Achieving Reproducibility by Combining Provenance with Service and Workflow Versioning

Simon Woodman
School of Computing Science
Newcastle University
Newcastle upon Tyne, UK
simon.woodman@ncl.ac.uk

Hugo Hiden
School of Computing Science
Newcastle University
Newcastle upon Tyne, UK
hugo.hiden@ncl.ac.uk

Paul Watson
School of Computing Science
Newcastle University
Newcastle upon Tyne, UK
paul.watson@ncl.ac.uk

Paolo Missier
School of Computing Science
Newcastle University
Newcastle upon Tyne, UK
paolo.missier@ncl.ac.uk

Abstract

Capturing and exploiting provenance information is considered to be important across a range of scientific, medical, commercial and Web applications [1, 2, 3, 4], including recent trends towards publishing provenance-rich, executable papers [5]. This article shows how the range of useful questions that provenance can answer is greatly increased when it is encapsulated into a system that can store and execute both current and old versions of workflows and services. e-Science Central provides a scalable, secure cloud platform for application developers. They can use it to upload data – for storage on the cloud – and services, which can be written in a variety of languages. These services can then be combined through workflows which are enacted in the cloud to compute over the data. When a workflow runs, a complete provenance trace is recorded. This paper shows how this provenance trace, used in conjunction with the ability to execute old versions of services and workflows (rather than just the latest versions) can provide useful information that would otherwise not be possible, including the key ability to reproduce experiments and to compare the effects of old and new versions of services on computations.

Categories and Subject Descriptors

H.4.0 [**Information Systems Applications**]: General

General Terms

Design

Keywords

e-science, provenance, reproducability

WORKS'11, November 14, 2011, Seattle, Washington, USA.
Copyright 2011 ACM 978-1-4503-1100-7/11/11 ...$10.00.

1. INTRODUCTION

Provenance traces are traditionally exploited to give users the answer to questions such as 'how was this data generated?' In e-Science infrastructure, the result can be visualised in the form of a directed graph that shows how data, workflows and services combined to create the data that is of interest. In scientific research, this information is important as it explains the process used to generate the result (the data) and so allows others to judge its value. In some cases, a scientist may be inspired to adopt the analysis process to use on his/her own data.

Investment in e-Science has driven forward work on provenance [6, 7, 8, 9], particularly through the use of workflow enactors that choreograph the execution of an analysis, and so provide a single point at which provenance can be captured. This allows the enactors to record a provenance trace automatically, without any effort on the part of the user running the workflow. Further, they do not require the services they call to have been specially designed to generate provenance – everything is done by the enactor (though this has limitations as will be described). This is important as in most cases the services are provided by external organisations and so are not in the control of the user writing the workflow. In a typical workflow engine such a Taverna [10] or Kepler [11], users design workflows using a graphical editor; when enacted, these call REST, Web or other types of services, and the workflow enactor records the provenance trace.

Typically, a provenance trace is represented as a directed graph whose nodes are either *data items* or *processes* that take in data items as input, perform a computation and produce data items as output. The graph's arcs link data to processes, indicating that an item of data was either produced by or consumed by a process. The graph can contain the sub-graphs of independent workflow enactments, linked together by data items that are common to more than one enactment – for example a data item that is generated by one workflow and consumed by others. These extended graphs that combine a set of enactments have been described as being the traces of 'virtual workflows' [12, 13].

The most general provenance query returns all direct and indirect data dependencies that involve a data item D_i (uniquely identified by i), i.e. the fragment of the provenance graph that includes all paths to and from D_i:

> Q1. Find all direct and indirect dependencies of data item D_i.

This is generally considered the baseline query that all provenance management systems should support, and some effort has been invested in ensuring its efficient processing [14, 9, 15].

A further distinction is often made between *backwards* and *forwards* provenance, as follows:

> Q2. *Backwards provenance*: Find all computations and data items used to generate D_i (ancestor query).

Type Q2 queries are typically used to explain the presence of a certain data item in the output, as they return the fragment of the input data set that has contributed, directly or indirectly, to that output. This often has an intuitive interpretation in settings where the workflow maps data from one domain (a set of genes, for example) onto another (a set of metabolic pathways that involve some of the input genes). In this example, a query of type Q2 corresponds to *inverting* the mapping for a specific pathway P in the output, and thus it answers the biologist's question "which genes contributed to P?".

> Q3. *Forward provenance*: Find all data items that are derived from data D_i, or from service S_j or workflow W_k (descendant query).

Type Q3 queries are used to perform impact analysis, as they return all and only the data items whose value is influenced, directly or indirectly, by a certain input (or intermediate data located upstream in the provenance graph).

Note that both types of queries can be used in particular for debugging purposes. Q2 queries are useful when an error is found in D_i, to determine a probable cause. Symmetrically, Q3 queries determine which results have been affected by a faulty D_i.

There are however restrictions on the questions that can be answered based on this trace information, and on how that information can be usefully exploited. Major issues surround the important concept of reproducibility.

Reproducibility has long been considered vital in scientific research as it allows others to gain confidence in, and validate, published results [16]. In most disciplines, papers that report new findings are expected to include a 'Methods' section that shows how the results were produced, so that another scientist could reproduce them. Recent attempts at producing "executable papers" such as [17] are spurred new effort towards making published papers "provenance-rich papers" [5].

Whilst capturing a provenance trace is necessary it is not sufficient to allow a computation to be repeated – a situation known as *workflow decay* [18]. The problem is that while provenance systems can store information on how the data was generated, they do not store copies of the key actors in the computation: the workflows, services and data. This may be less of a problem for data and workflows; there are now widely used systems to store copies of workflows [19] and so if the provenance system records the version of the workflow used, and if that version is still available in such a repository then it could be used in reproducing a computation. Also, a diligent scientist may keep versions of the data used in the analyses. Therefore, in some cases, two of the three types of actors involved in a computation may still be available for re-use. However, this still leaves services. There are two problems:

- external services may become inaccessible, for example if they are removed by their owner.
- services are often intermittently updated by their owners, for example to fix bugs and improve performance. Therefore, even if a service remains accessible, the version that can be used may differ from that identified in the provenance trace of a computation. And, as there is no standard way for provenance systems to access and record the version number of a service used in a computation, then it may not even be possible to know if the currently available version of a service is the one that was referenced in a provenance trace.

This paper describes a system designed to overcome these problems. e-Science Central allows data, workflows and services to be stored and executed in multiple cloud computing environments. As will be described, it overcomes the above problems in three ways:

- it automatically stores and retains all versions of all data, workflows and services used in computations. Users are able to delete these artifacts but are warned if this will impact on reproducing a computation.
- it automatically stores a full provenance trace for all computations. This includes the version numbers of all data, workflows and services.
- it can automatically transform a provenance trace into a workflow, selecting the exact versions of the data, services and workflows that appeared in the trace. It can then enact this workflow.

This allows the system to reproduce exactly a computation recorded in a provenance trace, enabling users to answer the additional question:

> Q4. If the computation that generated trace T_l is re-run, are the results the same?

Furthermore, the ability to select specific versions of services and data items in computations based on provenance traces also allows users to investigate the relationship between workflow outcomes and the specific service or workflow versions:

> Q5. What is the effect on the results of the computation that generated trace T_l if different versions of one or more of the data, workflows or services contained within it are used?

In particular, scientists are often interested in comparing the original outcome of a workflow with that obtained by running the same workflow with newer versions of some of its components and services:

Q5a. What is the effect on the results of the computation that generated trace T_l if the latest versions of all the data, workflows or services contained within it are used?

Conversely, there may be concern over the results produced by the new version of a service S_j (or a workflow W_k). In this case, it is useful to have a record of outcomes obtained in the past from the same workflow, using older versions of the components:

Q5b. Does running computation that generated trace T_l with the previous version of S_j give a different result?

The ability to select specific versions of data, services and workflows also allows users to produce consistent, comparable results when they are analysing different datasets using the same workflow over a period of time in which new versions of the workflow, and the services it contains, may become available. If this can be done, it overcomes a widely reported problem for scientists using electronic tools that are not under their control – that once their tool chain for analysis reaches a certain length, it becomes increasingly likely that at least one of those tools will have been updated before all their experimental analyses are complete. This makes it very difficult to compare results for different datasets unless all analyses are repeated – something that can be time-consuming and expensive.

The rest of the paper is structured as follows. Section 2 describes e-Science Central, focusing on: its capabilities to store versions of services, workflows and data; and, its provenance capture. Section 3 then describes key user questions that the system can answer (which are beyond the capabilities of systems that cannot store and execute versions of services), and show how they are answered through workflows that are automatically generated based on queries over provenance traces. This is illustrated with examples. Section 4 describes the application which users are presented with in order to leverage this functionality. Section 5 compares this to related work before indicating future directions and drawing conclusions.

2. E-SCIENCE CENTRAL

e-Science Central is a portable cloud 'platform-as-a-service' that can be deployed on either private clusters or public clouds, including Amazon EC2 and Windows Azure. Cloud computing has the potential to give scientists the computational resources they need, when they need them. However, cloud computing does not make it easier to build the often complex, scalable secure applications needed to support science. e-Science Central was designed to overcome these obstacles by providing a platform on which users can carry out their research, and build high-level applications. Figure 1 shows the key components of the e-Science Central Science 'Platform as a Service' sitting on an 'Infrastructure as a Service' cloud. It combines three technologies – Software as a Service (so users only need a web browser to do their research), Social Networking (to support sharing and community interaction) and Cloud Computing (to provide storage and computational power). Using only a browser, users can upload data, share it in a controlled way with colleagues, and analyse the data using either a set of pre-defined services, or their own, which they can upload for execution and sharing. A range of data analysis and programming languages are supported, including Java, Javascript, R and Octave. From the point of view of users, this gives them the power of cloud computing without them actually having to manage the complexity of developing cloud-specific software – they can create services in a variety of languages, upload them into e-Science Central, and have them run transparently on the cloud. They can then also compose services and automate analysis using the Workflow editing and enactment facilities.

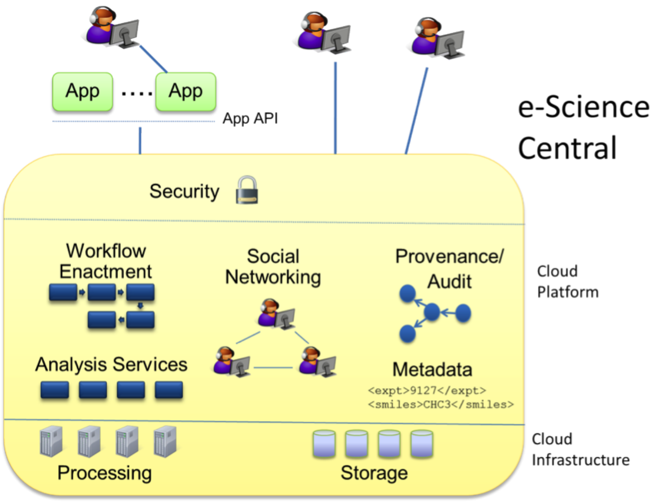


Figure 1: e-Science Central Components

Figure 2 shows a screenshot of the browser-based HTML 5 workflow editor. Below the editing pane are links to the results of previous runs of the workflow. Everything that can be accomplished by a user interacting with the system through a browser can also be accomplished through a programmatic REST based API [20]. This has allowed users to build, for example, mobile phone applications that use e-Science Central to store and analyse data that is captured 'in the field'. A complete description of e-Science Central is available in [21]. The rest of this section focuses only on those features that are key to collecting and exploiting provenance to answer the user questions posed in the introduction.

2.1 Versioning

Versioning is an integral storage feature in e-Science Central, allowing users to work with old versions of data, services and workflows. All objects (data, files and workflows) are stored in files through a virtual filestore driver than can be mapped onto a range of actual storage systems including standard local and distributed filesystems, and Amazon S3. When a file is stored, if a previous version exists then a new one is automatically created.

All operations that are made available through both the user interface and API allow a choice of version. For example in the workflow editor, shown in Figure 2, the user can select any service, and then choose any version of that service before running the workflow.

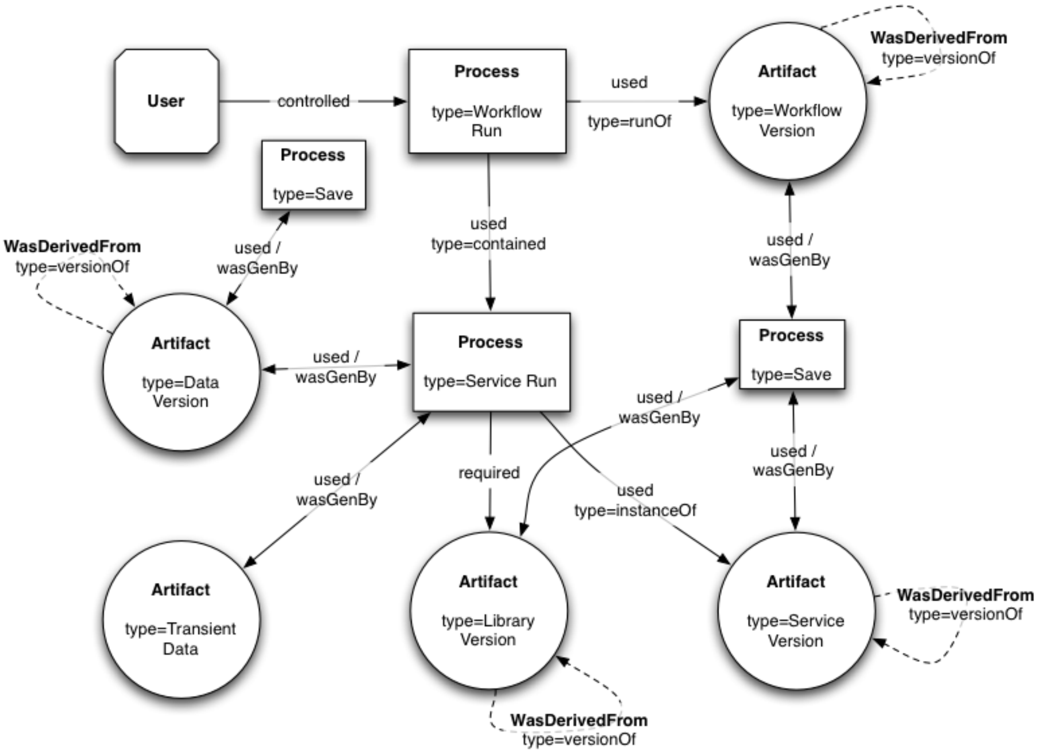


Figure 3: e-Science Central Provenance Model

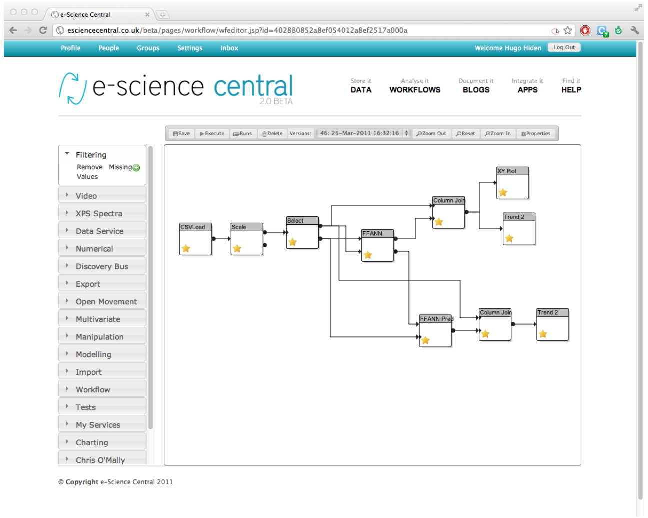


Figure 2: The HTML5 Workflow Editor

2.2 Capturing Provenance traces in e-Science Central

The provenance system within e-Science Central is used to capture the history and life cycle of every piece of data within the system. For instance: who created the data? who has downloaded the data? what version of which services (in what workflow) accessed the data? and who has the data been shared with?

The provenance data model, shown in Figure 3, is based on the Open Provenance Model (OPM) Version 1.1 [22], and can be used to produce a directed acyclic graph of the history of an object. Objects in OPM are categorised as either artifacts, processes or actors which correspond to nodes within the graph. Vertices in the graph represent relationships between two objects and are of types such as wasGeneratedBy, used, wasControlledBy and wasDerivedFrom. The OPM Core model has been extended with subclasses to identify the different types of processes and artifacts we are concerned with. For example, execution of a workflow and a service within a workflow have been differentiated. The relationship between workflow execution and service execution is of type *contained* is not strictly required by our model but its inclusion makes it easier to generate the different views of the process, as shown in Figure 4. The artifacts described in the model are also subclassed in order to differentiate between versions of data, services, libraries and workflows. Without this subclassing, it would be necessary to either encode the object type in the identifier (not desirable as it adds an unnecessary layer of obfuscated information) or perform many lookups to answer the question 'what type of object has a particular identifier'. The fact that the *User* controls all of the processes has been omitted from the diagram to aid clarity.

The relationships between processes and artifacts is specified in the OPM standard with the *Save* process taking one version of an artifact and generating a new version.

The self referential relationship between artifacts of type *WasDerivedFrom*, shown with a dashed line, indicates a second level, inferred relationship omitting the *Save* process. This implies that $D_{i,v}$ is in some way derived from $D_{i,v-1}$ when $v > 0$.

Our model deals with two different types of data artifact: *Data Version* and *Transient Data*. This is due to the semantics of workflows in e-Science Central: data generated by a workflow must be explicitly saved using a service which 'exports' the data back into the e-Science Central repository. Any data which is not explicitly exported will be discarded when the workflow completes. The reason for this design choice is that most intermediate data is of little use and may or may not be in a format which can be used again. If the user decides it is necessary to keep this data it is easy to do so, otherwise it is taking up unnecessary space. Section 6 discusses using the provenance information to determine which data items can be safely discarded and which should be persisted.

Service Versions in our model are annotated with information about their repeatability semantics. Specifically, whether they are *deterministic* and *idempotent.* These attributes are set by the author of the service. The *deterministic* attribute states whether for the same inputs the service will always provide the same outputs. The term *idempotent* is used in this case to indicate that the service does or does not have any side effects. For example, services that insert rows into a database would not be considered idempotent whereas those that updated rows would be.

The example in Figure 4 shows how a chemical informatics workflow creates two new models (one from a PLS service the other from a Neural Network) These were generated by a model building workflow which in turn used an input file containing chemical descriptor values and was controlled (executed) by the user 'Hugo Hiden'. A feature of OPM is that it allows for alternate views of how an artifact was generated. The white and grey sections of Figure 4 show differing levels of detail on how the two models were constructed. Whilst the white version treats the workflow as a single 'black box', the grey version of events breaks the workflow into its constituent parts (simplified for this example).

Users of e-Science Central can manually traverse the provenance graph using a textual or graphical representation from the object properties. The level of detail they will see is dependent on their access privileges to the constituent objects. For instance, if they have access to the workflow they can see the grey section of Figure 4 whereas if they do not, they can only see the white section. One side effect of this is that if a data set is made public, users of it can see that a number of workflows have been executed on it, but cannot necessarily see what those workflows do (unless their owner has made them public too). This makes as much information available as possible whilst respecting the privacy concerns of the workflow owner.

Provenance traces are stored in a graph database to be analysed. The next Section describes how this is done, and the queries that are used to answer the user questions posed in the Introduction.

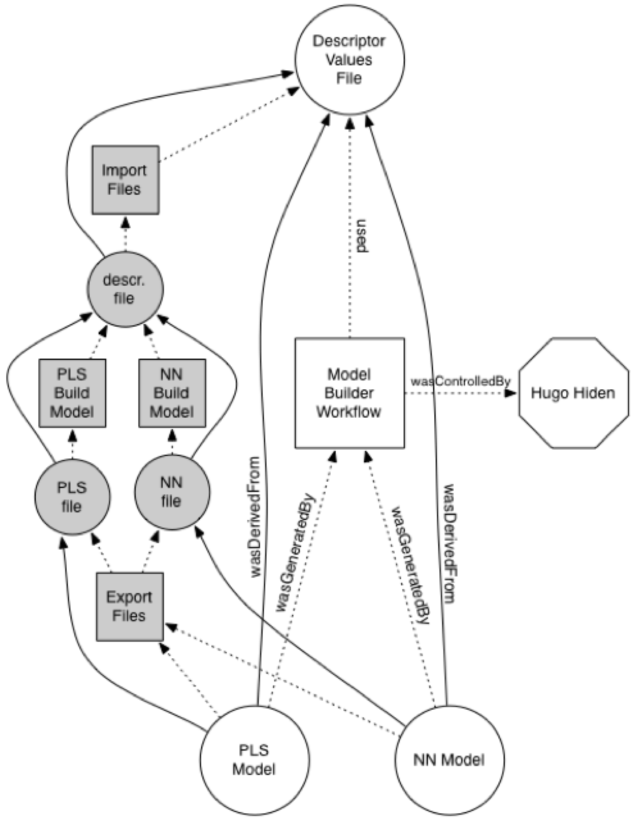


Figure 4: Provenance Graph Example

2.3 Provenance Database Implementation

Given that the provenance structure being stored is a directed acyclic graph, we chose to store it in the non-relational graph database, Neo4j [23] Neo4j differs from traditional relational databases as its structure is not in terms of tables and rows and columns, but in nodes, relationships and properties. This provides a much more natural fit to our model, and allows us to store the provenance graph directly instead of encoding it in a relational model. Neo4j has also been shown to scale well, and most importantly, to perform well in terms of queries even when storing a very large graph structure [24]. Libraries are provided to allow users to work with Neo4j directly in either Java or Ruby. We have used the Java library.[1]

Best practises with Neo4j indicate that one should wrap a node in a Java object with the member variables of the object mapped onto properties of the Neo4j node. We have followed this pattern and firstly created a small OPM library that is able to store processes and artifacts in Neo4j. Our implementation is built on top of the OPM library with subclasses to represent our specific artifacts and processes as described in the previous section. Relationship types are represented as a Java Enumerated type. Within the e-Science Central model most artifacts are identified by a combination

[1] A server version of Neo4j now exists but this work was started prior to it being available. We plan to update the implementation to use the server version in the future.

of *objectId* and *versionId*. These are stored in Neo4j indexes to allow quick lookups of the nodes and prevent duplicate nodes representing a single entity.

Various components within e-Science Central all log provenance information. From the outset the system was designed such that the order in which provenance events are received by the server is not important. For instance, we could have an arbitrary interleaving of the events which signify that a 'Service has run' and that the 'Service accessed some data'. Each of these events contains a subset of information which must be added to the provenance database: the first event contains the service name, start and completion time (and some other information) whereas the latter defines the identity of the data which was read. Using transactions and Neo4j indexes ensures that we will end with a single node in the database with properties containing the information from all events associated with that node.

Instead of SQL queries, Neo4j supports an operation known as a *traversal* whereby the user defines a starting node and rules about what types of relationship/node to 'traverse'. The result is a set of paths from the starting node to the acceptable ending nodes. Traversals are used extensively and are described in detail in Section 3. The semantics of the traversals are that they are guaranteed to visit all nodes in the set, but not all relationships. This is important when considering the provenance of some data as if the workflows contain branching and joining only one path will be returned. A solution to this is described in Section 3.

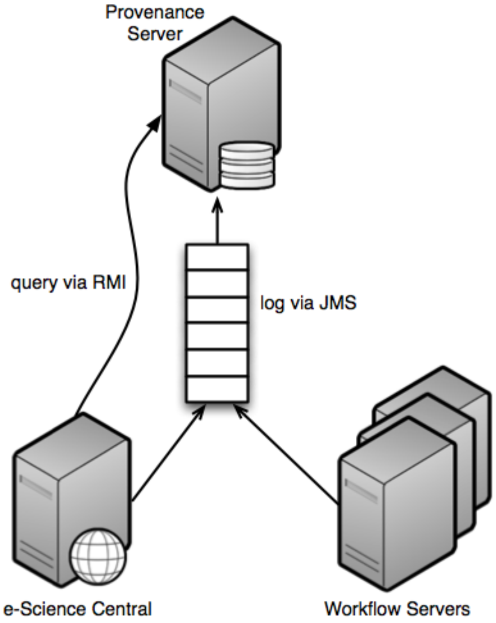


Figure 5: Provenance Capture Architecture

We have developed our provenance capture component as a standalone application 'outside' of the e-Science Central server. It is decoupled from e-Science Central with a durable JMS queue which allows e-Science Central to minimise the number of synchronous write operations which must be performed in a synchronous request. This also allows other components, such as the workflow engine, to log provenance directly without making another request to the server. For interrogating the provenance of an object the provenance server provides a Java RMI interface which can traverse the data store and return the provenance trace requested.

3. ANSWERING THE USER QUESTIONS

In order to answer the questions from Section 1, the algorithms shown in Table 1 need to be run over the provenance graph collected by e-Science Central. Because we are now operating over versions of data and services, we extend the notation to include a version number (e.g. $\mathrm{D_{i,v}}$). The subsequent sections will describe the implementation of each of the operations in turn.

3.1 TraceContains($\mathbf{D}_{i,v}$)

Finding all the direct and indirect dependencies of an artifact, $\mathrm{D}_{i,v}$ in this case, although the following also applies for $\mathrm{S}_{j,v}$ and $\mathrm{W}_{k,v}$, involves constructing the sub graph of all nodes directly reachable from $\mathrm{D}_{i,v}$ whilst following edges of types *Used*, *WasGeneratedBy* and *Contained*. This sub graph will contain all ancestors of $\mathrm{D}_{i,v}$, which were involved in its creation as well as all descendants which were derived from $\mathrm{D}_{i,v}$.

The direct and indirect dependencies of the PLS file ($\mathrm{D}_{i,v}$ in this case) artifact in Figure 4 would include the ancestors PLS Build, descr. file, Import Files and Descriptor Values File. It would also contain the descendants Export Files and PLS Model and the sibling Model Builder Workflow (which describes the process at a higher level). In order to construct this in the Neo4j Traverser framework we use the following query:

```
Traverser provenance = plsFile.traverse(
  Traverser.Order.DEPTH_FIRST,
  StopEvaluator.END_OF_GRAPH,
  ReturnableEvaluator.ALL,
  OpmRel.USED, Direction.BOTH,
  OpmRel.WAS_GENERATED_BY, Direction.BOTH,
  ProvRel.CONTAINED, Direction.INCOMING);
```

This traverser begins from a node in the graph database called `plsFile`, will perform a depth first search to the end of the graph (note that a Traverser can be set to only search to a particular depth if required), will return all types of Node and will traverse the following relationships: Used and WasGeneratedBy, both when the relationship is either *incoming* or *outgoing* from the node being inspected, and the Contained relationship when it is *incoming*. The former two build up the left hand side of the provenance in Figure 4 and the latter, whilst not strictly necessary, provides the Workflow Run which provides the user with some contextual information.

Section 2.3 describes how *Traversals* in Neo4j are guaranteed to visit every node but not every edge. This means that as there are two paths leading from PLS Model to Descriptor Values File, both nodes will be present in the results but one of the relationships will be omitted. In order to build up a complete representation of the provenance we must include those relationships too which can be done in the following way. The traverser contains an `Iterable` set of Nodes and it is possible to obtain the relationships of a particular type from the Node. Therefore, we iterate over the collection of Nodes in the Traverser, calling `getRelationships(Type,`

	Question	Operations Required	Result
1	Find all direct and indirect dependencies of $D_{i,v}$	TraceContains($D_{i,v}$)	{Trace}
2	*Backwards Provenance*: Find all computations and data items used to generate $D_{i,v}$ (ancestor query)	Provenance($D_{i,v}$)	Trace
3	*Forwards Provenance*: Find all data items that are derived from data $D_{i,v}$, service $S_{j,v}$ or workflow $W_{k,v}$ (descendant query)	DerivedFrom($D_{i,v}$), DerivedFrom($S_{i,v}$) or DerivedFrom($W_{i,v}$)	{Data}
4	If the computation that generated trace T_i is re-run, are the results the same?	W_i = mkWorkflow(T_i); Execute(W_i)	(Boolean, {($D_{i,v}$,$D_{i,v}$)})
5	What effect on the results of computation that generated trace T_i does it have if different versions of one or more of the data or services contained within it are used?	W_i = mkWorkflow (T_i); W_j = Substitute (W_i,{DataVersionSubstit}, {ServiceVersionSubstit}), Execute(Wj)	{Data}

Table 1: Operations Required for the Provenance Questions

`Direction`) for each Node (and using the same Relationship Types we used to configure the original Traverser).[2]

3.2 Provenance($D_{i,v}$)

This operation involves computing the ancestry of a data item $D_{i,v}$, showing the artifacts which were used in its creation. The output, in terms of a directed acyclic graph will be the ancestry part of the previous operation, omitting the descendant part. To include only the ancestry we modify the previous query with the direction in which the relationships are followed, choosing only to follow *outgoing* relationships.

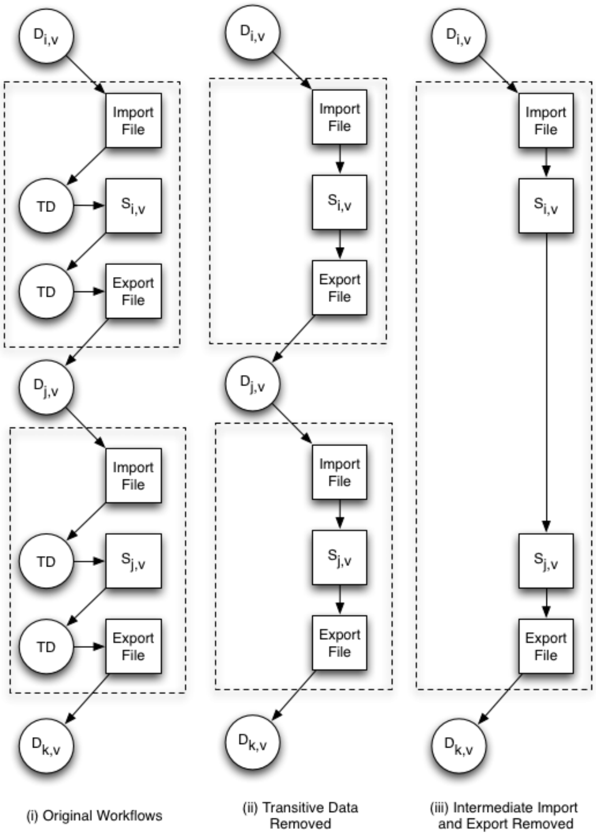
```
Traverser provenance = plsFile.traverse(
  Traverser.Order.DEPTH_FIRST,
  StopEvaluator.END_OF_GRAPH,
  ReturnableEvaluator.ALL,
  OpmRel.USED, Direction.OUTGOING,
  OpmRel.WAS_GENERATED_BY, Direction.OUTGOING,
  ProvRel.CONTAINED, Direction.INCOMING);
```

3.3 DerivedFrom($D_{i,v}$ or $S_{j,v}$)

This question is the dual of the Provenance($D_{i,v}$). Whereas the Provenance($D_{i,v}$) determines the ancestry of $D_{i,v}$, DerivedFrom($D_{i,v}$) is concerned about what effects $D_{i,v}$ has had on other data items, i.e. its descendants. Therefore, the solution to this question is identical to the one proposed in Section 3.2 but with the Traverser (and invocations of `getRelationships(...)` being configured to navigate *incoming* relationships for the types WasGeneratedBy and Used.

3.4 mkWorkflow(T_l)

This operation takes a provenance trace and generates from it a workflow description that can be executed. Workflows in e-Science Central take the form of a set of in-memory Java objects which can be serialised into JSON (Javascript Object Notation) for transfer to the web browser or XML for persisting into the database. The provenance trace we start with is an in-memory representation of nodes which represent either data or services and their relationships. The process of generating the e-Science Central workflow from this is an engineering challenge of parsing the graph and extracting the relevant parts.



(i) Original Workflows

(ii) Transitive Data Removed

(iii) Intermediate Import and Export Removed

Figure 6: Stages of the mkWorkflow(T_l) Operation

[2] This pattern is the standard solution to the problem of obtaining all the Nodes and Edges between two Nodes and documented in the Neo4j documentation.

There are a number of mis-matches which must be overcome. Firstly, the provenance graph contains nodes which represent Transient Data passing from one service to another. These must be removed as in e-Science Central services are directly connected and data is passed implicitly. Secondly, there are two ways that data may be passed into a service: as an edge representing the output of another service or as a property which the user is able to edit prior to running the workflow. Properties of the relationship in the provenance trace distinguish between these two cases and also contain unique names which allows multiple connections between two services in the same workflow.

The most challenging part of the mkWorkflow(T_l) operation, as shown in Figure 6 concerns dealing with traces which contain multiple workflows – that is, where the output of one workflow has been used as the input to another. In this case, we create one virtual workflow which spans multiple workflows which previously ran to produce $D_{i,v}$. The challenge here is that the workflows are not directly *linkable*, the edge blocks which do the data import and export must be removed, and the connections re-routed to the previous and next block in the workflow. This is safe to do in e-Science Central as the data import and export have no side-effects – they do not transform the data in any way.

Once we have the Java object representation of the e-Science Central workflow from T_l we can use it in the same way we would any other workflow. It can be saved, executed directly, or offered to a user to sanity check and run. At this point it is indistinguishable from any other workflow within e-Science Central.

3.5 Substitute(Wi,DataVersionSubstit, ServiceSubstit)

With the assumption that we are able to build the workflow from a provenance trace, substituting versions of services and data are simply a case of changing the version numbers in the representation of the workflow after it has been constructed from the provenance trace. This can either be done automatically when updating the versions of services and data to the most recent version, or by presenting the workflow to the users when they wish to change to specific versions of services or data.

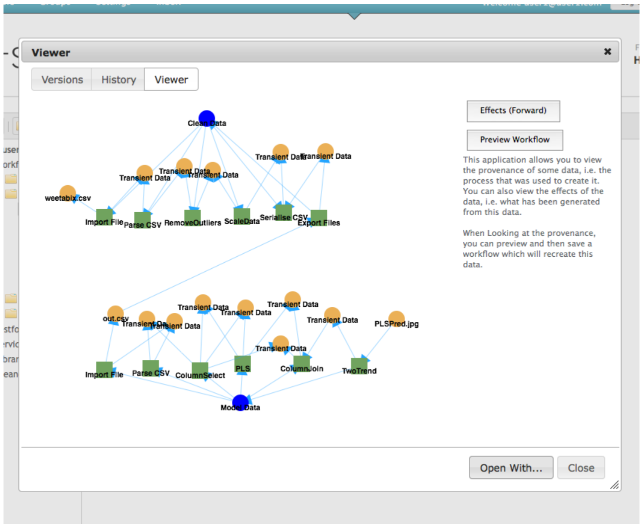
When a service, $S_{j,v}$ is created it can define a reliance on a library, $L_{i,v}$ which can be specified to always be the most recent version, or can be a specific version. In the case where $S_{j,v}$ uses the most recent version of the Library, when performing the substitution, the $L_{i,v}$ substituted will be the one that was most recent one when $S_{j,v}$ was created.

3.6 Execute(Wj)

Executing Wj is straightforward as we have an executable workflow. The provenance viewer application, described in Section 4, is used to create the workflow. The user is then offered the chance to save and check it prior to execution. When many workflows are created in order to view the effects of changing a service version the provenance viewer application can automatically execute the workflows and only alert the user to results which have changed.

4. USER INTERFACE

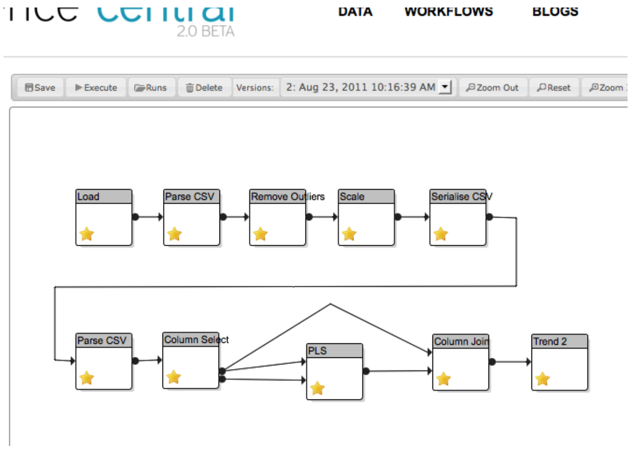
In order to present the user with an interface to be able to interact with the e-Science Central Provenance Service, we have developed an application which utilises the e-Science Central API. The application, shown within an iFrame in e-Science Central, allows users to view the *backwards* and *forwards* provenance of an data item. Shown in Figure 7, the user is presented with a visualisation of the provenance graph, built using the Javascript Infovis Toolkit [25]. The example shown here is an extension of the theoretical workflows shown in Figure 6, where two workflows are involved. The output of the first (shown in the upper line of Figures 7 and 8 correspond to the first workflow, whose output is used in the second workflow (bottom row). The user can choose to create an executable workflow from the graph prior to substituting service or data versions and then re-running the workflow. At this point, the virtual workflow generated by the Provenance Viewer is indistinguishable from any other workflow in e-Science Central, and subject to the same security constraints.



This application allows you to view the provenance of some data, i.e. the process that was used to create it. You can also view the effects of the data, i.e. what has been generated from this data.

When Looking at the provenance, you can preview and then save a workflow which will recreate this data.

Figure 7: Viewing the Provenance of a Data Item.



Versions: 2: Aug 23, 2011 10:16:39 AM

Figure 8: Re-running the Computation

5. RELATED WORK

De Roure et al. [18] discuss the likelihood of *workflow decay*, that is the probability that a workflow will execute as expected at some time in the future. They divide their analysis into whether the workflow and/or data and services have been updated and provide reasons why each of the four cases where something has been updated is relevant. They suggest caching of intermediate results as a solution for partial reproducibility where the data has been updated since the workflow run. Similar techniques are used in the Pegasus/Wings workflow system which analyses whether data should be stored and re-used or regenerated based on it's provenance [26, 27]. We suggest an optimisation to our system in Section 6 which is influenced by this research.

Taverna [10] and Kepler [11] are extremely popular workflow engines used in the scientific domain and capable of capturing provenance data. In many cases the facilities they offer are far greater than e-Science Central in terms of querying and exporting the provenance graph [28] and identifying collections used in computations [9]. However, e-Science Central is capable of dealing with service versions in a more graceful manner and, as this paper has shown, creating executable workflows from provenance traces.

Neo4j has previously been used by Wendell to capture provenance data about the software development process [29] and by Tylissanakis the to store the provenance of scientific workflows [30]. The latter is similar to our work although it focuses on coordinating Web Services and doesn't consider the issue versioning. However, they do mention briefly that the are able "reproduce simulation results" but do not go into detail.

6. FUTURE WORK

At present the virtual workflows described in Section 3.4 will construct a workflow which contains all service invocations back to data which was uploaded rather than produced from a workflow. However, it may not be necessary to execute this whole workflow if an intermediate data item is available and no services prior to it need to be changed. This could be considered as a checkpoint of the state and the re-run could start from here.

Extracting such information is possible but we have not included it in our initial application as we see it as an optimisation. However, we intend to implement it in the near future.

In addition, in Section 2.2 we described how Transient Data is generated during a workflow run and discarded upon completion. In some cases it may be desirable to automatically keep this Transient Data. For example when a service which is an ancestor of the Transient Data is either expensive to run (in terms of time or monetary cost) or is nondeterministic in nature. We plan to analyse the workflow prior to execution with respect to whether the services it contains are marked as *deterministic* or *idempotent* in order to determine whether Transient Data should be automatically persisted. This checkpoint could then be chosen as the start of a re-execution as described earlier.

There are many uses of the provenance information which we are capturing in order to predict future execution time based on historical events. Initially we are pursuing the idea of predicting required cloud computing resources based on factors such as queue length, services contained in each workflow, the input data size and execution time for each service. Clearly the algorithm is expensive to compute and so we will be relying on heuristics in order to provide an estimation of resources required.

7. CONCLUSIONS

This paper has described how the provenance storage system in e-Science Central can be used to answer questions useful to scientific research. Namely what is the provenance of a data item and what effects does changing the versions of ancestral artifacts have? We are able to answer these questions as we are leveraging the storage and versioning capabilities of e-Science Central to enable the system to execute previous versions of services on previous versions of data. The provenance storage system in e-Science Central is subject to the same security restrictions as the remainder of the system, whereby users with different roles are able to see different levels of information. Our model, which extends the Open Provenance Model, has been implemented in a Graph Database, Neo4j. We have shown how the questions we posed can be answered by traversing the graph structure and crating executable workflows.

8. ACKNOWLEDGEMENTS

This work has been part funded by the EU FP7 Grant, VENUS-C (RI-261565) and SiDE, the RCUK Digital Economy Research Hub on Social Inclusion through the Digital Economy, EP/G066019/1.

9. REFERENCES

[1] Simon Miles, Sylvia C Wong, Weijian Fang, Paul T Groth, Klaus-Peter Zauner, and Luc Moreau. Provenance-based validation of e-science experiments. *J. Web Sem.*, 5:28–38, 2007.

[2] S. Davidson and J. Freire. Provenance and scientific workflows: challenges and opportunities. In *Procs. SIGMOD Conference, Tutorial*, pages 1345–1350, 2008.

[3] Geetika T Lakshmanan, Francisco Curbera, Juliana Freire, and Amit Sheth. Guest Editors' Introduction: Provenance in Web Applications. *IEEE Internet Computing*, 15:17–21, 2011.

[4] Paul Groth, Simon Miles, Sanjay Modgil, Nir Oren, Michael Luck, and Yolanda Gil. Determining the Trustworthiness of New Electronic Contracts. In *Proceedings of the Tenth Annual Workshop on Engineering Societies in the Agents' World, (ESAW-09)*, Utrecht, The Netherlands, 2009.

[5] Bela Bauer, Jan Gukelberger, Brigitte Surer, and Matthias Troyer. Publishing Provenance-rich Scientific Papers. In *Procs. TAPP'11 (Theory and Practice of Provenance)*, Heraklyion, Crete, Greece, 2011.

[6] S Davidson, S Cohen-Boulakia, A Eyal, B Ludäscher, T McPhillips, S Bowers, M Kumar Anand, and J Freire. Provenance in Scientific Workflow Systems. In *Data Engineering Bulletin*, volume 30. December 2007.

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

[8] Khalid Belhajjame, Paolo Missier, and Carole Goble. Data Provenance in Scientific Workflows. In *Handbook of Research on Computational Grid Technologies for Life Sciences, Biomedicine, and Healthcare.* IGI Global, 2009.

[9] P. Missier, N. Paton, and K. Belhajjame. Fine-grained and efficient lineage querying of collection-based

workflow provenance. In *Procs. EDBT*, Lausanne, Switzerland, 2010.

[10] Duncan Hull, Katy Wolstencroft, Robert Stevens, et al. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34(suppl 2):W729–W732, 1 July 2006.

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

[12] Ilkay Altintas, Manish Kumar Anand, Daniel Crawl, Adam Belloum, Paolo Missier, Carole Goble, and Peter Sloot. Understanding Collaborative Studies Through Interoperable Workflow Provenance. In *Procs. IPAW 2010*, Troy, NY, 2010.

[13] Paolo Missier, Bertram Ludascher, Shawn Bowers, Manish Kumar Anand, Ilkay Altintas, Saumen Dey, Anandarup Sarkar, Biva Shrestha, and Carole Goble. Linking Multiple Workflow Provenance Traces for Interoperable Collaborative Science. In *Proc.s 5th Workshop on Workflows in Support of Large-Scale Science (WORKS)*, 2010.

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

[15] Manish Kumar Anand, Shawn Bowers, Timothy M McPhillips, and Bertram Ludäscher. Exploring Scientific Workflow Provenance Using Hybrid Queries over Nested Data and Lineage Graphs. In *SSDBM*, pages 237–254, 2009.

[16] Jill P. Mesirov. Accessible reproducible research. *Science*, 327(5964):415–416, 2010.

[17] Piotr Nowakowski, Eryk Ciepiela, Daniel Harężlak, Joanna Kocot, Marek Kasztelnik, Tomasz Bartyński, Jan Meizner, Grzegorz Dyk, and Maciej Malawski. The Collage Authoring Environment. *Procedia Computer Science*, 4(0):608–617, 2011.

[18] David De Roure, Khalid Belhajjame, Paolo Missier, Jose Manuel Gomez-Perez, Raul Palma, Jose Enrique Ruiz, Kristina Hettne, Marco Roos, Graham Klyne, and Carole Goble. Towards the preservation of scientific workflows. *iPress 2011*, 2011.

[19] David De Roure, Carole Goble, and Robert Stevens. The design and realisation of the myexperiment virtual research environment for social sharing of workflows. *Future Generation Computer Systems*, 25(5):561 – 567, 2009.

[20] Simon Woodman, Hugo Hiden, Paul Watson, and Jacek Cala. Developing applications using the e-science central api. In *Proceedings of the 2011 e-Science All Hands Meeting*, 2011.

[21] Paul Watson, Hugo Hiden, and Simon Woodman. e-Science Central for CARMEN: science as a service. *Concurrency and Computation: Practice and Experience*, 22:2369–2380, December 2010.

[22] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan, and Jan Van Den Bussche. The Open Provenance Model — Core Specification (v1.1). *Future Generation Computer Systems*, 7(21):743–756, 2011.

[23] http://www.neo4j.org. Neo4j, September 2011.

[24] D. Dominguez-Sal, P. Urbón-Bayes, A. Giménez-Vañó, S. Gómez-Villamor, N. Martínez-Bazán, and J. L. Larriba-Pey. Survey of graph database performance on the hpc scalable graph analysis benchmark. In *Proceedings of the 2010 international conference on Web-age information management*, WAIM'10, pages 37–48, Berlin, Heidelberg, 2010. Springer-Verlag.

[25] Javascript infovis toolkit. http://www.thejit.org, September 2011.

[26] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.*, 13:219–237, July 2005.

[27] Jihie Kim, Ewa Deelman, Yolanda Gil, Gaurang Mehta, and Varun Ratnakar. Provenance trails in the wings-pegasus system. *Concurr. Comput. : Pract. Exper.*, 20:587–597, April 2008.

[28] Paolo Missier, Satya Sahoo, Jun Zhao, Carole Goble, and Amit Sheth. Janus: From workflows to semantic provenance and linked open data. In Deborah McGuinness, James Michaelis, and Luc Moreau, editors, *Provenance and Annotation of Data and Processes*, volume 6378 of *Lecture Notes in Computer Science*, pages 129–141. Springer Berlin / Heidelberg, 2010.

[29] Heinrich Wendel, Markus Kunde, and Andreas Schreiber. Provenance of software development processes. In Deborah McGuinness, James Michaelis, and Luc Moreau, editors, *Provenance and Annotation of Data and Processes*, volume 6378 of *Lecture Notes in Computer Science*, pages 59–63. Springer Berlin / Heidelberg, 2010.

[30] Giorgos Tylissanakis and Yiannis Cotronis. Data provenance and reproducibility in grid based scientific workflows. *Grid and Pervasive Computing Conference, Workshops at the*, 0:42–49, 2009.