# Using Mediation to Achieve Provenance Interoperability

Tommy Ellkvist
Linköpings Universitet

David Koop
University of Utah

Juliana Freire
University of Utah and Linköpings Universitet

Cláudio Silva
University of Utah

Lena Strömbäck
Linköpings Universitet

## Abstract

*Provenance is essential in scientific experiments. It contains information that is key to preserving data, to determining its quality and authorship, and to reproducing as well as validating the results. In complex experiments and analyses, where multiple tools are used to derive data products, provenance captured by these tools must be combined in order to determine the complete lineage of the derived products. In this paper, we describe a mediator-based architecture for integrating provenance information from multiple sources. This architecture contains two key components: a global mediated schema that is general and capable of representing provenance information represented in different model; and a new system-independent query API that is general and able to express complex queries over provenance information from different sources. We also present a case study where we show how this model was applied to integrate provenance from three provenance-enabled systems and discuss the issues involved in this integration process.*

## 1 Introduction

One of the greatest research challenges of the 21st century is to effectively understand and leverage the growing wealth of scientific data. To analyze these data, complex computational processes need to be assembled, often requiring the combination of loosely-coupled resources, specialized libraries, distributed computing infrastructure, and Web Services.

Workflow and workflow-based systems have recently grown in popularity within the scientific community as a means to assemble these complex processes [4, 5, 13, 15, 18, 23–27]. Not only do they support the automation of repetitive tasks, but they can also systematically capture provenance information for the derived data products [3]. The provenance (also referred to as the audit trail, lineage, and pedigree) of a data product contains information about the process and data used to derive the product [6, 22]. It provides important documentation that is key to preserving the data, to determining its quality and authorship, and to reproducing as well as validating the results. These are all important requirements of the scientific process.

Most workflow systems support provenance capture. But each adopts its own data and storage models [3, 6]. These range from specialized Semantic Web languages (e.g., RDF and OWL) and XML dialects that are stored as files in the file system, to tables stored in relational databases. These systems have also started to support queries over provenance information [16]. Their solutions are closely tied to the data and storage models they adopt and require users to write queries in languages like SQL [1] and SPARQL [9, 14, 31]. Consequently, determining the *complete* lineage of a data product derived by multiple such systems requires that information from these different systems and/or their query interfaces be integrated. Consider the scenario shown in Figure 1. In order to determine the provenance of the visualization, it is necessary to combine the provenance information captured both by the workflow system used to derive the simulation results and by the workflow-based visualization system to derive the image. Without combining this information, it is not possible to answer important questions about the resulting image, such as for example, the specific parameter values used in the simulation.

**Contributions and Outline.** In this paper, we address the problem of provenance interoperability in the context of scientific workflow systems. Provenance interoperability is a topic that has recently started to receive attention in the scientific workflow community [8, 19]. In the Second Provenance Challenge (SPC), several groups collaborated in an exercise to explore interoperability issues among provenance models [19]. Part of the work described in this paper was developed in the context of the SPC.

We argue that although existing provenance models differ in many ways, they all share an essential type of information: the provenance of a given data product consists of a *causality graph* whose nodes correspond to processes

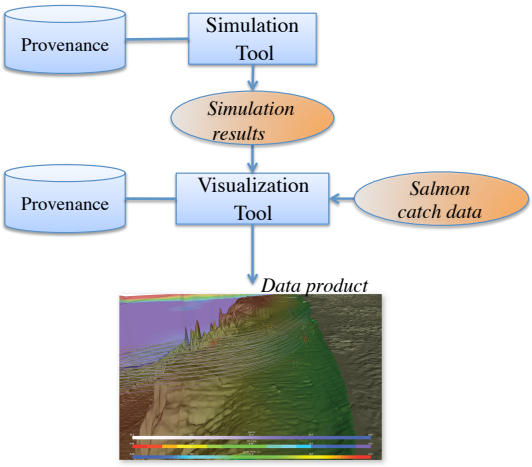


**Figure 1.** The visualization on the right shows salmon catch information superimposed with a model of the currents in the mouth of the Columbia River. The simulation was run on a cluster of computers using a grid-enabled workflow system. A workflow-based visualization system was then used to display the simulation together with the salmon catch data.

and data products, and edges correspond to either data or data-process dependencies. Inspired by previous works on information integration [28], we propose a mediator architecture which uses the causality graph as the basis for its global schema for querying disparate provenance sources (Section 2). The process of integrating a provenance source into the mediator consists of the creation of a wrapper that populates the global (mediated) schema with information extracted from the source. As part of the mediator, we provide a query engine and API that support transparent access to multiple provenance sources. We evaluate the effectiveness of this approach by applying it to integrate provenance information from three systems (Section 3). We discuss our experiences in implementing the system and its relationship to recent efforts to develop a standard provenance model.

## 2 A Mediation Approach for Integrating Provenance

Information mediators have been proposed as a means to integrate information from disparate sources. A mediator selects, restructures, and merges information from multiple sources and exports a global, integrated view of information in these sources [28]. In essence, it abstracts and transforms the retrieved data into a common representation and semantics. An information mediator consists of three key components (see Figure 2): a global schema that is exposed to the users of the system; a query rewriting mechanism that translates user queries over the global schema into queries over the individual data sources; and wrappers that access data in the sources and transform them into the model of the mediator.

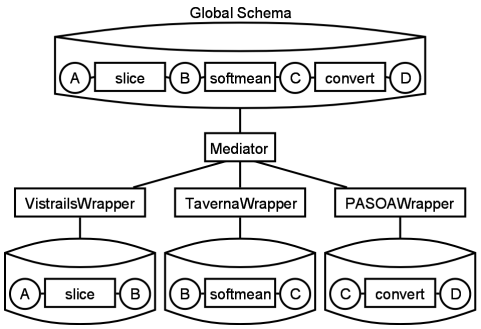


**Figure 2.** Mediator architecture used to integrate three provenance models. Queries over the global schema are translated by the wrappers into queries over the provenance sources, which are then executed and their results returned to the mediator. In this example, pieces of a complex workflow (*slice*, *softmean* and *convert*) were executed by the workflow systems. *A*, *B*, *C* and *D* are data items.

In what follows, we describe the mediator architecture we developed for integrating provenance information derived by scientific workflow systems. In Section 2.2, we present the global schema used and in Section 2.3 we discuss the query API supported by our mediator. Details about the wrappers are given later, in Section 3, where we describe a case study which shows that the data model and query API can be effectively used to support queries over (real) provenance data derived by different systems.

### 2.1 Scientific Workflows and Provenance

Scientific workflow and workflow-based systems have emerged as an alternative to ad-hoc approaches for constructing computational scientific experiments. They provide a simple programming model whereby a sequence of tasks (or modules) is composed by connecting the outputs of one task to the inputs of another. Workflows can thus be viewed as graphs, where nodes represent modules and edges capture the flow of data between the processes.

In the context of scientific workflows, provenance is a record of the derivation of a set of results. There are two distinct forms of provenance [2]: prospective and retrospective. *Prospective provenance* captures the specification of the workflow—it corresponds to the *steps that need to be followed* (or a recipe) to generate a data product or class of data products. *Retrospective provenance* captures the *steps that were executed* as well as information about the execution environment used to derive a specific data product—a detailed log of the execution of the workflow.

An important piece of information present in workflow provenance is information about *causality*: the dependency relationships among data products and the processes that generate them. Causality can be inferred from both prospective and retrospective provenance and it captures the se-

quence of steps which, together with input data and parameters, caused the creation of a data product. Causality consists of different types of dependencies. Data-process dependencies (e.g., the fact that the visualization in Figure 1 was derived by a particular workflow run within the visualization tool) are useful for documenting data generation process, and they can also be used to reproduce or validate the process. For example, it would allow new visualizations to be derived for different input data sets (i.e., different simulation results). Data dependencies are also useful. For example, in the event that the simulation code used to generate simulation results is found to be defective, data products that depend on those results can be invalidated by examining data dependencies.

Although different workflow systems use different data models, storage systems, and query interfaces, they all represent the notion of causality using a directed acyclic graph (DAG). In this graph, vertices are either data products or processes and the edges represent dependencies among them. As we describe below, the causality graph forms the basis for the global schema used in our mediator architecture.

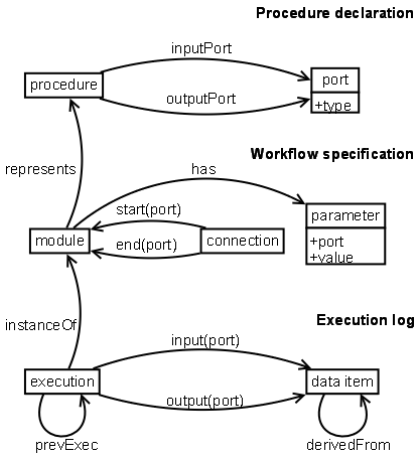


**Figure 3.** Overview of the Scientific Workflow Provenance model. Boxes represents entity types and arrows represents relationships between entities. Relationships can have attributes, shown after the relationship name. The *procedure declaration* specifies a list of typed input/output ports for each procedure—the signature of the procedure. The *workflow specification* contains the modules, connections and parameters that makes up the workflow. The *execution log* contains a record of executions of processes and used data items.

## 2.2 A Data Model for Scientific Workflow Provenance

A central component of our mediator is a general provenance model, the *Scientific Workflow Provenance Data Model (SWPDM)*. The model captures entities and relationships that are relevant to *both* prospective and retrospective provenance, i.e., the definition and execution of workflows, and data products they derive and consume. As a result, besides queries over provenance, our model also supports direct queries over workflow specifications. As we discuss later, this is an important distinction between SWPDM and the Open Provenance Model [17].

The entities and relationships of the SWPDM are depicted in Figure 3. At the core of the model is the *operation* entity, which is a concrete or abstract data transformation, represented in three different layers in the model: *procedure*, which specifies the type of an operation; *module*, which represents an operation that is part of an abstract process composition; and *execution*, which represents the concrete execution of an operation.

A *data item* represents a data product that was used or created by a workflow. A *procedure* represents an abstract operation that uses and produces data items. A *procedure declaration* is used to model procedures together with a list of supported *input* and *output ports*, which have types (e.g., integer, real). It describes the signature of a module. A port is a slot from/to which a procedure can consume/output a data item. A *workflow specification* consists of a graph that defines how the different procedures that compose a workflow are orchestrated, i.e., a set of *modules* that represent procedures and *connections* that represent the flow of data between modules. *Parameters* model pre-defined input data on specific module ports. The *execution log* consists of concrete *executions* of procedures and the data items used in the process.

Two points are worthy of note in our choice of entities. A workflow is assigned input data before execution, but besides these inputs, a module may also have parameters which serve, for example, to set the state of the module (e.g., the scaling factor for an image). From a provenance perspective, parameters are simply data items, but by using a finer grained division, we can support more expressive queries. Furthermore, by modeling workflow *connections* as separate from the data items, we are able to query for the structure of a workflow directly. These connections can then be used to answer queries like "which data items passed through this connection".

Another key component of workflow provenance is *user-defined information*. This includes documentation that cannot be automatically captured but records important decisions and notes. This information is often captured in the form of annotations, which are supported by most workflow systems. In our model, an *annotation* is a property of entities in the model. Annotations can add descriptions to entities that can later be used for querying. Examples are execution times, hardware/software information, workflow creators, descriptions, labels, etc. In the model, we assume that any entity can be annotated.

| Function | Description |
|---|---|
| outputOf(data) | get execution that created $data$ |
| inputOf(data) | get execution that used $data$ |
| output(execution) | get data created by $execution$ |
| input(execution) | get data used by $execution$ |
| execOf(execution) | get the module representing $execution$ |
| getExec(module) | get the executions of $module$ |
| represents(module) | get the process that $module$ represents |
| hasModule(process) | get module that represents $process$ |
| derivedFrom(data) | get data products used to create $data$ |
| derivedTo(data) | get data products derived from $data$ |
| prevExec(execution) | get execution that triggered $execution$ |
| nextExec(execution) | get executions triggered by $execution$ |
| upstream(x) | transitive closure operation, where $x$ is a module, execution or data |
| downstream(x) | transitive closure operation,where $x$ is a module, execution or data |

**Table 1.** List of API functions

## 2.3 Querying SWPDM

We have designed a new query API that operates on the entities and relationships defined in the SWPDM model. This API provides basic functions that can serve as the basis to implement a high-level provenance query language. In order to integrate a provenance source into the mediator, one must provide system-specific bindings for the API functions. Figure 4 illustrates the bindings of the API function *getExecutedModules*($wf\_exec$) for three distinct provenance models. As we describe in Section 3.2, each binding uses the data model and query language supported by the underlying system.

Note that a given workflow system may not capture all the information represented in our model (see Figure 3). In fact, the systems we used in our case study only partially cover this model (see Section 3). Thus, in designing API bindings for the different systems, the goal is to extract (and map) as much information as possible from each source.

Some of the API functions are summarized in table 1.[1] Since the API operates on a graph-based model, a key function it provides is graph traversal. The graph-traversal functions are of the form *getBFromA*, which traverse the graph from A to B. For example, *getExecutionFromModule* traverses the graph from a module to its executions, i.e., it returns the execution logs for a given module.

Additional functions are provided to represent common provenance operations which have to do with having both data- and process-centric views on provenance. For example, *getParentDataItem* returns the data items used to create a specific data item. Such parent/child functions also exist for modules and executions.

Note that the API contains redundant functions, e.g., *getParentExecution* can be also achieved by combining *getExecutionFromOutData* and *getInDataFromExecution*. If data items are not recorded by the given provenance system, the binding for *getParentExecution* might use another path to find the previous execution. Although these redundant functions are needed for some provenance systems, they can make query construction ambiguous. It is up to the wrapper designer to implement these based on the capabilities of the underlying system and its data model.

[1] The complete API, and the bindings to Taverna, PASOA and VisTrails, are available for download in:
http://twiki.ipaw.info/pub/Challenge/VisTrails2/api.zip

**VisTrails (XPath)**

```
def getExecutedModules(self, wf_exec):
    newdataitems = []
    q = '//exec[@id="' + wf_exec.pid.key + '"]/@moduleId'
    dataitems = self.logcontext.xpathEval(q)
```

**Pasoa (XPath)**

```
def getExecutedModules(self, wf_exec):
    q = "//ps:relationshipPAssertion[ps:localPAssertionId='" +
            wf_exec.pid.key + "']/ps:relation"
    dataitems = self.context.xpathEval(q)
```

**Taverna (SPARQL)**

```
def getExecutedModules(self, wf_exec):
    "  "
    q = '''
    SELECT ?mi
    FROM <''' + self.path + '''>
    WHERE
      { <''' + wf_exec.pid.key + '''>
        <http://www.mygrid.org.uk/provenance#runsProcess> ?mi }
    '''
    return self.processQueryAsList(q, pModuleInstance)
```

**Figure 4.** Implementations of the *getExecutedModules* function for different provenance systems.

Provenance queries often require transitive closure operations that traverse the provenance graph in order to trace dependencies forward or backward in time. Our API supports transitive closure queries in both directions: *upstream*, which traces provenance backwards (e.g., what derived a given data item); and *downstream*, which traces provenance forward (e.g., what depends on a given data item).

The *upstream* function is represented as *upstream*($x$) where $x$ is an entity and the output is all its dependencies. There is also a corresponding downstream function. Since these queries can be expensive to evaluate, it is useful to have additional attributes that can prune the search space. A *depth restriction* specifies the maximum depth to explore, and a *scope restriction* specifies entities that should be ignored in the traversal. These restrictions are capture by the function $y = upstream(x, depth, scope)$, and the corresponding downstream.

There are additional operations including *getAll* which returns all entities of a specific type and is used to create local caches of the provenance store. Two operations handle annotations: *getAllAnnotated* returns entities containing a specific annotation and is used for queries on annotations; and *getAnnotation* returns all annotations of an entity.

## 2.4 Discussion and Related Work

**Mediator Architecture.** There are different approaches to mediation [11]. In this paper, we explore the virtual approach, where information is retrieved from the sources

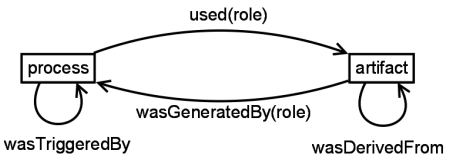


**Figure 5.** The basic concepts in the OPM. It maps directly to the execution log of our SWPDM. The entity types and relationships have different names but represent the same concepts. Here, data items are called *artifacts* and ports are called *roles*.

when queries are issued by the end user. Another approach is to materialize the information from all sources in a warehouse. The trade-offs between the two are well-known. Whereas warehousing leads to more efficient queries, not only can data become stale, but also the storage requirements can be prohibitive. Nonetheless, our architecture can also be used to support a warehousing solution: once the wrappers are constructed, queries can be issued that retrieve all available data from the individual repositories to be stored in a warehouse.

**Other Approaches to Provenance Interoperability.** Thirteen teams participated in the Second Provenance Challenge, whose goal was to establish provenance interoperability among different workflow systems. Most solutions mapped provenance data from one system onto the model of another. Although all teams reported success, they also reported that the mapping process was tedious and time consuming. To create a general solution, such approaches would require $n^2$ mappings, where $n$ is the number of systems being integrated. In addition, they require that all data from the different systems be materialized, which may not be practical. In contrast, by adopting a mediator-based approach, only $n$ mappings are required—one mapping between each system and the global schema. And as discussed above, both virtual and materialized approaches are supported by the mediator architecture.

**The Open Provenance Model.** One of the outcomes of the Second Provenance Challenge was the realization that it is indeed possible to integrate provenance information from multiple systems, and that there is substantial agreement on a core representation of provenance [19]. Armed with a better understanding of the different models, their query capabilities, and how the can interoperate, Moreau et al. [17] proposed a standard model for provenance: the Open Provenance Model (OPM). The OPM defines a core set of rules that identify valid inferences that can be made on provenance graphs. Important goals shared by the OPM and SWPDM include: simplify the exchange of provenance information; and allow developers to build and share tools that operate on the model. However, unlike the SWPDM, OPM supports the definition of provenance for any "thing", whether produced by computer systems or not. In this sense, OPM is more general than SWPDM. However, by focusing on workflows and modeling workflow representations, SWPDM allows a richer set of queries that correlate provenance of data products and the specification of the workflows that derived them.

We should note that many of the concepts model by OPM can also be modeled in SWPDM. Figure 5 shows the OPM representation of the relationships between processes and data items. This representation can be mapped directly to the execution log of SWPDM which contains the provenance graph. The OPM also contains a number of inferred relationships, namely transitive versions of the basic relationships. We support these by using transitive functions in the API (upstream and downstream, see Section 2.3). The OPM has an optional part component to represent time. SWPDM supports time as an annotation which means we can support any number of representations of time including the one in the OPM. In the OPM there is also a notion of an *agent* that is responsible for executions. These can also be modeled as annotations.

# 3 Case Study: Integrating Provenance from Three Systems

We implemented the mediator architecture described in Section 2 as well as bindings for the query API using three distinct provenance models: VisTrails [27], PASOA [10], and Taverna [24]. Figure 2 shows a high-level overview of our mediator.

In order to assess the effectiveness of our approach, we used the workflow and query workload defined for the Provenance Challenge [19]. The workflow entails the processing of functional magnetic resonance images, and the workload consists of typical provenance queries, for example: what was the process used to derive an image? which data sets contributed to the derivation of a given image? For a detailed description of the workflow and queries, see [19]. Before we describe our implementation and experiences, we give a brief overview of the provenance models used in this case study.

## 3.1 Provenance Models

*VisTrails* is a scientific workflow system developed at the University of Utah. A new concept introduced with VisTrails is the notion of *provenance of workflow evolution* [7]. In contrast to previous workflow systems, which maintain provenance only for derived data products, VisTrails treats the workflows (or pipelines) as first-class data items and keeps their provenance. The availability of this additional information enables a series of operations which simplify exploratory processes and foster reflective reasoning, for example: scientists can easily navigate through the space of workflows created for a given exploration task; visually

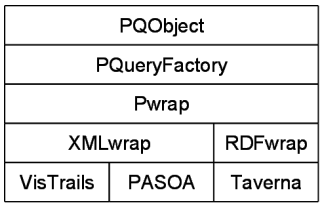


**Figure 6.** The different layers in the implementation of the query API. Queries are processed starting from a known entity (*PQObject*) and traversed by using relationship edges in the mediator (*PQueryFactory*). The mediator executes the query using the wrapper interface (*Pwrap*), that in turn executes the query using a specific wrapper for each data source.

compare workflows and their results; and explore large parameter spaces. VisTrails captures both prospective and retrospective provenance, which are stored uniformly either as XML files or in a relational database.

*Taverna* is a workflow system used in the myGrid project, whose goal is to leverage semantic web technologies and ontologies available for Bioinformatics to simplify data analysis processes in this domain. Prospective provenance is stored as Scufl specifications (an XML dialect) and retrospective provenance is stored as RDF triples in a MySQL database. Taverna assigns globally unique LSID [30] identifiers to each data product.

*PASOA* (Provenance Aware Service Oriented Architecture) relies on individual services to record their own provenance. The system does not model the notion of a workflow, instead, it captures assertions produced by services which reflect the relationships between services and data. The complete provenance of a task or data product must be inferred by combining these assertions and recursively following the relationships they represent. PReServ, an implementation of PASOA, supports multiple backend storage systems, including files and relational database, and queries over provenance can be posed using its Java-based query API or XQuery.

### 3.2 Building the Mediator

Our model was developed based on a study of the three models above. Each of the models covers only part of the mediated model. For both Taverna and PASOA, only the execution log was available: the workflow specifications were not provided. VisTrails stores both workflow specification and the execution log. It uses a normalized provenance model where each execution record points to the workflow specification where it came from. But the system does not explicitly identify data items produced in intermediate steps of workflow execution.

**Implementation.** We implemented the mediator-based architecture in Python. The different components are shown in Figure 6. *PQObject* represents a concept in the global schema; *PQueryFactory* the mediator; and *XMLwrap* (for XML data using XPath) and *RDFwrap* (for RDF data using SPARQL) are abstract wrappers. The concrete wrappers are in the bottom layer: for PASOA and VisTrails, wrappers were built by extending XMLWrap; and for Taverna using RDFWrap. These wrappers implement API functions defined in Section 2.3 using the query interfaces provided by each system. Due to space limitations, we omit the details of the API bindings. The source code for the mediator and bindings is available at http://twiki.ipaw.info/pub/Challenge/VisTrails2/api.zip.

**Using and Binding the API.** Here we show some examples of how the API functions can be used to construct complex queries. Since each function applies to an entity, as a first step, it is necessary to obtain a handle for the entity instance of interest. For example, to access the handle for a port, the node corresponding to that port needs to be extracted from the global schema:

$$m = \mathit{pqf.getNode}(\mathit{pModule}, \mathit{moduleId}, \mathit{store1.ns})$$

The *getNode* method accesses the components of an instance of the PQueryFactory (*pqf*), and it requires the specification of the entity type (*pModule*), unique entity identifier (*moduleId*), and a specific provenance store (*store1.ns*). Once the handle has been retrieved, the provenance graph can be traversed by invoking an API function (i.e., to get all executions of a module $e$, we would call $e.\mathit{getDataItemFromExecution}()$).

There are some issues that need to be considered during query construction. First, there are different ways to represent a workflow. For example: modules can contain scripts whose parameters are not properly exposed in the module signature; and parameters can be modeled as input ports. Thus, the actual implementation of a query depends on the chosen representations and semantics implemented within the wrapper. Another issue concerns the specification of data items and inputs. Some systems record the concrete data item used as input while others represents data items by names that are stored as parameters to modules that use the data item. This must also be resolved during the wrapper design, by modeling data items, inputs and parameters consistently across distinct provenance stores.

Consider, for example, the provenance challenge query 6 which asks for "the process that led to the image Atlas X Graphic", i.e., the provenance graph that led to this specific image. In VisTrails, the image is identified by *atlas-x.gif* that is specified as a parameter to a `FileSink` module in the workflow specification. Taverna uses the port name *convert1_out_AtlasXGraphic*—data items are handled internally and are not saved to disk. PASOA uses the string *atlas-x.gif* that is passed between the two modules. This means that the starting handle obtained for the different sys-

tems will be of different types. In VisTrails, it is a parameter of a module; in Taverna it is an output port of a module; and in PASOA it is a data item. Thus, once the handle for "Atlas X Graphic" is obtained, different methods need to be used in order to compute the required upstream modules. For VisTrails, we find the `FileSink` module that contains the file name, then find the executions of that module, and finally compute the upstream of those executions. For Taverna, we obtain the executions associated with the output port and compute the upstream. For PASOA, we get the executions that created the data item and compute the upstream.

### 3.3 Experiences

In what follows, we discuss some of our findings during this case study as well as issues encountered while implementing the mediator and how they have been addressed.

*Mismatches between models.* We found only a few mismatches between the models we considered. This indicates that it is possible to create a general provenance model that is effective. One mismatch was due to the labeling of modules. In Taverna, modules are assigned user-defined names whereas VisTrails uses the name of the module type. This can confuse users, for example, a VisTrails user looking at a Taverna workflow may falsely assume that the label of a module represents its type. In our implementation, this was resolved by creating an abstract *label* that describes the module. We mapped this label to the module label for Taverna and to the module type for VisTrails.

*Implementing wrappers.* Wrapper construction is a time-consuming process. A wrapper needs to provide a wide range of access methods so that it can efficiently support general queries. This is in contrast to $n \times n$ mapping approaches (see Section 2.4), where specifying an individual mapping between two models can be much simpler. The problem with the latter solution is that more mappings are required for a comprehensive integration system.

*Transitive closure.* The implementation of transitive closure (i.e., the upstream/downstream functions) was problematic. Neither XPath nor SPARQL supports transitive closure operations natively. Possible workarounds are to use recursive functions for XQuery [29] and to add inference rules to the SPARQL engine. However, in both solutions, queries are expensive to evaluate. To overcome this problem, we used two other methods: (1) wrappers implement the closure as recursive calls of single-step operations—this method is system-independent but not very efficient; (2) cache the transitive relations on the mediator and perform the closure computation on the client side—although fast, this method may not be scalable since it requires loading transitive data from the data stores. Support for transitive closure computation is a topic that has been well-studied in deductive databases (see e.g., [12]) and we plan to experiment with alternative and more efficient approaches.

We note that the optimizations for limiting the scope and depth of transitive closures were useful for some of the provenance challenge queries. For example, scope restrictions were used to discard everything before *softmean* in Query 2; and in Query 3, the *stages* can be interpreted as depth, in which case a depth restriction is useful.

*Provenance interoperability and data identifiers.* Another problem that needs to be addressed when reconstructing the provenance of a data item across multiple provenance stores is how to identify the data items used. If a data item is the product of a sequence of workflows, and the connections among the workflows are represented only by the data items they exchange, versioning of the files is required to correctly connect the provenance records. Furthermore, identifiers used in different models were of different types. As a result, one could not assume any specific format or that the identifiers were unique. We solved this by appending the source model type to the identifier and treat the identifier itself as a string. Essentially, our mediator creates its own namespace. It would be preferable to use the namespace already present in the source data but since they were of different types or not present at all, this was not possible. Another potential solution for this problem would be the adoption a common identifier type for provenance entities, such as the LSIDs used in the Taverna model.

## 4 Conclusions and Future work

We addressed the problem of provenance interoperability as data integration and proposed a mediator-based architecture for integrating provenance from multiple sources. We have identified a core set of concepts and access functions present in different provenance systems, and used them to develop a general provenance model and query API. To validate our approach, we implemented the mediator along with wrappers for three provenance models. We used the system to evaluate a series of queries over provenance data derived by the different systems. The preliminary results of our case study indicate that this approach can be effective for integrating provenance information.

There are several avenues we plan to explore in future work. An important goal of this work was to assess the feasibility of the proposed architecture. As a next step, we would like to evaluate the efficiency and scalability of different approaches to provenance interoperability. One challenge we currently face is the lack of a suitable benchmark for the evaluation. Another direction we plan to pursue is to implement a mediator-based architecture that uses OPM as the model for the global schema. Last, but not least, we would like to explore usable languages and interfaces for querying provenance. The query API we have developed is low-level and can be complex for end-users to specify queries. It can, nonetheless, be used as the ba-

sis for high-level languages and interfaces. Specifically, we plan to develop automatic mechanisms for translating high-level provenance queries into calls to the API, for example: queries defined through a query-by-example interface [21]; and through domain-specific provenance languages [20].

## References


[1] R. S. Barga and L. A. Digiampietri. Automatic capture and efficient storage of escience experiment provenance. *Concurrency and Computation: Practice and Experience*, 20(5):419–429, 2008.

[2] B. Clifford, I. Foster, M. Hategan, T. Stef-Praun, M. Wilde, and Y. Zhao. Tracking provenance in a virtual data grid. *Concurrency and Computation: Practice and Experience*, 20(5):565–575, 2008.

[3] S. B. Davidson and J. Freire. Provenance and scientific workflows: challenges and opportunities. In *Proceedings of ACM SIGMOD*, pages 1345–1350, 2008.

[4] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Scientific Programming Journal*, 13(3):219–237, 2005.

[5] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying and automating data derivation. In *Proceedings of SSDBM*, pages 37–46, 2002.

[6] J. Freire, D. Koop, E. Santos, and C. T. Silva. Provenance for computational tasks: A survey. *Computing in Science and Engineering*, 10(3):11–21, 2008.

[7] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing rapidly-evolving scientific workflows. In *IPAW*, pages 10–18, 2006.

[8] D. Gannon et al. A Workshop on Scientific and Scholarly Workflow Cyberinfrastructure: Improving Interoperability, Sustainability and Platform Convergence in Scientific And Scholarly Workflow. Technical report, NSF and Mellon Foundation, 2007. `https://spaces.internet2.edu/display/SciSchWorkflow`.

[9] J. Golbeck and J. Hendler. A semantic web approach to tracking provenance in scientific workflows. *Concurrency and Computation: Practice and Experience*, 20(5):431–439, 2008.

[10] P. Groth, S. Miles, and L. Moreau. Preserv: Provenance recording for services. In *Proceedings of the UK OST e-Science Fourth All Hands Meeting (AHM05)*, September 2005.

[11] A. Y. Halevy. Data integration: A status report. In *BTW*, pages 24–29, 2003.

[12] Y. E. Ioannidis and R. Ramakrishnan. Efficient transitive closure algorithms. In *VLDB*, pages 382–394, 1988.

[13] The Kepler Project. `http://kepler-project.org`.

[14] J. Kim, E. Deelman, Y. Gil, G. Mehta, and V. Ratnakar. Provenance trails in the wings/pegasus system. *Concurrency and Computation: Practice and Experience*, 20(5):587–597, 2008.

[15] Microsoft Workflow Foundation. `http://msdn2.microsoft.com/en-us/netframework/aa663322.aspx`.

[16] L. Moreau, editor. *Concurrency and Computation: Practice and Experience– Special Issue on the First Provenance Challenge*, 2008.

[17] L. Moreau, J. Freire, J. Futrelle, R. McGrath, J. Myers, and P. Paulson. The open provenance model, December 2007. `http://eprints.ecs.soton.ac.uk/14979`.

[18] S. G. Parker and C. R. Johnson. SCIRun: a scientific programming environment for computational steering. In *Supercomputing*, page 52, 1995.

[19] Second provenance challenge. `http://twiki.ipaw.info/bin/view/Challenge/SecondProvenanceChallenge`, 2007. J. Freire, S. Miles, and L. Moreau (organizers).

[20] C. Scheidegger, D. Koop, E. Santos, H. Vo, S. Callahan, J. Freire, and C. Silva. Tackling the provenance challenge one layer at a time. *Concurrency and Computation: Practice and Experience*, 20(5):473–483, 2008.

[21] C. Scheidegger, D. Koop, H. Vo, J. Freire, and C. Silva. Querying and creating visualizations by analogy. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1560–1567, 2007. Papers from the IEEE Visualization Conference 2007.

[22] Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.

[23] Y. L. Simmhan, B. Plale, D. Gannon, and S. Marru. Performance evaluation of the karma provenance framework for scientific workflows. In *International Provenance and Annotation Workshop (IPAW), Chicago, IL*, volume 4145 of *Lecture Notes in Computer Science*, pages 222–236, 2006.

[24] The Taverna Project. `http://taverna.sourceforge.net`.

[25] The Triana Project. `http://www.trianacode.org`.

[26] VDS - The GriPhyN Virtual Data System. `http://www.ci.uchicago.edu/wiki/bin/view/VDS/VDSWeb/WebMain`.

[27] The VisTrails Project. `http://www.vistrails.org`.

[28] G. Wiederhold. Mediators in the architecture of future information systems. In M. N. Huhns and M. P. Singh, editors, *Readings in Agents*, pages 185–196. Morgan Kaufmann, San Francisco, CA, USA, 1992.

[29] XQuery 1.0: An XML Query Language, `http://www.w3.org/TR/xquery/`, 2008.

[30] J. Zhao, C. Goble, and R. Stevens. An identity crisis in the life sciences. In *Proc. of the 3rd International Provenance and Annotation Workshop*, Chicago, USA, May 2006. LNCS. extended paper.

[31] J. Zhao, C. Goble, R. Stevens, and D. Turi. Mining taverna's semantic web of provenance. *Concurrency and Computation: Practice and Experience*, 20(5):463–472, 2008.