Electronically Querying for the Provenance of Entities

Simon Miles

School of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
`sm@ecs.soton.ac.uk`

Abstract. The provenance of entities, whether electronic data or physical artefacts, is crucial information in practically all domains, including science, business and art. The increased use of software in automating activities provides the opportunity to add greatly to the amount we can know about an entity's history and the process by which it came to be as it is. However, it also presents difficulties: querying for the provenance of an entity could potentially return detailed information stretching back to the beginning of time, and most of it irrelevant to the querier. In this paper, we define the concept of *provenance query* and describe techniques that allow us to perform *scoped provenance queries*.

1 Introduction

In order to understand, apply, or judge the accuracy or authenticity of an entity, whether electronic or physical, it is often crucial to know its *provenance*, i.e. from where it originated, how it was produced and what has happened to it since creation [9]. For example, in e-Science, to determine if an experiment's results are accurate, we look at the rigour of the experiment's process.

However, the amount of information making up the provenance of an entity may be vast [4]. The details of *everything* that ultimately caused an entity to be as it is would, generally, be an unmanageable amount. For example, to give the full provenance of an experiment's results, we have to describe the process that produced the materials used in the experiment, the provenance of materials used in producing *those* materials, the devices and software used in the experiment and their settings, the origin of the experiment design etc. Ultimately, if enough information was available, we would include details of processes back to the beginning of time. Similarly, given enough information, we could give finer and finer grained information on the processes that led to an entity, e.g. not just documenting that a sample was tested to see if a chemical was present, but the procedure by which this is done, the molecular interactions that took place in the testing procedure and so on. All the information about the provenance of an entity is potentially useful for someone with a particular question about that entity, but providing it all in all cases would be counter-productive.

Instead, anyone requiring the provenance of an entity should be able to get it by expressing the request as a *query* and *scoping* that query so that only the information relevant to them is returned. The contributions of this paper are to specify what a provenance query consists of and how it can be expressed, an algorithm for processing a provenance query to obtain the results and the form taken by results of a provenance query.

In the next section, we look at earlier work on retrieving provenance. Section 3 examines the properties data must have in order to extract the provenance of an entity from it, and the delineations used to scope a provenance query. Section 4 specifies a model of a provenance query and its results, illustrated by example in Section 5. We compare our work with others in Section 6 and conclude in Section 7.

2 Provenance

From existing work on provenance, we can derive two general properties. First, *the provenance of an entity depends on its relationships to other entities.* For instance, the results of an experiment are produced from processing intermediate data in an experiment, which are in turn produced from the inputs to the experiment. In myGrid [12], an entity is marked as being *derived from* another entity which, in turn, is derived from other entities. Using this approach, the provenance of an entity is determined from relations of that entity to other entities it is directly or indirectly derived from.

Second, *the provenance of an entity can vary in scope and, at widest scope, is everything that has had a causal effect on the entity.* Buneman [5] describes two differently scoped types of provenance, named *where* and *why provenance*, applied to database query results. Applied within the closed world of a database system, *where provenance* is the origin of the entity and its components, while *why provenance* is all data having a causal effect on the entity.

We argue that the provenance of an entity can be seen as a *process* that leads to the entity being in its current state. Full information on that process would include the origin of and anything having an effect on the entity. To obtain the provenance, we need documentation of how that process occurred, at some level of abstraction, e.g. we may infer the process from the database query or workflow whose execution resulted in the entity, or we may have details of each action performed by actors in the process. Documentation of past processes is called *process documentation.*

In our model, we separate two concepts: process documentation, documentation of executed processes, and provenance, a description of the process leading to an entity which is determined from process documentation. The separation allows us to tailor systems to best meet the requirements on each. For example, the details of a process may not be recoverable after it has completed, so should be documented as completely and accurately as possible during or immediately after [7], while provenance can be determined as long as process documentation is preserved unmodified, so should be scoped to include only that relevant to the querier.

3 Process Documentation

A provenance query is executed over process documentation and, in order to implement a scoped query, documentation must be structured so that it can be determined whether any piece of information is part of the provenance of an entity. Below, we introduce the properties and delineations of process documentation that can be used to answer provenance queries.

3.1 Assertions and Temporality

An *actor* is anything that performs actions, and each piece of process documentation is created by an actor: by recording data on processes it has taken part in or inferring what happened from information from other actors. We can provide no guarantee that documentation accurately reflects what occurred, so documentation is actually *assertions* by actors. A *p-assertion* is an assertion about a process and process documentation is comprised of *p-assertions.*

A process includes multiple *events* and the provenance of an entity as it exists as one event occurs is different from the provenance of the same entity later or earlier. For example, the provenance of a hospital patient, the process which led to that patient being in a particular state [1], would be different for the patient up to an operation starting and up to the operation finishing, because after the latter event the provenance includes data about the operation. An event that an assertion provides documentation about need not be instantaneous, but entities documented in the assertion must not change over the course of the event (else the provenance would be ambiguous). In our approach, the provenance of an entity is always the provenance of an entity as it exists when an event occurs, called the *entity's provenance up to the event*. In order to determine if a p-assertion is part of an entity's provenance up to a given event, it must be clear which event it documents.

3.2 Relationships

As stated in Section 2, the provenance of an entity depends on its relationships to other entities and so process documentation must include these relationships to be used in determining provenance. Relationships are directional, so we identify one entity in the relationship as the *subject* and the others as *objects*, e.g. the subject "12" *was the result of summing* objects "5" and "7". As one p-assertion may document an event in which several entities were involved, a relationship is between *parts* of p-assertions. A *p-assertion data item* is an entity within a p-assertion and a relationship is between two or more p-assertion data items. A p-assertion documenting the relationship of an entity in a p-assertion to entities in other p-assertions is a *relationship p-assertion.*

Relationship Types. Relationships can be of different types, the most abstract being a *causal* relationship, i.e. E was caused by C. While causal relationships are all that is needed to determine which documentation is part of an entity's provenance, they are inadequate for scoping the query. We need more information on *how* one entity is related to another to determine if some process

documentation is relevant. Therefore, *functional relationships* between entities can be asserted, stating that an entity was produced by performing a function on other entities, e.g. an actor may assert that a value produced in an experiment is a product of measuring the weight of a sample (there is a functional relationship between the value and the sample).

Parameter Names. A p-assertion often documents a relationship in the world in which the entities being related play *roles* in that relationship, e.g. in asserting that the results of a *divide* operation were derived from its inputs, we must mark the entities involved with the roles they play: divisor and dividend for the inputs, quotient and remainder for the outputs. The name of an entity's role in a relationship is a *parameter name*, which must be asserted with relationships for the documentation to be interpretable.

4   Provenance Queries

Below, we specify a model for expressing provenance queries. To execute a query, a *provenance query request* is sent to a *provenance query engine* by a *querying actor*. A provenance query request includes a *query data handle*, identifying the entity of which to find the provenance, and a *relationship target filter*, specifying the query's scope. These are shown in Table 1, and described in full below.

4.1   Query Data Handles

When a querying actor asks for an entity's provenance, it identifies the entity such that a query engine can find documentation of the entity. The identification is called a *query data handle*. For the actor, a query data handle identifies an entity at a given event. For the engine, a query data handle identifies a search for p-assertion data items in process documentation. A handle comprises three parts, discussed below.

A p-assertion documents an event in a manner dependent on the way the system is modelled. Part of a handle is an *event search*, a search for documentation of the event in which the entity occurred. Within documentation of that event, an *entity search* finds p-assertion data items documenting the entity. The *search space* of a data handle identifies the set of process documentation to be searched.

4.2   Relationship Target Filters

A *relationship target filter* is used to scope a query to the part of a process of interest to the querying actor. More concretely, we can say that, given that a p-assertion data item has been identified as part of a query's results, and that that data item is related to other data items (by relationship p-assertions), the relationship target filter specifies which related data items should also be included in the results.

A *relationship target* is a set of properties of a data item that is the object of a relationship p-assertion. The properties, e.g. the event that the p-assertion

Table 1. Data comprising a provenance query request

Provenance Query Request		
Query Data Handle	A search over process documentation to find the record of an entity at a given event for which the querying actor wishes to find the provenance.	
Relationship Target Filter	Criteria by which the querying actor specifies whether any given entity in the documentation, and its relations, should be included in the query results.	
Starting Search Space	The process documentation set from which to start searching for the provenance of the entity.	
Query Data Handle		
Event Search	A search for the relevant event involving the entity.	
Entity Search	A search in p-assertions for data items documenting the entity.	
Search Space	The process documentation set that will be searched over.	
Relationship Target		
Event	The event which the p-assertion is documenting.	
Global P-Assertion Key	The globally unique identifier for the p-assertion.	
Parameter Name	The role played by the object in the relationship.	
Provenance Store Address	The address where the p-assertion is stored.	
Data Accessor	The location of the data item within the p-assertion.	
Relationship	The relation (name) of which this target is an object.	
Asserter	The asserting actor's identity, if known.	
P-Assertion Content	The content of the p-assertion (the actual documentation).	

documents or the asserter's identity, are those described in Section 3 and are listed in Table 1. A relationship target filter is a function over a *relationship target* returning a boolean value specifying whether the relationship target is within scope. For example, a relationship target filter may return false for relationship targets where the asserter is a particular, untrusted, source. In this case, the provenance query results exclude all p-assertions by that asserter and p-assertions iteratively related to those assertions.

4.3   Provenance Query Results

A provenance query request is processed as follows. First, perform the search expressed by the query data handle to find a set of p-assertion data items. For each relationship of which one of those items is a subject, execute the relationship target filter on the information about each object of the relationship (i.e. each relationship target). Where an object is accepted by the relationship target filter, recursively apply the filter to objects of its own relationships. The final results of the query are comprised of two parts: the p-assertion data items from the query data handle search (the *start* data items); and, for every relationship object accepted by the filter, the relationship between that object and the subject in that relationship.

A query engine, with instantiations of the above model in XML, has been implemented as part of an open source distribution. Due to space restrictions,

we do not describe it here, but refer readers to `www.pasoa.org`, from which it can be downloaded, and a link to a functional specification of the engine's interface can be found.

5   Case Study

To demonstrate provenance queries completely but concisely, we use a simple, contrived example. A workflow, shown in Figure 1, is run and process documentation generated: a GUI actor calls an Averager service with two values (7, 5); Averager sums them and calls Divider with 12 as divisor, 2 as dividend; Divider sends the answer 6 to Averager, which sends it to the GUI; the GUI sends 6 to Store, along with the file location, e.g. `file1`, at which to store it.
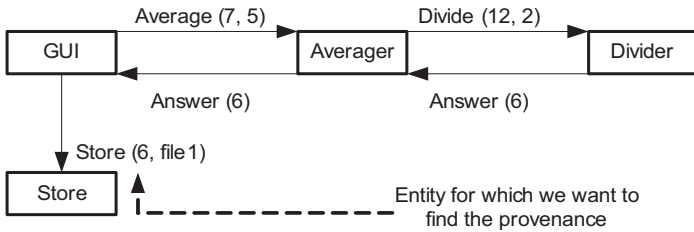

Fig. 1. An example workflow

Two types of p-assertion are recorded by actors in the scenario. An *interaction p-assertion* is a copy of the message sent between two actors. A relationship p-assertion asserts a relationship between data items exchanged in messages (and so documented in interaction p-assertions). A querying actor can derive a causal relationship from an interaction p-assertion: a message being received is caused by the same message being sent. The events to which p-assertions are declared to apply are the sending or receiving of messages. Each actor in the workflow makes interaction p-assertions about every message it sends and receives.

For this scenario, we show the results of a provenance query scoped in different ways. The query data handle represents a search for the "data requested to be stored at location `file1` (*entity search*) at the event of the Store actor receiving a message (*event search*)." The query returns results in which the start p-assertion data item is the 6 in the message from GUI to Store, and, if unrestricted in scope, includes all relationships recorded. The relationships form a directed acyclic graph linking data items in exchanged messages, shown in Figure 2, with the start data item shown at the bottom. Broken lines between data items depict asserted relationships, labelled with their functional relationships. Parameter names, "divisor" and "dividend" are shown for the objects of the "Division of" relation.

One way to scope the provenance is to exclude a type of relationship, e.g. by excluding "Average of", that relationship is removed from the graph returned by
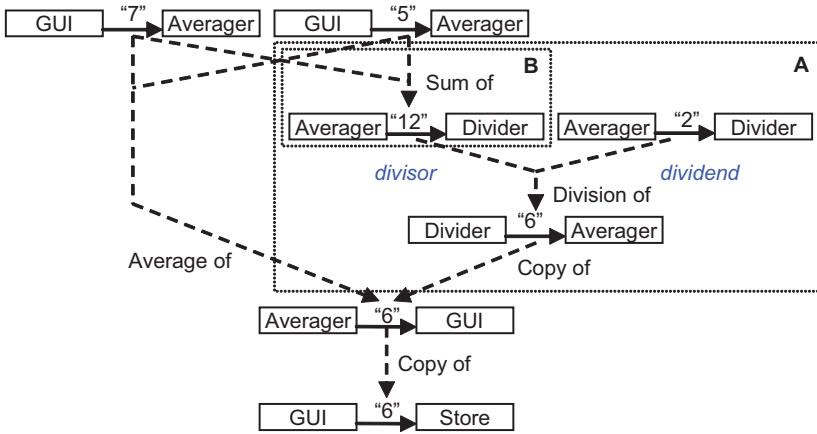
Fig. 2. The provenance of the data stored in `file1`

the query. As a comparable practical example, in asking for the provenance of a journal article, we may want to include the origins of its content and not the origins of the paper on which it is printed: these are types of relation from the journal paper to other entities. Another scope excludes the internal operations of an actor, such as Averager. This removes the documentation shown in box A in Figure 2 from the results. Comparably, we may want to know from which database we downloaded data, but not the database query: we make the results more coarse-grained by hiding part of the process. Finally, we can scope on the role that data plays in the process by excluding all "divisors" from the results (as specified in the parameter name), so removing the relationships shown in box B. Comparably, in querying for the provenance of compressed data, we may wish to know the data before compression but not the compression algorithm used: two roles in the compression function. The true benefits of the scoping process require a more detailed example than is possible to give in this paper, and will be in future work.

6   Related Work

In related work, provenance queries of a form are executed, but the approaches differ considerably from ours. In most systems, the mechanism is assumed to be an unspecified query of a database containing documentation [2,11], but a few specify another mechanism. Some systems, e.g. lab information management systems (LIMS) and work on deriving provenance data from database queries [5], operate in closed systems, allowing extra positive assumptions to be made on the quality of the query results, but preventing determination of an entity's provenance where the process that led to it spans multiple systems. Also, results cannot be scoped to arbitrary levels. Similarly, approaches for which queries are made with regards to workflows rather than their results [3], are limited to answering questions of the provenance of data within a workflow.

myGrid [12] and CombeChem [8] have advocated storing process documentation as functional and derivation relations between entities, named by URIs, as RDF triples. This has the advantage that query languages, e.g. SPARQL [10], exist to query relationship-based data, but there is no distinction between events in an entity's lifetime, e.g. in asking for the provenance of an experiment result, is an actor asking for how the result was produced, how it has been used as input to other processes (such as publication) since, or both? This means that the data on which a query can be scoped is limited. Some approaches visualise the provenance of data or records of workflow execution, allowing users to iteratively hide or retrieve data on the basis of what they have previously seen [6]. While this is productive and simple, it is only practical where the user is the direct consumer of the data, and does not address the problem where documentation is to be further processed to answer complex questions.

7   Conclusions

The provenance of an entity is essential, in many domains, for understanding and evaluating that entity, but the amount of information that makes up the provenance of an entity could be huge, so the results need to be *scoped* to retrieve only what is relevant for the querier. To execute a scoped provenance query, documentation over which the query is executed must have particular characteristics. In this paper, we have specified how a scoped provenance query can be expressed and executed. We exposed the necessary characteristics of the documentation over which the query is processed, criteria available for scoping, a data model for query requests and the algorithm used to execute a query.

References

1. Sergio Alvarez, Javier Vazquez-Salceda, Tamas Kifor, Laszlo Z. Varga, and Steven Willmott. Applying provenance in distributed organ transplant management. In *this volume*, 2006.
2. Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: A survey. *ACM Computing Surveys*, 37(1):1–28, March 2005.
3. Shawn Bowers, Timothy McPhillips, Bertram Ludascher, Shirley Cohen, and Susan B. Davidson. A model for user-oriented data provenance in pipelined scientific workflows. In *this volume*, 2006.
4. Miguel Branco and Luc Moreau. Enabling provenance on large scale e-science applications. In *this volume*, 2006.
5. P. Buneman, S. Khanna, and W.C. Tan. Why and where: A characterization of data provenance. In *Int. Conf. on Databases Theory (ICDT)*, 2001.
6. Juliana Freire, Claudio T. Silva, Steven P. Callahan, Emanuele Santos, Carlos E. Scheidegger, and Huy T. Vo. Managing rapidly-evolving scientific workflows. In *this volume*, 2006.

7. Paul Groth, Simon Miles, and Steve Munroe. Principles of high quality documentation for provenance: A philosophical discussion. In *this volume*, 2006.
8. Gareth Hughes, Hugo Mills, David de Roure, Jeremy G. Frey, Luc Moreau, m.c. schraefel, Graham Smith, and Ed Zaluska. The semantic smart laboratory: a system for supporting the chemical escientist. *Organic and Biomolecular Chemistry*, 2(2):1–10, 2004.
9. Simon Miles, Paul Groth, Miguel Branco, and Luc Moreau. The requirements of recording and using provenance in e-science experiments. Technical report, University of Southampton, 2005.
10. Eric Prud'hommeaux and Andy Seaborne. Sparql query language for rdf. http://www.w3.org/TR/rdf-sparql-query/, 2006.
11. Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.
12. J. Zhao, C. Goble, M. Greenwood, C. Wroe, and R. Stevens. Annotating, linking and browsing provenance logs for e-science. In *Proc. of the Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data*, October 2003.