A Proof of Concept: Provenance in a Service Oriented Architecture

Liming Chen+, Victor Tan+, Fenglian Xu+, Alexis Biller*,
Paul Groth+, Simon Miles+, John Ibbotson*, Michael Luck+ and Luc Moreau+
+School of Electronics and Computer Science
University of Southampton, Southampton SO17 1BJ, UK
*Emerging Technology Services, IBM
Hursley Park MP137, Winchester SO21 2JN
Email: lc@ecs.soton.ac.uk

Abstract

Provenance has been identified as an emerging and important concept within the Grid community for a variety of purposes, such as verifying or tracing results. We seek to provide a concrete conception of provenance and its possible utilisation through the process of designing and implementing a system prototype with some specific provenance requirements. This prototype, which is based on an idealised recipe for baking a cake, is developed within the context of a service oriented Grid computing environment and implemented using standard Web Services technologies. The issues surrounding the design of possible provenance system are also explored.

Keyword provenance, Grid community, service oriented grid computing, Web Services

1. Introduction

The general understanding of provenance is the source or history of derivation of a particular object. Provenance is an important requirement in many practical fields. For example, the American Food and Drug Administration requires that the record of a drug's discovery be kept as long as the drug is in use. In aerospace engineering, simulation records that lead up to the design of an aircraft are required to be kept up to 99 years after the design is completed. In museum and archive management a collection is required to have archival history regarding its acquisition, ownership and custody.

With the prevalence of distributed computing, in particular the availability of service-oriented Grid computing infrastructure, collaborative problem solving by exploiting and sharing resources in distributed environments has become a reality [1]. This has led to a growing demand for tracking, recording and managing data sources and derivation [2, 3]. While the utility of provenance has been explored recently in the arena of database systems [4, 5], and various Grid applications [6, 7] such as e-Diamond, myGrid, Combechem and DataMiningGrid have clearly identified provenance-related requirements within the scope of their functionality [8], there is no *commonly agreed conception* of provenance in the context of service-oriented grid computing, nor any concretely implemented prototypes.

In this paper, through a sample application we demonstrate our conception that *the provenance of a piece of data is the process that leads to the data*. Our primary contribution lies in the design and implementation of a system prototype which illustrates the various facets of provenance that are integral to the functionality of this sample application domain and, in the process identifying some of the issues underlying the design of a generic provenance system.

We adopt a simple cake baking scenario and conceptualise it as a process couched in a service-oriented architecture (SOA) framework. Within the context of this framework, provenance would enable users to trace and identify the individual services (or an aggregation of services) as well as their corresponding inputs and outputs that were involved in the production of a specific result data (a cake, in this instance). The provenance of the cake is stored in an appropriate location and can be retrieved for various purposes: this could include post hoc analysis of the cake baking process or replaying the entire process to produce new cakes. We focus on identifying, storing and querying provenance data in this scenario.

The paper is organised as follows. Section 2 outlines the application scenario. Section 3 analyses the application scenario from service-oriented perspective and presents our conception of provenance in a SOA. In Section 4 we describe details about provenance system design. We give an in-depth analysis and discussion in Section 5 and conclusions in Section 6.

2. Application scenario

The application scenario is based on the process of baking a cake, more specifically, Baking Victoria Sponge Cake (BVSC). We derive an

idealised recipe from an actual recipe [9]. This idealised recipe can be considered to be composed of four distinct steps or activities that correlate roughly with the original recipe; these are outlined below:

Step 1: Whisk together a certain amount of butter and sugar (in proportion) until light and creamy;

Step 2: Beat the required eggs for a certain duration and add it to the whisked sugar and butter mixture;

Step 3: Fold some flour into the beaten mixture and add the flavouring preferred (vanilla, lemon);

Step 4: Put the folded wet dough into an oven and bake it for a given time at a specified temperature;

Each activity requires a specific number of ingredients as inputs and produces an intermediate or final output.

Although baking cakes in accordance with a standard recipe appears to be a routine task for many people, it is quite common that cakes produced by different users or even a single user end up being different from each other. The differences in the quality of the cake in question can be reflected in a multitude of ways. Hence, some questions may be posed by the user (or other interested parties) to identify the contributing factor for a cake of relatively inferior quality. Some of these questions include:

- Was the correct sugar amount as specified in the recipe used?
- Was the correct oven temperature as specified in the recipe used?
- Was the correct amount of flour used for the oven baking activity at a given location?

The process of answering questions is in effect an inquiry pertaining to the provenance of a cake.

3. Provenance and provenance system

This section analyses the BVSC process, its entities and their interactions and information flow from a service-oriented perspective. We first cast the BVSC process in a service-oriented architecture in which entities provide services to each other through interactions. Then we use a sequence diagram to delineate the interaction and information flow of the BVSC process. From the above discussion we introduce the concept of provenance in the context of BVSC and the use of provenance stores. A sequence diagram is provided to demonstrate when and where provenance data are captured and recorded with the involvement of a provenance store.

3.1 A service-oriented view

A service-oriented view is a way of modelling large, complex systems in terms of services. Here a service can simply be viewed as an abstract characterisation and encapsulation of some content or processing capabilities. A corollary to the service-oriented view is the service-oriented architecture (SOA), in which services are the primitive building blocks. In a SOA, problem solving processes amount to the discovery, aggregation and execution of a set of loosely coupled services.

In service-oriented view, an activity in the BVSC scenario is in essence the behaviour of a service or an operation provided by a service. Thus each of the four activities in the idealised BVSC recipe can be viewed as a service. In addition, the user and the baker are also services by virtue of the activities they perform respectively in the BVSC process. In a service-oriented architecture (SOA), clients typically invoke services, which may themselves act as clients for other services; hence, we will use the term *actor* to denote either a client or a service in a SOA. Figure 1 depicts all the actors/services in the BVSC process.

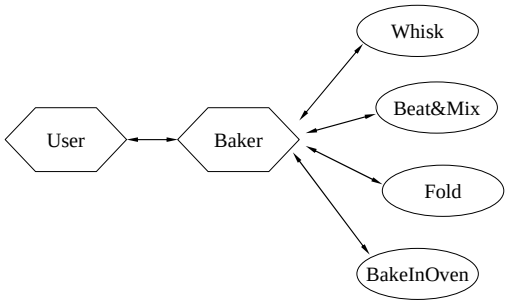

Figure 1 Services in the BVSC process

Most SOAs have a primary functional characteristic of executing workflows, a workflow being a process by which a series of services are executed in a specific sequence, including the specification of how outputs of services are routed to the services of other tasks. There is usually a component in the SOA known as the service enactment engine which undertakes the action of executing a workflow. In the context of the BVSC scenario, the BVSC process corresponds to a workflow run that produces a cake, with the baker assuming the role of a service enactment engine that executes the workflow specification (idealised recipe) provided by the user.

3.2 Information flow in a process

To expose the information flow of a process we use sequence diagram techniques to represent all interactions and their participants as a process unfolds. A sequence diagram depicts all services contained, all events taking place and all interactions between services in the process in a

temporal order. An RPC-like interaction between two services consists of two messages: an invocation message and a result message. The invocation message is defined by an operation name and parameters carried by the operation. The result message is defined by a name and the results returned by the service.

Figure 2 shows the sequence diagram of the BVSC process based on the service-oriented view. In this diagram, services are shown as rectangles and arranged horizontally from left to right in the order of invocation. Interactions are arranged vertically from top to bottom as the process proceeds from the start to the end. The sequence of invoking messages is represented as solid arrow lines and the return messages are represented as dashed arrow lines. The input parameters and the name of the messages are shown above the lines and separated by a colon.

From the diagram we can capture all information flows between services in a process. For example, the user interface initiates the BVSC process by invoking the Baker service with some control parameters (i.e. sugar, duration, flour and temperature). At the end of the process, the Oven service returns a cake to the Baker service that in turn returns the cake to the User Interface. Both invocation and result messages contain concrete input/output information. It is also clear that a process is actually composed of a number of interactions. In order to repeat or verify the result of a process we need to capture and record all the information flows and the concrete information in these interactions. This has led to our provenance conception as described below.

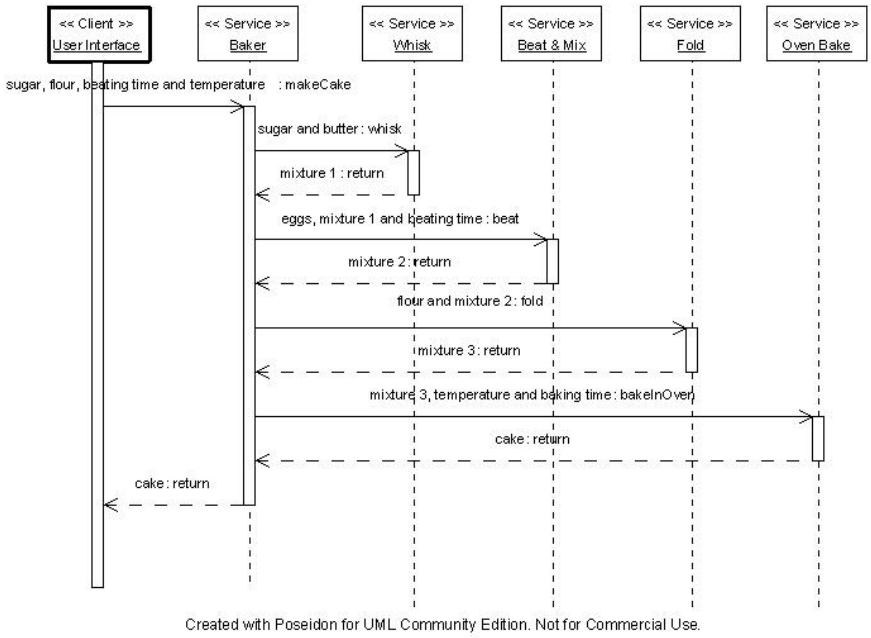

Figure 2: BVSC process sequence diagram

3.3 Provenance, provenance recording and provenance stores

We define that the provenance of a piece of data is the process that led to the data. We note that such a definition is concerned with provenance as a concept. Ultimately, our aim is to conceive a computer-based representation of provenance that allows us to perform useful automated analysis and reasoning to support our use cases. The provenance of a piece of data will be represented in a computer system by some suitable documentation of the process that led to the data. Furthermore, we distinguish a specific piece of information documenting some step of a process from the whole documentation of the process. The former shall be referred to as p-assertion, which is formally defined as *an assertion made by an actor pertaining to a process.*

Given that a SOA can be broken down into two types of actors: clients who invoke services and services that receive invocations and return results, we have identified two disjoint kinds of p-assertion: interaction p-assertions and actor state p-assertions. An interaction p-assertion is an assertion made by an actor about a message it has sent or it has received. We do not prescribe the nature of the assertion about a message; instead such decisions are left to the application. For instance, an interaction p-assertion could simply contain a copy of the message exchanged between two actors. Therefore, interaction p-

assertions can be obtained by recording the inputs and outputs of the various services involved in generating a result. Alternatively, if some data contained in the message is regarded as confidential by the actor or too large to be submitted, the assertion may consist of the message in which the concerned data have been replaced by an opaque proxy or a pointer.

An actor state p-assertion is the documentation provided by an actor about its internal state in the context of a specific interaction. Actor state documentation is extremely varied: it can include the function the actor performs, the workflow that is being executed, the amount of disk and CPU a service used in a computation, the floating point precision of the results it produced, or application-specific state descriptions. We note that in a distributed

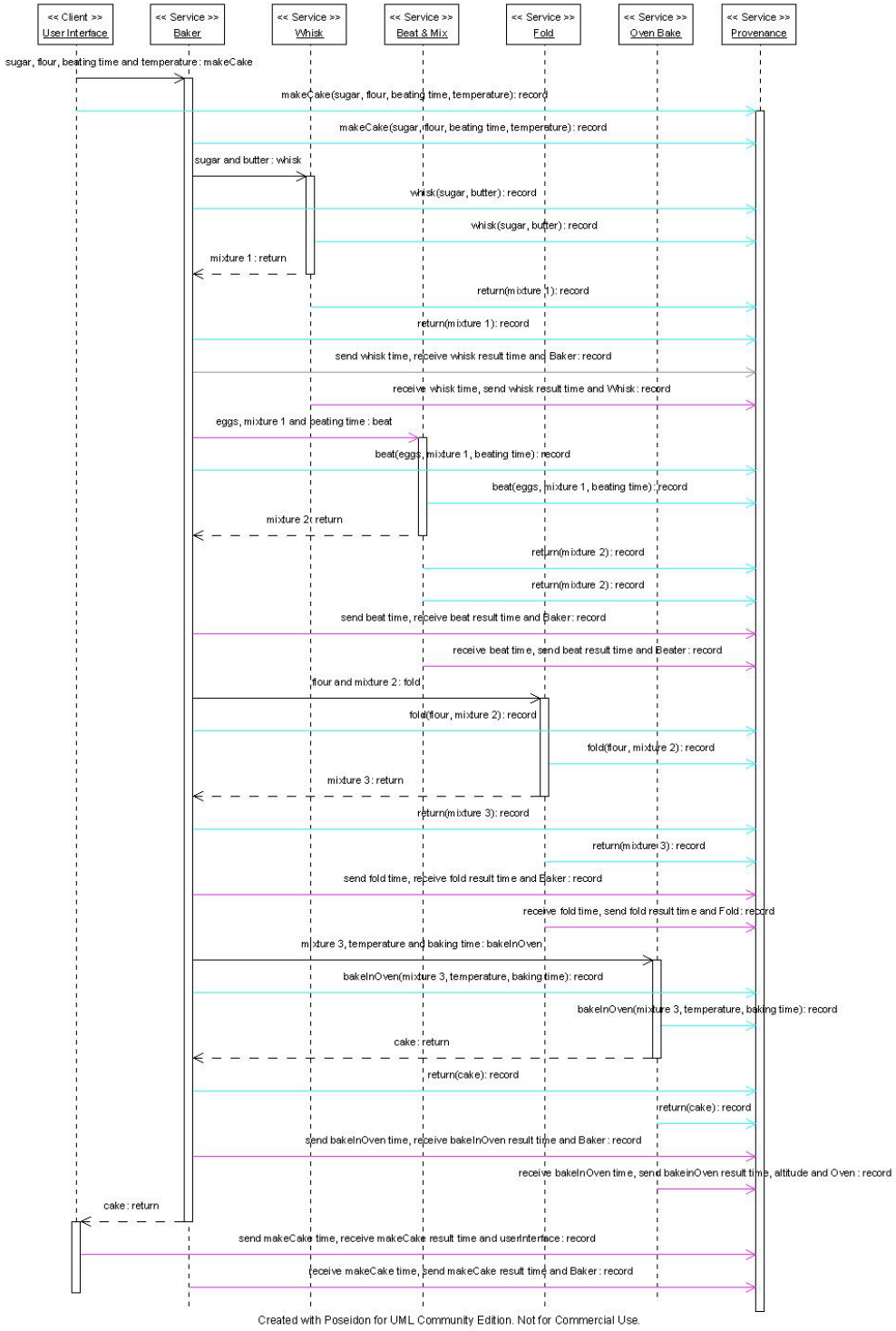

Figure 3: The BVSC process sequence diagram with provenance

system, an actor state is not externally observable, and therefore can only be captured by cooperative contribution of the actor itself.

In addition, an appropriate storage is needed for the recorded interaction and actor state p-assertions. This can assume the form of an additional service within the SOA whose primary activity is the archival of p-assertions generated from a process. We term this service the provenance store.

We handle interaction p-assertions in the following way. For each interaction between a client and service consisting of an invocation and a result, each party is required to submit their view of the interaction to a common provenance store. Even though the BVSC process considers multiple actors, the interaction between all these actors can be reduced down to a common triangular pattern of interaction, i.e. the client, service and provenance store.

The BVSC process sequence diagram in Figure 3 shows all messages exchanged between services and all p-assertions to the provenance store made by services. As can been seen, the provenance store service is added to the far right end. It is responsible for recording p-assertions from both client and service sides. For example, a message *makeCake(sugar, flour, beating time, temperature):record* is sent to the provenance store for recording by both the User Interface and the Baker service. The return message is recorded in the same way.

Actor state p-assertions are aimed at recording information for a participating actor of an interaction. It usually consists of such information as the time of sending, receiving a request or a response message and the property of actors themselves. As an example, the BakeInOven actor contains a location parameter and a temperature parameter for baking the cake. Further information about actor state p-assertion could be the brand, date of manufacture, etc.

Recording p-assertions is carried out in conjunction with the BVSC process. Once the process is completed, all p-assertions will be recorded in the provenance store.

4. System design and implementation

We have designed and implemented a provenance system for the BVSC application based on the above analysis within a service oriented view, which is described in detail below.

4.1 Provenance modelling and representation

Given that provenance consists of interaction p-assertions and actor state p-assertions, provenance modelling is actually the modelling of interaction and actor states. In a SOA, an interaction means an invocation or a result message between a client and a service. Interaction p-assertions are, in essence, the inputs and outputs of the services involved in the interaction.

Practically, provenance modelling consists of three aspects, i.e. the definition of data types used by messages and actor state p-assertions, the specification of messages that form the core of interaction p-assertions and the design of actor state p-assertions models. In line with common practice in Web Service design, we have used XML schema to model all data types, messages for the interactions and actor state p-assertions. We use the XML to represent concrete p-assertion data.

In developing a provenance system prototype for BVSC application we have designed a suite of data models. Figure 4 shows some of the data types, Figure 5 shows the message used in the BakeInOven service and Figure 6 shows the actor state p-assertions model for the BakeInOven service. All models are represented in UML.

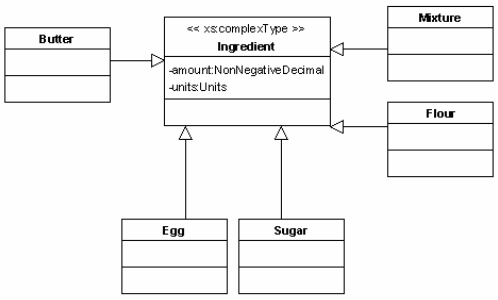

Figure 4: The BVSC data types

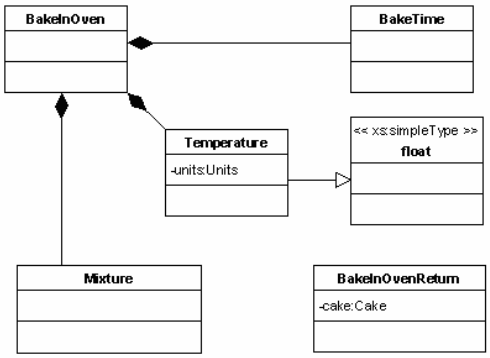

Figure 5: The BakeInOven service message

4.2 Provenance-aware web service design

A Web Service is a software system designed to support interoperable machine-to-machine interactions over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialisation in conjunction with other Web-related standards.

Web service design requires the specification of interfaces. In the BVSC application all services' interfaces, i.e., the creation of WSDL files, are specified in terms of the defined data types and messages.

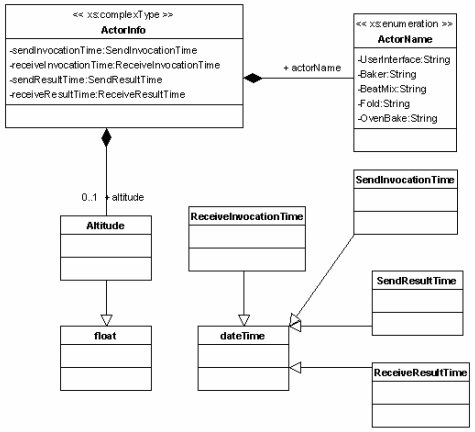

Figure 6: Actor state p-assertion data model

4.3 Provenance system implementation

The physical storage of provenance store is implemented in a file system that is organised in a structure as defined in Figure 7. The URL of the host where the provenance store is located is defined as the root node of the provenance store. Under this top-level node, there will be multiple sessions. Each session refers to a workflow run with a unique identifier (ID) and contains at least one activity with a unique activity ID.

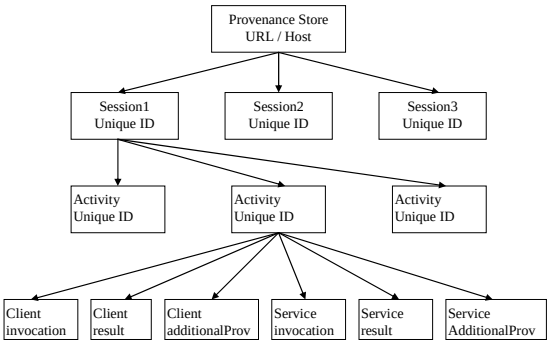

Figure **7:** The structure of a provenance store

An activity describes an interaction between a client and a service. An interaction is further split into an invocation message and a result message. Both messages are stored by both client and service sides so that conflicts among two p-assertions about the same interaction can be detected. Apart from interaction messages, the states of an actor involved in the interaction will also be recorded in the provenance store. While Figure 7 presents an abstract outline of the provenance data structure, there could be many different implementations.

The provenance recording system is implemented using the Axis Web Services framework deployed within the Tomcat servlet container [10], and utilised the provenance recording protocol [11] and APIs [12].

4.4 Provenance query and example uses

Once provenance is recorded and stored, an important question then is how to access, explore and reuse p-assertions in an optimally beneficial manner. From the end-users' point of view, the exploitation of provenance in enhancing problem solving processes (for example, speeding up or lowering its cost), is likely to be of greater consequence than the preliminary activities of provenance recording and archiving.

This section introduces a query algorithm used in a provenance store to find a general data item. Then we present a query example we have performed in the BVSC application to demonstrate the query mechanism and, most importantly, the usage of provenance data.

4.4.1 Query algorithm

Our query algorithm is based on the assumption that the final result of the BVSC process (the cake) has a unique ID, which we term resultID. The input to the query algorithm is the resultID and the name of a data item (which we term searchItem), and the output is the quantity or unit associated with searchItem. The algorithm is given in Figure 8 below.

4.4.2 Query example

As discussed in the BVSC application, the amount of sugar used in the whisking activity is one of the factors that may affect a cake's quality of taste; and there is generally a guideline on the minimum amount of sugar to be used in order to attain a minimum quality of taste.

To find out if the correct sugar amount as specified in the recipe was used, we need to retrieve the amount of sugar used for baking a specific cake from a provenance store, and subsequently perform a comparison with the guideline on the minimum amount. Assuming that a cake with a unique ID is given, the query trail is as follows based on the following instantiation of the query algorithm just described:

- Search through all messages in the provenance store until a Return message is located which contains the unique cake ID;
- Locate the makeCake activity which contains the Return message;

- Locate the session ID that contains this makeCake activity;
- Locate the Whisk activity corresponding to this session ID;
- Extract the amount of sugar shown in the Whisk message within the Whisk activity. Both the client and service view of the message should coincide.

Once the actual sugar amount used is obtained from the above steps, we can compare it with the guideline for the minimum amount of sugar and draw an appropriate conclusion.

```
foundID = false;
foundData = false;
located SessionID = false;

For all session IDs in a provenance store {
  For all activity IDs in current session ID {
    For all messages in current activity ID {
      if (resultID exists in current message)
      {
        located session ID = current session ID;
        foundID = true;
        break;
      }
    }
    if (foundID) break;
  }
  if (foundID) break;
}
if (not foundID)
  Show error message and exit;

For all activity IDs in located sessionID {
  For all messages in current activity ID {
    if (searchItem exists in current message)
    {
    foundData = true;
   return unitQuantity corresponding to searchItem;
    }
  }
}
if (not foundData)

   Show error message and exit
```

Figure 8: The query algorithm

5. Discussion

Provenance has been investigated in other contexts [2, 3, 4, 5] using definitions such as audit trail, lineage, dataset dependence and execution trace. By framing and analysing the BVSC application within a SOA we have chosen here instead to refer to provenance as the process that led to the data. This process-centred view of provenance is motivated by our observation that most scientific and business activities are usually accomplished by a sequence of actions performed by multiple participants. The recently emerging service-oriented computing paradigm, in which problem solving amounts to composing services into a workflow, is a further motivating factor towards the adoption of our process-centred view on provenance.

Through the analysis of the BVSC process and its information flow using a service-oriented view and sequence diagram, we further identify that provenance in the context of a SOA consists of two main types of provenance data: interaction p-assertions and actor state p-assertions. Interaction p-assertion is concerned with the capture of an execution trace while actor state p-assertion concentrates on the information pertaining to participating entities. We have placed special emphasis on interaction p-assertions, since services are usually dynamically discovered, aggregated, executed and discontinued in a virtual organisation on the Grid. In this context, information on how services are invoked, what messages are passed among them, and when they are invoked, are usually required in order for a workflow result to be analysed or for a workflow to be repeated.

We have identified the notion of classifying the recorded p-assertions into hierarchical groups on the basis of the relationships between service interactions. This idea is reflected in the modelling of the provenance store. We have also developed a provenance service to carry out p-assertion recording and storage. The decision to employ a service-oriented implementation is made based on several considerations. Firstly, provenance can provide added value for complex distributed applications that are increasingly adopting a service-oriented view for modelling and software engineering, as demonstrated in grid computing. Secondly, a service-oriented implementation of the provenance infrastructure simplifies its integration into a SOA, thus facilitating the adoption of the infrastructure in SOA-based applications. Finally, a service-oriented provenance infrastructure deploys easily into heterogeneous distributed environments, thus facilitating the access, sharing and reuse of provenance data.

Although as simple as they are, the query algorithm and the performed query sample demonstrate that provenance data can be accessed through the designed algorithm. Most importantly it demonstrates how provenance data can be used to answer questions. While there are undoubtedly many different questions in terms of application characteristics and many different ways of accessing and retrieving provenance data, the query algorithm and example present a showcase for the viability of provenance usage.

The benefits of developing a provenance system prototype in the context of BVSC are multiple. Firstly, it helps pin down the conception, modelling and representation of provenance in a SOA. Secondly, it helps define the characteristics of the provenance problems, and identify and clarify user requirements in the context of OSA-based applications. Thirdly, it helps identify and clarify the software

requirements for a provenance system, i.e. what a provenance system has to do. Fourthly, the successful design and operation of the entire provenance system prototype have demonstrated and proved our conception of provenance, its design approaches and implementation rationale. Finally, all findings, insights and experiences acquired in the development of the provenance system prototype will be used in future work towards a secure, scalable and generic provenance system architecture and its corresponding reference implementation.

However, the simplicity of the BVSC application scenario does impose some limitations on the investigation of other issues. For example, the BVSC process involves a linear sequence of services, and hence we have not considered the issue of iterative loops and/or parallel processing. The provenance system is implemented in a centralised fashion, and there are clearly additional issues of distribution and scalability to consider if a provenance infrastructure was deployed in a distributed grid environment. The actor state p-assertions in the BVSC application is currently vague and does not have explicit semantics associated with it. Other issues that have not been studied yet include the scalability and security of a provenance system.

6. Conclusions

In this paper we have defined the concept of provenance and further clarified it by analysing the BVSC process and its information flow within the context of a SOA. We have identified the core components and functionalities, i.e., provenance recording, storage and query, for a provenance system in providing provenance support for the BVSC application. We have also developed a suite of generic APIs and front end GUI in implementing the provenance system for the BVSC application, which can be used for the realisation of provenance systems for any other application domain.

Our contributions are threefold. Firstly, the research provides a proof of concept for provenance and provenance systems. Secondly, it provides guidelines towards the construction of a basic provenance system. Finally, it demonstrates a possible design and implementation pattern for provenance-enabled applications. In the future we shall focus on the specification and design of a generic provenance architecture, which will include the design of an appropriate query interface. We shall also tackle security and scalability issues.

Acknowledgement

This work is supported by the EU PROVENANCE project (IST511085) under FP6/IST programme. The BVSC implementation makes use of the PReServ provenance store developed in the PASOA project (EPSRC Grant GR/S67623/01).

References

[1] Foster I, Kesselman C, Nick J, Tuecke S., 2002. Grid Services for Distributed System Integration. Computer, 35(6).

[2] Workshop on Data Derivation and Provenance, October 17-18, 2002, Chicago, USA, http://www-fp.mcs.anl.gov/~foster/provenance/

[3] Data Provenance and Annotation, December 1-3, 2003, Edinburgh, UK, http://www.nesc.ac.uk/esi/events/304/

[4] Buneman P., Khanna S. and Tan W.-C., 2000, Data provenance: Some basic issues, In Foundations of Software Technology and Theoretical Computer Science

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

[6] Martin Szomszor and Luc Moreau. Recording and reasoning over data provenance in web and grid services. In International Conference on Ontologies, Databases and Applications of SEmantics (ODBASE'03), Lecture Notes in Computer Science, Catania, Sicily, Italy, November 2003.

[7] Mark Greenwood, Carole Goble, Robert Stevens, Jun Zhao, Matthew Addis, Darren Marvin, Luc Moreau, and Tom Oinn. Provenance of e-science experiments - experience from bioinformatics. In Proceedings of the UK OST e-Science second All Hands Meeting 2003 (AHM'03), page 4 pages, Nottingham, UK, September 2003.

[8] EU Provenance project: User Requirements Document. http://twiki.gridprovenance.org

[9] Victoria sponge cake recipe http://thefoody.com/baking/victoriasponge.html

[10] Fenglian Xu, Alexis Biller, Liming Chen, Victor Tan, Paul Groth, Simon Miles, John Ibbotson and Luc Moreau **"**A proof of concept design for provenance**",** Technical report, University of Southampton, 2005

[11] Paul Groth, Michael Luck, and Luc Moreau, 2004, A protocol for recording provenance in service-oriented Grids**,** In Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS'04), France.

[12] PReServ 0.1.5: Provenance Recording for Services, http://twiki.pasoa.ecs.soton.ac.uk/bin/view/ PASOA/SoftWare