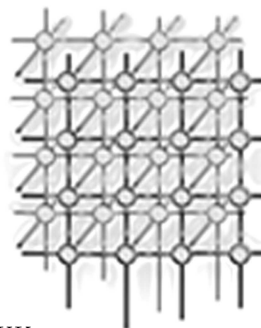# Extracting Causal Graphs from an Open Provenance Data Model

Simon Miles*, Paul Groth, Steve Munroe, Sheng Jiang,
Thibaut Assandri and Luc Moreau

School of Electronics and Computer Science, University of Southampton, UK

SUMMARY

The Open Provenance Architecture (OPA) approach to the challenge was distinct in several regards. In particular, it allows different components of the challenge workflow to independently record documentation, and for the workflow to be executed in different environments, made possible by an open, well-defined data model and architecture. Another noticeable feature is that we distinguish between the data recorded about what has occurred, *process documentation*, and the *provenance* of a data item, which is all that caused the data item to be as it is. In this view, provenance is obtained as the result of a query over process documentation. This distinction allows us to tailor the system to best address the separate requirements of recording and querying documentation. Other notable features include the explicit recording of causal relationships between both events and data items, an interaction-based world model, intensional definition of data items in queries rather than relying on explicit naming mechanisms, and *styling* of documentation to support non-functional application requirements such as reducing storage costs or ensuring privacy of data. In this paper, we describe how each of these features aid us in answering the challenge's provenance queries.

KEY WORDS: provenance, causation, service-oriented architectures, e-Science

Our approach to enabling the provenance of data to be determined, the *Open Provenance Architecture* (OPA), is the result of extensive requirements capture [12] in domains including bioinformatics, medicine, physics and chemistry [8, 10, 17]. To meet the requirements, we have specified a domain and technology-independent architecture [7]. The specification gives models and interfaces for capturing documentation on what has occurred in application processes and, from this documentation, determining the provenance of items in the processes' results. By being open, by which we mean that the specification is publicly available to all parties, every part of a system can independently record documentation without needing to know

---

*Correspondence to: Computer Science, King's College London, London, WC2R 2LS, UK

of any other part. This is critical in a distributed application with services from multiple organisations. We note that our independence from execution technology distinguishes us from most other approaches to the challenge, which were based around workflow or operating systems. A further benefit of this independence is that we do not rely on the workflow being available and interpretable at the time that queries are performed.

Our approach is unique in other ways. First, we make a distinction between *process documentation*, the record of what has occurred, and the *provenance* of an item, i.e. data regarding how that item came to be as it is. Provenance is determined by performing a query over process documentation. This distinction allows us to separately specify the data structures and algorithms to meet the non-functional requirements on each. For instance, process documentation is ideally recorded independently by disparate parts of an application and requires efficient storage due to its size, while provenance should be *scoped* to include only that relevant to the querier. Process documentation and provenance are connected by the concept of *causation*: the provenance of an item is everything that *caused* it to be as it is, and process documentation records the *causal relationships* between events and data in application processes. A second distinct feature of our approach is, therefore, that our data structures include an explicit, technology-independent representation of causation. Other notable features of our approach, detailed in the following sections and used in the challenge, include the intensional definition of items in queries rather than relying on global naming mechanisms, and *styling* of documentation to help meet application requirements such as reducing storage costs or ensuring privacy of data.

In this paper, we present the salient features of our approach in Section 1, describe how we used our software to implement the challenge in Section 2, and then detail how we answered each challenge query in Section 3. Finally, we discuss possible variations of the challenge that our system could support and conclude in Section 4.

1.   The Open Provenance Architecture

The Open Provenance Architecture enables processes to be documented and the provenance of items to be determined. It is depicted graphically on the provenance challenge TWiki [6], and described in full elsewhere [7]. For many applications, process documentation cannot be produced in a single, atomic burst, but instead its generation must be interleaved with execution. Given this, it is necessary to distinguish an item of documentation for part of a process from the whole process documentation. We see the former, referred to as a *p-assertion*, as an assertion made by an individual service about a process it is involved in. Documentation of a process consists of p-assertions made by the services in the process.

1.1.   Process Documentation

To minimise its impact on application performance, documentation needs to be structured such that it can be constructed and recorded autonomously by services. Otherwise, should synchronizations be required between services to agree on how and where to document

execution, application performance may suffer. To satisfy this requirement, three generic kinds of p-assertions have been identified.

- An *interaction p-assertion* documents the event of sending or receiving a message by a service to/from another service, and the contents of the message sent/received. As we cannot predict the form of applications that will use architecture, we do not restrict what form an application message may take.
- A *relationship p-assertion* documents causation between two send/receive events or data items, where the events/items are local to one service. For example, a relationship may document that a service sent one message because it had received another. The fact that we can relate both events and items distinguishes our approach from others. In REDUX [1], only events are related; in myGrid [18] relations are between data.
- An *actor state p-assertion* documents the state of a service at a given instant. Because there are no global clocks, an instant is defined in relation to events local to the service, i.e. the sending or receiving of a message.

One piece of information that is given for each interaction p-assertion is the *documentation style* used in asserting the contents of the message sent/received. The documentation style of an assertion is the transformation applied to its content when asserted. Documentation styles can, for example, be used to declare that a large message was not recorded in a p-assertion verbatim but stored separately and then referenced in the p-assertion. As another example, in the challenge implementation, we have transformed a binary file format into an XML one on inclusion in an interaction p-assertion. This allows easier querying over the p-assertion contents (see Section 2.3 for more details). Such declarations allow queriers to correctly interpret p-assertion contents. It is intended that documentation style transformations are independent of the application itself (they are performed only for creating p-assertions) and are trivial compared to the application. Concrete examples can be found on the TWiki page [13].

1.2.   Queries

A set of p-assertions can be organised into a document with a well-defined overall structure, the *p-structure*. In the p-structure, p-assertions are categorised by the send/receive event they document. This structure, and the schema by which it is realised in XML (`twiki.gridprovenance.org/bin/view/Provenance/OpenSpecification`), allows queries to be performed over p-assertions.

Our approach includes two types of query, exploring process documentation in different ways. *Process documentation queries* simply navigate the p-structure to retrieve particular data from the documentation recorded. In our implementation, they are XQuery expressions, where the XML over which the query is performed is a p-structure. A *provenance query* uses the causality asserted by interaction and relationship p-assertions to the events and data that caused an item to be as it is. A provenance query request consists of two parts.

- The *query data handle* is a specification of the data item of which to find the provenance.
- The *relationship target filter* is a specification of the *scope* of the query, i.e. what information is relevant to return.

2.   Implementation for the Challenge

The above specifies what is required for an application to use OPA. There is no requirement that particular actors exist, assertions are made, programming languages are used or formats are used for application messages or actor states. The current implementation of the OPA provenance store is as a Web Service, and so places a requirement on the client to send XML over HTTP (additionally, if the application makes assertions in XML it will benefit from ease in querying). In this section, we describe how the challenge workflow was implemented, what process documentation was recorded and how queries were instantiated in general.

2.1.   Workflow and Documentation

We simulated the workflow in Java, operating directly on the URLs for the input data, and producing URLs for the outputs and intermediary data. Our simulator uses one class for each of the five workflow operations, named AlignWarp, Reslice, Softmean, Slicer and Convert. Additionally, one class, named Challenge, acts as a workflow enactor calling each of the operations as per the challenge workflow. URLs and parameters are exchanged by instantiations of these classes. Data (URLs) is sent by the enactor to each operation on invocation, result data is returned by operation to enactor on completion. Data output by one operation may be used, by the enactor, as input to others, as specified by the workflow.

As the program executes, it creates and records p-assertions in a repository called a *provenance store* [9]. The store implements interfaces for recording p-assertions and performing process documentation and provenance queries. Recording is independent of the technology used to execute the workflow, so similar documentation would be produced by other workflow implementations. The following are recorded when the program is executed.

- For every call made by the enactor to an operation, an interaction p-assertion, containing the data item URL arguments, is recorded.
- For every response given by an operation to the enactor, an interaction p-assertion, containing the result data item URLs, is recorded.
- For every execution of an operation in transforming inputs into outputs, a relationship p-assertion is asserted relating those outputs to those inputs.
- For every operation output given as input to a subsequent operation in the workflow, a relationship p-assertion is asserted by the enactor from input to output.
- For every operation or enactor execution, an *actor profile* is recorded, including a user-friendly name for the service and a type (Java class name), in an actor state p-assertion.

In Figure 1 we depict a part of the workflow implementation, in which the enactor (Challenge) calls first the Softmean operation, then Slicer. Each horizontal arrow denotes an interaction between the enactor and an operation, in which request or result data is exchanged. Each black circle denotes an interaction p-assertion that is made about the interaction to which it is adjacent, by the actor in whose box it is contained. Each curved arrow denotes a relationship p-assertion made between data items in the connected interactions, by actor in whose box the arrow is contained. All challenge p-assertion data is on the TWiki page [13].
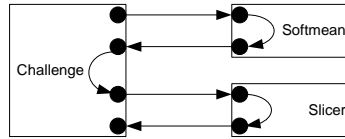
Figure 1. Part of the workflow implementation with p-assertions

Additionally, the annotations required by the challenge are recorded as actor state p-assertions including both the annotation (key-value pair) and a reference to the data item (as recorded in an interaction p-assertion) to which the annotation applies. Because there is no distinction made in our approach between workflows and other processes, annotations to workflow data items can be made within the workflow execution itself or outside in other processes. However, our approach does not by itself support arbitrary mutable annotations on process documentation, unlike systems such as Mindswap[5] or VDL[3]. Actor state p-assertions can also be used to record the local clock time of a service, but such timestamps are not relied on to determine the provenance of data, unlike systems such as USC/ISI[11]. A final feature that we mention here is *tracers*. These are unique tokens propagated through interactions by services and so are apparent in interaction p-assertions. By this mechanism, a tracer is used to delimit a process, and so groups together all process documentation regarding that process, a mechanism absent in other systems, such as Karma [16]. Complete examples of the use of tracers are available in other documents [7].

2.2.   Provenance Queries

A provenance query request is made up of two parts: the query data handle and the relationship target filter. In our implementation, a *provenance query builder* Java class helps clients specify these in a minimal, declarative way.

2.2.1.   Query Data Handle

The query data handle identifies the data item for which the provenance is to be found. More specifically, it states how documentation regarding that item can be identified in process documentation. A feature of our approach is to not rely on global naming mechanisms (as do systems such as ES3[4] and SDG[15]) which, in a distributed environment with independently executing components, can make coherence and uniqueness impractical to maintain, and may add substantially to the performance costs of recording documentation and in execution. Moreover, we wish to allow items to be identified at any level of granularity in queries, which means that every part of every document, for example, would have to be given a separate name. Instead, we identify a data item *intensionally*, i.e. by a set of criteria which, in combination, apply only to that item.

The same item, or items with the same properties, may occur at multiple times in application processes, so to uniquely identify an item, a querier may need to specify an instant at which it was present in a process. In distributed systems, particularly those that span multiple organisations, we cannot rely on a synchronised global clock being available, so we identify an instant by the event that occurred at that instant, either the sending or receiving of a message. The item itself will be present within the content of a p-assertion, usually an argument of the message sent or received, a copy of which is documented in an interaction p-assertion.

Because each interaction is uniquely identified and each p-assertion contains fixed data in a fixed order, at least one unique set of criteria always exists to refer to an item. However, whether such criteria are available to a given querier is a matter of appropriate software engineering [14]. The advantage of being able to refer to data items intensionally using sets of criteria rather than naming each individually is that we can ensure the criteria used for intensional definitions is information that will be available to the querier long after the documented process has completed. For example, in referring to a data item as "that data sent between actors A and B", we use the fact that actors A and B, and the means to identify them, still exist and are known to the querier after the process has completed. On the other hand, the data referred to may have been a temporary creation that have since been deleted. If the querier was to refer to such data items by name, it would have to have those names available at the time of query, thus requiring the querier to retain a potentially vast amount of information and defeating the purpose of a provenance system. Note also that the intensional definition is used only for the querier to identify the item of which to find the provenance: from that point on in determining the provenance of the item, unique identifiers, internal to the provenance store, are used, so there are no significant efficiency consequences of this approach.

As an example of an intensional definition, a query data handle for the Atlas X Graphic may be specified by the following set of properties. It was implemented in Java as shown in Figure 2 (top), where the four properties are specified in lines (3) to (6) in the figure.

P1	The item was sent in a message from the Convert service
P2	The item was received in a message by the Challenge service (workflow enactor).
P3	We are interested in the provenance from the instant it was received.
P4	The message was in XML, and the item can be found in it with a given XPath.

2.2.2.   Relationship Target Filter

Relationship p-assertions document the causal relationships between application items, collectively making a directed acyclic graph of causes and effects. Once an item has been identified by a query data handle, a provenance query request is evaluated by following the relationships in the process documentation starting from that item, i.e. traversing the causality graph. The graph of relationships causally related to the item, i.e. the set of all relationships traversed, is returned as the provenance of the item. However, only a part of this graph may be of interest to the querier. Therefore, a *relationship target filter* specifies the criteria by which process documentation is determined to be relevant for the querier, i.e. under what conditions the graph traversal terminates.

For example, in the challenge workflow, a relationship target filter may specify that the querier is not interested in anything prior to the operation of the Softmean service. The

```
(1) dataItem = "http://www.ipaw.info/challenge/atlas-x.gif";
(2) builder.getQDHBuilder ().
(3)        sourceURL (convertURL).                                    // P1
(4)        sinkURL (enactorURL).                                      // P2
(5)        receiver ().                                               // P3
(6)        contentItem ("/java/object/void/object[string='" + dataItem + "']");  // P4

(7) builder.getRTFBuilder ().
(8)        .notRelation ("http://relation.org/softmean");
```

Figure 2. Java code for specifying a query data handle (top) and relationship target filter (bottom) in building a provenance query

operation of the service is specified in a relationship p-assertion, relating the inputs and outputs of the service, with a given URL for the relation type. The implementation of this filter is seen in Figure 2 (bottom). This single criterion, exclusion of a relationship of a given type, can be extended with other exclusive and inclusive criteria. The exclusion of relationships of this type mean that not only are softmean relationships themselves excluded from the result, but also everything leading up to (prior to) those softmean relationships in the causal graph. Further provenance query examples can be found on the TWiki page [13].

2.3.   Documentation Style

In our challenge workflow implementation, we have transformed the binary image header files into XML representations. This is a two step process: first AIR's *scan_header* utility is used to convert from binary to a properties file (key-value pairs). Then, these properties are transformed into an XML depiction of the same. The documentation style of interaction p-assertions including the image header files is set to a particular value, 'http://www.pasoa.org/schemas/ontologies/DocumentationStyle.Java_Header_XMLEncoding', to indicate that this occurred. A querier can search for headers in the style that it knows how to parse, e.g. to check values of individual header properties.

2.4.   Process Documentation Queries

Process documentation queries are expressed as XQuery [2] expressions over the p-structure schema. An example is shown in Figure 3 (the XQuery's header part, in which namespaces and variables are defined, is excluded for brevity), which finds an interaction p-assertion that includes a given set of parameter options and the date. XQuery allows arbitrary documents to be constructed and returned, so there is no specified schema for the results. The amount of knowledge that a querier needs to know about the structure of application messages or actor state descriptions depends on the query. If the query concerns only the interactions between actors, then only the p-structure schema needs to be known, for example, but if the querier wishes to extract a specific argument from a message, the structure of the message must be

```
(1) $ps:pstruct/ps:interactionRecord/ps:sender
(2) [ps:interactionPAssertion/ps:content/java/object/void/object/void/object/long='1157626153750']
(3) [ps:interactionPAssertion/ps:content/java/object/void/array/void/string='-m 12 -q']
(4) /../ps:interactionKey
```

Figure 3. Process documentation query for Query 4

known. We envisage, and have in some applications used, tools which hide the implementation details of queries from the user. Due to space restrictions, we do not include all the XQueries used for the challenge in this paper, but refer the reader to our TWiki page [13].

3.   The Challenge's Provenance Queries

In this section, we summarise how the functionality above was exploited in answering the challenge queries. Several of the queries were answered by composite steps, each involving an XQuery or provenance query to be made to the provenance store. In these cases, the steps were joined by challenge-specific Java code. XQuery is a very expressive language, but can be verbose for complex expressions and is harder to check for correctness compared with programming languages. Also, it does not, by default, have functionality for recursive operations while the provenance query performs a particular type of recursion (over causal relationships). Nevertheless, some systems using our architecture may choose to express more of their query processing in XQuery so that more computation occurs within the store. Results and details are on the TWiki page [13].

Queries 1, 2 and 3: The first three challenge queries were implemented as provenance query requests with the query data handle shown in Section 2.2.1 and Figure 2 (top). The first query specified no scope, so the relationship target filter was empty. The second and third queries used the following filters to exclude events or data prior to the Softmean service and Reslice service respectively: *(1)* exclude the `softmean` relationship in the causality graph traversal; *(2)* exclude the `reslice` relationship.

Query 4: This query was implemented as the XQuery shown in Figure 3. This finds interactions in which a message was sent (line 1 in the figure) containing the parameters corresponding to the model required (line 2), and the required date was recorded (line 3). The interaction keys (unique identifiers) for these interactions are returned (line 4), and these can be used to retrieve whatever other documentation for those interactions is required, using another process documentation query.

Query 5: For Query 5, we use the image header documentation style as described in Section 2.3 in a process documentation query. The XQuery finds interactions where an image header is encoded in the XML style and the "global maximum" property is set to 4095 in that file, returning the interaction key which identifies the interaction where this took place. We then find out which processes included those interactions by first performing a provenance

query for each Atlas Graphic image (the same query as in Query 1), then iterating through the provenance query results to find the interaction keys found in the first step. Wherever a provenance query result contains one of those interaction keys, we have determined that the image of which those results are the provenance fulfils the criteria of Query 5.

Query 6: We answer this query in three steps. First, we find interactions involving the Softmean service, identified by its common name specified in an actor profile (with an XQuery). Next, we find the provenance of the outputs of those interactions, i.e. the process that led to those averaged images (a provenance query very similar to that in Query 1). Finally, we find within the provenance query results an invocation that takes "-m 12" as an argument by iterating through the results and, for each interaction, performing an XQuery which detects if the argument is present. If so, then we have found the output (the interaction from the first step) we are looking for.

Query 7: To answer this query, we first find all occurrences of Atlas X Graphic being output using an XQuery. For each output, we perform the same provenance query as in Query 1. We then iterate over the results and extract the actor types in every interaction included in the provenance query results. An actor type is given in an actor profile, and extracted with an XQuery. Finally, the sets of actor types for the different workflow runs are compared. In the second run (the adapted workflow), we find three new actor types. Therefore, the difference between the runs is the presence of these services.

Queries 8 and 9: As described in Section 2.1, the challenge's annotations are included in our documentation using actor state p-assertions. To answer Queries 8 and 9, we first retrieve the annotations found in the store. Then, for Query 8, we iterate through the annotations to find "center=UChicago" and retrieve the interactions to which it is attached. Using a series of XQueries we then: *(i)* get the argument, i.e. the anatomy image file, in the latter interaction; *(ii)* find the AlignWarp service that inputs that anatomy image file; *(iii)* find the interaction in which that *align_warp* service outputs the warp parameters for that image file; and finally, *(iv)* get the result, i.e. the warp parameter file, from that interaction. For Query 9, we simply iterate through the annotations to find all those attached to the same item as the one annotated with "studyModality".

4.   Conclusions

The Open Provenance Architecture approach to the challenge is distinct in several ways. There is a clear separation of the open, application-independent, technology-independent documentation of process and the provenance of items in those executions, provenance being obtained conveniently from process documentation by specifically designed queries that traverse causal relationships asserted in the documentation. There is no reliance on global naming mechanisms, instead the items for which we want the provenance are defined intensionally. Finally, data can be *styled* to allow queries to be better performed. Through this combination of features, we were able to answer all the challenge queries.

Our approach also allows for several interesting variants of the challenge to be tackled. Because relationship p-assertions may connect events at different levels of granularity, e.g. at the level of service invocations or at the level of Java procedure calls, the provenance of a

workflow result can be returned at the granularity which best satisfies the querier. The scoping mechanism provided by the provenance query allows this exclusion of unwanted granularities. Further, our approach has a comprehensive security architecture, that allows p-assertions to be signed and for those signatures to be checked by queriers to increase trust in the documentation recorded. Finally, our approach allows process documentation to be distributed and inter-linked over multiple provenance stores, addressing scalability.

REFERENCES

1. Roger S. Barga and Luciano A. Digiampietri. Automatic capture and efficient storage of escience experiment provenance. *Concurrency and Computation: Practice and Experience*, 2007.
2. Scott Boag, Don Chamberlin, Mary F. Fernandez, Daniela Florescu, Jonathan Robie, and Jerome Simeon. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/.
3. Ben Clifford, Ian Foster, Mihael Hategan, Tiberiu Stef-Praun, Michael Wilde, and Yong Zhao. Tracking provenance in a virtual data grid. *Concurrency and Computation: Practice and Experience*, 2007.
4. James Frew, Dominic Metzger, and Peter Slaughter. Automatic capture and reconstruction of computational provenance. *Concurrency and Computation: Practice and Experience*, 2007.
5. Jennifer Golbeck and James Hendler. A semantic web approach to tracking provenance in scientific workflows. *Concurrency and Computation: Practice and Experience*, 2007.
6. Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, Victor Tan, Sofia Tsasakou, and Luc Moreau. An architecture for provenance systems. http://twiki.ipaw.info/bin/view/Challenge/SouthamptonArchitecture.
7. Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, Victor Tan, Sofia Tsasakou, and Luc Moreau. An architecture for provenance systems. Technical report, Electronics and Computer Science, University of Southampton, October 2006. Available at http://eprints.ecs.soton.ac.uk/12023/.
8. Paul Groth, Simon Miles, Weijian Fang, Sylvia C. Wong, Klaus-Peter Zauner, and Luc Moreau. Recording and using provenance in a protein compressibility experiment. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC'05)*. IEEE Computer Society, July 2005.
9. Paul Groth, Simon Miles, and Luc Moreau. Preserv: Provenance recording for services. In Simon J. Cox and David W. Walker, editors, *Proceedings of the UK e-Science All Hands Meeting 2005*, Nottingham, UK, September 2005. EPSRC. Proceedings published on CD.
10. Tamas Kifor, Laszlo Z. Varga, Javier Vazquez-Salceda, Sergio Alvarez, Steven Willmott, Simon Miles, and Luc Moreau. Provenance in agent-mediated healthcare systems. *IEEE Intelligent Systems*, 21(6):38–46, November/December 2006.
11. Jihie Kim, Ewa Deelman, Yolanda Gil, Gaurang Mehta, and Varun Ratnakar. Provenance trails in the wings/pegasus system. *Concurrency and Computation: Practice and Experience*, 2007.
12. Simon Miles, Paul Groth, Miguel Branco, and Luc Moreau. The requirements of using provenance in e-science experiments. *Journal of Grid Computing*, 2006. To appear.
13. Simon Miles, Paul Groth, Steve Munroe, Sheng Jiang, Luc Moreau, and Thibaut Assandri. Provenance challenge results: Southampton. http://twiki.ipaw.info/bin/view/Challenge/Southampton, 2006.
14. Steve Munroe, Simon Miles, Luc Moreau, and Javier Valquez-Salceda. Prime: A software engineering methodology for developing provenance-aware applications. In *Proceedings of the Software Engineering and Middleware Workshop (SEM 2006)*. ACM Digital, 2006. To appear.
15. Karen Schuchardt, Tara Gibson, Eric Stephan, and George Chin. Applying content management to automated provenance capture. *Concurrency and Computation: Practice and Experience*, 2007.
16. Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. Querying capabilities of the karma provenance framework. *Concurrency and Computation: Practice and Experience*, 2007.
17. Sylvia C. Wong, Simon Miles, Weijian Fang, Paul Groth, and Luc Moreau. Provenance-based validation of e-science experiments. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *The Semantic Web ISWC 2005: 4th International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science*, pages 801–815, Galway, Ireland, November 2005. Springer-Verlag GmbH.
18. Jun Zhao, Carole Goble, Robert Stevens, and Daniele Turi. Mining taverna's semantic web of provenance. *Concurrency and Computation: Practice and Experience*, 2007.