PReServ: Provenance Recording for Services

Paul Groth Simon Miles Luc Moreau

School of Electronics and Computer Science
University of Southampton
{pg03r, sm, l.moreau}@ecs.soton.ac.uk

Abstract

The importance of understanding the process by which a result was generated in an experiment is fundamental to science. Without such information, other scientists cannot replicate, validate, or duplicate an experiment. We define provenance as the process that led to a result. With large scale in-silico experiments, it becomes increasingly difficult for scientists to record process documentation that can be used to retrieve the provenance of a result. Provenance Recording for Services (PReServ) is a software package that allows developers to integrate process documentation recording into their applications. PReServ has been used by several applications and its performance has been benchmarked.

1 Introduction

The importance of understanding the process by which a result was generated in an experiment is fundamental to science. Without such information, other scientists cannot replicate, validate, or duplicate an experiment. For generations, scientists have used lab notebooks to record documentation of the process by which an experimental result was produced. Using this process documentation, scientists are able to retrieve the *provenance* of an experimental result, where we use provenance to mean the process that led to a result. However, with large scale in-silico experiments, it may be difficult for a scientist to record adequate process documentation especially when an experiment is conducted using thousands of computers owned and operated by different organisations.

In order to support the recording and querying of process documentation for Grid based applications, such as myGrid or CombeChem, we have argued for standard mechanisms to record and query process documentation as well as work with provenance [7]. This work also presents the P-assertion Recording Protocol (PReP) as a initial method to record process documentation in a generic manner for Service Oriented Architectures (SOA), the *de facto* architecture for designing Grid systems. Fundamentally, an SOA consists of a set of services that communicate via sending input and receiving output messages. By capturing assertions (termed p-assertions) about the content of these messages (message exchange p-assertions) along with causal relationships between messages and assertions about the internal states of services (actor state p-assertions), the documentation of process that led to a result can be recorded. We use the term actor to denote either a client or service in a SOA. PReP specifies how p-assertions can be recorded in a separate entity, the Provenance Store. This paper presents an implementation of PReP, which takes the form of an open source Java-based Web Services implementation of the protocol, and is named Provenance Recording for Services (PReServ).

The PReServ software package contains a Provenance Store Web Service, a set of interfaces for recording and querying, a set of Java libraries for easily accessing those interfaces, and an Axis[1] handler for "automatically" recording message exchange p-assertions for Axis based Web Services. PReServ is not just a proof-of-concept; it has been used to make a number of applications provenance-aware. A provenance-aware application is one that records p-assertions and makes use of those assertions to reason about the provenance of results. PReServ's performance has also been evaluated in the context of one of these applications, namely a bioinformatics experiment.

The rest of this paper is organised as follows, we first describe a Provenance Store implementation. We then describe the Provenance Store's interface and how to record and query p-assertions using the three mechanisms provided by PReServ. A per-

[1] Axis is a set of commonly used libraries from the Apache organisation for implementing Web Services in Java. It is available at http://ws.apache.org/axis/

formance evaluation of PReServ is then presented, which is followed by a discussion of related work and a conclusion.

2 The Provenance Store

PReServ's Provenance Store Web Service[2] is a repository for storing p-assertions. P-assertions are assertions made by actors about their execution. This includes assertions about the content of messages actors send and receive as well as assertions about their state. The Provenance Store handles request both to store and query p-assertions. We now discuss the implementation of the Provenance Store in the context of a concrete architecture. Each architectural component is presented along with its implementation details. Before discussing the individual components, we briefly describe some implementation design choices.

Heterogeneity of platforms is common in both Grids and Web Services. This heterogeneity motivated our choice of Java as an implementation language. By using Java, PReServ runs under Windows, Solaris, Linux and Macintosh OS X platforms, unmodified. Another global design choice was to implement the Provenance Store as a Java servlet hosted using the Apache Tomcat servlet container. This choice is in contrast to a number other Java based Web Services, such as the Grimoires registry (www.grimoires.org), that use the Axis container. Axis is itself a Java servlet but offers a set of features in order to automatically bind XML data structures into Java Objects. However, Provenance Stores need to store SOAP messages directly. By implementing the Provenance Store as a servlet, we can directly retrieve SOAP messages without the extra overhead of Axis.

2.1 Provenance Store Structure

The Provenance Store presents a structure to the outside world dictated by PReP, termed the p-structure. This structure allows for identification of p-assertions in provenance stores, which facilitates both recording and querying of p-assertions. Each actor is allowed to record p-assertions pertaining to their execution in a Provenance Store of their choice. However, these p-assertions must be identified so that they are correctly positioned in the p-structure. The p-structure is organised hierarchy as follows. It contains a list of message exchange records, which reflects an application message between two actors in a SOA. Each message exchange record is uniquely identified by a group of identifiers. Contained in a message exchange record are two views one for the client and one for the service in an message exchange. These views in turn contain the message exchange p-assertions as well as actor state p-assertions pertaining to that application message exchange. In addition to these p-assertions, the p-structure can contain relationships between data stored in the provenance store. This allows causal dependencies to be expressed. Likewise, if p-assertions have been recorded across provenance stores, the p-structure can contain links that can be traversed to go from one provenance store to the next in order to retrieve related p-assertions. We note that the p-structure is reflected in the interface to the store and does not mandate any mechanism for the storage of data.

We now discuss the concrete architecture components shown in figure 1.

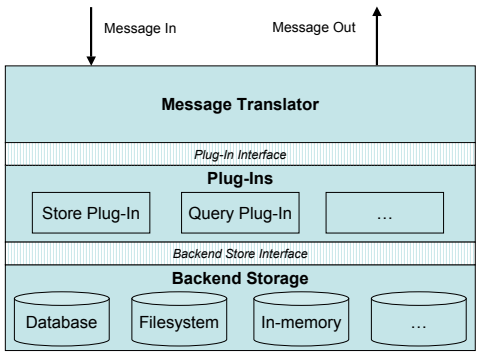

Fig. 1: The Provenance Store concrete architecture

2.2 The Message Translator

In order to isolate the Provenance Store's storage and query logic from the message layer, we introduce the Message Translator. The isolation of message processing from the Store's business logic allows the Store to be easily modified to support different underlying message layers. For example, the Store currently supports SOAP, however, by developing a new Message Translator, a Java RMI message layer could be supported. The Message Translator component is responsible for three operations.

1. Translating incoming messages into the internal format of the Provenance Store.

2. Determining the appropriate plug-in to handle incoming messages.

[2] Provenance Store used as a proper noun denotes the Provenance Store Web Service included in PReServ

3. Translating plug-in responses into messages and sending the message to the appropriate actor.

Our implementation of the Message Translator processes incoming SOAP messages by striping off the HTTP and SOAP message handlers from the message. It then translates the body of the SOAP message into a W3C DOM `Document` (http://www.w3.org/DOM/). The internal representation of the store is XML represented by a W3C DOM. We expect that all incoming messages will be either represented in XML or translated to it by the the Message Translator. The choice of XML is motivated by its use in the Web Services environment as a *lingua franca* and is buttressed by the development of specifications such as XML Binary (http://www.w3.org/XML/Binary/) for representing binary documents in XML. After translating the SOAP message body into a W3C DOM, the Translator then passes the document to a Plug-In. The Translator determines the Plug-In by looking at the HTTP context of the incoming message i.e. the URL that the message was passed to. For example, if the message was sent to http://www.provenance-store.com/record the context of the message would be /record. The Translator takes this context and passes it to a PlugInHandler which passes back a reference to the Plug-In that maps to the given context. The final operation the Message Translator performs is the serialisation of the Plug-In's returned DOM `Document` into a SOAP message. It then sends the message back to the requester.

2.3 Plug-Ins

The second component of the concrete architecture is a set of Plug-Ins. Each Plug-In implements a specific piece of functionality that the Provenance Store provides. Such functionality could include storing submitted p-assertions correctly, searching through already stored p-assertions via an XPath (http://www.w3.org/TR/xpath) query, or responding to requests to move provenance documentation to another store. Plug-Ins allow for new functionality to be easily added without having to modify already existing functionality. This design decision is particularly important given that PreServ is open source. It allows third party developers to easily add new functionality to the Store without having to delve into its inner workings in detail. Every plug-in must implement the Java interface shown below.

```
import org.pasoa.prep.service.BackendStore;
import org.w3c.dom.Document;

public interface PlugIn
{
    public Document process (Document soapBody,
                             BackendStore store);
}
```

The interface means that every Plug-In takes an XML document represented as a W3C DOM `Document` and a `BackendStore` as parameters and returns a `Document` as a result. The `BackendStore` parameter contains a reference to the data repository that the Plug-In can act on. As mentioned before, each Plug-In is bound to a particular HTTP context. Therefore, Plug-Ins are responsible for handling the functionality that the Provenance Store's interface expresses for that particular context. This means for our Web Services implementation of a provenance store, each Plug-In has its own WSDL operations and schema describing the offered functionality. Likewise, if a Plug-In is unable to handle an incoming request then it is responsible for generating an appropriate `Document` containing an error message. The Provenance Store, currently, implements two Plug-Ins.

1. The Store Plug-In is responsible for handling p-assertion storage requests.
2. The Basic Query Plug-In is responsible for handling basic queries on already stored p-assertions. This Plug-In allows stored p-assertions in the form of the p-structure to be traversed.

The interfaces to these two Plug-In make up the Provenance Store Interface, which will be discussed in detail later.

2.4 Backend Storage

Backend storage is where p-assertions are finally stored in the Provenance Store. Therefore, the third component of the concrete architecture is a set of backend stores implementing the same interface. The requirement that each backend store implements the same interface allows Plug-Ins to be implemented without regard to the underlying storage system used by the Provenance Store. For example, this allows a Plug-In to work correctly with a Provenance Store that uses the file system for storage as well as one that uses a database. However, a common interface does have one drawback; it hides any additional capabilities that a backend might provide. A database, for example, might allow SQL queries to be performed directly on it. If the common interface does not provide a method for SQL queries to be issued then Plug-Ins would not have access to this functionality. Given this drawback, it is critical for backend stores to be optimised for the common interface.

In our implementation, the common Java interface that each backend store must implement is

BackendStore. This Application Programming Interface (API) provides a set of getter and setter methods that map to the p-structure. It also includes several basic methods for traversing that structure. PReServ comes with file system, in-memory and database backends each of which implements the BackendStore interface. The file system backend stores p-assertions in a hierarchy of directories directly on the file system. The hierarchy maps directly to the p-structure. The file system backend is particularly useful in debugging as it allows the contents of the Provenance Store to be inspected by the user. The in-memory backend stores system uses hashtables and vectors to model the p-structure. The database backend is critical because a large amount of p-assertions can be stored using it. We use the Berkeley DB Java Edition database. Using this pure Java transactional object database maintains the portability of PReServ. The backend can be changed by modifying one line of code in the main servlet class of the Provenance Store. In the future, we would like to implement other backends. For example, a backend that uses the OGSA-DAI Grid data access and integration layer (http://www.ogsadai.org.uk/) would be of interest.

Here, we have shown a concrete architecture for a provenance store and explained an implementation of it as a Web Service. We now describe the interface of PReServ's Provenance Store.

3 The Provenance Store Interface

The Provenance Store Interface describes the functionality that the Store provides and how that functionality can be accessed. The interface is defined in WSDL (Web Service Description Language, http://www.w3.org/TR/wsdl) and defines two ports a RecordPort and a QueryPort. Both these ports take an input message and produce an output message defined in an external XML schema. Defining SOAP message contents in an external schema is known as the document/literal style of Web Services. One of the helpful attributes of this style is that it reduces the complexity of the WSDL file, which is already quite complex for even just two operations. There are two schemas used by the Provenance Store Interface, one reflects the recording interface and the other the query interface. We discuss the recording interface in detail here. The query interface is still being refined. Currently, it provides a mechanism for navigating a Provenance Store's p-structure. We are currently investigating an XQuery based interface for retrieving and searching p-assertions.

The recording interface maps directly to the messages defined by PReP. Below we show the schema (**Record Schema**) for both the Record and RecordAck messages defined in the Provenance Store's WSDL. The record element is referenced by the Record message in the WSDL and its structure is defined by the pr:Record type. The type specifies that the element will contain one or more elements of type IdentifiedContent, which contains an element identifiers and an element content. The type Identifiers contains the set of ids that the situate the content (a p-assertion) in the p-structure. The content element's type defines a choice between one of the messages of PReP. It is important to note that the elements in type Content and Identifiers are defined using the p-structure schema, which is an XML Schema file that models the p-structure. The types that come directly from the p-structure are prefixed by ps. By representing the p-structure as XML Schema, we were able to use it directly in the WSDL interface. Another important property of the interface is that it allows more than one protocol message to be bundled in a record message. For example, the Record type allows more than one element of identifiedContent to be inside the record element, which means that a client could record various messages from different interactions encapsulated in one record SOAP message. The IdentifiedContent type also supports this property, by allowing for multiple content elements. Therefore, a client could record all of an interaction's messages, identified by an identifiers element, at once.

The recordAck element is referenced by the RecordAck message and its structure is defined by the RecordAck type, which contains an element identifiers and an element contentName that contains the name of the message acknowledgement. The names are restricted to the strings listed in the schema. There can be more than one acknowledgement in the RecordAck message because the recordAck element contains an unbounded sequence of ack elements. The PRecord schema defines the structure of the messages Provenance Store clients can use to record p-assertions.

Record Schema

```
<xs:schema xmlns:pr=... xmlns:ps=...
  xmlns:xs=... targetNamespace=pr>

  <xs:element name="record" type="pr:Record"/>

  <xs:element name="recordAck"
              type="pr:RecordAck"/>

  <xs:complexType name="RecordAck">
    <xs:sequence>
      <xs:element name="ack"
        minOccurs="1" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="identifiers"
```

```
                    type="pr:Identifiers"/>
            <xs:element name="contentName">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration
                    value="messageExchagnePA"/>
                  <xs:enumeration
                    value="actorStatePA"/>
                  <xs:enumeration
                    value="relationship"/>
                  <xs:enumeration
                    value="submissionFinished"/>
                  <xs:enumeration value="ERROR"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
           <xs:element name="moreAckInfo"
             minOccurs="0" maxOccurs="1"
             type="xs:anyType"/>
         </xs:sequence>
       </xs:complexType>
     </xs:element>
     <xs:element name="ERROR" minOccurs="0"
        maxOccurs="1" type="xs:anyType"/>
 </xs:complexType>

 <xs:complexType name="Record">
   <xs:sequence>
     <xs:element name="identifiedContent"
              type="pr:IdentifiedContent"
              maxOccurs="unbounded"/>
   </xs:sequence>
 </xs:complexType>

 <xs:complexType name="IdentifiedContent">
   <xs:sequence>
     <xs:element name="identifiers"
              type="pr:Identifiers"/>
     <xs:element name="content"
              type="pr:Content"
              maxOccurs="unbounded"/>
   </xs:sequence>
 </xs:complexType>

 <xs:element name="clientId"
              type="ps:Identity"/>
 <xs:element name="serviceId"
              type="ps:Identity"/>

 <xs:complexType name="Identifiers">
   <xs:sequence>
     <xs:element name="messageExchangeId"
                       type="xs:anyURI"/>
     <xs:element ref="pr:clientId"/>
     <xs:element ref="pr:serviceId"/>
     <xs:element name="asserter">
       <xs:complexType><xs:choice>
         <xs:element ref="pr:clientId"/>
         <xs:element ref="pr:serviceId"/>
       </xs:choice></xs:complexType>
     </xs:element>
   </xs:sequence>
 </xs:complexType>

 <xs:complexType name="Content">
   <xs:choice>
     <xs:element
       name="messageExchangePAssertion"
       type="ps:MessageExchangePAssertion"/>
     <xs:element
       name="actorStatePAssertion"
       type="ps:ActorStatePAssertion"/>
     <xs:element
       name="relationship"
       type="ps:Relationship"/>
     <xs:element
       name="submissionFinished"
       type="xs:int"/>
     </xs:choice>
   </xs:complexType>
</xs:schema>
```

4 Using the Provenance Store

This section describes how a Provenance Store client can make use of the Store through the WSDL interface described above. There are three options an application developer has for accessing the Store through this interface: direct Web Service calls, the Java Client Side Library, or the Axis Handler. We now describe each of these options.

4.1 Direct Web Service Calls

The first option is to call the Provenance Store Web Service directly. This entails building a SOAP message that is compatible with the WSDL interface for every query or record operation, sending the message to the Store and processing the resulting acknowledgement message. This option shows the promise of Web Services. It allows any implementation language or platform that can understand XML to submit and query provenance information. For example, even though the Provenance Store is implemented in Java, a Perl script, or .Net program can still record information. This approach provides the maximum flexibility to a developer, however, it also requires the maximum amount of work. The developers must implement all SOAP message construction and processing themselves. Therefore, PReServ offers another option for those developers using Java.

4.2 The Java Client Side Library

The second option to call the Store is to make use of PReServ's Java Client Side Library. If a developer is using Java, the Library provides Java methods that correspond to the Provenance Store Interface. The Library produces the correct SOAP message for a given method and sends it to the specified Provenance Store. However, this option still requires the developer to implement the business logic for when to record p-assertions and what assertions to record.

4.3 The Axis Handler

The third option available to developers for recording p-assertions is the Axis Handler, which can only be used if applications are implemented using the Axis Web Services libraries. By adding the Axis Handler Jar file to the application classpath, all SOAP messages produced or received by a Web Service client or service will be recorded in a Provenance Store specified in a configuration file. Essentially, the Axis Handler wraps the specified client and service and intercepts all Web Service communication. The Axis Handler is best used for already

existing Axis based Web Service applications. It allows p-assertion recording to be added without modifying the existing application. There are, however, some drawbacks to this approach. First, it requires both the use of Java and Axis. Secondly, it does not directly support actor state p-assertions and recording relationships, which are key to documenting process. In the future, we would like to add more functionality to support these PReP messages.

This section described three mechanisms by which developers can access Provenance Stores from their applications. Each option provides varying degrees of ease of use and flexibility. By providing three different mechanisms, PReServ caters to the needs of a broad range of applications.

5 Performance Evaluation

We now evaluate the performance of the Provenance Store in the context a bioinformatics application, the Protein Compressibility Experiment. We analyse the performance of both the recording and querying aspects of the Store. The application is discussed in detail in another paper [8]. The performance evaluations were conducted on two Pentium P4, 2.8 Ghz, 1.5 GB RAM PCs. The Provenance Store Web Service was run under Windows XP on one machine, while the experiment itself was located on the other PC running Redhat Linux 9.1, running under a VMWare virtual machine. Like a number of authors, virtual machines were adopted for virtualising our Grid deployment [9, 4]. While some application slowdown was observed by running over VMWare, we note that provenance recording itself also suffers a similar slowdown. Hence, we conjecture that our results remain valid if similar benchmarks are run natively on a physical machine. The PCs were connected by a 100Mb local ethernet.

Before we evaluate the Provenance Store in an application context we examine a local benchmark. While running on a Windows XP PC with a Pentium P4, 2.8 Ghz, 1.5 GB RAM. It takes approximately 18 ms round trip to record one pre-generated message in the Provenance Store. These tests were conducted with both the client and server running on the same host. This benchmark gives us a basic understanding of the speed of the Store. We now evaluate the performance of PReServ's Provenance Store in the context of the application implementation.

5.1 Provenance Recording Evaluation

The purpose of this evaluation is to benchmark the overhead of p-assertion recording in a real scientific application. We note that the actors in this experiment were not Web Services but command line programs wrapped by a script that submitted p-assertions to the Provenance Store. In order to provide provenance for the scientific experiment, both message exchange and actor state p-assertions were recorded

Figure 2 plots the overall execution time (measured by the time difference between the last and first activities in the protein compressibility workflow), for an increasing size of data input (expressed as permutations of protein sequences), and for different configurations of p-assertion recording:

1. without p-assertion recording,
2. with asynchronous recording, in which p-assertions are accumulated locally in a file before being shipped to the Provenance Store after execution,
3. with synchronous recording by direct Web Service invocation of the Provenance Store, and
4. with synchronous recording with extra information being recorded as actor state p-assertions (such as the scripts that a service was running).

Our observations are as follows:

1. overall, the different execution times remain linear (each plot has a correlation coefficient greater than 0.99) with the number of permutations to be processed;
2. accumulating p-assertions to be submitted asynchronously has an overhead over no recording;
3. asynchronous recording has an overhead smaller than synchronous recording;
4. overall, the overhead of asynchronous performance recording remains less than 10%.

Like a scheduler requires a granularity coarse enough to offset the overhead of automatic scheduling, automatic p-assertion recording has an acceptable cost if the granularity of workflow activities is coarse enough. In this experiment, the run of a workflow for one 100Kb sample with 1 permutation takes approximately 4.5s and; each permutation involves the creation of 6 provenance records and their submission (averaged over a long running workflows).

5.2 Query Performance Evaluation

We now evaluate the Provenance Store's query performance. One use case for the query of provenance in this application was categorising scripts.

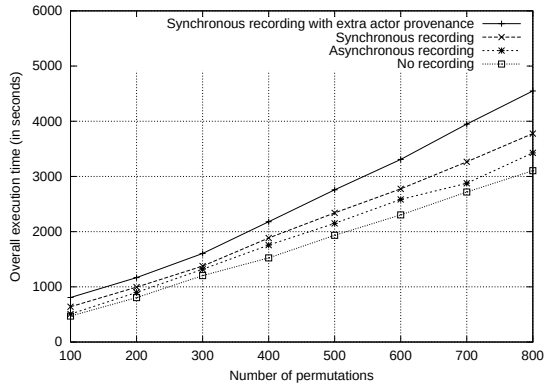

Fig. 2: Recording Performance

The bioinformatician wanted to know which scripts were used by which service in a given workflow run. The categorisation is performed by querying each activity in the Provenance Store for actor state p-assertions containing the script and creating a mapping from each set of exactly equivalent scripts to the workflow run in which that script is used for a given service.

As we analyse all message exchange records in the Provenance Store, the time taken to perform the categorisation is dependent on the size of the store. In Figure 3, we plot the time to query the store for all relevant actor state p-assertions and perform a full comparison against the number of records contained in the store. We observe a linear behaviour (the plot has a correlation coefficient greater than 0.99) with the size of the store; on average, it takes about 15ms to retrieve a script (through one store invocation) and categorise it.

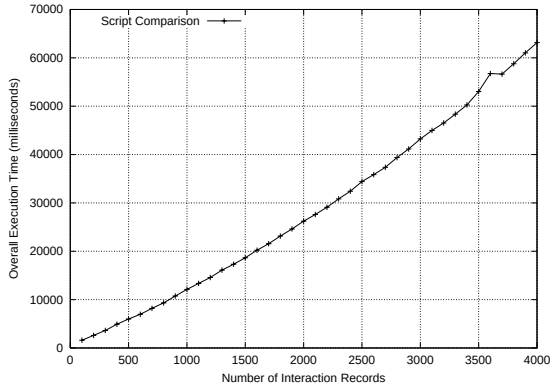

Fig. 3: Query Performance

5.3 Performance Analysis

The performance of the Provenance Store degrades linearly with the size of the data being stored or with the number of queries needing to be processed. For queries, it takes on average 15ms to retrieve a 100Kb data item from the store no matter the Store's size. The ability for the Provenance Store to record asynchronously is an advantage in that it allows actors to wait until an experiment is finished or there is a lull in processing to submit p-assertions. However, the Provenance Store could become a bottleneck if it had to handle a large number of connections simultaneously. Therefore, the support for multiple linked Stores in the PReP protocol is necessary. We will continue to investigate the performance of the Provenance Store in larger versions of this application but these first results show that p-assertion recording has a reasonable overhead given the use cases that provenance supports.

6 Related Work

Along with the protein compressibility experiment, two other applications have made use of PReServ. One application used PReServ to develop a provenance-aware fault tolerance system [14]. The other application used it in a prototype scenario [2].

There have been other systems that have been developed for capturing the provenance of a result. There have been several systems that are domain specific including work in Geographical Information Systems [10], bioinformatics[6] and sensor networks [11]. There has also been indepth study of provenance in database systems [1, 3]. Other work has been done in e-notebooks [12], metadata catalogs [5] and workflow centric systems [13]. Our work is different from all these in that it provides a technology and domain independent mechanism for recording process documentation that can be applied to any system modelled by a SOA.

7 Conclusion

PReServ provides a cross-platform means for recording and querying p-assertions to determine the provenance of a result. It is cross-platform because it is written in pure Java and has been tested on Mac OS X, Windows, and Linux. The Provenance Store has a flexible architecture allowing functionality to be added or changed. The architecture supports multiple backend storage systems including in-memory, file system, and database backends. Even with a change in the backend of a Provenance Store, queries are isolated from any changes through the use of the p-structure. Another benefit of PReServ is the three different options a developer has to make use of the Provenance Store. A developer can quickly add p-assertion recording to their Axis

based applications by using the Axis Handler. They can have fuller control using the Java Client Side Libraries and with the direct Web Service interface any implementation can use the Store, even command line programs. We demonstrated PReServ's application independent nature in its use in three different applications ranging from fault-tolerance [14], to bioinformatics [8], to baking[2]. Lastly, we showed that p-assertions could be recorded and queried in an acceptable amount of time for a real application.

In terms of future work, we plan to investigate in more detail the performance and issues related to distributed Provenance Stores. We will add more support for asynchronous recording to the Jave Client Side Library. In addition to these improvements, we intend on adding more functionality to the Axis Handler and implement an XQuery interface to the Provenance Store.

Acknowledgements
This research is funded in part by EPSRC PASOA project GR/S67623/01.

References

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

[2] L. Chen, V. Tan, F. Xu, A. Biller, P. Groth, S. Miles, J. Ibbotson, and L. Moreau. A proof of concept: Provenance in a service oriented architecture. In *Proceedings of the UK OST e-Science Second All Hands Meeting 2005 (AHM05)*, 2005.

[3] Y. Cui. *Lineage Tracing in Data Warehouses*. PhD thesis, Stanford University, December 2001.

[4] R. Figueiredo, P. Dinda, and J. Fortes. A case for grid computing on virtual machines. In *In Proceedings of the 23rd Internatinal Conference on Distributed Computing Systems (ICDCS 2003)*, 2003.

[5] I. Foster, J. Voeckler, M. Wilde, and Y.Zhao. Chimera: A virtual data system for representing, querying and automating data derivation. In *Proc. of the 14th Conf. on Scientific and Statistical Database Management*, July 2002.

[6] M. Greenwood, C. Goble, R. Stevens, J. Zhao, M. Addis, D. Marvin, L. Moreau, and T. Oinn. Provenance of e-science experiments - experience from bioinformatics. In S. J. Cox, editor, *Proc. UK e-Science All Hands Meeting 2003*, pages 223–226, September 2003.

[7] P. Groth, M. Luck, and L. Moreau. A protocol for recording provenance in service-oriented grids. In T. Higashino, editor, *Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS'04)*, volume Lecture Notes in Computer Science, pages 124–139, Grenoble, France, December 2004. Springer-Verlag.

[8] P. Groth, S. Miles, W. Fang, S. C. Wong, K.-P. Zauner, and L. 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)*, 2005.

[9] K. Keahey, K. Doering, and I. Foster. From sandbox to playground: Dynamic virtual environments in the grid. In *In Proceedings of the 5th International Workshop in Grid Computing (Grid 2004)*, Pittsburgh, PA, November 2004.

[10] D. Lanter. Lineage in gis: The problem and a solution. Technical Report 90-6, National Center for Geographic Information and Analysis (NCGIA), UCSB, Santa Barbara, CA, 1991.

[11] J. Ledlie, C. Ng, D. A. Holland, K.-K. Muniswamy-Reddy, U. Braun, and M. Seltzer. Provenance-aware sensor data storage. In *NetDB 2005*, April 2005.

[12] P. Ruth, D. Xu, B. K. Bhargava, and F. Regnier. E-notebook middleware for acccountability and reputation based trust in distributed data sharing communities. In *Proc. 2nd Int. Conf. on Trust Management, Oxford, UK*, volume 2995 of *LNCS*. Springer, 2004.

[13] M. Szomszor and L. Moreau. Recording and reasoning over data provenance in web and grid services. In *Int. Conf. on Ontologies, Databases and Applications of Semantics*, volume 2888 of *LNCS*, 2003.

[14] P. Townend, P. Groth, and J. Xu. A provenance-aware weighted fault tolerance scheme for service-based applications. In *In Proc. of the 8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2005)*, May 2005.