Logical Architecture Strawman for Provenance Systems

Luc Moreau, Liming Chen, Paul Groth, John Ibbotson, Michael Luck, Simon Miles, Omer Rana, Victor Tan, Steven Willmott, Fenglian Xu

April 26, 2005

Abstract

The purpose of this document is to propose a logical architecture for a provenance system. The logical architecture is specified independently of specific technologies. Specifically, we introduce our definition of provenance in the context of service-oriented architectures, and we identify the different roles that exist in a provenance system.

This document is the first in a series of documents aiming at specifying an architecture for a provenance system. In the first instance, we focus on a *logical* architecture, which we characterise as a technology-independent abstract description of the different roles involved in provenance systems. In order to ground this presentation, we consider a broad view of service-orientation, which encompasses many of the technologies used to build distributed systems. In this context, we define our notion of provenance, itself derived from the common sense definition of provenance, which we then rely upon to elaborate our proposed architecture. In this document, numbers in the margin of this document refer to comments in the appendix, which may discuss technology specific aspects.

Common Sense Definition We first introduce the common sense definition of the word 'provenance'. Its etymology is the French verb 'provenir', which means to come forth, originate. According to the Oxford English Dictionary, provenance is defined as *(i) the fact of coming from some particular source or quarter; origin, derivation. (ii) the history or pedigree of a work of art, manuscript, rare book, etc.; concr., a record of the ultimate derivation and passage of an item through its various owners.* Likewise, the Merriam-Webster Online dictionary defines provenance as *(i) the origin, source; (ii) the history of ownership of a valued object or work of art or literature.* From such definitions, we can distinguish two meanings for provenance: first, *as a concept*, it denotes the source or derivation of an object; second, *more concretely*, it is used to refer to a record of such a derivation. We will come back to such a distinction when we define the notion of provenance we adopt in this project.

Context We take the broad view that open, large-scale systems are typically designed using a *service*-oriented approach; by service, we do not intend to restrict ourselves to a specific technology, instead we mean to refer to components that take inputs and produce outputs. Such services are brought together to solve a given problem typically via a *workflow* definition that specifies their composition. In this abstract view, interactions with services take place with *messages* that are composed in accordance with services *interface* specifications. 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. Finally, the execution of a workflow will be referred to as a *process*. *(1)* *(2)* *(3)* *(4)*

In real life, actors may have internal *states* that change during the course of execution. An actor's state is not directly observable by other actors; to be seen by another actor, the state (or part of it) has to be communicated as part of a message sent by its owner actor. *(5)*

Our broad, technology-independent approach to SOAs has formal foundations in the π-calculus [Mil99] and asynchronous distributed systems [Lyn95, Tel94]. According to such a conception of the world, messages are the only mechanism used to transfer information between actors. The π-calculus is of interest in this context because of its approach to defining events that are internal to actors as hidden communications; an asynchronous view of distributed systems is, however, a better match to service-oriented architectures. In both cases, π-calculus and asynchronous distributed systems consider that messages are also the only observable events; this is the only assumption that can be made of open distributed systems, in which actors may not necessarily wish or be able to reveal their internal behaviour or state, despite being involved in the computation of results we wish to track the provenance of.

Provenance Definition We propose a definition of provenance that is inspired from previous work [GLM04a, GLM04b, Gro04, TGX05, MGBM05, SM03], the EU Provenance project pre-prototype [XBC+05] and its user requirements document [AV05]. In the context of this document, *the provenance of a piece of data is the process that led to the data*. Referring to the two common sense definitions of provenance, 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 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. While our applications will specify the form that such documentation should take, we can identify several of its general properties. Documentation can be complete or partial (for instance, when the computation has not yet terminated); it can be accurate or inaccurate; it can present conflicting or consensual views of the actors involved; it can be descriptive or conceptual. *(6)*

Provenance System and Architecture In the context of this document, a *provenance system* is defined as a computer system that deals with all issues pertaining to the recording, maintenance, visualisation, reasoning and analysis of the documentation of the process that underpins the notion of provenance. Such a system is a software implementation of a *provenance architecture*, which identifies the different roles in such a system, their interactions and the kind of provenance representation they are expected to support.

As far as a provenance architecture is concerned, we distinguish the activity that consists of *recording* the representation of provenance of some data from the activity that *makes use* of recorded provenance representation. We now further detail our notion of "documentation of a process", and we describe a logical architecture for recording it and making use of it.

Provenance Representation In this section, we introduce the key elements that form the representation of provenance in a SOA; further refinement will ultimately lead to data types for provenance representation. In our discussion, given the provenance of some data, we shall make the distinction between the whole of provenance and one of its constituents, i.e., a specific piece of information documenting part of the process that led to the data. Hence, a given element of the provenance representation will be referred to as a *p-assertion* (assertion, by an actor, pertaining to provenance). We note that a given p-assertion may belong to the provenance representation of multiple pieces of data. A p-assertion that is recorded documents a step of a process in progress, which ultimately will lead to a piece of data. At the time of the recording, we may ignore the piece of data that will be produced; however, the p-assertion being recorded constitutes an element of the provenance representation of the data. For instance, when some quality wood is being transported in the Amazon forest, one may ignore that it will be used for creating the frame for a future famous painting, still to be painted.

Computer science has a long tradition of focusing on communications and interactions as a central concept used in the study and modelling of complex systems, e.g., programming languages semantics, process algebrae and more recently in biological systems models. In the context of SOAs, interactions consist of the messages exchanged between actors. By capturing all the interactions that take place between actors involved in the computation of some data, one can replay an execution, analyse it, verify its validity or compare it with another execution. Given the open nature of the distributed systems that we consider, interactions (i.e., message exchanges) are the only events that we can observe. Hence, describing such interactions is core to the documentation of process.

Therefore, the documentation of a process that leads to a piece of data includes a set of *interaction p-assertions*, each describing an interaction between actors involved in the computation of the data. Practically, an interaction p-assertion contains a message exchanged between two actors. *(7)*

Interaction p-assertions capture the observable interactions between actors of a sys-

tem. In some circumstances, however, actors' internal states may also be necessary to understand the functionality, performance or accuracy of actors, and therefore the nature of the result they compute. Hence, we introduce the notion of an *actor state p-assertion* as 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 system, an actor state is not externally observable, and therefore can only be captured by cooperative contribution of the actor itself.

In summary, p-asssertions can be of two disjoint kinds: interaction p-assertions and actor state p-assertions. We note that both interaction and actor state p-assertions are independent of the actual service technology used to implement applications.

Provenance Architecture Roles In order to support the capture and querying of these categories of p-assertions, we have specified a provenance architecture that takes into account a broad range of use cases [MGBM05, AV05]. It is summarised in Figure 1, which we discuss in the rest of this section. Central to the architecture is the notion of a *provenance store*, which is designed to store and maintain provenance representation beyond the life of a Grid application. In a given application, one or more provenance stores may be used in order to archive the representation of provenance: multiple provenance stores may be required for scalability reasons or for dealing with the physical deployment of a given application, possibly involving firewalls.

In order to accumulate p-assertions, a provenance store provides a *submission interface* that allows different actors to submit p-assertions related to their interactions and internal states. A provenance store is not just a sink for p-assertions: it must also support some query facility that allows, in its simplest form, browsing of its contents, and, in its more complex form, search, analysis and reasoning over the provenance representation so as to support use cases. To this end, we introduce *query interfaces* that offer multiple levels of query capability. Finally, since provenance stores need to be configured and managed, an appropriate *management interface* is introduced.

Some *actor-side libraries* facilitate the tasks of submitting p-assertions in a secure, scalable and coherent manner and of querying and managing provenance stores. They are also designed to ease integration with legacy applications. Interfaces and libraries have different purposes: the former specify the messages accepted and returned by provenance stores, and will be the focus of a standardisation proposal to ensure that applications can inter-operate with different implementations of provenance stores; the latter are convenience libraries offering bindings for specific programming languages.

(8)

During an application's execution, all *application services* are expected to submit p-assertions to a provenance store; this not only applies to *domain-specific services*, but also to *workflow enactment engines* and *registries*. Additionally, users may have

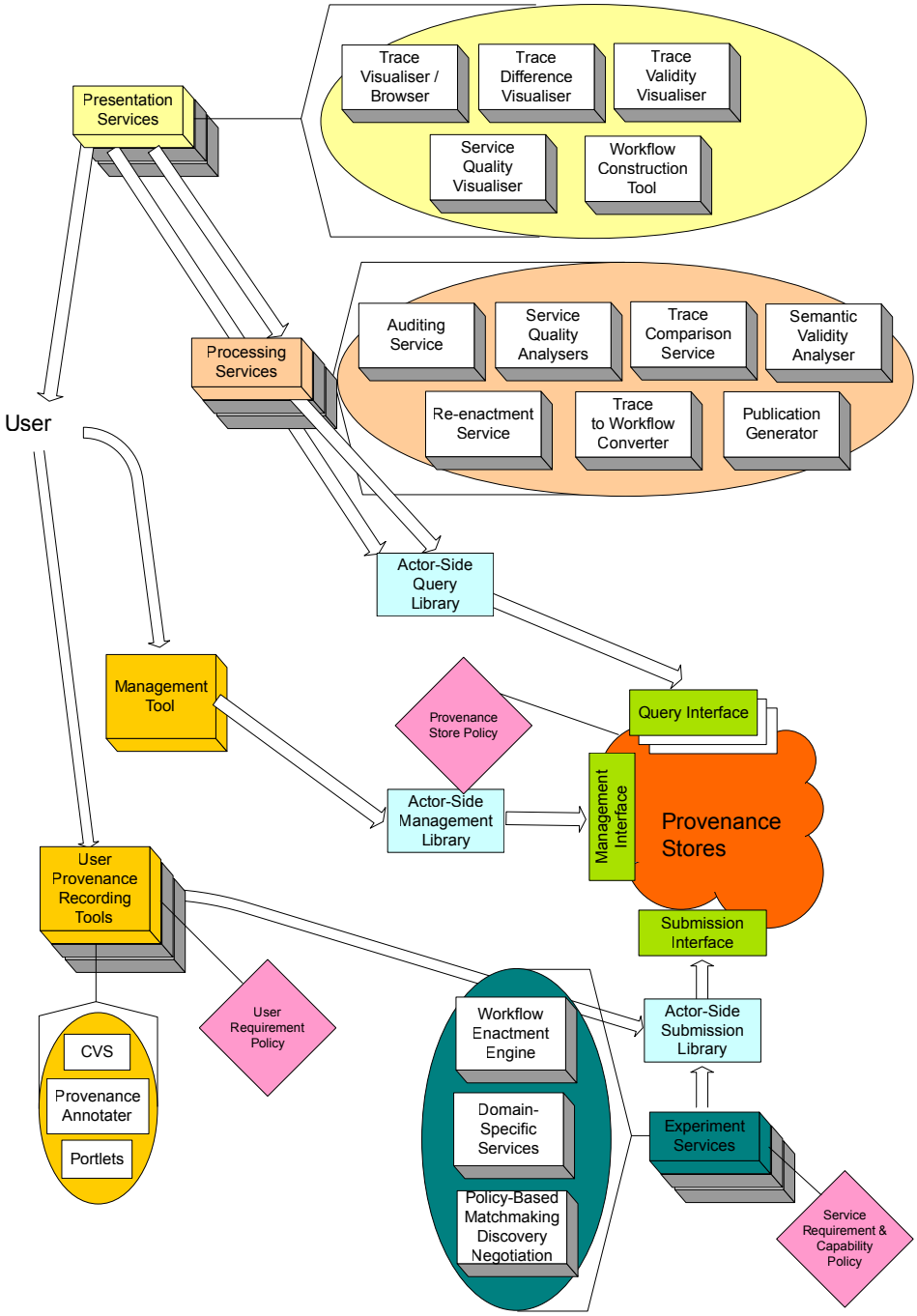

Figure 1: Provenance Logical Architecture

access to *tools* to manage provenance stores and to submit information to provenance stores, such as annotations about previous execution.

Once p-assertions have been recorded in a provenance store, provenance representation can be used by *processing services* and *presentation services*. The former provide added-value to the query interfaces by further searching, analysing and reasoning over recorded provenance, whereas the latter essentially visualise the contents of the store and of processing services' outputs. Figure 1 provides examples of such processing and presentation services offering functionality discussed in [MGBM05]. For instance, processing services can offer auditing facilities, can analyse quality of service based on previous execution, can compare the processes used to produce several data items, can verify that a given execution was semantically valid, can identify points in the execution where results are no longer up-to-date in order to resume execution from these points, can re-construct a workflow from an execution trace, or can generate a textual description of an execution. Presentation services can for instance offer browsing facilities over provenance stores, visualise differences in different execution, illustrate execution from a more semantic viewpoint, can visualise the performance of execution, and can be used to construct provenance-based workflows. We note that such a list of processing and presentation services is illustrative and not exhaustive; furthermore, it does not represent a commitment by the project to deliver these services specifically.

To be generic, a provenance architecture must be deployable in many different contexts and has to support user preferences. To adapt the behaviour of the architecture to the prevailing circumstances and preferences, several *policies* are introduced to help configure the system in its different aspects. Specifically, *(i)* policies state user requirements about recording, e.g., to identify the provenance stores to use, the level of documentation required by the user, desired security aspects; *(ii)* policies specify capabilities of documenting execution that services may wish to advertise (such as their ability to provide some type of actor states documentation), but in order to fulfill these, that they may also require from other services they rely upon (such as their need for high throughput or highly persistent provenance stores); *(iii)* policies define configurations of provenance stores, from a deployment and security viewpoint (e.g., resources they use, their access control list, or registry where they should be advertised). By making explicit all these policies, it becomes possible to *discover* services that *match* user or other service needs. When requested policies conflict with discovered policies, *negotiation* can be initiated to find a compromise between the offer and demand.

Protocol for Recording p-assertions We introduce PReP, the Provenance Recording Protocol, which specifies the messages that actors can asynchronously exchange with the provenance store in order to record p-assertions [GLM04a, GLM04b]. While PReP identifies *how* p-assertions should be recorded, it does not specify *when* to do so. Such flexibility must be provided in order to customise recording according to the application's needs: if the application requires provenance to be used immediately while

execution still proceeds, p-assertions may be submitted synchronously with execution; alternatively, when provenance is used after application completion, p-assertions may be recorded asynchronously so as to reduce recording overhead. Actor-side libraries offer support for such asynchronous recording of p-assertions.

Query Interfaces In order to be useful, the provenance stores must support queries over the provenance representation, whatever the state of the process documentation (complete or not, detailed or not). Furthermore, while we have identified application-independent constituents of provenance representation, we also expect provenance queries to refer to domain specific aspects. Therefore, multiple query interfaces will be provided in order to support various query capabilities. These interfaces remain to be specified.

Conclusion This document has introduced the notion of provenance of some data in the context of SOAs, which we expressed in terms of interaction and actor state p-assertions. It has also identified some key roles involved in a provenance architecture. This document is a living document which will continue to evolve in order to meet future requirements.

The document is complemented by a security logical architecture, which analyses the security issues relevant to the provenance architecture and identifies its core components. In the future, we will analyse the proposed logical architecture against the captured technical requirements. Instantiations of the logical architecture to specific technologies and applications will then be proposed.

1 Appendix: Notes

This appendix refers to annotations introduced in the margin of the logical architecture document.

Note 1: Specifically, the following are all considered as "services" because they all take some inputs and produce some outputs: Web Service, Corba or RMI objects, command line program.

Note 2: With such a broad definition, we see that WS-BPEL, WSFL, VDL, Dagman's DAGs or Gaudi are all workflow frameworks capable of expressing the composition of services. Likewise, a script calling several command line commands is also regarded as a workflow.

Note 3: Such messages take the form of SOAP messages for Web services. In the case of command line executables, we do not have explicit messages; instead, they take some explicit arguments potentially representing both inputs and outputs. We also see a memory shared by two threads as a way of implementing such message-passing mechanism; the message itself is the information stored in the shared memory.

Note 4: Our definition of process, like the Unix notion of process, refers to an instance of a running program (workflow here).

Note 5: At this stage of the specification, we do not make the distinction between resource and service [CaIFF+04] since they are defined in the context of the specific Web Services technology. Our broad view of message allows us to include in a message the necessary reference to resources, as required by WSRF.

Note 6: We note that the definitions introduced here are in no way restricted to electronic data; they are equally applicable to physical objects. Specifically, *the provenance of some object is the process that led to the object*; *it is represented by the documentation of the process that led to the object*. For a discussion of the relationship between the provenance of an object and a data, we refer the reader to a companion document in preparation.

Note 7: In a Grid application based on command line executables, an interaction p-assertion includes the executable fully qualified name, its inputs and its outputs, whereas in a Web Services based approach, interactions documentation will include input and output SOAP messages, and a reference to the service, port and operation being invoked. In the latter case, we note that an interaction p-assertion includes not only the SOAP message body, but also its envelope, containing valuable information such as security, addressing, resource or coordination contexts.

Note 8: In a concrete instantiation of the logical architecture for Java and Web Services technologies, interfaces will be specified by WSDL, whereas libaries will be Java classes, generated by `wsdl2java`, implementing the stubs necessary to communicate with the provenance stores, with possibly some basic convenience functions.

2 Terminology

In this appendix, for reference, we provide definitions of concepts that are related to provenance.

A *log/logbook* is an official record of events during the voyage of a ship or aircraft (OED). An *audit trail* is a record of the computing processes which have been applied to a particular set of source data, showing each stage of processing and allowing the original data to be reconstituted; a record of the transactions to which a database or a file has been subjected (cf. TRACE). (OED) A *trace* is the detailed examination of the execution of a program or part of one (usu. to investigate a fault) with the aid of another program that can cause individual instructions, operands, and results to be printed or displayed as they are reached by the first program; the analysis so obtained; also, a trace program. (OED)

In the context of databases, the *why-provenance* of a piece of output data is the set of all witnesses to why that piece of data exists in the output, whereas *where-provenance* describes which pieces of source data contribute to a piece of output data [BKT01]. Given a warehouse data item i, finding the exact set of source data items from which i was derived is termed the *data lineage* problem [CWW00].

References

[AV05] Árpád Andics and Laszlo Varga. User requirements document. Technical report, MTA SZTAKI, February 2005.

[BKT01] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Why and Where: A Characterization of Data Provenance. In *International Conference on Database Theory (ICDT)*, 2001.

[CaIFF+04] Karl Czajkowski, Donal Ferguson adn Ian Foster, Jeffrey Frey, Steve Graham, Igor Sedukhin, David Snelling, Steve Tuecke, and William Vambenepe. The WS-Resource Framework, March 2004.

[CWW00] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2):179–227, 2000.

[GLM04a] Paul Groth, Michael Luck, and Luc Moreau. Formalising a protocol for recording provenance in grids. In *Proceedings of the UK OST e-Science second All Hands Meeting 2004 (AHM'04)*, Nottingham, UK, September 2004.

[GLM04b] Paul Groth, Michael Luck, and Luc Moreau. A protocol for recording provenance in service-oriented grids. In *Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS'04)*, Grenoble, France, December 2004.

[Gro04] Paul T. Groth. Recording provenance in service-oriented architectures. 9 month report, University of Southampton; Faculty of Engineering, Science and Mathematics; School of Electronics and Computer Science, 2004.

[Lyn95] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, December 1995.

[MGBM05] Simon Miles, Paul Groth, Miguel Branco, and Luc Moreau. The requirements of recording and using provenance in e-science experiments. Technical report, University of Southampton, 2005.

[Mil99] Robin Milner. *Communicating and mobile systems: the π-calculus*. Cambridge University Press, 1999.

[SM03] 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)*, volume 2888 of *Lecture Notes in Computer Science*, pages 603–620, Catania, Sicily, Italy, November 2003.

[Tel94] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.

[TGX05] Paul Townend, Paul Groth, and Jie Xu. A provenance-aware weighted fault tolerance scheme for service-based applications. In *Proc. of the 8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2005)*, May 2005.

[XBC+05] 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, February 2005.