# Applying the Provenance Data Model to a Bioinformatics Case

Paul GROTH [a] and Steve MUNROE [a] and Simon MILES [b] and Luc MOREAU [a,1]

[a] *School of Electronics and Computing Science, University of Southampton, UK*

[b] *Department of Computer Science, King's College London, UK*

**Abstract.** Scientists and, more generally end users of computer systems, need to be able to trust the data they use. Understanding the origin or *provenance* of data can provide this trust. Attempts have been made to develop systems for recording provenance, however, most are not generic and cannot be applied in a general manner across different systems and different technologies. Moreover, many existing systems confuse the concept of provenance with its representation. In this article, we discuss an open, technology neutral model for provenance. The model can be applied across many different systems and makes the important distinction between provenance and the way it can be generated from a concrete representation of process. The model is described and applied to a grid-based example bioinformatics application.

**Keywords.** Provenance, data model, bioinformatics.

## Introduction

With the ever increasing proliferation and complexity of computational systems that underpin so much of human activity, people are increasingly becoming dependent on the data that these systems produce. The amount of data produced is growing exponentially, and ensuring that it can be trusted is becoming ever more important as increasing numbers of decisions we make rely so heavily upon it. Consequently, methods are required that can provide the necessary guarantees that the data they produce can be trusted, validated and replicated. In the context of Grid and High Performance Computing, the amount of data produced can be overwhelming and, in order to aid users of such systems better understand and trust their data, techniques are required that can provide a historical account of how such data is produced. The history, lineage or *provenance* of a given piece of data provides understanding of how it was that the data came to be as it is. This understanding enables users to validate data by providing the means to examine the processes that produced it for fitness for purpose, compliance to regulations, replication, validation and examination.

Recent work conducted at the University of Southampton under the PASOA and EU Provenance projects has addressed the problem of determining the provenance of data for open and heterogeneous applications. Work has been conducted to define an architec-

[1]Corresponding Author: Luc Moreau, L.Moreau@ecs.soton.ac.uk

tural framework that developers can use to enable the capture of information about processes within their applications enabling them to answer questions about the provenance of data. The work defines a technology neutral framework for representing, capturing, and querying the provenance of data, and an *open data model* that provides the means for developers to collect and organise provenance-related information. The combination of the architecture and the open data model enables provenance capability to be incorporated into a wide range of applications for a variety of application domains. For example, the architecture and data model have been applied to the health industry in an organ transplant management application [2], design engineering in an aerospace engineering simulator [12], medicine via a fMRI brain atlas imaging example [2] and bioinformatics in the context of a compressibility application [10].

Many existing papers exist that describe various aspects of the work by both the PASOA and EU Provenance projects [3], however, in this article we focus on describing how the open data model can be applied to an updated version of the above cited bioinformatics example. By presenting such a description, we aim to provide an account of the data model use that will aid other developers adopt it for their own applications. This paper makes two contributions:

1. an explanation of our open data model tailored to developers and;
2. an evaluation of the usage of the open data model in a grid-based bioinformatics experiment.

The article proceeds as follows: In the next section, we provide a high level overview of the open data model. In Section 2, we introduce the bioinformatics example. Section 3 describes in detail how the open data model was applied to the bioinformatics example. In the context of this example, we then discuss the performance of our approach in Section 4. In Section 5 we discuss related work, and in Section 6 we offer some concluding remarks.

## 1. A Conceptual Model For Provenance

Provenance, understood as it is in the world of Art, refers to the documented history of a given object such as a painting — who painted it, who owned it, who restored it, what restoration work has been done to it, where it has been kept and so on. This documented history enables scholars, owners and viewers to both better appreciate the significance of the object and to have confidence of its authenticity. The importance of provenance in the context of Art scholarship and appreciation is mirrored in the world of computer applications and data. Applications produce, consume and operate on data, and to ensure that we can trust applications it is necessary to be able to inspect their data, i.e. its origins in the processes that produced it. Having this information enables trust in the data and also provides other benefits besides, such as the ability to reproduce the data, understand why data is not as expected and prove that the creation of the data complies with stated policies and meets assumptions.

The start of our conceptual model for provenance begins with our definition of provenance for computational systems, which is *the provenance of a piece of data is the*

[2]http://twiki.ipaw.info/bin/view/Challenge/WebHome
[3]http://www.gridprovenance.org/biblioPages/

*process that produced that piece of data*. A unique aspect of our model that differentiates it from other approaches (See Section 5 for a review of related work) is the distinction we make about the information we use to obtain provenance and provenance itself. The former relates to information we gather about the process that produced the data — termed *process documentation* — and the latter refers to the results returned from performing a *query* over process documentation. In other words, the provenance of a given data item is what is returned when a query is performed over the documentation of the process that produced it. This distinction is vital because it allows the results returned from provenance queries to evolve and become more comprehensive as more documentation is created.

## *1.1. A Concept Map for Provenance*

In order to provide an overview of our model for provenance, Figure 1 shows a concept map inspired from our previous work on specifying an open provenance architecture [9]. The concept map reflects the above mentioned distinction between provenance and its representation in a computer system.

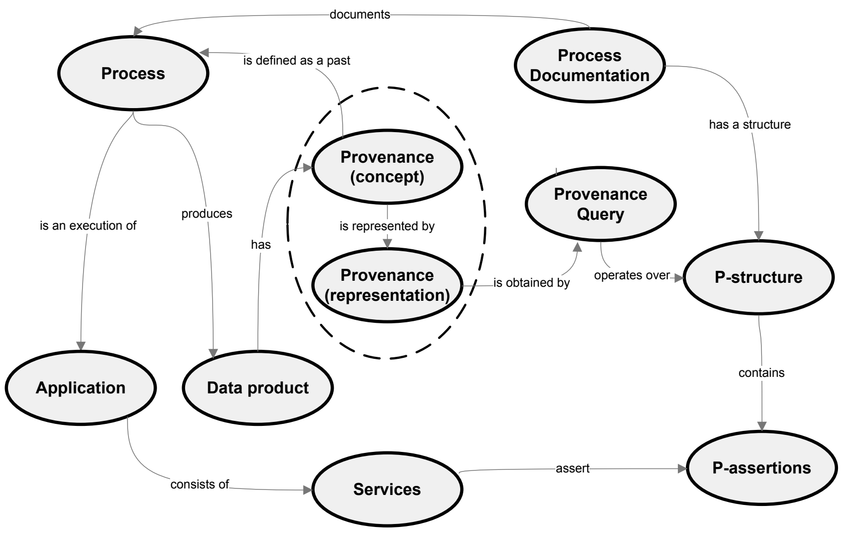


**Figure 1.** Concept Map for Provenance

A user who cares about the provenance of a data product has an interest in specific kind of information related to the process that led to the data product (cf. restoration vs ownership for a work of art); likewise, their needs can identify how far in the past the information should come from, hence the aforementioned representation of provenance as the result of a provenance query operating over the documentation of a process. Such documentation has a concrete representation, which we refer to as the *p-structure*. It consists of a structured set of *p-assertions* made by the different components of the application, whose execution produced the data product we consider the provenance of. Such p-assertions are asserted by software components and describe the components' involvement in a process. If all software components produce a description of their execution, process documentation would be very complete, and from it, many useful questions about the application could be answered through provenance questions. Within our model for provenance, such generated process documentation is stored in special repositories called *provenance stores*. These stores provide standard interfaces to allow software components to record and query the stored process documentation.

We note that, while our discussion focused on electronic data and computer-based applications, nothing in this concept map restricts us to the digital world. In fact, the provenance of a physical artefact could also be obtained by querying the documentation of the physical process that led to this artefact.

## 2. A Bioinformatics Use Case: The Amino Acid Compressibility Experiment

With the above overview, we are now in a position to show how our model represents data in a concrete domain application. Specifically, we will explain how the representation of provenance, i.e. process documentation, is structured and how this maps onto the example application. Note, that in this article we do not consider the techniques used to map our model of provenance onto an application, i.e. what must be done in order to make an application capable of generating process documentation. For this, we have specified the PrIMe methodology, a full description of which can be found in [15].

Briefly, the methodology consists of three phases. In Phase 1, use cases for provenance within the application are identified. From these use cases, important information items are identified whose provenance may need to be determined after application execution. Phase 2 maps the application into the interaction (i.e. Service Oriented Architecture) based perspective of the open data model and identifies which components within the application need to be modified to assert process documentation. Finally, in Phase 3, the application is modified to generate and record process documentation. While we have followed this methodology when adapting this use case, the aim of this paper is not to discuss the use of the methodology; instead, it aims to describe how the application's runtime information can be represented by our model after it has been made provenance-aware.

We choose a bioinformatics case (however many other domains could be, and have been, used, cf. [12,2]), in which a bioinformatician is conducting experiments to find protein sequences with interesting properties. This particular use case was chosen because it is high performance and has fine grain parallelism, which implies that recording process documentation may be difficult. Hence, showing that our approach performs in this difficult application provides support for the conclusion that the approach will work for a large set of applications with less demanding requirements.

In this section, we provide an overview of this bioinformatics example.

### *2.1. Biology*

Proteins are the essential functional components of all known forms of life; they are linear chains of typically a few hundred building blocks taken from the same set of about 20 different amino acids. Protein sequences are assembled following a code sequence represented by a polymer (mature messenger RNA). During and following the assembly, the protein will curl up under the electrostatic interaction of its thousands of atoms into a defined but flexible shape of typically 58 nm size. The resulting 3D-shape of the protein determines its function.

Amino acids can be *grouped* together by their chemical or physical properties. Those in the same group can often be substituted for one another in a protein sequence and the sequence will, in many cases, fold in the same manner. The ability to substitute amino acids is useful when trying to change or modify protein function. The aim of the Amino acid Compressibility Experiment (ACE) is to find other possible groups of amino acids that can be substituted for one another.

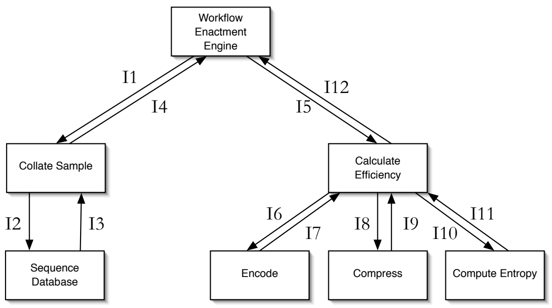
## *2.2. Experiment Description*

In this experiment, it is assumed that protein sequences that occur in nature are efficient, i.e. they use the least number of amino acids possible to represent their function. Based on this assumption, a group is tested for interest by substituting the amino acids specified by the group with a symbol representing the group and then measuring the efficiency of the recoded sequence. The efficiency of a protein sequence can be quantified in a computational setting through compression. If a sequence compresses well, then it is not efficient, whereas if the compression causes little reduction, then the sequence is efficient. The ACE, therefore, uses compression to attempt to find possible groups of interest.

The workflow for the experiment is shown in Figure 2. It starts with the creation of a sample, which is composed from individual sequences obtained from sequence databases made available on the Web (see www.ebi.uniprot.org). This collation provides enough data for the statistical methods employed by the compression algorithms. The experiment requires that the samples be composed from dissimilar sequences. This dissimilarity is determined by using a culling service such as PISCES [17]. Once a sample is created, the symbols in it are substituted with those of a given group (Encode). This recoded sequence is then compressed with compression algorithms, e.g., gzip, bzip2 or ppmz, to obtain the length of the compressed sequence (Compress). The Shannon entropy is then computed on the recoded sequence to provide a standard for comparison (Compute Entropy). This standard removes the influence of two factors from the calculation of compressibility: the particular data encoding used to represent the groups, and the non-uniform frequency of groups. From the results, an information efficiency value is computed for the sample that is relative to both the compression method and group coding employed and takes into account the size of the sample (Calculate Efficiency). The information efficiency values for different groups can then be plotted to find those that are the largest and thus are good candidates for further investigation.

## *2.3. The Provenance-Aware ACE Application*

Applications can be made provenance-aware by applying the PrIMe methodology [15], which analyses the application in the context of use case questions provided by the user and identifies those components in the application that should be given process documentation recording functionality. These components, or *actors* in PrIMe, record their part in the application's processes, recording p-assertions to a provenance store. Our model defines three forms of p-assertions that together provide all the information necessary to



| | |
|---|---|
| I1: collate sample request | I7: encoded sample |
| I2: database request | I8: encoded sample |
| I3: sequences | I9: compressed sample |
| I4: collated sample response | I10: encoded sample |
| I5: calculate efficiency request | I11: entropy |
| I6: sample | I12: information efficiency value |

**Figure 2.** ACE workflow.

document all the relevant information about an application's execution. The three forms of p-assertion are as follows.

**Interaction p-assertions** record the details of an actor's interactions — its role in the interaction (receiver or sender) the identity of the other party in the interaction and details about the data sent in the interaction.

**Relationship p-assertions** record the causal connections between incoming interactions and outgoing interactions, for example that a message was sent because another message was received.

**Actor state p-assertions** record important information the actor has access to *as it relates to an interaction*.

For example, each box in Figure 2 would be modelled as an actor and would record various different p-assertions as required in order to capture the right kinds of process documentation to answer the use case questions. In the figure, each arrow is annotated by an interaction identifier (I1 through I12), thus each would have a corresponding interaction p-assertion recorded for it as well as any necessary relationship and actor state p-assertions. Everytime the application is executed new p-assertions are created documenting that execution.

## 3. A Realisation of the Conceptual Model: The P-Structure

In what follows, we walk the reader though the *p-structure* — the model by which process documentation is organised in order to facilitate later querying. In Figure 3, we have the high level view of the p-structure as represented by an XML document. The fundamental method of organisation within the p-structure is the *interaction record*, thus the p-structure will contain multiple interaction records all of which together will contain the process documentation recorded for all the interactions made in the application.

```
– <ps:pstruct>
   + <ps:interactionRecord></ps:interactionRecord>
   + <ps:interactionRecord></ps:interactionRecord>
   + <ps:interactionRecord></ps:interactionRecord>
   + <ps:interactionRecord></ps:interactionRecord>
   + <ps:interactionRecord></ps:interactionRecord>
   + <ps:interactionRecord></ps:interactionRecord>
   + <ps:interactionRecord></ps:interactionRecord>
   + <ps:interactionRecord></ps:interactionRecord>
   + <ps:interactionRecord></ps:interactionRecord>
   + <ps:interactionRecord></ps:interactionRecord>
   + <ps:interactionRecord></ps:interactionRecord>
  </ps:pstruct>
```

**Figure 3.** The p-structure

### *3.1. Interaction Records*

The interaction record structure contains all the information that relates to one interaction. An interaction is defined as the passing of one message from one actor to another (therefore the response to an interaction is stored within its own interaction record). Thus, the interaction record stores all the p-assertions made about that interaction from *both* parties involved in it (i.e. the sender and the receiver). [4] In our ACE example, every time an interaction occurs between two actors (for example the workflow enactment engine — termed the ACE Enactor and the calculate efficiency actor) a new interaction record would be added to the p-structure.

```
– <ps:interactionRecord>
   + <ps:interactionKey></ps:interactionKey>
   + <ps:sender></ps:sender>
   + <ps:receiver></ps:receiver>
  </ps:interactionRecord>
```

**Figure 4.** Interaction Records

In Figure 4, we show the XML document for an interaction record. It contains the *sender* and *receiver* views on the interaction as well as the *interaction key* — a unique identifier for this interaction. The interaction key (shown in Figure 5) contains a unique identifier for this interaction as well as information relating to the sender and receiver; specifically, in this case, we use the WS standard for addressing, which involves specify-

[4]The architecture allows for the process documentation from the sender and receiver for one interaction to be stored in separate provenance stores and thus is physically different interaction records. However, linking mechanisms enable these to be linked effectively making the physically different interaction records logically the same.

ing an *endpoint reference* that contains a URL that locates the sender/receiver. Thus, for this interaction we can see that the message source is the ACE Enactor who is sending a message to the Collate Sample actor, located by the given endpoint reference.

```
- <ps:interactionKey>
  - <ps:messageSource>
    - <ns1:EndpointReference>
      - <ns1:Address>
          http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/AceEnactor
        </ns1:Address>
      </ns1:EndpointReference>
    </ps:messageSource>
  - <ps:messageSink>
    - <ns2:EndpointReference>
      - <ns2:Address>
          http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/CollateSample
        </ns2:Address>
      </ns2:EndpointReference>
    </ps:messageSink>
    <ps:interactionId>e71acbcc-6e05-46d0-b2ab-a65c1d1d453d2</ps:interactionId>
  </ps:interactionKey>
```

**Figure 5.** An interaction key

The sender and receiver views within the interaction record each contain the sets of p-assertions made by each actor as well as their asserter information, which contains the identity of the responsible party for the p-assertions in the interaction record. In this example, this is simply their endpoint reference (see Figure 6 for the XML document representing the sender view). In this case, the sender is the ACE Enactor and so all the p-assertions within this view will originate from this actor.

```
− <ps:asserter>
  − <ns9:EndpointReference>
    − <ns9:Address>
        http://pasoa-vmware1.ecs.soton.ac.uk/experiments/ace/AceEactor
      </ns9:Address>
    </ns9:EndpointReference>
  </ps:asserter>
```

**Figure 6.** The Sender View

## *3.2. P-Assertions*

Above we briefly describe the data structure that represents high level information about an interaction, such as the interaction identifier, the parties involved in the interaction as well as their endpoint references. Apart from this information, each interaction record contains all the p-assertions made by actors for the interaction (where the p-assertions made by each actor are stored in the appropriate actor view). In what follows, we describe in detail each of the three p-assertions mentioned earlier and explain their contents.

### *3.2.1. Interaction P-Assertions*

As previously explained, interaction p-assertions are generated for each interaction that takes place. An application will, therefore, generate sets of p-assertions from both parties involved in the interaction. As such, each interaction record will contain interaction p-assertions generated from both involved actors (i.e. one set for each *view*). Figure 7 shows the XML document for an interaction p-assertion. We can see that within this structure, there are three components. A *local p-assertion id*, a *documentation style* and a *content* element. The local p-assertion id identifies and distinguishes this p-assertion from every other that has been generated for this interaction.

```
− <ps:interactionPAssertion>
    <ps:localPAssertionId>2</ps:localPAssertionId>
    <ps:documentationStyle>http://www.pasoa.org/docstyle/AceVerbatim</ps:documentationStyle>
  + <ps:content></ps:content>
  </ps:interactionPAssertion>
```

**Figure 7.** An interaction p-assertion

The documentation style element, refers to one or more of a number of different ways of encoding the content of the p-assertion. Recall that interaction p-assertions include a description of the content of the message that has been sent or received. The form of this description is determined by the documentation style. The simplest documentation style is one in which the original contents of the message are simply copied verbatim within the p-assertion (this is the case with the example given in Figure 7). Other documentation styles may be used, however, to address security concerns or other non-functional requirements. For example, there are documentation styles that will encrypt the information within the p-assertion that relates to the content of the original message, or a pointer can be used to reference the original data if it is decided that the original information should not be kept with the p-assertion (for example due to space requirements). Other documentation styles are also available and descriptions of such can be found in [9].

Within an interaction p-assertion, the content element contains the payload of the message the p-assertion is documenting. As mentioned, it may have been encoded using a documentation style or it may be a verbatim copy of the information contained in the original message.

### *3.2.2. Actor State P-Assertions*

To record actor state p-assertions, the model provides the data structure represented in Figure 8. This allows for internal data of an actor to be recorded as it refers to an interaction. The reference to an interaction is indicated by the local p-assertion id element. The *data* element is used to contain any application specific data that the actor is using within the interaction (the example given is a *collate summary* used by the collate sample actor).

```
- <ps:actorStatePAssertion>
    <ps:localPAssertionId>2</ps:localPAssertionId>
  - <ps:content>
    - <ns7:data>
      + <ns8:collateSummary></ns8:collateSummary>
      </ns7:data>
    </ps:content>
  </ps:actorStatePAssertion>
```

**Figure 8.** An actor state p-assertion

### *3.2.3. Relationship P-Assertions*

To connect outgoing interactions to incoming interactions, we provide the relationship p-assertion. This provides the means to link information arriving in an interaction with the information sent out in another information. The combination of both interaction p-assertions and relationship p-assertions gives a complete account of the data flow within an application and allows a causal graph to be generated from process documentation. Thus, working backwards from a particular result, it is possible, using both relationship and interaction p-assertions to trace back the causes of that result from earlier interactions and any associated actor state p-assertions.

```
− <ps:relationshipPAssertion>
    <ps:localPAssertionId>3</ps:localPAssertionId>
  + <ps:subjectId></ps:subjectId>
  − <ps:relation>
      http://www.pasoa.org/schemas/ace/relationships/happenedAfter
    </ps:relation>
  + <ps:objectId></ps:objectId>
  </ps:relationshipPAssertion>
```

**Figure 9.** A relationship p-assertion

The relationship p-assertion (see Figure 9) contains information about the id of the interaction it is associated with, the name of the relation, e.g. *caused by* or *depends upon* or, in the figure: *happened after*, as well as information relating to the inputs of the relation — called the *objects* of the relation, and the outputs — called the *subjects* of the relation.

A relation can have multiple objects but only one subject. In this way, they are very much like functions and reflect the idea that the actor is performing some operation on some input and producing an output. Figure 10 shows the data structure for the *objectID*.

This contains an *interaction key* indicating from which interaction this input was from, a *view kind*, a *local p-assertion id* identifying the relationship p-assertion to which

```
− <ps:objectId>
   + <ps:interactionKey></ps:interactionKey>
   − <ps:viewKind xsi:type="pasoa:SenderViewKind">
        <ps:description>isSender</ps:description>
     </ps:viewKind>
     <ps:localPAssertionId>1</ps:localPAssertionId>
     <ps:parameterName>request</ps:parameterName>
   − <ps:objectLink>
        <ps:provenanceStoreRef/>
     </ps:objectLink>
  </ps:objectId>
```

**Figure 10.** An objectId

this object id belongs, a *parameter name*, which is a name given to this *input* data reflecting the role it plays in the relationship (in the figure, the role that is played by this data item is that of a request) and, finally, an *object link*. The object link can refer to the location where the input data representing the object can be found, which may be the location of another provenance store where the actor who sent this information is sending its p-assertions.

Finally, the relationship p-assertion contains a *subjectId* element that identifies the parameter name of the data item that is the *output* of the relation (again, in this example it represents a request) as well as the id of the p-assertion it is related to (shown in Figure 11).

```
− <ps:subjectId>
     <ps:localPAssertionId>2</ps:localPAssertionId>
     <ps:parameterName>request</ps:parameterName>
  </ps:subjectId>
```

**Figure 11.** A subjectId

We have now completed the description of the p-structure. With the information contained in this model and with this organisation, it is a simple matter to provide components that can search the p-structure to obtain information related to provide users with information about their data. To see a detailed description of how this can be achieved see [13].

## 4. Performance

Enabling provenance-awareness in applications inevitably carries some overhead. In order to examine the performance implications of recording process documentation, we evaluate an implementation of the provenance store, PReServ, both independently from ACE and when integrated with it. We describe PReServ, a Java-based Web Service in detail elsewhere [11].

The experiments were run on the Iridis Computing Cluster at the University of Southampton. Iridis contains several sets of nodes (i.e. computers). The set used in the experiments consisted of 237 nodes each with two AMD Opteron processors running at 2.2 GHz and 2 GB of RAM. A local storage of 25 GB is made available as scratch disk space for running jobs. Each node has access to a shared file system where results and executables can be stored. The Provenance Service runs on a node with 4 Dual Core AMD Opteron processors running at 2.4 Ghz and 2 GB of RAM. The database stores its files on a shared filesystem with 1.4 TB of space available for the storage of process documentation. All nodes are connected by Gigabit Ethernet. Each job run on Iridis obtains an entire node for itself and due to the policy of Iridis's maintainers, a maximum of 40 jobs can be run in parallel by any one user.

One measure for accessing the performance of PReServ in a generally applicable way is to measure its throughput. In Figure 12, we see that as the number of clients increases the throughput also increases. The x-axis shows the number of jobs being run in parallel. The y-axis shows how many threads or clients to the provenance store are being run on each node and the z-axis shows the throughput in number of p-assertions per 10 minute period. Typically, a systems throughput will increase until a maxima and then throughput levels off and may slightly decrease. We attempted to find such a maxima for the provenance store by increasing the number of threads (i.e. clients) per node. However, throughput continued to increase up to the point where 32 nodes each had 16 threads recording p-assertions at the same time (i.e. 560 client connections). At this point, 234025 p-assertions (2.2 GB of p-assertions) were recorded in a 10 minute period, which means that on average a 10K p-assertion was recorded every 2.6 milliseconds.

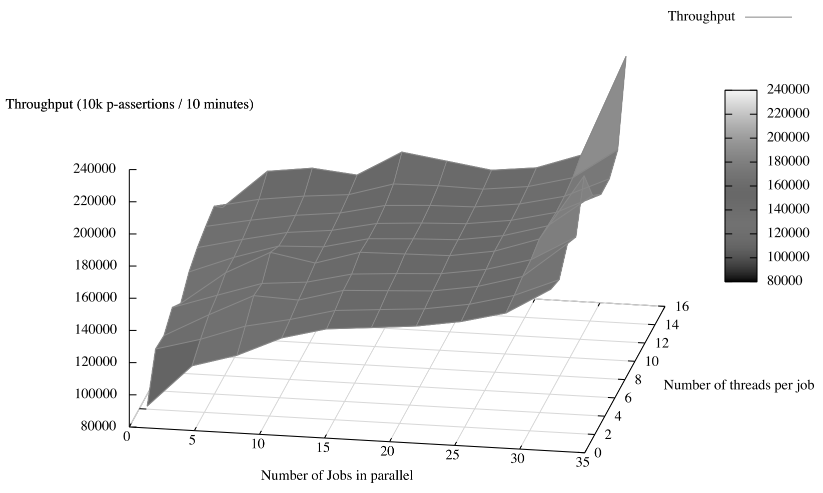


**Figure 12.** Throughput as the number of jobs and threads per jobs increases.

From this measure of throughput, developers can begin to ascertain the impact recording p-assertions will have on application performance. We now discuss the impact making ACE provenance-aware had on its performance.

The ACE workflow shown in Figure 2 was run as a set of 80 jobs on the Iridis cluster. Each job analysed 900 unique groups on 5 different 100K collated samples, thus, a job generates 4500 information efficiency values. A set of 900 groups is a 50K file. Process documentation that represents the provenance of each information efficiency value is stored across two provenance stores. After one run of ACE, roughly 14 GB of information is stored in the provenance store. We measured the impact that integrating p-assertion recording with ACE had on the runtimes of ACE's component jobs. The impact on the average, maximum, and minimum job runtimes is shown in Table 1.

| | Job Runtime | Job Runtime with Recording | Difference |
|---|---|---|---|
| Maximum | 23:20 | 26:24 | 3:04 |
| Average | 22:24 | 25:17 | 2:53 |
| Minimum | 20:39 | 23:09 | 2:30 |

**Table 1.** Maximum, Minimum and Average job record times both with and without p-assertion recording

From this data, we conclude that there is a 13% overhead for provenance-awareness in this detailed and highly parallel use case. As such we argue that the performance drop that results in adding provenance-awareness to the application is acceptable and more than offset by the added functionality provided to the user. Moreover, because of the difficulty of ACE, this result should give confidence to developers that their applications can be made provenance-aware without unacceptable degradation in performance.

## 5. Related Work

The subject of provenance has not gone without notice in the literature. Under the heading of lineage, Bose and Frew present a comprehensive overview of provenance related systems [3]. Likewise, Simmham et al. give a survey of provenance in the domain of e-Science [16]. A compilation of the current state of the art is given by Moreau and Foster [14]. From an analysis of these works, we assert that the focus of provenance research has been on the implementation of concrete systems for provenance in the context of either specific domains (i.e. geographic information systems, chemistry, biology) or technologies (i.e. databases). In contrast, this work focuses on a conceptual organisation of process documentation independent of technology or domain.

Work in the database community has focused on the data lineage problem, which can be summarised as: given a data item, determine the source data used to produce that item. Cui et al. present a number of algorithms for determining the lineage of data in relational databases [6] and data warehouse environments [7]. Buneman et al. also develop a formal model of provenance for database systems that applies to both hierarchical and relational databases [4]. Our data model differs from these approaches because it can be used to represent processes that occur both inside and outside database environments.

In the e-Science community, work has focused on provenance for workflow based environments. For example, Zhao et al. present a model for provenance in the Virtual Data System (VDS), which uses the workflow graph to tie together various data elements recorded during the execution of the workflow [19]. The benefits of this approach over the p-structure is that information stating causal dependencies is inferred from the workflow and thus must not be created by the actors within the workflow. However, because a workflow is a plan and not what actually occurred, the p-structure captures the actual causal connections according to execution, as opposed to the original workflow which implicitly captures all possible connections for all possible executions. The p-structure also differs from workflow centric systems like VDS, myGrid [18], and Kepler [1] in that it supports any type of execution environment. For example, process documentation compatible with the p-structure can be generated by Java programs, shell scripts or workflow enactment engines.

Developments in the Semantic Web community have concentrated on adding annotations to Resource Description Framework (RDF) graphs that describe the provenance of the nodes of the graph [5,8]. As with the other systems presented, these approaches are technology dependent as they rely wholly on RDF. Comparatively, the p-structure, because of its conceptual definition, can be represented using multiple technologies including RDF.

Our approach differs from all of these systems with its distinction of the *concept* of provenance from its *implementation*, as we described earlier.

## 6. Conclusion

In this article, we have described how the provenance open data model can be used with an example bioinformatics application. We have described how the kinds of information produced by the application can be captured in a structured way by the data model so that queries can be performed later. Having such an open data model enables it to be applied to any application domain, allowing for its wide adoption across many such domains as we described earlier. The structured format of the captured data provides many benefits that allow the model to scale up to large amounts of data that can be distributed across many provenance stores.

## References

[1] Ilkay Altintas, Oscar Barney, and Efrat Jaeger-Frank. Provenance collection support in the kepler scientific workflow system. In Luc Moreau and Ian Foster, editors, *International Provenance and Annotation Workshop, IPAW 2006*, volume 4145 of *Lecture Notes in Computer Science*, pages 118–132. Springer-Verlag, 2006.

[2] S. Álvarez, J. Vázquez-Salceda, T. Kifor, L.Z. Varga, and S. Willmott. Applying provenance in distributed organ transplant management. In L.Moreau and I.Foster, editors, *LNCS: Proceedings of the International Provenance and Annotation Workshop (IPAW'06)*, volume 4145, pages 28–36, Chicago, Illinois, May 2006. Springer-Verlag.

[3] Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: a survey. *ACM Computing Surveys*, 37(1):1–28, 2005.

[4] P. Buneman, S. Khanna, and W.C. Tan. Why and where: A characterization of data provenance. In *Int. Conf. on Databases Theory (ICDT)*, 2001.

[5] Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 613–622, New York, NY, USA, 2005. ACM Press.

[6] Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *Proceedings of the 16th International Conference on Data Engineering (ICDE'00)*, San Diego, California, February 2000.

[7] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. *The VLDB Journal*, 12(1):41–58, 2003.

[8] Joe Futrelle. Harvesting rdf triples. In Luc Moreau and Ian Foster, editors, *International Provenance and Annotation Workshop, IPAW 2006*, volume 4145 of *Lecture Notes in Computer Science*, pages 64–72. Springer-Verlag, 2006.

[9] 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, 2006.

[10] 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)*, pages 201–208, July 2005.

[11] Paul Groth, Simon Miles, and Luc Moreau. PReServ: Provenance Recording for Services. In *Proceedings of the UK OST e-Science Fourth All Hands Meeting (AHM05)*, September 2005.

[12] Guy K. Kloss and Andreas Schreiber. Provenance Implementation in a Scientific Simulation Environment. In *Proceedings of the International Provenance and Annotation Workshop (IPAW)*, pages 37–45, Chicago, Illinois, USA, May 2006.

[13] Simon Miles, Paul Groth, Steve Munroe, Sheng Jiang, Thibaut Assandri, and Luc Moreau. Extracting Causal Graphs from an Open Provenance Data Model. *Concurrency and Computation: Practice and Experience*, 2007.

[14] Luc Moreau and Ian Foster, editors. *Provenance and Annotation of Data — International Provenance and Annotation Workshop, IPAW 2006*, volume 4145 of *Lecture Notes in Computer Science*. Springer-Verlag, May 2006.

[15] S. Munroe, S. Miles, L. Moreau, and J. Vazquez-Salceda. Prime: A software engineering methodology for developing provenance-aware applications. In *Proceedings of Sixth International Workshop on Software Engineering and Middleware, SEM 06*, pages 39–46, Portland, Oregon, 2006. The ACM Digital Library.

[16] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.

[17] G. Wang and Jr. R. L. Dunbrack. Pisces: a protein sequence culling server. *Bioinformatics*, 19:1589–1591, 2003.

[18] 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.

[19] Yong Zhao, Michael Wilde, and Ian Foster. Applying the virtual data provenance model. In Luc Moreau and Ian Foster, editors, *International Provenance and Annotation Workshop, IPAW 2006*, volume 4145 of *Lecture Notes in Computer Science*, pages 148–161. Springer-Verlag, 2006.