Workflows to Open Provenance Graphs, round-trip

Paolo Missier*, Carole Goble*

*School of Computer Science, The University of Manchester, Oxford Rd., Manchester, UK

Abstract

The Open Provenance Model is designed to capture relationships amongst data values, and amongst processors that produce or consume those values. While OPM graphs are able to describe aspects of a workflow execution, capturing the structure of the workflows themselves is understandably beyond the scope of the OPM specification, since the graphs may be generated by a broad variety of processes, which may not be formal workflows at all. In particular, OPM does not address two questions: firstly, whether for any OPM graph there exists a plausible workflow, in some model, which could have generated the graph. And secondly, which information should be captured as part of an OPM graph that is derived from the execution of some known type of workflow, so that the workflow structure and the execution trace can both be inferred back from the graph. Motivated by the need to address the Third Provenance Challenge using Taverna workflows and provenance, in this paper we explore such notion of lossless-ness of OPM graphs relative to Taverna workflows. For the first question, we show that Taverna is a suitable model for representing plausible OPM-generating processes. For the second question, we show how augmenting OPM with two types of annotations makes it lossless with respect to Taverna. We support this claim by presenting a two-way mapping between OPM graphs and Taverna workflows.

1. Introduction

The Open Provenance model, or OPM [13], specifies a set of properties that describe the relationships amongst data values, called artifacts, as they undergo a sequence of transformations through some composition of processors. An instance of the model is a graph consisting of a collection of statements that involve OPM properties, for example \( a \) wasGeneratedBy \( p \) where \( a \) is an artifact and \( p \) is a processor. These causal relationships, as they are known in OPM, may be derived in a variety of ways. Here we are going to focus exclusively on OPM graphs that describe data transformations through a workflow, that is, a formal specification of the orchestrated execution of software components (the processors) that operate on and transform data (the artifacts).

Initiatives such as the provenance challenges [8, 12] have over the years been casting the OPM as a core causal graph model for exchanging workflow provenance, i.e., a detailed trace of an execution [6], that may have been generated by heterogeneous workflow models. These challenges have provided the provenance community with anecdotal evidence regarding the expressiveness of the OPM, making a convincing case for its role. Providing an explicit representation of the workflow specification in addition to its execution, is, however, out of the scope of OPM: understandably so,
as this would entail capturing a broad variety of process models. Yet, it is not hard to imagine how an OPM graph $G$ may contain sufficient information to entail a plausible workflow that, once encoded using a formal workflow model and executed on suitable input, would generate an execution trace that is described by $G$.

Exploring this hypothesis is the main purpose of this paper. This idea, which essentially amounts to “reverse-engineering” a causal graph into a process description, can be cast as a special case of the general process mining problem, an area of research first pioneered over ten years ago, both as a theoretical investigation into the problem of generating a formal process description from a collection of execution logs [5], and as a practical workflow-generation tool for IBM’s MQSeries workflow system [1]. The proven appeal of this idea extends naturally to the case of OPM graphs, for several reasons. First and foremost, we can use it to show the potential of using the OPM not only to exchange provenance information across different workflow models, but also, implicitly, to provide a formal encoding of the originating process, using some workflow model, say $M$. This means that $M$ can be adopted as a common model to be used in the context of interoperable provenance for exchanging process description information.

Secondly, consider OPM graphs that are produced from non-workflow processes, for example from the observation of an interactive user session with a system. Karma [16] is an example of such a provenance management system, and [3] provides an early example of an OPM-generating system based on it. In a similar spirit, the OurSpaces social networking site collects user interaction using OPM [14], and more generally, process mining techniques are now being applied to mine behavioural patterns from online user interactions [17]. In this setting, the generated workflow represents a new model for such interaction which can be used both for documentation, as well as to provide a third party, who has received the OPM graph, with a formal model that describes the user session.

Finally, workflow generation leads to the notion of losslessness of OPM graphs relative to a specific workflow model, in the following sense. Consider a workflow specification $W$ and one trace $T$ of its execution. The trace may include a variety of information, depending on the specific workflow model $M$. In this work we focus on the Taverna dataflow model, described in some detail later, which accounts for multiple activations of a processor in combination with the processing of data collections. In this setting, we say that $G$ is lossless relative to $M$, if we can define a pair of complementary functions: $W2G_M()$ that maps workflow specifications and their traces to OPM graphs, and $G2W_M()$ that generates a workflow specification and execution trace pair $(W, T)$ from an OPM graph, with the property that, for any pair $(W, T)$ the following round-trip property holds: $G2W(W2G(W, T)) = (W, T)$.

The index $M$ denotes the dependency of this construction on the specific workflow model. This dependency is reflected in the amount of information contained in the trace $T$. In the case of Taverna, for example, the trace captures the relationship between a processor and the set of its activations. The general question we address in the paper is then, whether OPM is sufficiently expressive to provide losslessness in the presence of a variety of such trace elements, or, to put it more constructively, what kind of annotations should be added to the core OPM specification in order to provide lossless-ness for workflow models that are interesting in practice.

In order to gain any insight into the limitations of OPM, however, we must restrict the nature of the allowed annotations. Indeed, one could define $W2G$ to compute an invertible encoding $e(W)$ of the workflow $W$ itself, annotate an OPM graph with it, and define $G2W$ to extract the annotation and compute $e^{-1}(e(W))$. Although this pair of functions trivially satisfies the lossless-ness property, it does not tell us anything interesting about OPM. Furthermore, it violates the principle that OPM should only contain information about process executions, rather than about process specifications.
In a sense, achieving lossless-ness is not the problem here, but rather it is the criterion we use to identify, in a principled way, which elements of a trace it would be useful to include as part of OPM. As anticipated, in this work we focus on two specific types of information that are rather common in scientific workflow models, namely the association between a processor and the set of its activations, and membership information for collection data structures — both of which are instance-level and thus can naturally added to the trace. In the rest of this paper we use the Taverna workflow model to carry out this analysis. Taverna in this case provides in a concrete setting where both these aspects are captured as part of a fully implemented provenance management component.

1.1. Practical motivation: addressing the third Provenance Challenge

This work has been originally motivated by the need to address the third provenance challenge\(^1\), an exercise in testing the effectiveness of the OPM as an interoperable provenance exchange model, which involved about 15 participating teams over a period of several months in 2009. The challenge involved one workflow and a set of questions regarding the provenance of its data products, following one execution. Each team was responsible for a different workflow management system (Taverna amongst them), and was given three tasks. (1) to implement the workflow, execute it, and export its provenance as an OPM graph. (2) to show that a foreign graph, produced by some other team’s system in task (1), could be successfully imported into their own system. And (3), that the provenance information imported in task (2) could be used to answer the questions posed as part of the challenge.

In the particular query architecture of the native Taverna provenance component, efficient query processing is achieved by using the workflow graph as an index into the provenance graph. While this has proven to be an effective strategy for native workflow executions [11], it meant that suitable workflow graphs had to be constructed from the OPM graphs imported as part of task (2), leading to the implementation of the \(G2W\) function mentioned earlier. In turn, the notion of plausible generating workflow leads naturally to the more general round-trip framework that is the object of this paper.

1.2. Paper Contributions

Summarising the previous discussion, we can state the specific contributions of this paper as follows. Firstly, we propose the notion of lossless-ness of the round-trip translation between workflow/trace pairs and OPM graphs, in particular as a way to establish a precise correspondence between the features of a workflow computational model, and the types of annotations needed in OPM to keep track of those features. Secondly, we use the Taverna workflow model to show the framework in action. Specifically, we provide simple algorithms for both \(G2W\) and \(W2G\) functions, to show:

- how a Taverna workflow \(W\) and associated trace \(T\) can be constructed from an OPM graph \(G\), i.e., \(G2W(G) = (W, T)\). The workflow is plausible in the sense that \(W2G(G2W(G)) = G\);

- how an OPM graph augmented with processor activation and collection membership annotations is lossless relative to the Taverna model, in the sense that \(G2W(W2G(W, T)) = (W, T)\).

\(^1\)http://twiki.ipaw.info/bin/view/Challenge/ThirdProvenanceChallenge
We believe the framework to be applicable to other popular workflow models and systems, such as those surveyed in [7] as part of a special issue on scientific workflow systems [20], which may lead to principled recommendations for further types of OPM annotations. Indeed, it would be interesting to analyse, and this is left as further work, whether functions similar to $G2W$ and $W2G$ can be defined for systems like Kepler [10], VisTrails [2], Triana [4], and others.

The rules for generating OPM graphs from a Taverna execution, and vice versa, are fully implemented and will be part of the open-source Taverna software release in the near future.

1.3. Related work

As mentioned, our work relates naturally to the general area of process mining, studied at length for the past ten years [19]. Amongst recent results, the work of Van der Aalst et al. on workflow mining [18], further extended in [15], stands out as the most relevant, when one views the OPM as a form of restricted workflow log. It provides a formal reference workflow model, called workflow-nets, or WF-nets (a special form of Petri Nets), as well as theoretical results on mining algorithms that rediscover specific classes of WF-nets, and corresponding necessary conditions on the workflow logs used for the mining task\(^2\). The rediscoverability problem, as defined in the paper, is in principle similar to our notion of lossless-ness of the OPM graph.

Substantial differences, however, separate this from our work, where different assumptions on goals, scope, and approach, make ours a simpler problem. Firstly, our scope is not as general as mining a collection of logs, rather it is limited to defining rules that operate on one log at a time. Furthermore, the $\alpha$-algorithm in [18] include steps for the discovery of causal dependencies from the log. In our work there is no need for this step, because OPM graphs capture such dependencies by definition. Also, we use a specific target workflow model, namely Taverna\(^3\) [9], which only contains data dependencies, i.e., with no control flow and no explicit task precedence, and thus we are not concerned with issues of concurrency. Finally, our goal is somewhat complementary to that of general workflow mining: rather than producing the best possible model given a set of arbitrary logs, we recommend adding specific annotations to the OPM and justify them in a principled way in terms of lossless-ness properties.

2. Taverna workflows and OPM graphs

We begin by presenting a summary of the Taverna workflow model, and of the fragment of OPM that we are going to consider in this paper.

2.1. Taverna workflows and their execution traces

Let $\mathcal{P}_r$ and $\mathcal{P}_o$ be two disjoint sets of processor names and port names, respectively. A workflow $W = \langle P, \text{link}, \text{ip}, \text{op} \rangle$ is defined by a set $P \subseteq \mathcal{P}_r$ of processors, each having input and output ports $\text{ip}(p), \text{op}(p)$, where:

$$\text{ip} : \mathcal{P}_r \rightarrow 2^{\mathcal{P}_o}, \quad \text{op} : \mathcal{P}_r \rightarrow 2^{\mathcal{P}_o}$$

and by a relation

$$\text{link} \subseteq \mathcal{P}_r \times \mathcal{P}_o \times \mathcal{P}_r \times \mathcal{P}_o$$

\(^2\)The choice of Petri Nets as the target workflow model extends the early work in [5], where finite state machines were used as target process models.

\(^3\)http://www.taverna.org.uk.
representing the set of all dataflow links, which connect one output port of a processor to an input port of another, i.e., \( \text{link}(p_1, x, p_2, y) \) where \( x \in \text{op}(p_1) \) and \( y \in \text{ip}(p_2) \).

Ports have a type \( \tau \), which is either the base type \( s \) (for string), or an arbitrarily nested list of elements of type \( s \), according to the syntax: \( \tau ::= s | [\tau] \). Let \( \mathcal{V} \) be the set of all values of type \( \tau \). Ports \( x \in \mathcal{P}_o \) are assigned values \( v \in \mathcal{V} \), denoted \( \langle x, v \rangle \). Values are assigned to the ports of a processor \( p \) when \( p \) is activated. As any processor can be activated multiple times during the course of a workflow execution, we introduce a new set \( \mathcal{A}_{ct} \) of activation names, disjoint from \( \mathcal{P}_r \), and a function

\[
\text{act} : \mathcal{P}_r \to 2^{\mathcal{A}_{ct}}
\]

that associates a set of activations (possibly empty) to a processor. For convenience, we also write \( \text{proc}(p_a) \) to denote the processor \( p \) for which \( p_a \) is an activation, i.e., \( \text{proc}(p_a) = p \) iff \( p_a \in \text{act}(p) \) (we are assuming that the set of executions for two different processors are disjoint).

Let \( p_a \in \mathcal{A}_{ct} \) be an activation, and let

\[
\text{IN}(p_a) \subseteq \{ \langle x, v \rangle | x \in \text{ip}(\text{proc}(p_a)), v \in \mathcal{V} \}, \text{OUT}(p_a) \subseteq \{ \langle x, v \rangle | x \in \text{op}(\text{proc}(p_a)), v \in \mathcal{V} \}
\]

be the assignments on the input (resp., output) ports of \( \text{proc}(p_a) \).

We also introduce a collection membership relation\(^4\):

\[
\text{member}(v_i, c, i) \iff v_i \text{ is the } i \text{-th element of ordered collection } c
\]

An execution trace \( T \) for a workflow, consisting of processors \( P \subset \mathcal{P}_r \) includes:

- a finite set of trace events, which are relations of the form

\[
\text{xform}(p_a, \text{IN}(p_a), \text{OUT}(p_a))
\]

where \( p_a \in \text{act}(p) \) for some \( p \in P \) (note that we are not concerned with the temporal ordering of these tuples);

- the activation function (1) introduced above, and

- a collection membership relation.

Note that a trace reflects a black box semantics for processors, i.e., the only available information about \( p_a \)'s execution is the set of inputs and output values bound to its interface.

The execution model includes a particular case of multiple processor activation, which involves iterations over ordered data collections. The Taverna type system requires that each input port \( x \) be annotated with a \textit{depth}, indicating the list depth of the expected input. A processor that is designed to operate on a simple list like \{‘foo’, ‘bar’\} on input \( x \), for example, will carry the annotation \( \text{depth}(x) = 1 \), while if \( p \) expects an atomic value to be assigned to \( x \), it will have \( \text{depth}(x) = 0 \). In practice, the list depth of the actual input assigned to \( x \) may be greater than

\[^4\text{In its current state, the OPM specification (v1.1) included in this same special issue of FGCS does not include specific provisions for representing data collections. However a recent proposal, still in draft form, can be found in the OpenProvenance online archive: http://mailman.ecs.soton.ac.uk/pipermail/provenance-challenge-ipaw-info/2009-June/000120.html.}\]
depth(x), for instance \( \langle x, v \rangle \) with \( depth(x) = 0 \) but \( v = [v_1 \ldots v_k] \). This happens for example when a processor that returns the result of a database query in the form of a list of elements, is connected to a second processor that accepts one element at a time. When the collection depth of the inputs on its ports is greater than the declared depth on the port, a processor \( p \) is activated once for each element of an input collection. For example, in the simplest case of one input port with value \( v = [v_1 \ldots v_n] \) and one output value \( w \), this behaviour is defined by \( w = (map \ p \ v) = [(p \ v_1) \ldots (p \ v_n)] \).

The general case of multiple inputs and outputs extends the use of the \( map \) higher-order function to the cross-products of the inputs, and is described in detail in [11]. For the purpose of this paper we can abstract from these details, and simply assume that, in the presence of input collections that trigger iterations, a processor \( p \) is activated multiple times, producing collections on the output ports. Thus, the following trace fragment is representative of a generic iterating processor:

\[
\begin{align*}
(4) & \quad \{xform(p, IN_l, OUT_l)\}_{l:1\ldots k} \quad \{\{\text{member}(a_{il}, a_{il}, l)\}_{l:1\ldots k} \\
(5) & \quad \text{act}(p) = \{p_1 \ldots p_k\} \quad \{\{\text{member}(a'_{jl}, a'_{jl}, l)\}_{j:1\ldots m} \}
\end{align*}
\]

where \( p \) is activated \( k \) times, and \( IN_l \) and \( OUT_l \) are the inputs and output assignments to the \( l \)-th activation, respectively. These assignments are of the form: \( IN_l = \{\langle x_1, a_{i1} \rangle \ldots \langle x_n, a_{nl} \rangle\} \) and \( OUT_l = \{\langle y_1, a'_{j1} \rangle \ldots \langle y_m, a'_{ml} \rangle\} \), where values \( a_{il} \) and \( a'_{jl} \) are elements of lists \( a_i \) and \( a'_j \), respectively.

2.2. OPM provenance graphs

An OPM provenance graph \( G = \langle N, E \rangle \) is defined by

- three disjoint sets \( N = P \cup V \cup Ag \) of node labels: \( Processor \) nodes \( P \subseteq P_r \), \( Artifact \) nodes \( V \subseteq V \) representing values, and \( Agent \) nodes \( Ag \);
- a set \( R \) of role labels, disjoint from the nodes labels, and
- two main types of edges: \( E = \text{used} \cup \text{wgby} \), defined as relations on the nodes and roles:

\[ \text{used} \subseteq P \times R \times V, \text{wgby} \subseteq V \times R \times P \]

Thus, \( \text{used}(p, r, a) \) denotes that \( p \) has consumed artifact \( a \) in role \( r \), while \( \text{wgby}(a, r, p) \) denotes that \( a \) was produced by \( p \) in role \( r \).

For simplicity, in the rest of the paper we are only considering a core fragment of OPM that does not include Accounts, although the current implementation of the G2W function for Taverna does support the OPM notion of provenance Accounts in a very simple way, namely by mapping each
account to a separate workflow. We do not account for Agents, either, mainly because they have been shown to be less of a priority in practice, as none of the participants to the third provenance challenge included them in their graphs. Finally, for simplicity we are not going to consider edges of type wasDerivedFrom, and instead assume that these edges are always present between each input and output values of each processor.

3. Mapping between OPM graphs and Taverna workflows

In this section we construct a bi-directional mapping between the set \( \mathcal{W} \) of all Taverna workflows and their traces \( \mathcal{T} \), and the set \( \mathcal{G} \) of OPM graphs (with the restrictions just mentioned). More specifically, we want to define two complementary functions that map a graph to a pair \( (\mathcal{W}, \mathcal{T}) \), and vice versa:\(^5\)

\[
\text{G}_2\text{W} : \mathcal{G} \rightarrow \mathcal{W} \times \mathcal{T}
\]

and

\[
\text{W}_2\text{G} : \mathcal{W} \times \mathcal{T} \rightarrow \mathcal{G}
\]

We translate our earlier informal “round-trip” property requirement between OPM graphs and workflow graphs, into the following two requirements on these functions:

1. for any graph \( G \in \mathcal{G} \):

\[
\text{W}_2\text{G}(\text{G}_2\text{W}(G)) = G
\]

2. for any Taverna workflow \( W \in \mathcal{W} \) and valid execution trace \( T \in \mathcal{T} \) for \( W \):

\[
\text{G}_2\text{W}(\text{W}_2\text{G}(W, T)) = (W, T)
\]

The first property ensures that the \( (W, T) \) pair generated by \( \text{G}_2\text{W} \) can be mapped back to \( G \), i.e., \( W \) indeed represents a “plausible” workflow. The second property is a more specific losslessness property which says that an OPM graph that represents one execution of a Taverna workflow, contains enough information to reproduce both the workflow and the execution trace that the graph represents.

We show that, for the class of Taverna workflows where each processor is activated exactly once, and all values are atomic, OPM is sufficient to support both properties. When the workflow involves multiple activations (or some of the processors are never activated), however, additional annotations must be captured as part of the OPM graph in order to support property (2). We present both cases in the following two sub-sections.

3.1. Simple mapping with no OPM annotations

We begin by presenting \( \text{G}_2\text{W} \) on an un-annotated OPM graph, such as one that could have been generated by a third party workflow. \( \text{G}_2\text{W} \) is based on the composition of two types of elementary mappings, from used and wgby OPM edges onto elements of a workflow and trace. These mappings are illustrated in Fig. 2(a). An edge of the form used\((p, x, v)\) translates into a workflow processor \( p \) having an input port \( x \), plus a trace event of the form xform\((p, IN, OUT)\), where IN includes the

---

\(^5\)Note that we have dropped the index \( M \), for the model, used in the introduction, as the model is implied to be Taverna’s throughout.

7
assignment \langle x, v \rangle$. Similarly, an edge of the form `wgby(y, w, p)` translates into a workflow processor `p` having an output port `y`, plus a trace event of the form `xform(p, \langle x, v \rangle)`, where `OUT` includes the assignment \( \langle y, w \rangle \). Furthermore, a data link from a port `x` of processor `q` to a port `y` of `p`, is generated in response to a path from `p` to `q` in the OPM graph, of the form `used(p, x, v)`, `wgby(y, w, q)`, i.e., when `p` consumes a value `v` (through port `x`) that has been produced by `q` through port `y`. The composition also avoids duplicating processor and port names when the same processor node is involved in multiple edges of the same type; in this case, all port assignments are collected into a single `xform` event.

Alg. 1 shows the pseudo-code for `G2W`. As an example, Fig. 2(b) shows a simple OPM graph and the corresponding workflow and execution trace.

Algorithm 1 `G2W`: Simple generation of Taverna workflows from OPM graphs

\begin{verbatim}
\text{G2W}: Simple generation of Taverna workflows from OPM graphs

P = \emptyset, T = \emptyset, L = \emptyset
for all `used(p_a, x, v) \in E_{used}` do
    P ← P \cup \{p\}
    ip(p) ← ip(p) \cup \{x\}
    if `xform(p_a, IN, OUT) \in T` then
        IN ← IN \cup \{(x, v)\}
        \text{\textgreater{} add assignment to IN set for existing trace event}
    else
        T ← T \cup \{xform(p_a, \{(x, v)\}, \emptyset)\}
        \text{\textgreater{} create new trace event}
    end if
end for
for all `wgby(v, y, p_b) \in E_{wgby}` do
    L ← L \cup \{link(p_b, y, p_a, x)\}
    \text{\textgreater{} used followed by wgby \rightarrow new link}
end if
\end{verbatim}

A symmetrically constructed `W2G` function needs only consult the trace `T`, and can be defined as
follows. For each trace event of the form  
\[ xform(p, \{ \langle x_1, v_1 \rangle, \ldots, \langle x_n, v_n \rangle \}, \{ \langle y_1, w_1 \rangle, \ldots, \langle y_m, w_m \rangle \}) \]
the function generates a set of used edges: \( \{ \text{used}(p, x_i, v_i) \}_{i=1...n} \), and a set of used edges: \( \{ \text{usedby}(y_j, w_j, p) \}_{j=1...m} \), one for each input and output assignment in the event. Alg. 2 shows the pseudo-code for W2G.

**Algorithm 2 W2G: Generation of OPM graphs from Taverna workflow execution**

\[
\begin{align*}
E_{used} & \leftarrow \emptyset, E_{usedby} \leftarrow \emptyset \\
\text{for all } & xform(p_i, \text{IN}(p_i), \text{OUT}(p_i)) \in T \text{ do} \\
& \quad \text{for all } \langle x, v \rangle \in \text{IN}(p_i) \text{ do } E_{used} = E_{used} \cup \{ \text{used}(p_i, x, v) \} \\
& \quad \text{end for} \\
& \quad \text{for all } \langle y, w \rangle \in \text{OUT}(p_i) \text{ do } E_{usedby} = E_{usedby} \cup \{ \text{usedby}(w, y, p_i) \} \\
& \quad \text{end for} \\
& \text{end for} \\
\text{return } & E_{used}, E_{usedby}
\end{align*}
\]

### 3.2. The case for annotations in OPM

When each \( p \in P \) is activated exactly once, every processor appears in exactly one trace event, resulting in an OPM graph fragment like the one shown in Fig. 2(b) for P3. In this case it is easy to verify, by construction, that the application of \( W2G \) to the resulting graph \( G \) reproduces precisely \( W \) and \( T \).

In this section we deal with the more general case where (i) a processor is activated zero or more times, and (ii) list values may be consumed and produced. As we recall from Sec. 2, in Taverna these two circumstances occur together, as multiple activations of a processor are generated implicitly in response to a mismatch in the depth of the input lists, relative to the expected depth on the processor ports, as shown in Fig. 1.

This pattern is illustrated in Fig. 3, where list values in some or all of the input ports trigger a map operation, as explained in Sec. 2, resulting in \( k \) activations of \( p \). This produces the graph at the bottom of the figure, where all the activations are represented explicitly. Note also that the information about the nested lists that provide structure to artifacts \( a_{ij} \) and \( a'_{ij} \) is lost.

The consequence of this loss of information, both on processors and on data, is that the subsequent application of \( G2W \) produces the new workflow in Fig. 4, which includes one processor for each \( p_i \). The corresponding trace events are of the form \( \{ xform(y_i, \text{IN}(y_i), \text{OUT}(y_i)) \}_{i=1...k} \) where \( \text{IN}(i) = \{ \langle x_1, a_{i1} \rangle, \ldots, \langle x_n, a_{in} \rangle \} \), and \( \text{OUT}(y_i) = \{ \langle y_1, a'_{i1} \rangle, \ldots, \langle y_m, a'_{im} \rangle \} \). In this case we have

\[
G2W(W2G(W, T)) = (W', T') \neq (W, T)
\]

and \( W' \) is a workflow structure in which all the activations that occurred in \( W \)'s execution are represented explicitly. Note also that it is impossible to reconstruct the list structure of the initial input on which the original workflow was executed (this is the set of artifacts in the OPM graph that are the destination of used edges but are not the source of any usedby edges).

In order to prevent this information loss and make the \( G2W, W2G \) pair completely loss-less, we extend their definition to keep track explicitly of each processor’s activation, as well as of the list structures defined in a trace by the member relation. Note that, while the extra OPM annotations that we propose are non-standard, they are not needed to interpret a standard OPM graph obtained from a Taverna workflow. Indeed, a graph \( G \) where all the iterations are unfolded is perfectly valid, and furthermore, \( \langle W', T' \rangle = G2W(G) \) is still a “plausible” workflow, in the sense that, since all of its processors now represent exactly one activation, we have \( G2W(W2G(W', T')) = (W', T') \) as in the
base case from the previous section. In view of this, the purpose of this section is to justify the introduction of a minimal set of OPM annotations that make the round-trip construction possible when the graph represents an execution of a workflow that supports multiple processor activations and data collections.

As the annotation framework currently proposed for OPM is quite generic, prescribing only the use of property-value pairs, we are simply going to use the same relational notation (1) and (2) in OPM that we used for Taverna to represent activation and membership annotations, respectively.

The definition for \( W_2G_{\text{ann}} \), the annotation-aware version of \( W_2G \), simply copies all \( \text{act} \) and \( \text{member} \) annotations from the workflow trace to the OPM graph (see Alg. 3, where \( \text{Ann}_{\text{act}} \) denotes the set of activation annotations)\(^6\).

Alg. 4 shows the annotation-aware version \( G_2W_{\text{ann}} \) of \( G_2W \). In this version, a single processor \( p \) is generated for all OPM processors \( p_a \) such that \( \text{proc}(p_a) = p \). If no such annotations are available (as would be the case for any OPM graph produced by a function other than \( W_2G_{\text{ann}} \)), then a new Taverna processor is generated for \( p_a \), as in \( G_2W \). Fig. 5 shows the effect of combining \( W_2G_{\text{ann}} \) (Alg. 3) with \( G_2W_{\text{ann}} \) (Alg. 4) when annotations are carried throughout both constructions.

Due to space limitations, we omit a formal proof that \( G_2W_{\text{ann}} \) and \( W_2G_{\text{ann}} \) do indeed satisfy the round-trip properties (6) and (7). The proof is fairly straightforward, however, and its intuition is illustrated in the patterns of Fig. 5\(^7\).

\(^6\)Note that collection annotations are simply copied through by each of the two functions, which is not explicitly described.

\(^7\)One borderline case, which is left for future work, occurs when the workflow does nothing at all. In this case the resulting trace is empty, and so is the OPM graph. This means that Alg. 4 will generate an empty workflow — which is indeed plausible, but breaks the round-trip property.
Algorithm 3 W2G\textsubscript{ann}: Annotation-aware generation of OPM graphs from Taverna workflow execution

\begin{algorithm}
\begin{algorithmic}
\State $E_{\text{used}} \leftarrow \emptyset$, $E_{\text{wgby}} \leftarrow \emptyset$, $\text{Ann}_{\text{act}} \leftarrow \emptyset$
\ForAll {xform($p_a$, IN($p_a$), OUT($p_a$)) ∈ $T$}
\ForAll {\langle x, v \rangle ∈ \text{IN}(p_a)}
\State $E_{\text{used}} = E_{\text{used}} \cup \{\text{used}(p_a, x, v)\}$
\EndFor
\ForAll {\langle y, w \rangle ∈ \text{OUT}(p_a)}
\State $E_{\text{wgby}} = E_{\text{wgby}} \cup \{\text{wgby}(w, y, p_b)\}$
\EndFor
\State $\text{Ann}_{\text{act}} = \text{Ann}_{\text{act}} \cup \{\text{act}(p_a, \text{proc}(p_a))\}$
\EndFor
\Return $E_{\text{used}}, E_{\text{wgby}}, \text{Ann}_{\text{act}}$
\end{algorithmic}
\end{algorithm}

Algorithm 4 G2W\textsubscript{ann}: Annotation-aware generation of Taverna workflows from OPM graphs

\begin{algorithm}
\begin{algorithmic}
\ForAll {\text{used}(p_a, x, v) ∈ $E_{\text{used}}$}
\If {\text{proc}(p_a) is defined} \Comment{$p$ is set to the processor rather than its activation}
\State $p = \text{proc}(p_a)$
\Else \State $p = p_a$
\EndIf
\If {$P \leftarrow P \cup \{p\}$ ; \State $\text{ip}(p) \leftarrow \text{ip}(p) \cup \{x\}$}
\If {xform($p_a$, IN, OUT) ∈ $T$}
\State IN \leftarrow IN \cup \{(x, v)\}
\Else \State $T \leftarrow T \cup \{\text{xform}(p_a, \{(x, v)\}, \emptyset)\}$
\EndIf
\EndIf
\If {\text{wgby}(w, y, p_a) ∈ $E_{\text{wgby}}$ \Comment{$p$ is set to the processor rather than its activation}}
\State $L \leftarrow L \cup \{\text{link}(p_b, y, p_a, x)\}$
\EndIf
\EndFor
\ForAll {\text{wgby}(w, y, p_a) ∈ $E_{\text{wgby}}$}
\If {\text{proc}(p_a) is defined} \Comment{$p$ is set to the processor rather than its activation}
\State $p = \text{proc}(p_a)$
\Else \State $p = p_a$
\EndIf
\If {$P \leftarrow P \cup \{p\}$ \State $\text{op}(p) \leftarrow \text{op}(p) \cup \{x\}$}
\If {xform($p_a$, IN, OUT) ∈ $T$}
\State OUT \leftarrow OUT \cup \{(y, w)\}
\Else \State $T \leftarrow T \cup \{\text{xform}(p_a, \emptyset, \{(y, w)\})\}$
\EndIf
\EndIf
\EndFor
\Return $P, \text{link}, T$
\end{algorithmic}
\end{algorithm}
4. Summary and ongoing work

In this paper we have made the case for the selective introduction of specific annotations into OPM, based on a notion of losslessness of a graph relative to a particular workflow model. We have described a general framework for losslessness in this context, and presented the specific case of the interplay between Taverna workflows and OPM graphs. This work was triggered by the third Provenance Challenge, which required a means to generate a "plausible" Taverna workflow from an OPM graph. Both G2W\textsubscript{ann} and W2G\textsubscript{ann} are implemented (in Java) as part of the provenance management component of Taverna, and will be part of the next release of the workflow manager.

We find two main directions for further work interesting. Firstly, while the bulk of our work has so far been Taverna-centric, we hope to apply the framework to other workflow models, in order to provide a justification for other types of OPM annotations. In particular, when a workflow trace is not limited to "black-box" representation of a processor's execution, it may contain enough information to allow for the generation of a workflow that is not just plausible, but that is actually executable on some input. We view this as a step in the direction of reproducible workflow-based science. Finally, we are planning to address the more general case of workflow mining in the presence of multiple OPM graphs.


