5.1.3. Reproducibility

The use of provenance to support reproducibility is a major concern for scientific artifacts. Provenance information is often used in computation-intensive scientific applications as a checkpoint mechanism to restart applications and workflows [12]. As a checkpoint, OPM has been successfully used to support scientific workflow executions. Reproducibility, however, is not restricted to the case above. For example, publications should include enough provenance information to allow results to be reproduced elsewhere. In this context, both justification data and metadata are required to support reproducibility of an artifact. It is necessary to know which tools were used but it is also necessary to know the algorithms implemented by these tools, which according to Use Case 3, cannot be accomplished with OPM. For example, in the sensor data, it is necessary to know the sensor maker, the accuracy and the conditions required to use a sensor. The sensor, according to Use Case 1, cannot be assigned to an artifact and thus the information about the sensor cannot be used to reproduce experiments encoded in OPM.

5.1.4. Trust

Trust recommendations are often computed from information about trust relations among agents [13, 14]. Provenance is an important element for trust computation since it identifies dependencies between artifacts (or information) and thus between agents asserting those artifacts [15, 16]. In other words, *attribution* as discussed in Section 5.1.1 is essential for trust.

In addition to attribution, we understand that one should not trust any information coming from a single source. If this assumption is correct, trust does need multiple account capability that both OPM and PML

provide. However, information coming from multiple sources may use distinct identifiers for common entities and that is not supported by OPM as shown in the analysis of Use Case 6. Furthermore, we need to accept that the knowledge about complex events may be far more extensive than anyone can know and that accounts need to leverage other accounts, a capability that OPM does not have according to Use Case 5.

5.2. On the Usefulness of Provenance Languages

Useful provenance languages support provenance applications and provenance interoperability while keeping the property of being easily adopted.

Interoperability is mainly achieved by a well-grounded specification supported by a solid theory. Interoperability is not just a syntax matter but of a comprehensive set of constraints in support of a proper semantics for each construct and construct property. The fact that the OPM specification is in English is already limiting since it allows multiple interpretations for provenance models. Further, the OPM specification itself is obscure due to the notions of "strong" and "weak" interpretations mentioned several times in the documentation. These are interpretation variations that we see being propagated to OPM implementations. For example, OPM has two serialization approaches in place, which makes it difficult for tools to effectively encode provenance in a format that claim to support interoperability. PML does not present many of the OPM challenges for interoperability. The fact that PML specification uses an ontology encoded in the standard OWL language limits the number of interpretations. The fact that PML is based on time-tested proof theory limits the capability of one drawing new and distinct interpretations for PML constructs.

OPM claims to be a high-level provenance model comprised of basic constructs. On the other hand, PML has a relatively large number of concepts required to meet most of the provenance requirements the PML community has identified so far. Considering OPM's relatively small number of concepts and a notation designed for human consumption, we have been inclined to accept OPM as an easier-to-understand-and-adopt provenance language. However, in light of the mapping discussed in this article, we are inclined to conclude that OPM's simplicity is misleading. There is no question that OPM's notation is more human-friendly than PML and that a large community can understand it. However, *OPM either does not have constructs in support of critical provenance applications supported by PML or it has more constructs for modeling provenance that are accomplished with fewer constructs in PML*. For example, OPM does not have support for assertions, sources, source usage, and inference rule as discussed in Use Cases 1 and 2. But to model temporal dependencies, OPM has five kinds of "causal" dependencies, i.e., "used", "was generated by", "was triggered by", "was derived from", "was controlled by" , while PML has basically the *hasAntecedentList* property of inference steps. Moreover, as described in Use Case 5, PML does not need to have an account construct and all the other additional constructs in support of accounts because PML assertions and derivations have been accounted for.

6. Conclusions

This article presented six small and fundamental provenance use cases that pose challenges for OPM, PML, and most of the available provenance languages, i.e., the languages listed in the PROV-XG state of the art report on provenance. From the description of these use cases in Section 3, we discussed possible provenance encodings in OPM and PML as described in Section 4. An immediate goal of these encodings was to understand the similarities between OPM and PML. Another goal was to identify limitations OPM and PML have encoding provenance scenarios and using the encodings to answer questions about the scenarios. Later in Section 5, we contrasted the identified limitations on the use of OPM and PML encodings to support attribution, understanding, reproducibility, trust and interoperability.

From the encodings, we see that both notations have their limitations and that probably none of them can fully support interoperability in its current state. OPM is easier for humans to understand and to initiate some use. OPM terms are derived from scientific workflows and are relatively intuitive for human understanding. OPM, however, does not support assertions, has limitations on supporting derivations including no mechanism to describe process specifications, has a rigid strategy for handling provenance element identifiers, no mechanism is provided for provenance users to specify and use the semantics for these roles, and provenance accounts are not scalable. PML has support for assertions, derivations, derivation roles and accounts. PML, however, is difficult to learn and is not user-friendly. PML terms are not intiuive for humans since they are derived from proof theory, which is not easily mapped into day-by-day activities.

Revisiting PROV-XG's goal of creating a state-of-the art understanding of technologies in the area of provenance for the Semantic Web, we understand that

the modeling of the use cases discussed in this article exposes a number of substantial difficulties when using provenance notations. These difficulties may be lost in the discussion of three exciting but complex use cases under consideration by PROV-XG: Disease Outbreak, Business Contract and News Aggregator scenarios. More important, however, is our concern over the use of OPM as the target language for establishing an understanding of the state-of-the-art: mappings to OPM will not include substantial areas of concern for provenance and thus impact the capability of provenance to support a broad range of provenance applications.

Acknowledgements

The authors would like to express their gratitude to many people for sharing their time and effort with us allowing us to better understand OPM and PML. From the PML team, we would like to thank Timothy Lebo, Li Ding and Deborah McGuinness for all their initial discussion that led us to establish some of the use cases used in this article. From the OPM team, we thank Paul Groth, Simon Mile and Luc Moreau for the useful conversations during IPAW that led us to understand some aspects of OPM that are more challenging to understand from the OPM specification itself. We also thank Jim Myers for his effort to map PML into OPM that gave us useful insight about OPM and PML mappings from an OPM perspective. Finally, we would like to thank the students at the Trust Laboratory at the University of Texas at El Paso for their useful comments during a couple of meetings we had to review and discuss both OPM and PML.

Appendix A. Formal Specification of Inference Rules

PML inference rules can use InferenceML schemas (or any other formal specification language) to state such transformations [17]. In the particular case of InferenceML, we define a schema to be a pattern, which is any expression of Simple Common Logic (SCL) [18] in which some lexical items of a certain grammatical category (typically things like Sent(ence), Name, Rel(ation symbol) etc.) have been replaced by a schematic variable (or meta-variable), paired with a set of syntactical conditions, which record the corresponding type of each meta-variable (and possibly other conditions, described later). So, an SCL schema has the general form

```
pattern ;; syntax-conditions
```

where:

- the pattern is an SCL expression with schematic variables. (InferenceML also uses several other categories of meta-variable, noted below.) In order to distinguish schematic variables from SCL text in a pattern, we enclose literal SCL text in single quotes and leave schematic variables unquoted. This syntax is intended to indicate any piece of SCL core syntax text that can be obtained by substituting suitable lexical items for the schematic variables and concatenating the fragments in the order shown.
- the syntax-conditions are specialized expressions specifying the types of the schematic variables. These are written in the SCL core syntax using a special vocabulary, but the meta-variables are treated as normal variables.

For example,

```
(implies p q  );; (Sent p q)
```

is a schema that matches any SCL implication sentence. Here, `p` and `q` are schematic variables for SCL sentences. The syntax condition predicate `Sent` takes any number of arguments and is true exactly when the arguments are sentences.

Appendix B. On the Use of SPARQL to Query PML

Figure A.8 shows a SPARQL query used to retrieve the information sources that have contributed to a given artifact called `QuickLook.owl`. The SPARQL query relies on the use of the `isConsequentOf` property of a node set that connects a node set to other node sets through inference steps. By traversing the node sets, the query reaches all the nodes that were used to derive `QuickLook.owl`. In this case, a list of these node set URIs is assigned to the query variable `?x`. For each value of `?x`, the query further inspects the property *hasSourceUsage* of each node set' s inference step. For each value of `?x` is assigned a list of Source Usage URIs is assigned to the query variable `?z`. Finally, for each value of `?z`, the query inspects the source usage to identify the URIs of the sources that are assigned to the query variable `?z`.

The SPARQL query in Figure A.8 demonstrates that PML can be used to answer complex provenance queries with the use of standard Semantic Web tools and without any addition implementation effort.

```
PREFIX ql: <http://astronomy/solar/QuickLook.owl#>
SELECT ?w
WHERE { ql:answer pmlj:isConsequentOf ?x .
        ?x pmlj:hasSourceUsage ?z .
        ?z pmlp:hasSource ?w .}
```

Figure A.8: A SPARQL query used to retrieve all the known sources of an artifact from a PML encoding.

References

[1] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, J. Van den Bussche, The Open Provenance Model — Core Specification (v1.1), Future Generation Computer Systems.

[2] P. Pinheiro Da Silva, D. L. McGuinness, R. Fikes, A Proof Markup Language for Semantic Web Services, Information Systems 31 (4-5) (2006) 381–395.

[3] O. Hartig, J. Zhao, Using Web Data Provenance for Quality Assessment, in: Proceedings of the 1st Int. Workshop on the Role of Semantic Web in Provenance Management (SWPM) at ISWC, Washington, USA, 2009.

[4] PREservation Metadata: Implementation Strategies Working Group, Data Dictionary, version 1.0. , http://www.oclc.org/research/projects/pmwg/premis-dd.pdf.

[5] A. S. Troelstra, H. Schwichtenberg, Basic Proof Theory, 2nd Edition, Cambridge University Press, Cambridge, UK, 2000.

[6] Dictionary.com, http://dictionary.reference.com/browse/assertion, as accessed on September 19, 2010 at 2pm MDT.

[7] J. W. Murdock, P. Pinheiro Da Silva, D. Ferrucci, C. Welty, D. L. McGuinness, Encoding Extraction as Inferences, in: Proc. of AAAI Spring Symposium on Metacognition on Computation, AAAI Press, Stanford University, USA, 2005, pp. 92–97.

[8] R. Fikes, D. Ferrucci, D. Thurman, Knowledge Associates for Novel Intelligence (KANI), Tech. Rep. KSL-03-16, Knowledge Systems Laboratory, Stanford University, Stanford, CA, USA (October 2003).

[9] S. Russel, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, Upper Saddle River, NJ, USA, 1995.

[10] D. L. McGuinness, F. van Harmelen, OWL Web Ontology Language Overview, Tech. rep., World Wide Web Consortium (W3C), proposed Recommendation (December 9 2003). URL `http://www.w3.org/TR/owl-features/`

[11] D. L. McGuinness, P. Pinheiro da Silva, Explaining Answers from the Semantic Web:The Inference Web Approach, Journal of Web Semantics 1 (4) (2004) 397–413.

[12] S. Bowers, T. McPhillips, S. Riddle, M. Anand, B. Ludaščher, Kepler/pPOD: Scientic Workow and Provenance Support for Assembling the Tree of Life, in: International Provenance and Annotation Workshop (IPAW'08), Salt Lake City, Utah, 2008.

[13] S. D. Kamvar, M. T. Schlosser, H. Garcia-Molina, The eigentrust algorithm for reputation management in p2p networks, in: Proceedings of the 12th international conference on World Wide Web, 2003.

[14] R. Guha, R. Kumar, P. Raghavan, A. Tomkins, Propagation of trust and distrust, in: Proceedings of the 13th international conference on World Wide Web, ACM Press, 2004, pp. 403–412.

[15] Y. Gil, D. Artz, Towards content trust of web resources, Journal of Web Semantics 5 (4).

[16] I. Zaihrayeu, P. Pinheiro Da Silva, D. L. McGuinness, IWTrust: Improving User Trust in Answers from the Web, in: Proceedings of 3rd International Conference on Trust Management (iTrust2005), Springer, Paris, France, 2005, pp. 384–392.

[17] P. Pinheiro Da Silva, P. Hayes, D. L. McGuinness, R. Fikes, P. Deshwal, Towards Checking Hybrid Proofs, Tech. Rep. KSL-05-01, Knowledge Systems Laboratory, Stanford University, Stanford, CA, USA (2005).

[18] M. Altheim, B. Anderson, P. Hayes, C. Menzel, J. F. Sowa, T. Tammet, SCL: Simple Common Logic, http://www.ihmc.us/users/phayes/CL/SCL2004.htm (2005).