A Proposal for Handling Collections in the Open Provenance Model

Paul Groth, Simon Miles, Paolo Missier, Luc Moreau

June 4, 2009

1 Introduction

The Open Provenance Model (OPM) is a community-driven data model for Provenance that is designed to support inter-operability of provenance technology. The original OPM specification [1] was refined into [2] following the first OPM workshop[1].

During this workshop, an agreement emerged that there needed to be a mechanism in OPM to deal with collections. Specifically, it was felt that it would be useful to express that artifacts belong to input lists used by a process, or that artifacts were contained in lists generated by a process. There was debate about whether there should be extra syntax to achieve this. It was demonstrated that collections could be modeled in the current OPM syntax explicitly using the wasDerivedFrom edge and accounts. However, there were concerns about the scalability of this approach, in terms of the amount of documentation needed to be produced for collections with many elements, and verbosity, in terms of inability to easily understand the relationship between collections. To acknowledge the importance of collections, a whole section on collections was introduced in OPM 1.01 [2], though it did not address the aforementioned concerns.

An important suggestion was to introduce *intensional properties* through annotations to the graph. This suggestion however was felt to go against the notion that OPM represents an *explicit account* of provenance. Therefore, it was suggested that that rules may be used to explain how a refined description of the details of a collection can be derived automatically from a higher-level description of the same collection. Another suggestion was to introduce a set of subclasses of process from which one can infer the wasDerivedFrom relationships using rules.

The purpose of this document is to put forward a proposal to represent collections in OPM. Our approach is fully compatible with OPM as it specialises OPM with a *Collection Profile* by which operations at the level of collections (construction, accessor, mutation, transformation, communication) can be expanded

[1] http://twiki.ipaw.info/bin/view/Challenge/OpenProvenanceModelWorkshop

into operations at the level of elements of collections; the result of this expansion remains a full OPM graph.

Such a collection profile could be used in two different ways. Either OPM graphs containing collection descriptions would be expanded automatically, and reasoning would take place on the expanded graph (as per defined by OPM); or, reasoners would be aware of this profile, and would be able to reason on the high level descriptions directly.

This paper is organised as follows. First we identify some requirements that we aim to address (Section 2). We then review the purpose of profiles in OPM (Section 3). We then discuss the collection profile in Section 4, and we illustrate its application on some examples (Section 5).

2 Requirements

In this section, we enumerate the requirements and/or issues that have been identified with respect to collections.

Requirement 1 (Intensional representation) *We need to be able to determine the causal connection between elements of collections output from and input to processes, without modelling all those connections explicitly in the OPM graph.*

Requirement 2 (Communications) *When communicating a collection of items to a remote service (or procedure), it is necessary to be able to know that one item in the received collection is the same as the corresponding item in the sent collection. In Figure 1, collection (a1,a2,a3) was exchanged between two services. How can the recipient know that the first element of this collection (referred to as a1') is the same as a1?*

Requirement 3 (Constructor/Mutator) *We need to be able to model the effect of mutator functions in OPM. How can we represent the provenance of* `b` *produced by the following program in OPM?*

```
Item a = new Item();
List b= new List();
b.add(a);
a.setF(g);
```

Indeed, the statement `a.setF(g)` *modifies* `a`*, and therefore* `a`*'s provenance now includes a reference to* `g`*. Given that* `b`*'s provenance contains* `a`*, how is the modification affecting* `b`*'s provenance?*

Requirement 4 (Inputs or outputs) *We need to be able to address Requirement 1 even when there are not collection artifacts to be modelled, but simply processes with large numbers of input or output artifacts*

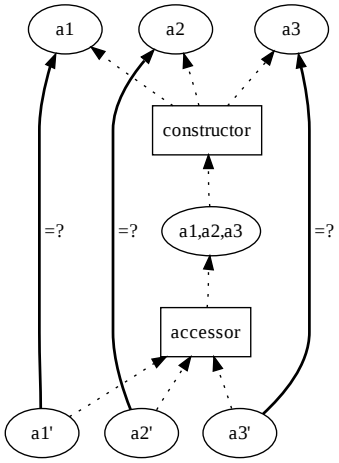
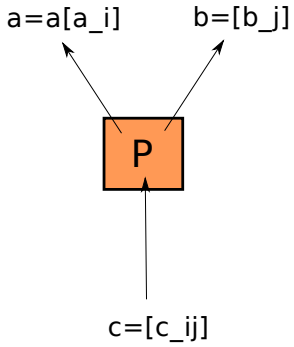


Figure 1: Collections and Communications

Requirement 5 (Workflow constructs) *OPM must be able to express some high-level constructs supported by workflow engines in a convenient manner. For instance in Taverna, in the following picture, the process P takes two input lists and produces an ouput list. The list* `c` *is defined as* `(map p (a x b))`*. So, each* `c_ij=p(a_i,b_j)`*; Can we express such dependencies in OPM in a compact way like some workflow languages do?*



Requirement 6 *OPM should avoid redefining yet again a new set of collections. Instead, we should reuse an existing set. It is suggested that a minimalistic approach, as followed by JSON, may be suitable, for instance, supporting ordered lists and associative arrays.*

3 Design Philosophy of OPM Specialisation by Profile

This document does not seek to define what an OPM profile is but instead makes some assumptions about what an OPM profile specification might require. It is assumed that:

1. The notion of OPM profile has been defined elsewhere (maybe refer to WS-I or OGSA profiles).
2. Profiles identify specific pattern of usages of OPM constructs and associated semantics.
3. Profiles convert without loss of information to regular OPM. The semantics defined in the profile should still be valid in the resulting OPM graph.
4. Reasoning can be done on the raw OPM graph, or directly on the profile.

4 Proposal of a Collection Profile in OPM

The proposal for handling collections in OPM is a profile consisting of specific types of artifacts, processes and edges, and accounts defined in terms of these types, as follows.

4.1 Collections and Artifacts

OPM defines an artifact as an entity (specifically a node) allowed in OPM graphs. We assume here that artifacts can be typed in OPM: by this we mean that artifacts can be declared to belong to a subtype of the toplevel type Artifact. Such typing information can be seen as an annotation on the original Artifact entity.

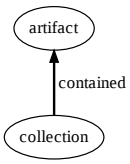
We introduce a new type of artifact: a collection. Collections are generic types, and therefore need to identify the type of artifacts they contain. The collection type is abstract, it is subclassed by the concrete collection types supported by OPM (cf. Requirement 6). In the following, we consider ordered lists.

Artifact	<	Artifact	<	Artifact
type : *Artifact*		*type* : *Collection*		*type* : *Orderedlist*
		contains : ...		*contains* : ...

Likewise, OPM edges are typed. Their type is represented graphically as a label attached to them.

4.2 Structural level

We introduce the containment relationship between collections and the artifacts they contain. Such containment relationship is expressed by a wasDerivedFrom edge of type CONTAINED, which means for a collection to exist in a given state, it was dependent on the artifacts it contained.



4.3 Constructor and Accessor Level

We introduce a level of description that is concerned with how collections are constructed and accessed. Collections are generated by constructors, which are processes with type CONSTRUCTOR, which used several artifacts to construct such collections. As a result, in an OPM graph, a collection is connected to the artifacts that were used for its construction, by a wasDerivedFrom edge of type CONTAINED.

Symmetrically, a process of type ACCESSOR used a collection in order to generate (some of the) collection constituents. Each artifact generated by an accessor was derived from this collection; such a kind of dependency is captured by a WASPARTOF edge.

The edges WASPARTOF and CONTAINED may appear to be symmetrical from a collection structural viewpoint: an artifact WASPARTOF a collection and a collection CONTAINED an artifact. However, these relationships are subtype of wasDerivedFrom, expressing a causal dependency. So, when a collection was constructed from some artifacts, the collection is the effect, and the artifacts the causes; we then use a CONTAINED edge. When an artifact was extracted from a collection, the effect is the artifact and the cause the collection; so we use a WASPARTOF edge.

Finally, an accessed artifact is the same as the corresponding artifact that was used by the collection constructor (which is captured by the WASIDENTICALTO edge). This description is illustrated by Figure 2.

In practice, with the collection profile, it is not expected that a collection, its elements used at construction time, its elements generated at access time be all extensionally described in an opm graph. Instead, under the Collection profile, the following inference is allowed.

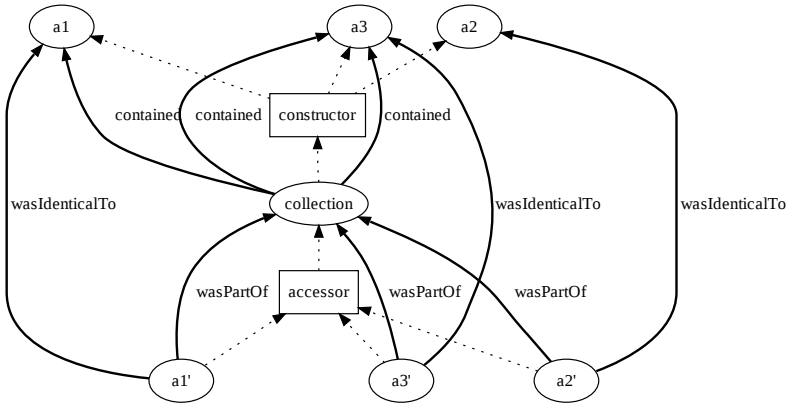


Figure 2: Constructor Level

Inference 1 (Accessor/Constructor) *Given a process p_c of type* CONSTRUCTOR*, a set of used artifacts a_i, and a generated collection c, we can make the following inferences:*

1. *There exists* CONTAINED *edges from c to each a_i to mark that the collection contained each artifact a_i used by the constructor p_c.*

2. *For each process p_a that has type* ACCESSOR *and that uses c, for each each a_i, one may infer an artifact a_i' that was generated by p_a, that is connected to c by a* WASPARTOF *relation, and that is connected by a* WASIDENTICALTO *relation to a_i.*

Symmetrically, the presence of artifacts used by a constructor can be inferred from the presence of generated artifacts by an accessor of a collection.

4.4 Communication level

Similarly to the constructor level, a description captures the details of how collections are communicated. We assume here that an edge WASCOPYOF encapsulates the facts that an artifact was marshalled, communicated, and unmarshalled. So, whenever a collection was a copy of another (by means of communications), the constituents of a received collection are also the same as the corresponding artifacts in the sent collection. We express that they were communicated, and hence, were copied, by the WASCOPYOF relation (cf. Figure 3).

In practice, with the collection profile, we do not expect all these elements and artifacts to be specified extensionaly. Instead, they can be inferred with the following inference rule.

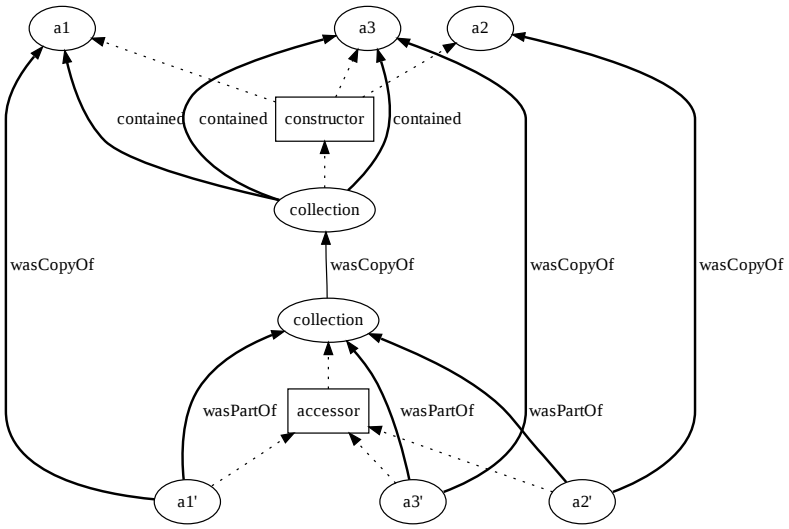


Figure 3: Communication Level

Inference 2 (Communication) *Given a process p_c of type* CONSTRUCTOR*, a set of used artifacts a_i, a generated collection c, and a collection c' resulting from communication, (c' was a copy of c), we can make the following inferences:*

1. *There exists edges* CONTAINED *from c to each a_i to mark that the collection contained each artifact a_i used by the constructor p_c.*

2. *For each process p_a that has type* ACCESSOR *and used c', for each each a_i, one may infer an artifact a'_i that was generated by p_a, that is connected to c by a* WASPARTOF *relation, and that is connected by a* WASCOPYOF *relation to a_i.*

Symmetrically, an inference rule can also derive the presence of sent artifacts from artifacts that were part of a received collection.

4.5 Operation level

This level describes the operations that are performed on collections. The semantics of these operations is expressed, by means of inferences, in terms of the operations that are performed on the collection elements. These inferences are expressed in plain English. It is not suggested at this stage that a formal rule language should be used to encode such rules. Indeed, such languages are still the focus of active research, and adopting an evolving, unstable inference language may hinder the specification of a collection profile.

A complete profile for collections may define inferences for several collection operations. In Figure 4, we illustrate the kind of inference that prevails for a mapping operation, and specify it as follows.

Inference 3 (Mapping) *Let C_1 and C_2 be two collections, such that C_2 is connected to C_1 by a wasDerivedFrom edge, with subtype* WASMAPPEDFROM *and parameter f (denoting a type of derivation). Let $a_{1,i}$ be an artifact member of C_1. The collection-level inference allows us to derive an artifact and two edges. There exists an artifact $a_{2,i}$ that C_2 contained such that $a_{2,i}$ was derived from $a_{1,i}$, with an edge of type f.*

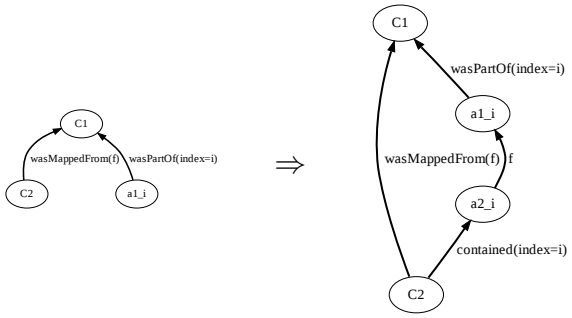


Figure 4: Mapping inference

A symmetric inference allows us to infer the presence of an artifact $a_{1,i}$ in C_1 from any artifact $a_{2,i}$ in C_2.

Is it worth noting that the inferred pattern (on the right of Figure 4) can be expressed more concisely and without loss of information as either (the pattern on the left of Figure 4) or the symmetric pattern, not shown in the figure.

4.6 Summary

In summary, our proposal regards collections as artifacts (of type collection). Different operations are supported for collections: construction, access to elements, communication, and transformation (such as mapping). Rules (expressed in English) allow us to infer dependencies related to collection elements from the collections themselves. The collection profile and its inference rules offer the benefit of more concise representation since all transformations into plain OPM and back are lossless.

5 Examples

5.1 Collection: constructors and updates

We revisit the example illustrating Requirement 3. We assume that collection `a` contains two fields `F` and `G`. Collection `a` is added to collection `b`, which also contains an artifact `c`.

We assume that the following lines of code have been executed.

```
Item a = new Item();  // collection with fields F and G
Item c = new Item();  // an artifact
List b = new List();  // list
b.add(c);             // add c to the list
a.setF(f);            // update f
```

Figure 5 displays the OPM graph, showing the provenance of all entities following two processes (`setG` and `append`), corresponding to the following instructions.

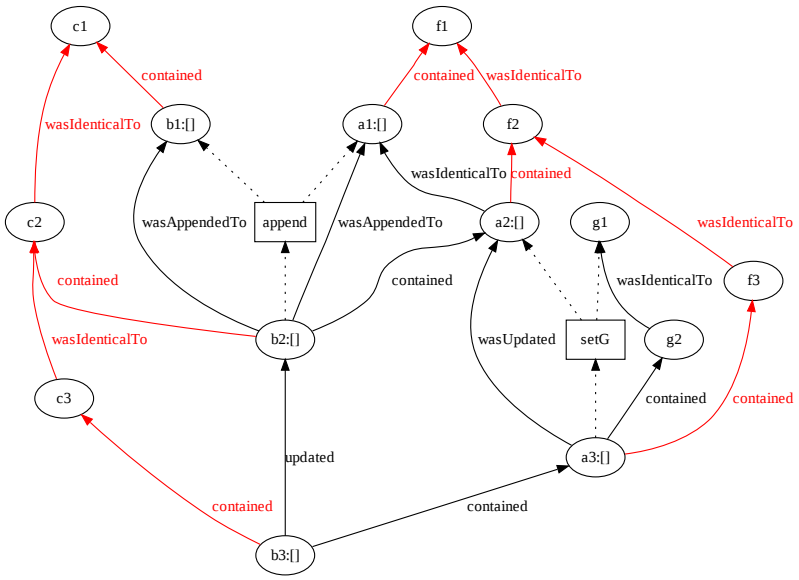
```
b.add(a);             // add a to the list, keep c
a.setG(g);            // update g, leave f unchanged
```

Figure 5 shows how `c` (with snapshots `c1`, `c2`, `c3`) and `f` (with snapshots `f1`, `f2`, and `f3`) remain unchanged (and are respectively contained in `b` and `a`). The dependencies in the red account can be inferred.

5.2 Collection: map

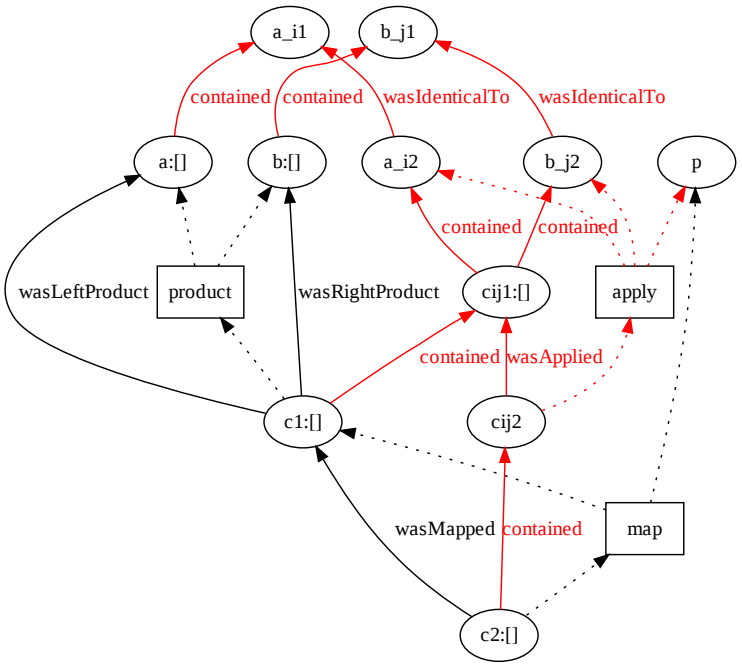
We revisit the example illustrating Requirement 5. A function `p` is applied to the elements of the cartesian product of two collections `a` and `b`.

The representation as an OPM graph is illustrated in Figure 6. In black, we see a description at the collection level, very similar to the workflow specification. In red, we see the representation that can be inferred for elements of these collections.



a1	The collection named a in its initial state, not yet added to collection b, with field F set to f1 and field G not set.
f1	The value of artifact a1's F field.
b1	The collection named b in its initial state, containing only value c1.
c1	The value contained in collection b1.
b2	The collection named b after a2 has been added to it.
a2	The collection named a in the state of being within collection b, with field F set to f2 and field G not set, identical in value to a1.
f2	The value of artifact a2's F field, identical in value to f1.
c2	The value contained in collection b2, identical in value to c1.
g1	The value of variable g prior to insertion into a collection.
a3	The collection named a in the state of being within collection b, with field F set to f3 and field G set to g2.
f3	The value of artifact a3's F field, identical in value to f2.
g2	The value of variable g when inserted in collection a3, identical in value to g1.
b3	The collection named b containing values c3 and a3.
c2	The value contained in collection b3, identical in value to c2.

Figure 5: Example1



`a`	A collection containing values `a_i1`, for each index `i`.
`b`	A collection containing values `b_j1`, for each index `j`.
`a_i1`	The value at the `i`'th index of collection `a`.
`b_j1`	The value at the `j`'th index of collection `b`.
`c1`	The set of cartesian products of `a_i2` and `b_j2`, for each `i` and `j`.
`cij1`	The cartesian product of `a_i2` and `b_j2`.
`a_i2`	The value extracted from the `i`'th index of collection `a`, identical in value to `a_i1`, when part of the product `cij1`.
`b_j2`	The value extracted from the `j`'th index of collection `b`, identical in value to `b_j1`, when part of the product `cij1`.
`p`	The function applied to pairs of values by the `apply` process.
`cij2`	The result of applying `p` to the pair of values `a_i2` and `b_j2`.
`c2`	The collection produced by applying `p` to all pairs of values in collection `c1`.

Figure 6: Example 2

6 URI Instantiation of Identifiers

For those systems which represent common vocabulary as URIs, we supply the following mapping.

OPM Name	URI
Relations	
CONTAINED	`http://openprovenance.org/collections#contained`
WASPARTOF	`http://openprovenance.org/collections#wasPartOf`
WASIDENTICALTO	`http://openprovenance.org/collections#wasIdenticalTo`
WASCOPYOF	`http://openprovenance.org/collections#wasCopyOf`
WASMAPPEDFROM	`http://openprovenance.org/collections#wasMappedFrom`
Processes	
CONSTRUCTOR	`http://openprovenance.org/collections#constructor`
ACCESSOR	`http://openprovenance.org/collections#accessor`
Roles	

7 Conclusion

In this document, we have put forward a proposal for an OPM profile that allows us to express operations related to collections, namely construction, copying, updating, accessing, and transforming. Adopting such a profile will promote inter-operability between systems.

We note that we chose to model a set of artifacts in a collection in the OPM graph because it has meaning (such a collection does exist in our application) and because we want to be able to derive the provenance of the collection as a whole. The same would also apply to a meaningful grouping of processes, e.g. all the actions by a single actor, all the processes enacted by a single project team etc. A future revision of this profile will consider collections of processes.

This document is a starting point for a fully specified Collections Profile. Such a profile would contain more information about the collection types supported, a more comprehensive set of inferences for operations on collections, and would be compatible with the community's definition of a profile.

References

[1] Luc Moreau, Juliana Freire, Joe Futrelle, Robert E. McGrath, Jim Myers, and Patrick Paulson. The open provenance model (v1.00). Technical report, University of Southampton, December 2007.

[2] Luc Moreau (Editor), Beth Plale, Simon Miles, Carole Goble, Paolo Missier, Roger Barga, Yogesh Simmhan, Joe Futrelle, Robert McGrath, Jim Myers,

Patrick Paulson, Shawn Bowers, Bertram Ludaescher, Natalia Kwasnikowska, Jan Van den Bussche, Tommy Ellkvist, Juliana Freire, and Paul Groth. The open provenance model (v1.01). Technical report, University of Southampton, July 2008.

A Appendix: Example of serialisation

This appendix contains the XML serialisation of the OPM graph of Figure 6. Type annotations on edges are made explicit with the `opmext` namespace.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<opm:opmGraph xmlns:opm="http://openprovenance.org/model/v1.01.a"
              xmlns:opmext="http://openprovenance.org/model/extension/v1.01.a"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://example.com/">
    <opm:accounts>
        <opm:account id="orange"/>
        <opm:account id="black"/>
    </opm:accounts>
    <opm:processes>
        <opm:process id="x">
            <opm:account id="black"/>
            <opm:value xsi:type="xsd:string">product</opm:value>
        </opm:process>
        <opm:process id="map">
            <opm:account id="black"/>
            <opm:value xsi:type="xsd:string">map</opm:value>
        </opm:process>
        <opm:process id="apply">
            <opm:account id="black"/>
            <opm:value xsi:type="xsd:string">apply</opm:value>
        </opm:process>
    </opm:processes>
    <opm:artifacts>
        <opm:artifact id="a">
            <opm:value xsi:type="xsd:string">a:[]</opm:value>
            <opm:account id="black"/>
        </opm:artifact>
        <opm:artifact id="a_i1">
            <opm:value xsi:type="xsd:string">a_i1</opm:value>
            <opm:account id="black"/>
        </opm:artifact>
        <opm:artifact id="a_i2">
            <opm:value xsi:type="xsd:string">a_i2</opm:value>
            <opm:account id="black"/>
        </opm:artifact>
        <opm:artifact id="b">
            <opm:value xsi:type="xsd:string">b:[]</opm:value>
            <opm:account id="black"/>
        </opm:artifact>
        <opm:artifact id="b_j1">
            <opm:value xsi:type="xsd:string">b_j1</opm:value>
            <opm:account id="black"/>
        </opm:artifact>
        <opm:artifact id="b_j2">
            <opm:value xsi:type="xsd:string">b_j2</opm:value>
            <opm:account id="black"/>
        </opm:artifact>
        <opm:artifact id="c">
```

```
        <opm:value xsi:type="xsd:string">c1:[]</opm:value>
        <opm:account id="black"/>
    </opm:artifact>
    <opm:artifact id="cij1">
        <opm:value xsi:type="xsd:string">cij1:[]</opm:value>
        <opm:account id="black"/>
    </opm:artifact>
    <opm:artifact id="cij2">
        <opm:value xsi:type="xsd:string">cij2</opm:value>
        <opm:account id="black"/>
    </opm:artifact>
    <opm:artifact id="pp">
        <opm:value xsi:type="xsd:string">p</opm:value>
        <opm:account id="black"/>
    </opm:artifact>
    <opm:artifact id="c2">
        <opm:value xsi:type="xsd:string">c2:[]</opm:value>
        <opm:account id="black"/>
    </opm:artifact>
</opm:artifacts>
<opm:agents/>
<opm:causalDependencies>
    <opm:used>
        <opm:effect id="x"/>
        <opm:role value="left"/>
        <opm:cause id="a"/>
        <opm:account id="black"/>
    </opm:used>
    <opm:used>
        <opm:effect id="x"/>
        <opm:role value="right"/>
        <opm:cause id="b"/>
        <opm:account id="black"/>
    </opm:used>
    <opm:used>
        <opm:effect id="map"/>
        <opm:role value="collection"/>
        <opm:cause id="c"/>
        <opm:account id="black"/>
    </opm:used>
    <opm:used>
        <opm:effect id="map"/>
        <opm:role value="function"/>
        <opm:cause id="pp"/>
        <opm:account id="black"/>
    </opm:used>
    <opm:used>
        <opm:effect id="apply"/>
        <opm:role value="left"/>
        <opm:cause id="a_i2"/>
        <opm:account id="orange"/>
    </opm:used>
    <opm:used>
        <opm:effect id="apply"/>
        <opm:role value="right"/>
        <opm:cause id="b_j2"/>
        <opm:account id="orange"/>
    </opm:used>
    <opm:used>
        <opm:effect id="apply"/>
        <opm:role value="fun"/>
        <opm:cause id="pp"/>
        <opm:account id="orange"/>
    </opm:used>
    <opm:wasGeneratedBy>
```

```
    <opm:effect id="c"/>
    <opm:role value="product"/>
    <opm:cause id="x"/>
    <opm:account id="black"/>
</opm:wasGeneratedBy>
<opm:wasGeneratedBy>
    <opm:effect id="c2"/>
    <opm:role value="result"/>
    <opm:cause id="map"/>
    <opm:account id="black"/>
</opm:wasGeneratedBy>
<opm:wasGeneratedBy>
    <opm:effect id="cij2"/>
    <opm:role value="result"/>
    <opm:cause id="apply"/>
    <opm:account id="orange"/>
</opm:wasGeneratedBy>
<opm:wasDerivedFrom xsi:type="opmext:NamedWasDerivedFrom">
    <opm:effect id="c"/>
    <opm:cause id="a"/>
    <opm:account id="black"/>
    <opmext:type>wasLeftProduct</opmext:type>
</opm:wasDerivedFrom>
<opm:wasDerivedFrom xsi:type="opmext:NamedWasDerivedFrom">
    <opm:effect id="c"/>
    <opm:cause id="b"/>
    <opm:account id="black"/>
    <opmext:type>wasRightProduct</opmext:type>
</opm:wasDerivedFrom>
<opm:wasDerivedFrom xsi:type="opmext:NamedWasDerivedFrom">
    <opm:effect id="a_i2"/>
    <opm:cause id="a_i1"/>
    <opm:account id="orange"/>
    <opmext:type>http://openprovenance.org/collections#wasIdenticalTo</opmext:type>
</opm:wasDerivedFrom>
<opm:wasDerivedFrom xsi:type="opmext:NamedWasDerivedFrom">
    <opm:effect id="b_j2"/>
    <opm:cause id="b_j1"/>
    <opm:account id="orange"/>
    <opmext:type>http://openprovenance.org/collections#wasIdenticalTo</opmext:type>
</opm:wasDerivedFrom>
<opm:wasDerivedFrom xsi:type="opmext:NamedWasDerivedFrom">
    <opm:effect id="a"/>
    <opm:cause id="a_i1"/>
    <opm:account id="orange"/>
    <opmext:type>http://openprovenance.org/collections#contained</opmext:type>
</opm:wasDerivedFrom>
<opm:wasDerivedFrom xsi:type="opmext:NamedWasDerivedFrom">
    <opm:effect id="b"/>
    <opm:cause id="b_j1"/>
    <opm:account id="orange"/>
    <opmext:type>http://openprovenance.org/collections#contained</opmext:type>
</opm:wasDerivedFrom>
<opm:wasDerivedFrom xsi:type="opmext:NamedWasDerivedFrom">
    <opm:effect id="cij1"/>
    <opm:cause id="a_i2"/>
    <opm:account id="orange"/>
    <opmext:type>http://openprovenance.org/collections#contained</opmext:type>
</opm:wasDerivedFrom>
<opm:wasDerivedFrom xsi:type="opmext:NamedWasDerivedFrom">
    <opm:effect id="cij1"/>
    <opm:cause id="b_j2"/>
    <opm:account id="orange"/>
    <opmext:type>http://openprovenance.org/collections#contained</opmext:type>
</opm:wasDerivedFrom>
```

```
        <opm:wasDerivedFrom xsi:type="opmext:NamedWasDerivedFrom">
            <opm:effect id="c"/>
            <opm:cause id="cij1"/>
            <opm:account id="orange"/>
            <opmext:type>http://openprovenance.org/collections#contained</opmext:type>
        </opm:wasDerivedFrom>
        <opm:wasDerivedFrom xsi:type="opmext:NamedWasDerivedFrom">
            <opm:effect id="c2"/>
            <opm:cause id="cij2"/>
            <opm:account id="orange"/>
            <opmext:type>http://openprovenance.org/collections#contained</opmext:type>
        </opm:wasDerivedFrom>
        <opm:wasDerivedFrom xsi:type="opmext:NamedWasDerivedFrom">
            <opm:effect id="cij2"/>
            <opm:cause id="cij1"/>
            <opm:account id="orange"/>
            <opmext:type>wasApplied</opmext:type>
        </opm:wasDerivedFrom>
        <opm:wasDerivedFrom xsi:type="opmext:NamedWasDerivedFrom">
            <opm:effect id="c2"/>
            <opm:cause id="c"/>
            <opm:account id="black"/>
            <opmext:type>wasMapped</opmext:type>
        </opm:wasDerivedFrom>
    </opm:causalDependencies>
</opm:opmGraph>
```