Linking Multiple Workflow Provenance Traces for Interoperable Collaborative Science

Paolo Missier, Carole Goble
School of Computer Science
University of Manchester, Manchester, UK
{pmissier,carole}@cs.man.ac.uk

Saumen Dey, Anandarup Sarkar
Dept. of Computer Science
University of California, Davis
{scdey,asarkar}@ucdavis.edu

Biva Shrestha
Dept. of Computer Science
Appalachian State University, Boone, NC
ivashrestha@gmail.com

Bertram Ludäscher
Dept. of Computer Science & Genome Center
University of California, Davis
ludaesch@ucdavis.edu

Shawn Bowers
Dept. of Computer Science
Gonzaga University
bowers@gonzaga.edu

Ilkay Altintas, Manish Kumar Anand
San Diego Supercomputer Center
University of California, San Diego
{altintas, mkanand}@sdsc.edu

***Abstract*—Scientific collaboration increasingly involves data sharing between separate groups. We consider a scenario where data products of scientific workflows are published and then used by other researchers as inputs to their workflows. For proper interpretation, shared data must be complemented by descriptive metadata. We focus on *provenance traces*, a prime example of such metadata which describes the genesis and processing history of data products in terms of the computational workflow steps. Through the reuse of published data, virtual, *implicitly collaborative* experiments emerge, making it desirable to compose the independently generated traces into global ones that describe the combined executions as single, seamless experiments. We present a model for provenance sharing that realizes this holistic view by overcoming the various interoperability problems that emerge from the heterogeneity of workflow systems, data formats, and provenance models. At the heart lie (i) an abstract workflow and provenance model in which (ii) data sharing becomes itself part of the combined workflow. We then describe an implementation of our model that we developed in the context of the *Data Observation Network for Earth* (DataONE) project and that can "stitch together" traces from different Kepler and Taverna workflow runs. It provides a prototypical framework for seamless cross-system, collaborative provenance management and can be easily extended to include other systems. Our approach also opens the door to new ways of workflow interoperability not only through often elusive workflow standards but through shared provenance information from public repositories.**

I. Introduction

One of the tenets of the emerging paradigm of "open" experimental, data-intensive science [14] in which information is the main product, is that scientists should have both the incentive and the ability to share some of their findings with other members of their community, as well as to reuse their peers' data products. Indeed, the scientists' natural resistance to sharing their data and methods is increasingly being replaced by the realization that the benefits of data sharing may outgrow the risks of losing exclusive ownership of data. This phenomenon is amplified by new requirements to make data available prior to publication, along with the definition of standard formats for data exchange in many domains of science [1].

We will concentrate on the particularly common setting where the structure and the steps of the transformation process is formally encoded as a scientific workflow [26], [18], and where provenance traces results from the observation of the workflow execution. In this setting, implicit collaboration between two or more parties involves the execution of workflows which uses some of the results of another workflow's execution as part of its inputs. The following scenario, used as a running example through the paper, clarifies this setting (see Fig. 1).

A. *Workflow Collaboration Scenario: Alice, Bob, and Charlie*

Alice and Bob are two experts in image analysis for medical applications who occasionally collaborate on joint projects. Alice has developed a workflow W_A using her favorite workflow system. W_A consists of a data pipeline that performs various transformations of input image(s) X to produce a set Z of new images.[1] Alice decides to publish the results of some of her workflow runs via a shared data space so that her collaborators (or any other users) can use them. Bob retrieves a copy of one of those images, $z \in Z$, as he would like to use it as input to his own workflow W_B, which he developed (incidentally using a different workflow system than Alice). He first applies some format transformation $u = f(z)$, then runs W_B with u (and possibly some additional local input data), obtaining a new result set v of data products. He then in turn publishes v, together with a trace T_B of how v was derived in a shared data space. Along comes Charlie, our third expert, who is interested in the results v and wants to understand how they have been derived. The commonly accepted approach to answering Charlie's question is to collect a provenance trace T_B during the execution of Bob's workflow, which describes in detail the data transformation and generation process, as well as the dependencies amongst data products involved in the process. The trace T_B is a directed graph whose nodes represent either data or computations, and where arcs represent dataflow or

[1] E.g., we use a workflow for image analysis of brain scans from the First Provenance Challenge [24], [23] to demonstrate our system [12].

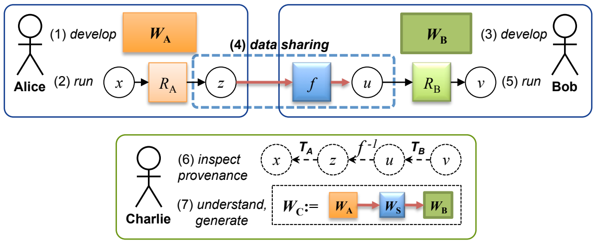

Fig. 1. Alice runs workflow W_A; the run R_A generates data z she shares. Bob imports and transforms this data (via f), so he can use it as input u to his workflow W_B. Charlie understands how Bob's result v depends on u via the trace T_B, but can also trace back u's provenance via Alice's trace T_A to the original input x. This is possible since T_A, T_B, and the data sharing provenance $z \dashleftarrow u$ have been published. Charlie understands that there is an implicit, combined workflow $W_C = W_B \circ W_S \circ W_A$ that he might automate provided Bob also publishes his data import/transformation script W_S.

other dependencies between nodes. In our example scenario, the provenance graph includes nodes for the input and output images, as well as for all of the intermediate data produced at each stage of the processing pipeline. User questions on the origins of a workflow-generated data product can then be answered by traversing the provenance graph, possibly as part of a more complex query. The standard scenario for a single-run provenance model (e.g. Alice's) is shown in Fig. 2. The annotation at the bottom, $\mathtt{ddep}^*(X, z_1)$, denotes a query that recursively traverses the trace graph T_A, following data dependencies (`ddep`) from one of the outputs z_1 back to all data X (inputs and intermediates) on which z_1 depends.

In the multi-user, multi-run scenario we should not assume that the provenance of v (Fig. 1) extends only as far as Bob's inputs: the actual dependencies—due to data sharing between Alice and Bob—can be carried all the way up to the primary data from which some of Bob's inputs were first derived, i.e., Alice's outputs and (transitively) her inputs to W_A. This is desirable for two reasons: firstly, it provides Charlie with a more complete explanation of v's origins (the provenance will still include all intermediate data, including Bob's inputs, as a special case); and secondly, automatic attribution is facilitated, e.g., here by giving Alice due credit for having contributed to the derivation of v via her shared data z.

B. Goals and Challenges

We explore the hypothesis that provenance traces produced by two different workflow systems can indeed be seamlessly connected whenever the outputs of the first overlap with the inputs of the second. The overall scenario appears in Fig. 3 and will be discussed in detail in the next sections. We have pointed out the potential appeal of this idea in recent work [3]: We can view a compound provenance trace as a manifestation of the execution of a *virtual workflow* W_C that includes both W_A and W_B and some data sharing inbetween. This is a workflow that Alice and Bob never agreed to define jointly, and yet the resulting joint provenance trace as observed by Charlie represents a formal model of their collaboration, which in reality has only occurred through shared data products.

However, a number of practical challenges must be overcome when realising such end-to-end provenance queries, e.g., due to the heterogeneity of local data spaces in which different workflow system operate, of the workflow models themselves, and also of the internal provenance models. Firstly, if the workflows are run on different systems (say Kepler and Taverna), then their provenance traces follow different native data models, and thus a common model that encompasses both must be defined, along with mappings from each local model to the common model. Secondly, each system may adopt a different naming scheme to provide references for the data involved in the process (and thus for the data that appears in the traces), once again requiring a mapping across such naming schemes. Finally, implicit collaboration through data sharing may involve further data transformations and preparation steps, e.g., to make Alice's data suitable for Bob's workflow. The provenance of these intermediate steps may initially not be part of a provenance-generating workflow, so the dependencies can be lost, breaking the combined trace graph.

Regarding the first problem, we propose a traditional data integration approach, which involves the definition of a global provenance model and of local-to-global mappings that allow local provenance traces to be materialised in a common provenance store (CPS in Figures 3 and 5). To address the second and third problem, we observe that, as long as any data sharing or transformation occurs *outside* of the control of a workflow system, there can be no guarantee that the provenance traces generated by the systems "join up" seamlessly despite the fact that they share some of the data, because the additional operations typically break the continuity of the relations required by the transitive provenance queries. This complication has indeed been the main stumbling block for the participants of the Third Provenance Challenge [25], which focused on provenance interoperability and successfully showed the potential of the Open Provenance Model (OPM) [22] for the purpose, only to fail to consider the heterogeneity of the local data naming spaces. This work is motivated in part by this experience and led to a collaborative DataONE project [11], [12] in which these technical challenges were re-examined and solved. Specifically, our strategy is to ensure that every time a transformation occurs "out-of-band" (e.g., the copy of a data value z from a local store, e.g., Alice's S_A, to a public store S_P), new, explicit transitive relations are produced, e.g., "z_0 in S_A *is the same as* z_0' in S_P" (cf. Fig. 3). Here, we follow the principle that data and provenance sharing itself is a process, and that the provenance traces provided by the workflow systems must be complemented with the *provenance of the data sharing process*.

C. Contributions and Outline

In the remainder of the paper, we first define an abstract model of workflows and an associated model of provenance that can be used as a least common denominator of, and target model for, different models of computation used by scientific workflow systems (Section II). Our model can be seen as a version of OPM with an associated workflow-level structure that is used to link instance-level trace information with workflow-level schema information. We also discuss how to extend and query the model.

In Section III, we elaborate on the notion of the "provenance of the publication process". We introduce a *copy* operator that is used for data sharing and which keeps track of data equivalences across different data spaces. Another operator is used to map traces from their native models to our common model of provenance. Taken together, we obtain global provenance graphs that are "stichted together at the seams", and which allow us to trace back data dependencies across multiple workflow runs, systems, and user groups.

The study and the prototype have been realised in the context of the Data Observation Network for Earth (DataONE) project [11]. DataONE is dedicated to large-scale preservation and access to multi-scale, multi-discipline, and multi-national Earth observational data. We describe our prototype implementation in Section IV and conclude with a discussion of related work in Section V.

II. Modeling Workflows and Provenance

A. Abstract Workflow and Provenance Model

In the following, we define a simple, abstract model for workflows and the provenance information that is generated by running them. Provenance information is captured in the form of *trace graphs*, which record key observables of workflow runs, e.g., the ids of data tokens consumed and produced by workflow processes. While our model is similar to OPM (see below), its main advantage is its simplicity and generality: We have used it as a common model that can accommodate both Kepler and Taverna provenance information and that can be easily extended to capture additional observables as needed.

More formally, a **workflow specification** (short: *workflow*) W is a directed graph $W = (V_W, E_W)$ consisting of vertices $V_W = \mathbf{A} \cup \mathbf{C}$ which either represent **actors** $\mathbf{A}$ (a.k.a. *processors* in Taverna) or data **channels** $\mathbf{C}$, and edges $E_W = E_{\mathtt{in}} \cup E_{\mathtt{out}}$ that are either **input edges** $E_{\mathtt{in}} \subseteq \mathbf{C} \times \mathbf{A}$ or **output edges** $E_{\mathtt{out}} \subseteq \mathbf{A} \times \mathbf{C}$.

A **trace graph** (short: *trace*) T is a directed, acyclic[2] graph $T = (V_T, E_T)$ consisting of vertices $V_T = \mathbf{I} \cup \mathbf{D}$ which either represent **invocations** $\mathbf{I}$ of actors or **data** objects $\mathbf{D}$, and edges $E_T = E_{\mathtt{read}} \cup E_{\mathtt{write}}$ that are either **read edges** $E_{\mathtt{read}} \subseteq \mathbf{D} \times \mathbf{I}$ or **write edges** $E_{\mathtt{write}} \subseteq \mathbf{I} \times \mathbf{D}$.

Intuitively, when running a workflow specification W, a trace graph T can be recorded, such that actors in W can have multiple invocations in T, with data flowing in and out of actors via channels, which in turn is captured through read and write edges in T.

We capture this intuition formally, using the following *minimal* requirement that we expect any trace to satisfy: A trace T is a (possibly partial) **instance of** a workflow W if the edges of the trace graph can be understood as instances of the workflow graph, i.e., there exists a graph homomorphism $h : T \to W$, such that for any data object $d \in \mathbf{D}$ and any invocation $i \in \mathbf{I}$:

- if $(d, i) \in E_{\mathtt{read}}$ then $(h(d), h(i)) \in E_{\mathtt{in}}$, and
- if $(i, d) \in E_{\mathtt{write}}$ then $(h(i), h(d)) \in E_{\mathtt{out}}$.

Thus, h associates reads and writes in T with input and output edges in W, respectively. Note that this also implies that h yields for every data item d a unique channel $h(d) \in \mathbf{C}$ and for every invocation i a unique actor $h(i) \in \mathbf{A}$.

Consider, e.g., Alice's workflow W_A and trace T_A on the top and bottom in Figure 2, respectively. A suitable h maps, e.g., the edges $x_1 \xrightarrow{\mathtt{read}} a_1$ and $x_2 \xrightarrow{\mathtt{read}} a_2$ in T_A to $X \overset{\mathtt{in}}{\to} A$ in W_A; and the edges $a_1 \xrightarrow{\mathtt{write}} y_1$ and $a_2 \xrightarrow{\mathtt{write}} y_2$ to the edge $A \overset{\mathtt{out}}{\to} Y$. It is easy to see that $h(T) \subseteq W$ if T is an instance of W.

Our model is similar to the one in [7] in that they also use a graph homomorphism to define when a trace (called *run* there) T is *valid* w.r.t. a given workflow graph W. There are a few differences, however, some of them subtle. First, we avoid the term "valid" since our model is a least common denominator for different models of computation (MoCs) M: e.g., SDF or COMAD in Kepler, or the Taverna MoC will have additional observables and corresponding requirements that need to be satisfied before a trace can be called "valid" w.r.t. W and M.[3] Another difference is that the model in [7] only has actor/invocation nodes but *no* channel/data nodes. The latter are crucical for us, since it is precisely at the data

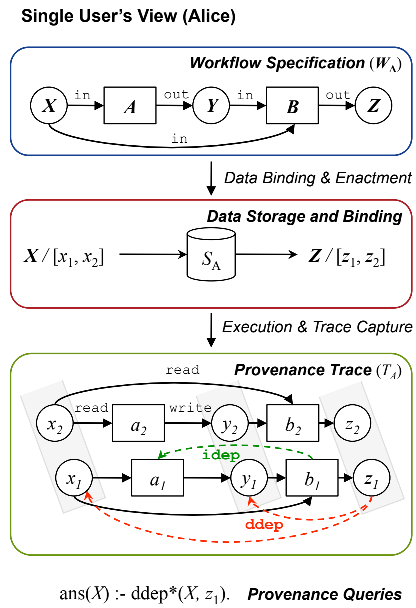

Fig. 2. Alice's workflow W_A (top), with input and output data bindings $X/[x_i]$ and $Z/[z_j]$ (middle), and trace graph T_A (bottom). T_A is an instance of W_A since there is a homorphism $h : T_A \to W_A$ that maps read (write) edges in T_A to corresponding in (out) edges in W_A. In addition, data (ddep) and invocation (idep) dependencies may be part of the trace.

[2] Note that we allow cycles in workflow specfications but not in trace graphs. This corresponds to the fact that when running a workflow, loops are unrolled, giving rise to an acyclic trace graph.

[3] In an SDF trace, e.g., all invocations $i \in \mathbf{I}$ of A, i.e., with $h(i) = A$, should have the same number of token reads and of writes (indicated by the number of read/write edges incident with i), so that the fixed token consumption and production rates of A can be used for compile-time workflow scheduling [16].

nodes where we stitch together multiple traces.

As our core model, we deliberately chose a very simple model that can be seen as a variant of OPM [22]: in particular, our `read` and `write` edges correspond to *used* and *was_generated_by* relations in OPM, respectively. Extensions are modeled simply by additional requirements on traces (possibly using further relations to capture additional observables) or on workflow graphs. For example, we can avoid write-conflicts at the *trace level* by disallowing multiple actors to output to the same channel at the *workflow level*, which in turn is the case iff the first column of $E_{\texttt{out}}(\mathbf{A}, \mathbf{C})$ is a key (no channel C can be a receiver for different actors A_1, A_2).

Despite its simplicity, our model can represent, e.g., loop unrolling and parallel execution of multiple instances: a feedback loop in W, e.g., with edges $A \xrightarrow{\texttt{out}} D$ and $D \xrightarrow{\texttt{in}} A$ may yield the trace edges $a_i \xrightarrow{\texttt{write}} d_i \xrightarrow{\texttt{read}} a_{i+1} \xrightarrow{\texttt{write}} d_{i+1} \xrightarrow{\texttt{read}} \cdots$ It can also accommodate apparently unusual traces: e.g., for a workflow with $X \xrightarrow{\texttt{in}} A \xrightarrow{\texttt{out}} Y$ we might have two *disconnected* trace edges $x_1 \xrightarrow{\texttt{read}} a_1$ and $a_2 \xrightarrow{\texttt{write}} y_2$.[4] It is easy to see that such traces, if desired, can be eliminated via additional constraints on T, e.g., that every node of T must be on a path from a distinguished input node to a distinguished output node.

The **signature** $W\colon X_1, \ldots, X_k \to Y_1, \ldots, Y_\ell$ of a workflow W consists of **input channels** X_i and **output channels** Y_j, i.e., those data channels which have no incoming and no outgoing edges, respectively. For example, W_A in Figure 2 has the signature $W_A\colon X \to Z$; here, channel Y represents internal, intermediate data that is *not* part of W_A's signature. After **binding** input data to input channels, W can be **run** (or *enacted*, *executed*), generating a trace T: e.g., $X/[x_1, x_2]$ in Figure 2 binds two data items x_1, x_2 to the input channel X, in this case resulting in two independent instances in the trace graph. When the run is complete, the binding $Z/[z_1, z_2]$ associates the results of the run with the output channel Z.

Finally, in addition to the external `read` and `write` observables, further observables might be recorded or inferred in a provenance trace T. For example, the rule

$$\texttt{idep}(a_1, a_2) :- \texttt{write}(a_1, d), \texttt{read}(d, a_2)$$

allows one to derive an **invocation dependency** $a_1 \overset{\texttt{idep}}{\dashleftarrow} a_2$, stating that invocation a_1 has written data d that was read by invocation a_2, so the latter depends on the former (cf. Fig. 2). Similarly, a **data dependency** $d_1 \overset{\texttt{ddep}}{\dashleftarrow} d_2$ can be derived via

$$\texttt{ddep}(d_1, d_2) :- \texttt{read}(d_1, a), \texttt{write}(a, d_2)$$

stating that d_2 depended on d_1 in T, whenever d_1 was read (input) by an actor invocation a that has written (output) d_2. This rule for `ddep` is a default assumption in many models of provenance: unless further information is available, one assumes that all outputs of invocation a may depend on all inputs of that invocation a. In some models of computation (MoCs), however, this is an overestimate of the true dependencies: e.g., sometimes not all output ports depend on all input ports [27], or not all input data items are used in the computation of the output items (e.g., in a sliding-window aggregate [19], or in MoCs such as COMAD that pass some (out-of-scope) items through an actor, without "seeing" those items [6]). In the following, we focus on data dependencies `ddep` as the main provenance relation, whether it has been derived via the above rule or recorded directly by the workflow system.

[4]Such traces do, however, occur in practice, e.g., for the `Delay` actor, which is used to "prime the pump" of feedback loops, first writing data independently of (and thus disconnect from) any read operation.

B. Queries on the Provenance Model

The model just described supports several types of queries over provenance traces. In our DataONE project [12] we have defined a set of reference queries that are designed to test the capabilities of our provenance interoperability system. We have grouped them into three classes: (i) queries on the common data store CDS, (ii) non-closure queries on the traces in the common provenance store CPS, and (iii) transitive closure queries on the traces. The first two classes include queries that can be easily answered using the public data store and the common provenance model, including for example:

- What data and what traces did Alice and Bob publish?
- What are the data inputs, outputs, and intermediate data products of trace T?
- What actors and channels were used in T?
- What are the inputs and outputs of invocation a_i in T?

For example, the answer to the third query are simply the nodes in $h(T)$.

More interesting from our perspective is the third class, where we find queries that require the traversal of *multiple, independently generated* provenance traces. Answering these questions requires (inductive or recursive) querying along dependency paths of unknown length, making them computationally expensive [10], [7], [13] as they involve the transitive closure of data dependencies `ddep`. A particular challenge of our collaborative e-Science scenario is to ensure that the traces (e.g., T_A and T_B in Figs. 1 and 3) can be "stichted together" or "bridged" so that the closure operation can traverse both of them seamlessly. The transitive closure $\texttt{ddep}^*$ of $\texttt{ddep} \subseteq \mathbf{D} \times \mathbf{D}$ is defined as usual:

$$\texttt{ddep}^*(d_1, d_2) :- \texttt{ddep}(d_1, d_2)$$
$$\texttt{ddep}^*(d_1, d_2) :- \texttt{ddep}(d_1, d), \texttt{ddep}^*(d, d_2)$$

which allows us to use the query $:- \texttt{ddep}^*(D, D')$ to find (i) all dependants D' of a given d, (ii) all D that a given d' depends on, or (iii) whether d' is reachable from d.

Note that such closure queries can be combined with additional conditions on elements of the data or of the traces, e.g., "find all dependants of d that have been observed to flow through channel c", or "find all upstream d that contributed to the computation of d' through a specific actor a". Clearly, these queries can be decomposed into a closure query, plus queries from classes (i) and (ii), and thus in the rest of the paper we are going to concentrate on closure queries only.

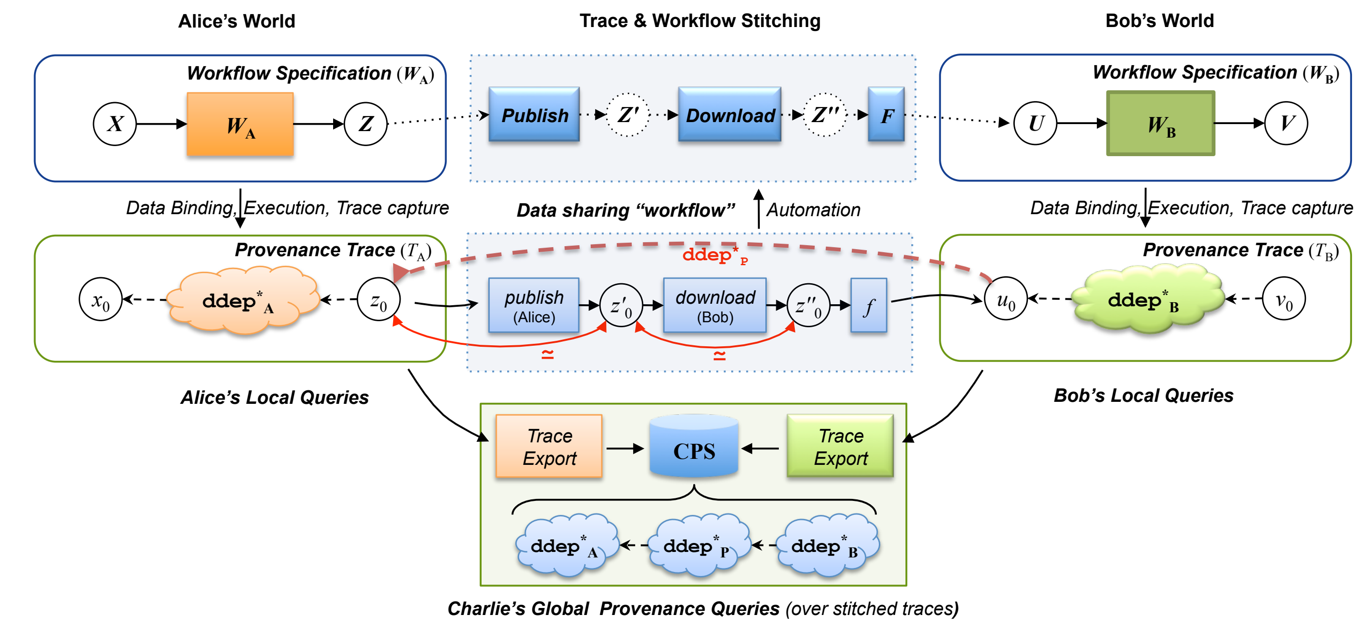

Fig. 3. "Stichting" the traces of Alice and Bob to enable seamless provenance queries by Charlie, spanning heterogeneous systems: Alice publishes her result z_0, yielding a copy with public-id z'_0. Bob downloads it, obtaining his local copy z''_0, followed by transformations f to prepare his input data u_0. By recording equivalances $z_0 \simeq z'_0$ and $z'_0 \simeq z''_0$ and the dependency $\mathtt{ddep}(z''_0, u_0)$ via f, a "seamless" trace is obtained in the common provenance store CPS.

III. Provenance of the Publication Process

With reference to our scenario, the goal of our architecture is to let Charlie trace the provenance of Bob's data products all the way to the primary data, in this case the inputs used by Alice. This requires the two independently generated graphs to be *connected* in such a way that Alice's inputs are reachable (by transitivity over a dependency relation, such as `ddep`) fromBob's outputs. As we noted earlier, however, this connectivity becomes problematic when the two systems use distinct naming schemes to address the data, and when additional intermediate transformations are involved, e.g. to ensure that Alice's data products are compatible with the input formats expected by Bob's workflow. Our approach to ensure that multiple traces "join at the seams" involves augmenting the provenance graphs with additional statements that capture the effect of publishing local data to a shared store, and downloading shared data for use in a new local context, for example a workflow.

Fig. 3 depicts the overall approach. The figure is organised into two primary layers, corresponding to a process and trace view of the scenario. The process view includes the two formally specified workflows, W_A and W_B (the workflows of Alice and Bob, respectively), as well as the "stitched" together workflow (middle-top) based on collected provenance information. Data objects (e.g., x_o and z_o) are bound to input and output channels at the beginning and end of workflow runs, and are subsequently stored in local repositories. The middle-center box shows the result of Alice publishing data onto a public data store, and Bob downloading (some or all of) the data into his local store, possibly transforming it using some function f, and using it as input to W_B. At the same time, provenance traces are captured, not only for the workflow executions T_A and T_B, but also for the data replication steps of the intermediate process. The latter information bridges the workflow traces, so that once they have all been published to the CPS, they can be seamlessly traversed, as shown at the bottom of Fig. 3.

This section formalizes these ideas by introducing: (i) a copy operator for publishing local data to a shared store, (ii) an operator for mapping provenance graphs expressed using a local model to the common model described in Section II, and (iii) an operator for publishing the transformed trace. To each of these operators we associate one or more new provenance statements. We then show how these augmented traces can be used to extend the queries introduced in the previous section.

A. Copy Operator

Let $\mathbf{D}$ be the universe of data items that may appear in a workflow, and S a *data store* into which data can be uploaded. S is a service that provides two operations:

- **store**: $r = S.\mathtt{put}(d)$,
- **retrieve** by key: $d = S.\mathtt{get}(r)$

where $d \in \mathbf{D}$, and r is a reference to d, local to S, such that $S.get(S.put(d)) = d$. The naming schemes used for the references can take various forms. A typical example consists of a web-accessible store S with URI-formatted references. We denote the set of all valid references for a store S with $\mathbb{R}_S$. Because of the differences in naming schemes across stores, a reference $r \in \mathbb{R}_S$ generated by S is not, in general, valid for a different store S', so $S'.\mathtt{get}(r) = \bot$ for all $S' \neq S$.

Operation	*Provenance Statements*
Alice:	
• (1) $\langle z_0, T_A\rangle = \mathtt{exec}(W_A(x_0))$ where T_A is W_A's execution trace	$b_t = \emptyset$
• (2) for each reference r in T_A: $r' = \mathtt{copy}(r, S_A, S_P)$, $\rho.\mathtt{put}(r, r')$	$b_t \leftarrow b_t \cup \{S_P.\mathtt{get}(r') \simeq_t S_A.\mathtt{get}(r)\}$
• (3) CPS.$\mathtt{put}(\Gamma_{A\to C}(T_A, \rho))$	
Bob:	
Let $r \in \mathbb{R}_{S_P}$ be the reference for z'_0 in S_P.	
• (4) $z''_0 = S_P.\mathtt{get}(r)$	$b_t \leftarrow b_t \cup \{z''_0 \simeq_t z'_0\}$
• (5) $u_0 = f(z''_0)$	$b_t \leftarrow b_t \cup \{\mathtt{ddep}(u_0, z''_0)\}$
• (6) $\langle v_0, T_B\rangle = \mathtt{exec}(W_B(u_0))$ where T_B is W_B's execution trace	
• (7) for each reference r in T_B: $r' = \mathtt{copy}(r, S_B, S_P)$, $\rho.\mathtt{put}(r, r')$	$b_t \leftarrow b_t \cup \{S_P.\mathtt{get}(r') \simeq_t S_B.\mathtt{get}(r)\}$
• (8) CPS.$\mathtt{put}(\Gamma_{B\to C}(T_B, \rho))$	

Fig. 4. Provenance publication process with corresponding provenance statements. The reference r for z'_0 before line (4) can be obtained from S_P in various ways, e.g., through a search/browse interface into S_P that is available to Bob.

Given two data stores S and S', one can copy a value from S to S' using its reference $r \in \mathbb{R}_S$:

$$\mathtt{copy}(r, S, S') = S'.\mathtt{put}(S.\mathtt{get}(r)) = r' \in \mathbb{R}_{S'}$$

where in general $\mathbb{R}_{S'}$ is different from, and incompatible with, $\mathbb{R}_S$, i.e., $S.\mathtt{get}(r') = \perp$. We consider two data items d and d' *trace-equidalent*, written $d \simeq_t d'$, if they are related to each other through a copy operation. Trace equivalence is captured by the following inference rule:

$$\frac{d{=}S.\mathtt{get}(r) \qquad r'{=}\mathtt{copy}(r, S, S') \qquad d'{=}S'.\mathtt{get}(r')}{d \simeq_t d'} \tag{1}$$

Note that the consequent of this rule is a provenance statement, which does not rely on any notion of value equality. Thus, we are only observing the copy operator in action.

B. General Data Transformation

In our scenario (Fig. 3), Bob transforms the data item z''_0 downloaded into his local store S_B into $u_o = f(z''_0)$, where f is a generic transformation (e.g., a format conversion), so that u_o can be used in his local workflow, W_B. The transformation f makes u_0 dependent on z''_0, but since f is in general a black-box operation, we cannot assume that $u_0 \simeq_t z''_0$. We can, however, make a weaker statement that u_0 is dependent upon z''_0. That is, for a data item d, we have:

$$\frac{d' = f(d)}{\mathtt{ddep}(d, d')} \tag{2}$$

again where the dependency asserted in the consequent is recorded as an additional provenance statement. For instance, in Fig. 3 we have: $z_0{\simeq_t}z'_0$, $z'_0{\simeq_t}z''_0$, and $\mathtt{ddep}(z''_0, u_0)$, which together imply that $\mathtt{ddep}^*(z_0, u_0)$, thereby "stitching" together T_A and T_B.

C. Mapping Data References in Traces

Exporting a trace T involves three operations, each performed by both Alice and Bob on their respective systems: (i) publishing all data that occur in T to a shared store S_P, (ii) mapping the trace itself to the common provenance model, and (iii) uploading the mapped trace to the CPS.

Step (i) is defined simply in terms of one copy operation for each d that occurs in T. Assuming that T contains data *references* $r \in \mathbb{R}_S$, i.e., which are valid for a local store S, this translates into $\mathtt{copy}(r, S, S_P) = r'$ for each r found in the trace. According to rule (1) above, each copy operation generates a provenance statement $d \simeq_t d'$ where $d = S.get(r)$ and $d' = S'.get(r')$. Additionally, we add the pair $\langle r, r'\rangle$ to a *renaming* map $\rho : \mathbb{R}_S \to \mathbb{R}_P$, which we use to replace local references with shared, public ones in the next step. If T contains data values d rather than references, then publishing d reduces to $r' = S'.put(d)$, and $d \simeq_t d'$ where $d' = S'.get(r')$.

Step (ii) is defined by mappings from each of the local provenance models to the common model, which we abstractly define via a function $\Gamma_{S\to P}(T_S, \rho) = T_P$ that maps a local trace T_S of workflow W_S to a shared, public version T_P. In the result T_P each reference r of T_S is replaced with $\rho(r)$.

Step (iii) is accomplished by uploading T (one trace each from Alice and Bob) to CPS, which behaves as just another store: CPS.$\mathtt{put}(T)$.

D. Queries over Augmented Traces

Table 4 summarizes our initial scenario, reformulated in terms of the operations introduced in this section. The rightmost column shows the corresponding new provenance statements, which we accumulate in the new "stitching" trace T_s.

These additional statements can now be used to extend $\mathtt{ddep}$ to include both the published versions of Alice and Bob's traces, and the statements in T_s:

$$\mathtt{ddep}^*_{\simeq}(d_1, d_2) :- \mathtt{ddep}^*(d_1, d_2).$$
$$\mathtt{ddep}^*_{\simeq}(d_1, d_2) :- \mathtt{ddep}^*(d_1, d), d \simeq_t d', \mathtt{ddep}^*(d', d_2).$$

where $d \simeq_t d' \in T_s$. This allows us to seamlessly answer provenance queries of the form $:-\mathtt{ddep}^*_{\simeq}(X, Y)$, e.g., between $z_0 \in T_A$ and $u_0 \in T_B$.

Ultimately, the consequence of our approach is that our third scientist, Charlie, has been able to compute the provenance of data products (produced via different workflows as well as workflow systems) as if they had been produced by the "virtual experiment" at the top of Fig. 3. Note also that we

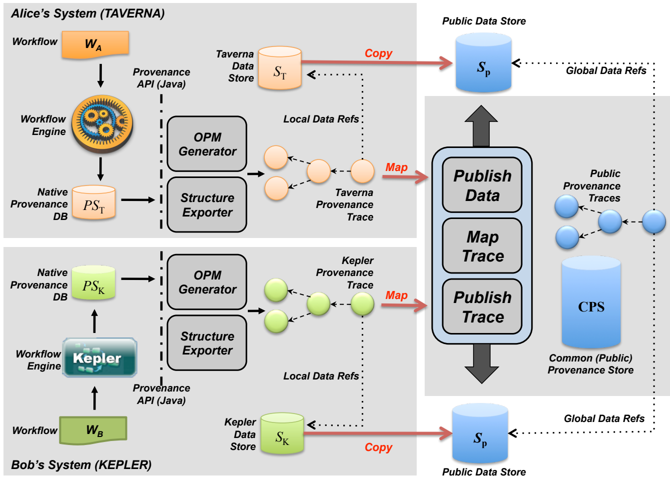

Fig. 5. DToL system architecture [12], implementing trace-level interoperability between Kepler and Taverna: data is published by copying from local stores S_T, S_K to the public store S_P, generating public-ids in the process. Local traces are mapped to a common model and published, replacing local data references with global ones in the common provenance store CPS.

can define a workflow specification for the virtual experiment: In this case, the virtual workflow consists of W_A and W_B combined via a "glue" workflow comprised of general publish and download operations as well as any transformation steps used by Bob. Further, if Bob had not applied a transformation f, or if this transformation is repeatable (i.e., via automation), then Charlie could execute the virtual experiment (e.g., over new input data) even though W_A and W_B are implemented within otherwise "non-interoperable" workflow systems. One can repeteadly apply the idea to multiple trace-generating workflows with shared data, obtaining larger tree-structured virtual experiments.

IV. Prototype Implementation

The approach described in the previous section brings both completeness to Charlie's queries, and fairness to all the contributors along the paths in this "Data Tree of Life" [12].

In the context of a DataONE project [11] we have implemented a prototype provenance sharing architecture that realises the abstract architecture of Fig. 5, using the Kepler [17] and Taverna [15] scientific workflow systems as testbeds.

To test the hypothesis that traces from different workflows and systems can be joined together using the approach described in Section III-D, we have taken the First Provenance Challenge [24], [23] workflow W_{PC}, and split it into two parts W_{PC_1} and W_{PC_2}. Both sub-workflows were encoded in Kepler and Taverna, and combined in a "crossing-over" manner in several ways. Overall, three different models of computation (MoCs) and corresponding designs were used for each of W_{PC_1} and W_{PC_2}: Taverna, "conventional" Kepler [4], and Kepler/COMAD [20], a novel MoC with its own provenance recorder to handle fine-grained dependencies within nested data collections [9], [6]. In the implementation, we have used a combination of native provenance models, available from both workflow systems, as well as OPM provenance graphs derived from those models (Fig. 5). The former include all the read/write observable events, as well as the information about the workflow structure, while the latter contains instances of the data dependency relation `ddep`. A description of how these relations are obtained for each of the two systems is beyond the scope of the paper; in general, they are either asserted explicitly or inferred, depending on the additional knowledge that is available to the provenance component.

The Kepler and Taverna native traces differ in their data model, but can be mapped to the common provenance model with relatively little effort. The mapping tools obtain their source trace information using an API that operators over a relational representation of the native provenance model, as well as over the OPM provenance graphs. In the case of Kepler and Taverna, the common model turns out to be less expressive than the native model, specifically with regards to the representation of nested lists, the only data structure used in Taverna (and one of many data structures supported by Kepler). This is not a problem, however, because the representation of the data structure (essentially a tree) is still maintained in the common data store, when the data is published through a "deep" version of the `copy` operator, while only the dependencies among list elements (e.g., the i-th element of an output list may depend on the j-th element of an input list) needs to be represented in the common model.

The Kepler and Taverna storage model follow our conceptual model closely, in that all process data is stored in an internal, system-dependent data store. Traces typically contain data references, but can also have embedded data values (for small data sizes). This early architectural choice makes it possible to avoid storing value equivalence statements altogether, because for each statement $d \simeq_t d'$ inferred when d is published, the trace contains local reference r, while r' is obtained from S_P upon upload. Thus, all that is needed is to record the pair $\langle r, r' \rangle$ into the renaming map ρ, so that the published trace will contain the new reference. When d' is reused, by symmetry r' will appear in the corresponding trace, which will then join naturally with the first.

Once the two traces have been published as described in the previous section, we were able to verify that the answers to the closure queries on CPS were consistent with those obtained by manually "joining" the partial answers obtained from each of the two systems on the shared data values. For this test, we used the available provenance query models available for each of the two systems [21], [6], [5], while on the CPS, lacking a dedicated provenance query facility, ad-hoc queries were constructed for the sake of this exercise.

V. Discussion of Related Work and Conclusions

We have presented an abstract model for scientific workflows and their associated provenance traces. Trace graphs can be seen as (homomorphic) instances of the given workflow graphs for which provenance is to be recorded. The model is deliberately simple and provides a least common denominator that can be easily extended and accommodate various models of computation. While building upon OPM concepts, our model also links an abstract workflow specification W with concrete provenance traces T resulting from the execution of W. We have developed our model to provide end-to-end user support for "implicit collaborations", i.e., where scientist build upon each others work by data sharing and then using published data products. We have described and implemented a scenario that "stitches together" provenance traces from different users, employing different workflow systems (Kepler, Taverna) with different underlying workflow languages and data models. Our implementation demonstrates that e-Science collaborations are possible "through the data", and that these collaborations can be correctly tracked, e.g., to provide automatic attribution support, i.e., where scientists whose data (and indirectly: workflows) have been used can be given credit through acknowledgements that are derived from public data dependency (provenance trace) graphs. [9] describe a framework and similar ideas as those presented here, including the notion of a graph over multiple workflow runs, highlighting different types of relationships among runs based on the level of data sharing between them. However, unlike the present work, [9] does not deal with the problem of "broken" traces when integrating provenance across different provenance models and workflow systems. In [3], [2] the idea of implicit collaboration through shared provenance has been discussed and a data model and query language have been introduced. In contrast, the present paper for the first time considers in detail and then solves the problem of instance-level bridging of provenance information. The problem has emerged previously [24], [25] but only now has a prototype been developed [12] that addresses the general problem, using concrete scientific workflow systems as a testbed.

In addition to having global provenance traces, our approach also facilitates a new way of achieving workflow system interoperability, i.e., by going "through the data" instead of waiting for a "one-size-fits-all" language standard for scientific workflow systems.[5] We will explore the practicality of this approach in future work.

A number of proposals are emerging for modelling and managing *Research Objects* (RO) [8], which augment primary and derived data with ancillary elements, such as a description of the experimental process of data generation and transformation, publications based on the data products, and more, which can enable improved capabilities for data interpretation. In our work we focus on a particular type of descriptive metadata that may find its place in an RO, namely *provenance traces*, which describe the dependencies of data products obtained through a process consisting of a sequence of transformations, from other data elements that participated in the process, namely its inputs along with any intermediate results.

References

[1] *Nature, Special Issue on Data Sharing*, volume 461. Sept. 2009.

[2] I. Altintas, M. K. Anand, S. Bowers, B. Ludäscher, S. Sun, P. Missier, and P. M. Sloot. A Data Model for Analyzing User Collaborations in Workow-Driven eScience, 2010. submitted for publication.

[5] Existing standards developed in other domains, e.g., BPEL have not (at least so far) been widely embraced by the scientific community.

[3] I. Altintas, M. K. Anand, D. Crawl, A. Belloum, P. Missier, C. Goble, and P. Sloot. Understanding Collaborative Studies Through Interoperable Workflow Provenance. In *IPAW*, Troy, NY, 2010.

[4] I. Altintas, O. Barney, and E. Jaeger-Frank. Provenance Collection Support in the Kepler Scientific Workflow System. In *Intl. Provenance and Annotation Workshop (IPAW)*, pages 118–132, 2006.

[5] M. K. Anand, S. Bowers, and B. Ludäscher. Techniques for efficiently querying scientific workflow provenance graphs. In *EDBT*, pages 287–298, 2010.

[6] M. K. Anand, S. Bowers, T. M. McPhillips, and B. Ludäscher. Exploring Scientific Workflow Provenance Using Hybrid Queries over Nested Data and Lineage Graphs. In *SSDBM*, pages 237–254, 2009.

[7] Z. Bao, S. Cohen-Boulakia, S. Davidson, A. Eyal, and S. Khanna. Differencing Provenance in Scientific Workflows. In *ICDE*, Mar. 2009.

[8] S. Bechhofer, D. De Roure, M. Gamble, C. Goble, and I. Buchan. Research Objects: Towards Exchange and Reuse of Digital Knowledge. In *Int'l Workshop on Future of the Web for Collaborative Science (FWCS) – WWW'10*, 2010.

[9] S. Bowers, T. McPhillips, M. W. Wu, and B. Ludäscher. Project Histories: Managing Data Provenance Across Collection-Oriented Scientific Workflow Runs. In *Data Integration in the Life Sciences*, volume 4544 of *Lecture Notes in Computer Science*, pages 122–138. Springer Berlin / Heidelberg, 2007.

[10] A. Chapman and H. V. Jagadish. Issues in Building Practical Provenance Systems. *IEEE Data Eng. Bull.*, 30:38–43, 2007.

[11] Data Observation Network for Earth (DataONE), 2010. http://dataone.org.

[12] Data Tree of Life (DTOL) Project: Storing and Querying Deep Provenance, 2010. public release scheduled for Sept. 1, 2010 at http://sites.google.com/site/datatolproject.

[13] T. Heinis and G. Alonso. Efficient Lineage Tracking For Scientific Workflows. In *Proceedings of the 2008 ACM SIGMOD conference*, pages 1007–1018, 2008.

[14] T. Hey, S. Tansley, and K. Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009.

[15] D. Hull, K. Wolstencroft, R. Stevens, C. A. Goble, M. R. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34:729–732, 2006.

[16] E. A. Lee and D. G. Messerschmitt. Synchronous Data Flow. In *Proceedings of the IEEE*, volume 75, pages 1235–1245, 1987.

[17] B. Ludäscher, I. Altintas, and C. Berkley. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice and Experience*, 18:1039–1065, 2005.

[18] B. Ludäscher, I. Altintas, S. Bowers, J. Cummings, T. Critchlow, E. Deelman, D. D. Roure, J. Freire, C. Goble, M. Jones, S. Klasky, T. McPhillips, N. Podhorszki, C. Silva, I. Taylor, and M. Vouk. Scientific Process Automation and Workflow Management. In A. Shoshani and D. Rotem, editors, *Scientific Data Management: Challenges, Existing Technology, and Deployment*. Chapman & Hall/CRC, 2009.

[19] B. Ludäscher, N. Podhorszki, I. Altintas, S. Bowers, and T. M. McPhillips. Models of Computation and Provenance: The RWS Approach. In *Concurrency and Computation: Practice & Experience* [23], pages 507–518.

[20] T. McPhillips, S. Bowers, and B. Ludäscher. Collection-Oriented Scientific Workflows for Integrating and Analyzing Biological Data. In *Data Integration in the Life Sciences (DILS)*. Springer, 2006.

[21] P. Missier, N. Paton, and K. Belhajjame. Fine-grained and efficient lineage querying of collection-based workflow provenance. In *EDBT*, Lausanne, Switzerland, 2010.

[22] L. Moreau, B. Clifford, J. Freire, Y. Gil, P. Groth, J. Futrelle, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, Y. Simmhan, E. Stephan, and J. V. den Bussche. OPM: The Open Provenance Model Core Specification (v1.1). http://openprovenance.org/, December 2009.

[23] L. Moreau and et al. The First Provenance Challenge, Special Issue of CCPE. *Concurrency and Computation: Practice and Experience*, 20:409–418, Apr. 2008.

[24] First Provenance Challenge. http://twiki.ipaw.info/bin/view/Challenge/FirstProvenanceChallenge, 2006. Washington, DC.

[25] Third Provenance Challenge. http://twiki.ipaw.info/bin/view/Challenge/ThirdProvenanceChallenge, 2009. University of Amsterdam.

[26] I. J. Taylor, E. Deelman, D. Gannon, and M. S. Shields, editors. *Workflows for eScience*. Springer, 2007.

[27] Y. Zhou and E. A. Lee. Causality Interfaces for Actor Networks. *ACM Transactions on Embedded Computing Systems*, 7(3):1–35, 2008.