Architecture for Provenance Systems

Editors: Paul Groth
Simon Miles
Victor Tan
Luc Moreau

Contributors: Andics Árpád
Alexis Biller
Miguel Branco
Liming Chen
Arnaud Contes
Frank Dannemann
Vikas Deora
Neil Hardman
Guy Kloos
Michael Luck
John Ibbotson
Omer Rana
Andreas Schreiber
Laszlo Varga
Javier Vazquez
Fenglian Xu
Steven Willmott

October 10, 2005
Version 0.4

Abstract

This document covers the logical and process architectures of provenance systems. The logical architecture identifies key roles and their interactions, whereas the process architecture discusses distribution and security. A fundamental aspect of our presentation is its technology-independent nature, which makes it reusable: the principles that are exposed in this document may be applied to different technologies.

Members of the PROVENANCE consortium:

IBM United Kingdom Limited	United Kingdom
University of Southampton	United Kingdom
University of Wales, Cardiff	United Kingdom
Deutsches Zentrum fur Luft- und Raumfahrt s.V.	Germany
Universitat Politecnica de Catalunya	Spain
Magyar Tudomanyos Akademia Szamitastechnikai es Automatizalasi Kutato Intezet	Hungary

Members of the PASOA project:

University of Southampton	United Kingdom
University of Wales, Cardiff	United Kingdom

Contents

Chapter 1

Introduction

1.1 Motivation

The importance of understanding the process by which a result was generated is fundamental to many real life applications (science, engineering, medical domain, supply management, etc). Without such information, users cannot reproduce, analyse or validate processes or experiments. Provenance is therefore important to enable users, scientists and engineers to trace how a particular result had been arrived at.

We propose a definition of provenance that is suited to the computational model underpinning service oriented architectures, an architectural style regarded as suitable for large scale, open systems. Based on such a definition, we conceive a computer-based representation of provenance that allows us to perform useful reasoning about the origin of results.

The purpose of this document is to present an architecture for provenance systems, its rationale and a methodology guiding its use. According to Kruchten [Kru95], several views of an architecture can be considered:

- The *logical architecture* primarily supports the functional requirements, i.e. what the system should provide in terms of services to its users: the system is decomposed into a set of abstractions, and their high level interactions are identified.

- The *process architecture* takes into account some non-functional requirements, such as performance and availability. It addresses issues of concurrency and distribution, of system integrity, of fault-tolerance and how the main abstractions from the logical view fit within the process architecture.

- The *development architecture* focuses on the actual software module organisation, including libraries.

- Finally, the *physical architecture* takes into account primarily the non-functional requirements of the system such as availability, reliability, performance and scalability.

This document covers the logical and process architectures of provenance systems. The logical architecture identifies key roles and their interactions, whereas the process architecture discusses distribution and security. A fundamental aspect of our presentation is its technology-independent nature, which makes it reusable: the principles that are exposed in this document may be applied to different technologies.

The development and physical architectures will be presented in separate documents, explaining how the architectural design is mapped onto the Web Services stack of standards, and how each individual service is implemented.

1.2 Structure of Document

This document is structured as follows.

Chapter 2: Provenance Definition Based on the common sense definition of provenance, we propose a new definition of provenance that is suited to the computational model underpinning service oriented architectures. Since our aim is to conceive a computer-based representation of provenance that allows us to perform useful reasoning about the origin of results, we examine the nature of such representation, which is articulated around the documentation of execution.

Chapter 3: Logical Architecture We then examine the architecture of a provenance system, centered around the notion of a provenance store. We also examine models of execution documentation.

Chapter 4: Security Architecture Although security is a non-functional requirement, software engineering methodology strongly recommends that security considerations be integrated into the development life-cycle as early as possible. Many of the application domains in which a provenance architecture could potentially be deployed have stringent requirements on access to data manipulated within the system. A security architecture that helps addressing these issues is discussed in this chapter.

Chapter 5: Distribution Architecture This chapter discusses distribution in the provenance architecture. First, it introduces a set of patterns that identify communications between key architecture roles; second, it explains how the data model underpinning the architecture allows for data that is geographically distributed; finally, it explains how deployments of core architecture components can cater for high load.

Chapter 6: Identifying Data Items When using a provenance-aware application, a user or software client may ask for the provenance of a piece of data at any time after that data has been produced. There are several factors which make asking the provenance question difficult: a piece of data may not have a unique identifier, may be moved from where it was initially stored after being produced etc. These factors mean that identifying data items to determine their provenance is a non-trivial task. In this chapter, we analyze a set of solutions for identifying arbitrary pieces of data to be applied when making an application provenance-aware.

Chapter 7: Provenance Modelling This chapter describes the various data models for the information recorded in the provenance store. The modelling describes how this information can be organised, identified, and extended.

Chapter 8: Functionality This chapter provides a more detailed description of the functionality supported by a provenance system. It relies on an overall model of information recorded in the provenance store, which is acted upon by and recording, querying and managing capabilities. This presentation is in natural language, informal, and will be used to derive more formal specifications expressed in UML.

Chapter 9: Actor Behaviour This chapter describes the expectations on actors behaviour so that process documentation can correctly be recorded and provenance questions usefully answered.

Chapter 10: Justification This chapter describes how the software requirements identified by the EU Provenance project for a provenance system are satisfied by the architecture.

Chapter 11: Related Work The chapter presents related work and discusses how our approach to provenance differs from existing systems.

Notes A set of technology-specific comments.

Index An index of terms defined in this document.

1.3 Status of this Document

This document will keep on evolving during the course of the EU Provenance project. Different chapters contribute to different milestones of the project, summarised in the following table, with schedules of drafts, reviews and final revisions.

Once a chapter has been finalised, following changes will be agreed and clearly documented.

Milestone	Chapters	Draft by	Review by	Final by
Logical architecture frozen:	2, 3	24/6	8/7	15/7
Functional architecture frozen:	4, 6 10	7/10	21/10	28/10
Final architecture frozen:	5, 7, 8, 9, 11, ...	TBC	TBC	TBC

For reference, the versions of individual chapters are summarised in the following table.

Chapter	Revision
Chapter 1	Revision: 1.25
Chapter 2	Revision: 1.25
Chapter 3	Revision: 1.25
Chapter 4	Revision: 1.25
Chapter 5	Revision: 1.25
Chapter 6	Revision: 1.25
Chapter 7	Revision: 1.25
Chapter 8	Revision: 1.25
Chapter 9	Revision: 1.25
Chapter 10	Revision: 1.25

1.4 Acknowledgements

The architectural design presented in this document is the ouput of research funded in part by the EU Provenance project (IST 511085) and EPSRC PASOA project (GR/S67623/01). It draws on the recording protocol (PReP), the P-Structure, the query interface and requirements of the PASOA (Provenance-Aware Service Oriented Architecture) project (EPSRC GR/S67623/01). The approach for identifying data items for provenance purposes is jointly developed by both EU Provenance and PASOA projects.

Specifically, this document is inspired by the following EU Provenance publications [AV05b, AV05a, MCG+05, XBC+05] and PASOA publications [Gro04, GLM04a, GLM04b, MGBM05, GMF+05, Gro05, GMM05, WMF+05, Bra05]. We would also like to thank Jim Myers for reading and providing feedback on drafts of this document.

Chapter 2

Provenance Definition

2.1 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 follows.

Definition 2.1 (OED Provenance Definition) (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 follows.

Definition 2.2 (MWO Provenance Definition) (i) *the origin, source;* (ii) *the history of ownership of a valued object or work of art or literature.* □

Both definitions are compatible since they regard provenance as the derivation from a particular source to a specific state of an item. The nature of the derivation, or history, may take different forms, or may emphasise different properties according to interest. For instance, for a piece of art, provenance usually identifies its chain of ownership. Alternatively, the actual state of a painting may be understood better by studying the different restorations it underwent.

From definitions 2.1 and 2.2, we can also distinguish two different understandings of 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 shall return to such a distinction when we define the notion of provenance we adopt in this project.

2.2 Context: Service Oriented Architectures

Given that our work predominantly focuses on Grid and Web Services, we summarise some relevant terminology in this section. We take the broad view that

open, large-scale systems are typically designed using a *service*-oriented approach [SH05], usually referred to as service-oriented architectural style [Bur00]. As far as services are concerned, we do not intend to restrict ourselves to a specific technology; instead, we take services to be components that take inputs and produce outputs. Such services are brought together to solve a given problem typically via a *workflow* that specifies their composition. In this abstract view, interactions with services take place using *messages* that are constructed in accordance with service *interface* specifications. In a service-oriented architecture (SOA), clients typically invoke services, which may themselves act as clients for other services; hence, we use the term *actor* to denote either a client or a service in a SOA. Finally, the execution of a workflow is referred to as a *process*. *(1)* *(2)* *(3)* *(4)*

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 within 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 view 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.

2.3 Definition of Provenance

In this section, we focus on data produced by computer systems, and we define the provenance of a piece of data (or data item). Specifically, we consider service-oriented architectures, as discussed in Section 2.2, since they constitute the architectural style adopted to build large scale open systems. (In Section 2.6, we examine how our definition of provenance can be extended to cater for objects or events of the physical world.)

The two common sense definitions consider provenance to be the derivation from a particular source to a specific state of an item. We have identified a process in a SOA as the execution of a workflow, which we broadly see as a specification of a service composition. Hence, by having a description of the process that resulted in a data item, we can explain how such a data item has been obtained. Inspired by previous work [GLM04a, GLM04b, Gro04, TGX05, MGBM05, SM03b], the EU Provenance project pre-prototype [XBC+05], its requirements documents [AV05b, AV05a], and an architecture strawman [MCG+05], we propose the following definition of provenance, which makes explicit the notion of process.

Definition 2.3 (Provenance of a piece of data) *The provenance of a piece of data is the process that led to that piece of data.* □

In relation to the two common sense definitions of provenance, we note that Definition 2.3 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. Consequently, the provenance of a piece of data is to be represented in a computer system by some suitable documentation of the process that led to the data.

While specific applications determine the actual 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; and it can abstract more or less from reality.

2.4 Representation of Provenance

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 (cf. Chapter 7).

In the previous section, we stated that provenance of a data item is to be represented in a computer system by some suitable documentation of the process that led to it. To this end, 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 we define as follows.

Definition 2.4 (p-assertion) *A p-assertion is an assertion that is made by an actor and pertains to a process.* □

From this definition, we derive the notion of process documentation.

Definition 2.5 (Process Documentation) *The documentation of a process consists of a set of p-assertions made by the actors involved in the process.* *(6)* □

We note that a given p-assertion may belong to the provenance representation of multiple pieces of data. When a p-assertion is created (and later recorded), it documents a step of a process in progress, which ultimately will lead to a piece of data. At the time of the p-assertion creation, we may not know 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 not know that it will be used for creating the frame for a future famous painting, still to be painted and framed.

Among all the p-assertions, we now introduce two kinds of p-assertions that allow us to capture an explicit description of the flow of data in a process: *interaction p-assertions* and *relationship p-assertions*.

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 language 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. Describing such interactions is then core to the documentation of process.

Therefore, the documentation of a process includes a set of *interaction p-assertions*, each describing an interaction between actors involved in the process.

Definition 2.6 (Interaction p-assertion) *An interaction p-assertion is an assertion of the contents of a message by an actor that has sent or received that message; the message must include information that allows it to be identified uniquely.* □

We do not prescribe the nature of the assertion of the message contents; instead, such decisions are left to the specific application. For instance, an interaction p-assertion could simply contain a copy of the message exchanged between two actors. Alternatively, if some data contained in the message is regarded as confidential by the actor or too large to be manipulated, the assertion may consist of the message in which the data concerned has been replaced by some other data or a pointer. *(7)*

A crucial element of an interaction p-assertion is information to identify a message uniquely. Such information allows us to establish a flow of data between actors. Indeed, let us consider two interaction p-assertions: actor A making an assertion α_A that it sent actor B a message with identity i, and actor B making an assertion α_B that it received from A a message with the same identity i. Such a pair of interaction p-assertions α_A, α_B is said to be "matching"; it identifies a flow of data from actor A to B.

Actors may directly return outputs for the inputs they receive; alternatively, they may invoke other actors in order to obtain intermediate results that help them return their outputs. In both circumstances, the relationship between the outputs and the inputs to the actor is not explicit in the messages themselves, and can only be understood by an analysis of the actor's business logic, which is private to the actor.

We do not expect the source code of the actor to be made available, because it may not be feasible, or the code may not be at a suitable level of abstraction. Instead, in order to permit some understanding of the flow of data, an actor may decide to "volunteer" some information that is only available to it. An actor

may provide *relationship p-assertions* that identify the relationship between its outputs (whether as returned result or invocation message to other actors) and its inputs (or intermediary results received from invoked actors).

Definition 2.7 (Relationship p-assertion) *A relationship p-assertion is an assertion about an interaction, made by an actor, that describes how the actor obtained output data or the whole message sent in that interaction by applying some function to input data or messages from other interactions. A relationship p-assertion is directional.* □

While matching interaction p-assertions denote a flow of data between actors, relationships explain how data flows inside actors. Relationship p-assertions are directional since they explain how some data was computed from other data.

Figure 2.1 illustrates two actors. The first is a primitive actor, i.e., one that does not invoke other actors, or alternatively, an actor that does not make assertions of the messages it sends to other actors (say, for privacy reasons). In order to contribute some information about its internal flow of information, it can indicate that its output data (in the output message) is a function of the input data (contained in the input message). The second actor of Figure 2.1 is not primitive, and makes assertions of the contents of the messages it sends to and receives from another actor. Like the first actor, it may indicate that its output is a function of its input; alternatively, it may explain how the data contained in the secondary invocation message and its result relate to the input and output.

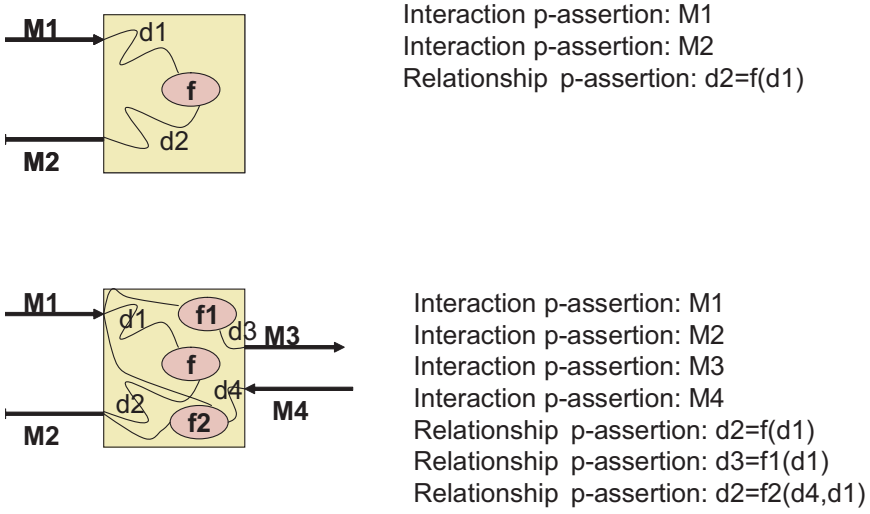


Figure 2.1: Data flow assertions by opaque and transparent actors

Figure 2.1 displays the ideal case of purely functional actors, which do not maintain a persistent state across invocations. The same approach generalises

to stateful actors: the data in an output message can be a function of the data received during a previous interaction and kept in a persistent store. *(8)*

Hence, interaction p-assertions denote data flows between actors, whereas relationship p-assertions denote private ones. Such data flows are core elements to reconstitute functional data dependencies in execution. In the most general case, such data flows constitute a directed acyclic graph (DAG). From a specific data item, the data flow DAG indicates where and how the data item is being used; vice-versa, following relationships in reverse helps us identify how a data item was produced. The data flow DAG is thus a core element of provenance representation, but it is not the only one; other p-assertions can provide further information about internal states of actors during execution, as we now explain.

Interaction and relationship p-assertions capture the flow of data in a process. 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.

Definition 2.8 (Actor State p-assertion) *An actor state p-assertion is an assertion made by an actor about its internal state in the context of a specific interaction.* □

Actor state p-assertions can be extremely varied: they may 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.

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

2.5 Provenance Lifecycle and Three Provenance Views

In the previous section, we characterised the syntactic nature of p-assertions, in the form of a broad classification in three different categories, according to whether they document interactions, relationships or actor states. We now focus on a dynamic characterisation of p-assertions and, in particular, when they are created, recorded, queried and managed. These different phases identify a *provenance lifecycle*, which we now describe.

Before discussing the provenance lifecyle, it is necessary to introduce an architectural element, which we expand upon in Chapter 3. Since we aim to provide a long-term facility for storing the provenance representation of data items, we

delegate the role of making persistent, managing and providing controlled access to such provenance representation to a specific element, which we refer to as a *provenance store*. The choice of an explicit architectural element to embody this role in no way implies any form of physical deployment; instead, it helps us identify the kind of functionality that is necessary in order to offer support for provenance.

The provenance lifecycle is composed of four different phases. As execution proceeds, actors create p-assertions that are aimed at representing their involvement in a computation. After their creation, p-assertions are stored in a provenance store, with the intent they can be used to reconstitute the provenance of some data. The provenance store therefore acts as storage of p-assertions. After a data item has been computed, users (or applications) may need to obtain the provenance of this data item: they can do so by querying the provenance store. At the most basic level, the result of the query is the set of p-assertions pertaining to the process that produced the data. More advanced query facilities may return a representation derived from p-assertions that is of interest to the user. We will come back to this aspect in Section 2.7. Finally, as time progresses, the provenance store and its contents may need to be managed (subscription management, content relocation, etc). In summary, the provenance lifecyle is composed of four different phases: *(i)* creating, *(ii)* recording, *(iii)* querying and *(iv)* managing. A provenance system should provide support for all these phases.

We previously discussed the two understandings of provenance that Definitions 2.1 and 2.2 imply: conceptual and representational (in a computer system). In light of the provenance lifecycle, we can refine this view and distinguish three understandings of provenance. *(i)* As before, provenance can be seen as a concept from which we can explain how a result has been achieved. *(ii)* The recording phase of the provenance lifecycle results in a set of p-assertions accumulated in the provenance store. These p-assertions constitute a documentation of execution, which we regard as a *recording-time representational understanding of provenance*; by this, we mean that among all the p-assertions accumulated in the provenance store, there is a representation of the provenance of the data we are interested in. *(iii)* Alternatively, the lifecycle querying phase suggests that provenance queries filter out p-assertions and make them available in some representation (whether as a set of p-assertions or in some other form), which constitutes a *query-time representational understanding of provenance*.

When designing a generic provenance system, we cannot anticipate all forms of queries that users may wish to issue. Hence, to be able to support complex querying functionality, it is important to provide a complete and detailed set of p-assertions about the aspect of execution we are permitted to document. This inevitably may raise scalability concerns that have to be addressed by the architectural design for the lifecycle recording phase. Symmetrically, the challenge for a query facility is to identify a subset of useful p-assertions, by selecting, scoping and filtering p-assertions. (These aspects are discussed further in Section 2.7.)

2.6 Beyond Computer Data

We specifically restricted Definition 2.3 to the provenance of electronic data contained in a computer system. Our rationale was that our primary focus is service oriented architectures, used in building open, large scale sytems. However, objects in the real world also have a provenance. The purpose of this section is to examine how the approach we propose to track provenance of data can be extended to track provenance of physical world entities.

Initially, we consider a restrictive deployment, as illustrated in Figure 2.2. On the left hand side, we see a computer application, in a SOA style, composed of a set of actors and producing some result. With the approach presented in this chapter, p-assertions describing execution are stored in a provenance store. The actors however are not traditional processing actors that take inputs and produce outputs as result of their internal behaviour. Instead, such actors are directly wired to "actuator/sensor" pairs that operate on objects in the physical world and sense their environment, all represented on the right hand of the picture. (The actual "wiring" is represented by dashed lines.) Such actuators can be robots, taking objects as input and assembling them, painting them, wrapping them, or even shipping them. Sensors perceive events in the physical world, such as movement sensors, cameras, radar. Information can transit from an actor to an actuator: it can be seen as control order for the actuator; vice-versa, sensors can feedback information to the computer system. We assume here that the mapping is one to one, i.e., for one actor there exists one and only one actuator/sensor, that an actuator is directly driven by an actor, and that an actor reacts to information provided by a sensor. The outcome of the chain of actuators/sensors is a physical artefact. We note that either the actuator or the sensor functionality in an actuator/sensor pair may be void.

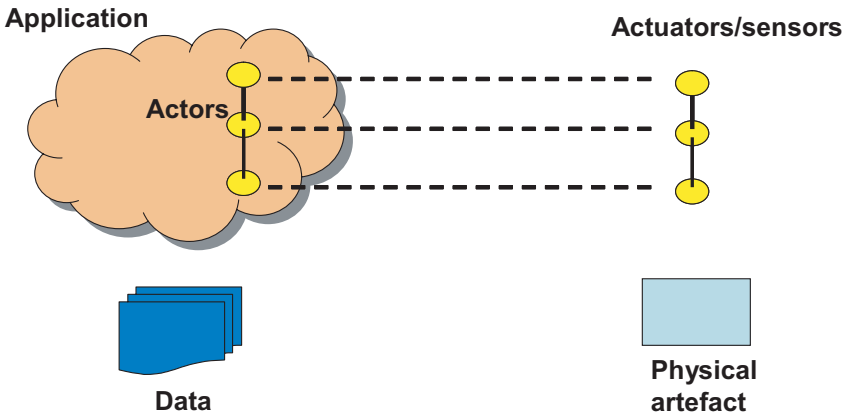


Figure 2.2: Mapping to the Physical World

Given this mapping assumption, the computer system's workflow mirrors a physical process in the physical world. The ultimate electronic data produced by the computer system is thus an electronic proxy for the physical world artefact. By querying the provenance of the electronic data, we can therefore obtain an accurate representation of the provenance of the physical artefact, due to the one to one mapping assumption. This requires some explicit actor state p-assertions to be recorded by actors in the computer application, which describe the activated actuators and the sensed data they return.

In practice, the one to one assumption may not necessarily hold, which means that the physical process may not directly be mirrored in the computer system. Specifically, we consider the case in which there may be actuators or sensors that are not directly driven by the computer systems, but by other entities (e.g., humans), not directly under the control of the computer application. In such circumstances, the provenance of the electronic data only helps us to derive a partial representation of the provenance of the physical artefact. Such limitation may be alleviated if an actor is capable of recording p-assertions about the part of the physical process that is not directly mirrored in the computer system, *as if* a one-to-one mapping existed. (We note that this also applies to any process where actors are not able to record documentation of process themselves.)

The discussion in this section has focused on physical artefacts. However, the principles just exposed remain applicable to other "things" in the real world, such as choices made by users, outcomes of a decision making process, or events observed by sensors or users. What the provenance system requires is either a user interface or sensor to act as an actor, recording p-assertions about the actions that occurred in the physical world, or another actor to relate such actions on behalf of the physical process that is not observed by sensors or users.

Consequently, we can now extend our definition of provenance to encompass the physical world.

Definition 2.9 (Provenance of an entity) *The provenance of an entity (whether computer based or in the physical world) at a given point in execution is the process that led to that entity at that point.* □

In the rest of the document, we continue to refer to the provenance of "data items" unless we specifically wish to refer to the provenance of physical world entities.

Additionally, we note that earlier we used the term *actor* to denote either a client or a service in a SOA. As the physical world is not so clearly describable in terms of clients and services, we broaden the definition of actor to mean any entity that acts.

2.7 The Nature of Queries

The purpose of a *provenance query* about a given data item is to identify a set of p-assertions that were submitted to the provenance store during some execution that resulted in the data item. The intent of such a query is that the selected p-assertions, which we refer to as the *query result*, provide a description of the process that led to the data, i.e., the provenance of the data, expressed at a level of abstraction that is suitable for the requester.

Hence, given a query, the purpose of a *query engine* is to decide which p-assertions belong to a query result. Several factors can be taken into account in order to decide if a p-assertion belongs to a query result. It is the purpose of the query to specify such factors. In the rest of the section, we discuss some of the factors that a provenance system needs to support.

Open systems may introduce an understanding of a process's scope that differs from the one in closed systems. Indeed, in a traditional batch system, the beginning of a process is marked by its submission to the batch system (or by its scheduling) and its end is defined by the termination of execution and deallocation of resources. While such a clearly defined beginning and end of process can still be achieved in a well-structured and controlled closed computation performed in an open environment, it no longer applies when previous results are opportunistically and serendipitously discovered and reused to produce some data. As an illustration, consider a process p_1 producing a result r_1, which is itself later discovered and used by a distinct process p_2 producing r_2. In this example, the end of process p_1 is marked by the production of result r_1, while process p_2 begins after the production and discovery of r_1 and terminates with result r_2. Another design could have conceived process p_3 producing a similar final result r'_2, where p_3 is the composition of p_1 and p_2. If we are not interested in temporal details, and the fact that intermediary result r_1 was stored and discovered, both results r_2 and r'_2 have similar provenance, but were produced by apparently different processes, p_2 and p_3, respectively. The reason for this difference is that p_3 is conceived as a closed experiment, producing r'_2, whereas p_2 opportunistically reused an existing result. There is no right or wrong interpretation in this example: whether p_2 or p_3 is the process of interest is to be decided at query time, by the querier.

Let us now assume that the provenance representation we discuss here is made available for all data or objects. Given that the state of our universe, including all electronic data, is derived from the "Big Bang", we do not expect provenance queries to return all p-assertions back to such a point. Hence, we need mechanisms to specify how far back in the execution we include p-assertions in the query result. Such mechanisms can be varied: we introduce them briefly here and discuss them later in Section 8.2. *(i)* A limit can be set on the length of the relationship chains, or on the number of successive processes. *(ii)* Relationship chains can be traversed until the data being transferred satisfies some property,

such as being of a given type. *(iii)* Given that actors can describe themselves by the functionality they perform on their inputs, functionalities of interest identify p-assertions that belong to the query result or that are to be rejected. *(iv)* Actors may record p-assertions describing their state and message they exchange; a query may identify that an actor should be seen as private, as if itself and the other actors it invoked did not record any p-assertions.

2.8 Conclusion

In this chapter, we have introduced our definition of provenance of a data item and how it can be extended to physical world entities. We have shown how provenance can be represented in a computer system, and have identified a provenance lifecycle composed of four phases: creating, recording, querying and managing. In the next chapter, we introduce an architecture that provides support for these four lifecycle phases.

Chapter 3

Logical Architecture

3.1 Architecture vs System

In the context of this document, a *provenance system* is defined as a computer system that deals with all issues pertaining to the recording, maintaining, visualising, reasoning and analysis of the documentation of 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.

The provenance lifecycle is composed of four phases concerned with creating, recording, querying and managing p-assertions. We now describe the roles of the actors involved in each phase of the lifecycle and then present a logical architecture that supports these actors in performing the activities of the lifecycle.

3.2 Role Definition

We can classify the actors involved in the provenance lifecycle according to their *role* in a provenance system. Briefly, the responsibilities of each role are as follows.

- An *application actor* is responsible for carrying out the application's business logic.
- A *provenance store* is responsible for making persistent, managing and providing controlled access to recorded p-assertions.
- An *asserting actor* is an actor that creates p-assertions about an execution.
- A *recording actor* is an actor that submits p-assertions to a provenance store for recording.
- A *querying actor* is an actor that issues provenance queries to a provenance store.

- A *managing actor* is an actor that interacts with the provenance store for management purposes.

3.3 Logical Architecture

In order to support capturing, recording, querying and managing the categories of p-assertions introduced in the previous chapter, we have specified a provenance architecture that takes into account a broad range of use cases [MGBM05, AV05b]. The architecture is summarised in Figure 3.1, which we discuss in the rest of this section. Central to the architecture is the notion of a *provenance store*, which is a service designed to store and maintain provenance representation beyond the lifetime of a Grid application. Such a service may encapsulate at its core the functionality of a physical database, but also provides additional functionality pertinent to the requirements of the provenance architecture. In particular, the provenance store's responsibility is to offer long-term persistence of p-assertions.

In a given application, one or more provenance stores may be used in order to act as storage for p-assertions: 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 *recording interface* that allows recording actors to submit p-assertions related to their interactions and internal states, for recording purposes. The recording interface supports the second phase of the provenance lifecycle (storing). 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 process documentation so as to support use cases. To this end, we introduce *query interfaces* that offer multiple levels of query capability; the query interfaces support the third phase of the provenance lifecycle (querying). Finally, since provenance stores need to be configured and managed, an appropriate *management interface* is introduced, which supports the fourth phase of the provenance lifecycle.

Some *actor-side libraries* facilitate the tasks of recording 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. We also expect actor-side libraries to provide some support to create common forms of p-assertions (the first phase of the provenance lifecycle). The creation of p-assertions here will need to take into account the expression of causal relationships in an appropriate manner, as previously discussed.

The interfaces and libraries shown in Figure 3.1 have different purposes: the interfaces specify the messages accepted and returned by provenance stores, and will be the focus of a standardisation proposal to ensure that applications, written in multiple programming languages, can inter-operate with different implemen-

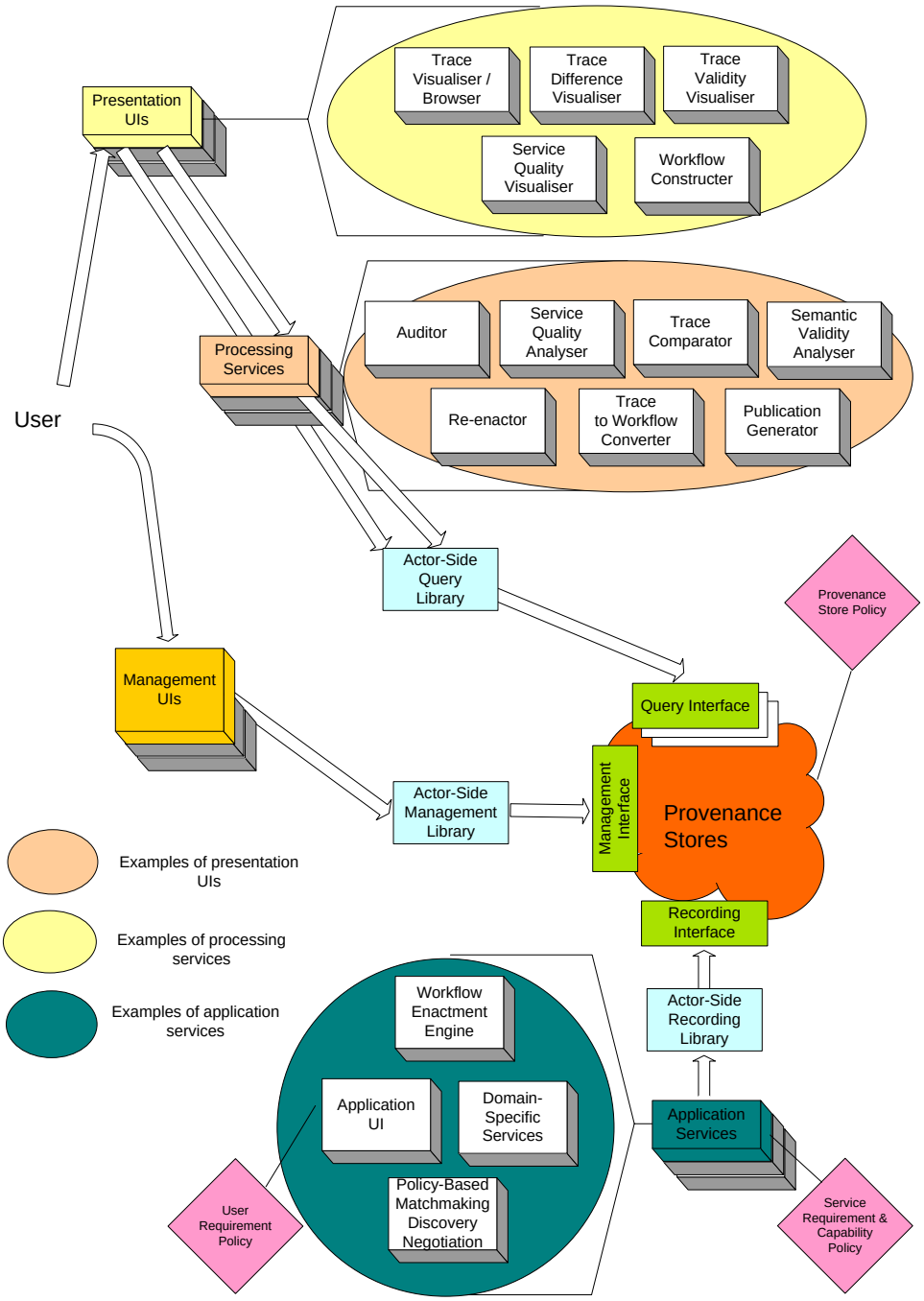


Figure 3.1: Provenance Logical Architecture

tations of provenance stores; the libraries are convenient mechanisms for offering bindings to the interfaces for specific programming languages. 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 generic middleware, such as *workflow enactment engines*, *registries* and *application user interfaces*. *(9)*

Once p-assertions have been recorded in a provenance store, process documentation can be used by *processing services* and *presentation user interfaces*. The former provide added-value to the query interfaces by further searching, analysing and reasoning over recorded p-assertions, whereas the latter essentially visualise query results and processing services' outputs. Figure 3.1 provides examples of such processing services and presentation UIs 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 user interfaces can, for instance, offer browsing facilities over provenance stores, visualise differences in different execution, illustrate execution from a more semantic viewpoint, visualise the performance of execution, and be used to construct provenance-based workflows. We note that such a list of processing services and presentation UIs is illustrative and not exhaustive; furthermore, it does not represent a commitment by the project to deliver these services specifically.

Another kind of user interface to the provenance store is also identified in the architecture. This is the *management user interface*, which allows users to manage the contents of the provenance store.

To be generic, a provenance architecture must be deployable in many different contexts and must 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 process that services may wish to advertise (such as their ability to provide some type of actor states p-assertions), and any requirements they have on other services they rely upon in order to perform this documenting (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.

3.4 The P-Header

In Section 3.2, we introduced the roles of actors involved in the provenance life-cycle and their general responsibilities. Roles place more specific obligations on actors with respect to supporting actors in other roles. Largely, this is a matter of providing adequate information in the correct format: for example, an asserting actor must create p-assertions in a format that a provenance store can make persistent and a provenance store must provide p-assertions in a format that querying actors can interpret. We specify how p-assertions and other data should be modelled to provide such consistency in Chapter 7.

In order for p-assertions to be created, asserting actors need to identify which process they are making an assertion about, which requires some *shared context* between asserting actors. The asserting actors receive the information, such as messages sent or received or actor states, from application actors, and therefore it is the application actors which need to share context information. We therefore place a further obligation on application actors to pass context information between each other regarding the process being executed. As this would often be achieved by putting the context information in the header of an application message (it could be exchanged by other, application-specific means), we call this information the *p-header*, defined as follows.

Definition 3.1 (p-header) *The p-header of an interaction is provenance-related context information, sent along with the interaction's message.* □

In practice, the p-header can contain an identifier for the interaction to which the context information applies and the locations of provenance stores where p-assertions documenting the same process are stored. Additionally, the p-header can contain a set of *tracers*, which are used to demarcate where one process starts and ends. A tracer is a token added to a p-header by an application actor, where the same tracer is added to the p-headers of all interactions in the same process by the same application actor. Additionally, where a tracer is included in the p-header of a message received by an application actor, that actor is obliged to copy the tracer into the p-header of all interactions within the same process. Using tracers, a querying actor can determine which interactions were part of a single process, because their p-headers will all contain the same tracer, and whether one process is contained within another, because the tracers of the former's interactions will be a subset of the tracers of the latter's interactions. The structure of p-headers and tracers is discussed in more detail in Chapter 7.

3.5 Conclusion

In this chapter, we have presented the logical architecture that underlies our provenance system and the roles of the actors that interact within that architecture. During the provenance lifecycle, the actors perform several roles: application actors execute processes; asserting actors create p-assertions about these processes; and recording actors record p-assertions in provenance stores, which allow querying actors to retrieve p-assertions and managing actors to maintain them. The recording, query and management functions of the provenance stores are made available through fixed, pre-specified interfaces, making it possible to program an application to take advantage of the architecture. Policies control the run-time behaviour of architectural components deployed in different contexts, and each of the roles places obligations on the actors playing them.

The remaining chapters of this document examine the issues that affect the fundamental parts of the architecture or that cut across a provenance system as a whole.

Chapter 4

Security Architecture

One of the key features for a provenance architecture within the context of this project is security. Many of the application domains in which a provenance architecture could potentially be deployed in have stringent requirements on access to data manipulated within the system. Correspondingly, p-assertions that incorporate or are derived from these data are likely to have similar security restrictions on them as well. Although security is a non-functional requirement, software engineering methodology strongly recommends that security considerations be integrated into the development life-cycle as early as possible. With this as a motivating factor, we proceed in this chapter to outline a security architecture for the logical architecture that we described in Chapter 3. In addition, the remaining chapters of this document will contain a security section (if relevant) that may make reference to the material presented in this chapter.

In Section 4.1, we briefly define some of the common security concepts that we use in this document. In Section 4.2, we survey the security issues relevant to the conception of provenance. Following that, we present the security architecture for the provenance store and describe the functionality and interaction between its constituent components in 4.3. In Section 4.4, we discuss the security issues pertaining to other components in the logical architecture. We then outline the security issues that remain unaddressed in Section 4.5, and conclude in Section 4.6.

4.1 Background

This section provides a brief narrative that encompasses some of the more common terminologies encountered in the field of electronic security. It is not intended to be a comprehensive treatise of the area, and merely seeks to provide a conceptual background for the security discussion in the remaining sections of this document.

We consider a system that offers some functionality through a set of resources

that can be accessed and manipulated. It is usually the case that these resources can only be accessible or manipulated in specific ways in order to ensure that the functionality offered by the entire system is unaffected. The *integrity* of a resource is a property of that resource that is preserved as long as the resource is accessed or manipulated in the prescribed manner. It is assumed that these restrictions on resource manipulation necessary to preserve its integrity are known to the entity responsible for the system resources, which we shall term as the *system administrator*. Hence for the trivial case where a system administrator accesses or manipulates a system resource, there is no risk of intentional resource integrity violation. The role of a system administrator would be roughly analogous to that of a managing actor within the context of the provenance architecture.

However, systems are generally useful only where their functionality (as provided by their internal resources) is accessible to external entities. In situations such as this, the system administrator may not have direct control over these external entities and cannot ensure that their behaviour is compliant with preservation of resource integrity. There is therefore the need to perform *access control* to these resources, and this is typically achieved by restricting access (out of the overall group of entities that are capable of accessing the system resources) to a specific group of entities that are trusted by the system administrator. We do not consider in our discussion the context of trust and how it is established in the first instance between the system administrator and a group of external entities.

A preliminary and necessary requirement for access control is *authentication*, which is the process of producing an *identity* based on some *credentials* submitted by the entity to the security infrastructure. An identity produced from a successful authentication process can subsequently be used in access control to ascertain whether an entity's accompanying request to access some resource in a specific manner is permitted or not. An entity that is allowed to access a given resource in a specific manner is said to be *authorised* to perform that access on the specific resource, and such an authorisation can be expressed in different ways. For example, in a *mandatory access control* system, entities are authorised to access resources on the basis of the relationship between different security labels or clearance levels assigned to the various resources and entities. In a *discretionary access control system*, authorisations are generally expressed in the form of a direct relationship between a given identity and the resources accessible to it.

The set of authorisations in a system is typically predetermined by the system administrator according to some existing *security policies*, and the scope of enforcement of this policy is generally known as a *security domain*. It should be noted that the identity produced from an authentication process is only meaningful to the system performing the authentication; it is entirely possible that a single entity may be represented in different systems with different internal system identities. Authentication and access control are often tightly interlinked components in a security architecture.

Situations may arise when a data resource (or a copy of it) has to be transported across an open medium, such as a network connection, where it is no longer protected by the security infrastructure of the system. *Privacy* is a property of this data that is achieved in this context by transforming the data into a form (typically via the use of *symmetric cryptographic mechanisms*) that is unintelligible to entities that were not originally authorised to access it. Integrity of this data is achieved in this context by ensuring that any processing or modification of the data while in transit becomes detectable. This generally involves the use of *asymmetric cryptographic* mechanisms such as *digital signatures*.

Signatures are generated by using the private portion of a *public/private key pair* to generate a message digest on a piece of data. Only the owner of the private key is capable of generating a digest on that data that can subsequently be verified successfully by any entity possessing the public portion of the key pair. This ensures that any modification of the data by any entity other than the owner would be detectable via an unsuccessful verification attempt. In addition, the uniqueness of the private key enables the establishment of a direct link between the key owner and a piece of data signed with that key. This is sometimes useful in attempting to guarantee the property of *non-repudiation*, which seeks to ensure that an individual is held accountable for an action in the system and cannot deny having undertaken this action post hoc. If such an action is expressible in the form of a data item, then a signature on this item undisputedly establishes corresponding responsibility for the action on the key owner.

Certificates are electronic documents used to link a public key with an identity of an entity possessing the corresponding private key. The reliability of this link is established by a signature on the certificate by a party trusted by all entities that use the certificates. This trusted party is usually a *certificate authority (CA)*, who is the primary entity responsible for the life cycle management of these certificates within a *Public Key Infrastructure (PKI)*.

Interactions across different security domains can sometimes occur, particularly in large scale, distributed systems exemplified by the Grid or the Web Services environment. For example, a workflow initiated by an individual may interact with resources from several systems, each with separately administered access control schemes. Here, the individual would need to authenticate to the relevant security components of each of these systems, since the individual would very likely have distinct internal identities in the different security domains. *Federation of identity* is a method which seeks to simplify the security procedure, and hence the overall workflow process, by requiring the individual to authenticate only once (usually known as *single sign-on*) in order to access resources across several security domains. In order to accomplish this while still retaining the original level of security, the infrastructure of each of these security domains needs to be structured to communicate relevant information, particularly pertaining to actor identities, between themselves.

Another requirement that arises within a distributed environment is the need

for *delegation of access control rights*. For example, during the process of workflow execution by an enactment engine, a service invoked by the workflow engine might need to invoke another service in order to fulfil the requested functionality. If these services exist in different security domains, then the individual responsible for initiating the workflow would need to authenticate twice: once to each of them. Once again, a single sign-on capability can be provided if a mechanism is implemented in the security infrastructure that empowers the first service to invoke the second service based on the access control rights transferred to it from the individual concerned. Note here that while the conception of single sign-on is the same as is the case in identity federation, the motivating situations are slightly different. Delegation of access control generally also carries the implication that the delegated access rights are only qualified within a certain context: for example, during the duration of a workflow or to access specific resources only. There must be a way to ensure that a service that has been delegated some rights from an individual does not maintain the ability to use these rights indefinitely outside of the given context, nor to delegate it further onwards to other entities unless permitted to do so.

It needs to be borne in mind that delegation of access control and federation of identity are not novel security methodologies nor do they enhance the security capabilities of a system. They merely provide a way to maintain the existing level of security in individual security domains while attempting to simplify the security requirements that arise when complex interactions between these different domains occur.

4.2 Provenance Related Security Issues

In this section, we outline the security issues that we believe are relevant pertaining to our notion of provenance. We note however that not all of these issues are relevant in the context of the software requirements (see Chapter 10), and the eventual security architecture will only address those that are.

1. *Access control to the provenance store.* This is the primary security issue as the provenance store is considered to be central to the logical architecture. While the access control mechanisms utilised are situated in the context of the specific requirements of the project, this notion of security here is conceptually identical to the general case of securing a database with multiple users.

2. *Integrity and non-repudiation of p-assertions.* Recording actors store p-assertions created by asserting actors in the provenance store. In the event that the asserting actor is not the recording actor, there is a need to ensure that information within the p-assertion is not altered unintentionally or maliciously by either the recording actor or provenance store. This can

be achieved by having the asserting actor sign the p-assertion it creates. The signature also serves the additional purpose of ensuring that the asserting actor cannot deny responsibility for the creation of the p-assertion in question. This can be necessary when legal or other requirements mandate establishment of liability for the consequences arising from utilizing the information in a p-assertion. This issue is discussed further in Section 7.8.

3. *Ascertaining asserter identity in a p-assertion.* The structure for holding p-assertions created by an asserting actor will also hold the identity of this actor (Figure 7.9). By implication of the previous point we discussed, the asserter identity should correlate with the identity associated with the signature on the p-assertion, since only the asserting actor should sign the p-assertion. A check can be done to ascertain whether this is true, and can be undertaken by either the provenance store to which the p-assertion is recorded to, or by the querying actor retrieving the p-assertion in question.

4. *Derivation of authorisation information relating to p-assertions.* It is likely that p-assertions will contain or be derived in some fashion from an existing piece of data in the system. For example, an application actor with access to a database may send a message containing an item from that database to another actor. This item is likely to have certain access control restrictions enforced upon it within the security domain of the database in question. When a p-assertion is created for the transmitted message and recorded to the provenance store, appropriate access control restrictions (or authorisations) must now be established for this new entry to ensure that any future access to it is in accordance with the security policies of the provenance store.

   In some situations, it may be useful to relate the authorisation for the newly recorded p-assertion in some way to the access control restrictions on the original database item that the p-assertion is based upon. This effectively allows for a more flexible specification of authorisations on p-assertions by taking into account information other than that found in statically predefined security policies on the provenance store. A possible approach towards this end is for the recording actor to submit additional information along with the p-assertion to be stored. This additional information would be created by the asserting actor and can then be utilised in an automated manner by the provenance store to generate appropriate authorisations for the new p-assertion.

5. *Context-based authorisation specifications.* As we have seen in Chapter 3, processing services provide added-value to the query interfaces by further searching, analysing and reasoning over recorded p-assertions. Some of the

operations that can be performed by a processing service have a well defined functionality; for example, comparing processes used to produce several data items. In order to perform this operation, a certain set of p-assertions identified by certain criteria will need to be retrieved from the provenance store. Another operation, for example, verifying that a given execution was semantically valid, will require the retrieval of another set of different p-assertions. Situations may arise where it is useful to ensure that certain actors are authorised to access only the relevant p-assertion subset necessary for a specific operation (or more generally, any type of context in which provenance representations can be used in). This would require an ability to express authorisations at this level, as well as some way to translate these context-based authorisations into finer grained authorisations at the p-assertion level.

4.3 Provenance Store Security Architecture

In this section, we present the logical design for a security architecture for the provenance store, having identified it as being the central component in the provenance architecture. Security issues pertaining to the other components in the logical architecture are addressed in the following section. An overview of this architecture is illustrated in Fig. 4.1; components enclosed in ovals indicate that they potentially (although not necessarily) exist in security domains separate from the domain of the provenance store. We first describe the functionality of each of these components and then proceed to outline the possible interactions between them. Finally, we discuss some of the broader security issues that are not considered in this architecture.

4.3.1 Components of Security Architecture

The provenance store in the logical architecture exposes three different interfaces (recording, management, query) for different purposes. All of these interfaces can have optional operations that access the required security functionality in the actor side libraries. The *identity validator* accepts all incoming requests and accompanying credentials (such as certificates) over a secure link supporting either transport or message level encryption. It is also important that the validator and actor interacting with it mutually authenticate each other during this secure transmission. This is necessary from the viewpoint of the provenance store as the identity of the actor is the first step towards enforcing appropriate access control. However, it is equally relevant as well for the actor who needs to circumvent potential impersonations of a valid provenance store by malicious parties that would then gain unauthorised access to the p-assertion.

The identity validator then performs four functions:

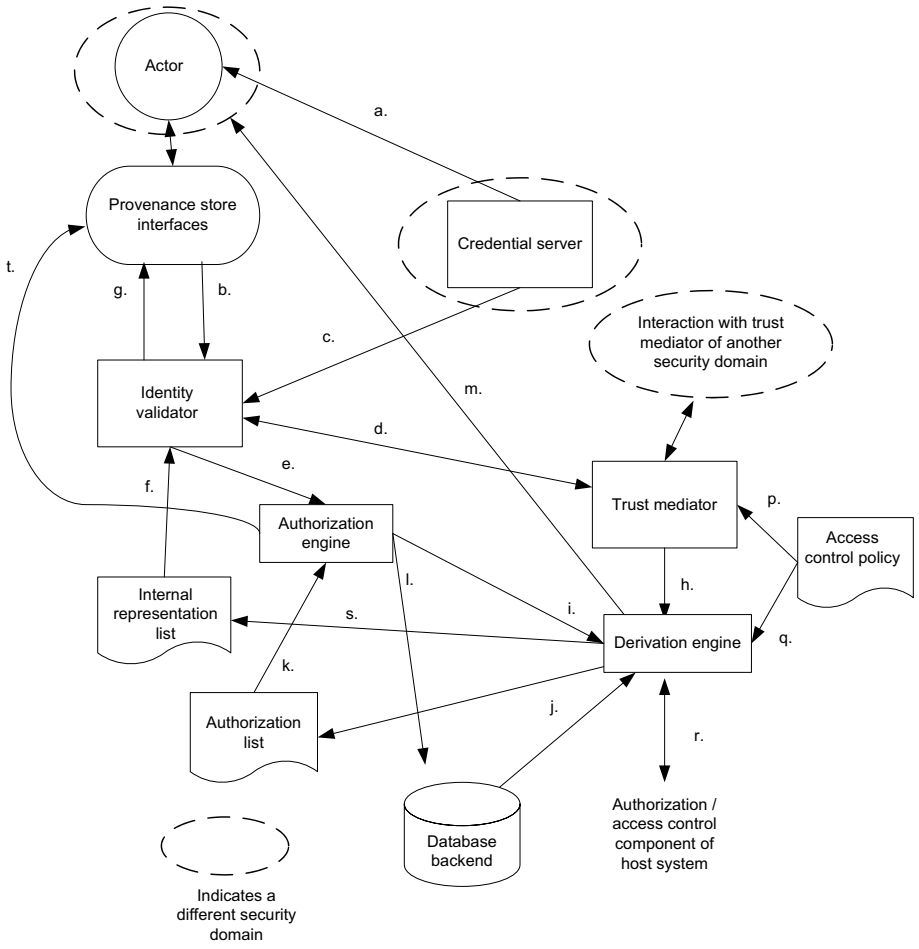


Figure 4.1: Provenance store security architecture

1. Verifies that the submitted credentials are valid within the context of the domain. This may require interaction with the *trust mediator* in the event where federated identity validation is required. It also needs to take into account that the submitted credentials may imply some form of delegation.

2. Maps these credentials to an internal representation (IR). This could assume a combination of various forms (an identity, a role, a list of attributes, a list of privileges, etc). This should include basic role information to support an RBAC implementation. A common way of doing this is to map the identity to a role which has a predefined set of authorisations or privileges.

3. Ensures that the asserter identity on the submitted p-assertion tallies with the identity associated with the signature, if any, on the p-assertion. This operation is optional, and correlates with the third security issue in Section 4.2. If the asserter identity is to be utilized in the access control decision, then it needs to be mapped to a corresponding IR as well.

4. Formats the request into an appropriate representation for access control purposes.

The first two functions are performed with help from an *internal representation list* that specifies the appropriate mapping relationships, including roles.

The *credential server* fulfils the role of being a trusted third party holding identity-related information for all potential users of the provenance system within a given security domain, as well as providing them with suitable credentials and other related security tokens for authentication purposes. The *authorisation engine* essentially performs the access control functionality in two main ways based on the authorisations specified in the *authorisation policy* and the IR produced from the identity validator:

- The request is granted or denied solely on the basis of the information from the authorisation policy and the IR related to the identity of the requesting actor. It is also possible that the IR of the asserting actor is taken into account in the access control decision as well; we assume that this possibility exists, but use the term IR to refer to the IR of the recording actor for the sake of brevity in the remaining discussion. If granted, the requested operation is performed and the appropriate acknowledgment or data item is returned directly to the requestor without further intervention from the authorisation engine.

- The granting of the request may additionally be dependent on information contained within the data item that the request is related to (such a condition would be specified accordingly in the authorisation policy). For example, a read operation associated with an IR on a given p-assertion

might be permitted only if the p-assertion contained relevant information pertaining to that IR. In this case, the p-assertion in question would have to be retrieved first and assessed accordingly by the authorisation engine before a final decision can be made on granting or denying the request.

Depending on the nature of the authorisation engine, it may be necessary that the assignment of a role to an IR for the case of a RBAC should be achieved by the authorisation engine instead of the identity validator. In addition, it is possible to employ either one or both of these two approaches to specifying authorisation:

- an identity / role is assumed to have no authorisations in the initial case, and explicit authorisations have to be granted;
- an identity / role is assumed to have complete authorisation in the initial case, and explicit restrictions have to be placed.

The *access control policy* is a higher level security policy that specifies the ways in which the authorisation policy and/or internal representation list can be modified by the components which access them. Both the access control policy and authorisation policy could be subsumed under the broad umbrella of provenance store policy in the logical architecture. The *database backend* provides actual physical storage for the p-assertions. The *trust mediator* is an optional component of the architecture that is required only if federated identity management is to be supported. It provides the interface to other security domains, and is the component through which security assertions are exchanged about local internal representations and authorisations. The *derivation engine* provides the following functionality:

1. Derives new authorisations from existing authorisation information. For the case of the third security issue for provenance as identified in Section 4.2, the authorisation information would originate with the p-assertion as part of submitted request from the actor. Alternatively, there may be a need to correlate the authorisations as specified in the access control policy with the authorisations in the host systems security architecture, in the event that a tighter integration between both architectures is required.
2. Creates a set of appropriate authorisations corresponding to a higher level context-based authorisation specification. This corresponds to the fourth security issue for provenance, and can be considered to be optional functionality.

4.3.2 Interaction Between Components

We illustrate the interaction between these components using some simple scenarios in a technology independent manner. The flow of information are denoted

by labelled arrows in Fig. 4.1 and our description makes reference to them accordingly in brackets.

Scenario 1: Submission of a p-assertion to be stored by a recording actor

1. The p-assertion along with other relevant information is submitted as a an invocation message (b.) in accordance to the schema of the recording interface. The submission link is secured using appropriate client-side library functionality. The entire message could be signed, or some of its contents could be signed (for example, signing the p-assertion for non-repudiation). The required credentials can be obtained from the credential server prior to the invocation process. (a.).

2. The identity validator intercepts the message and verifies the signature if there is one; this may involve another interaction with the credential server (c.). An attempt is made to resolve the supplied credential information with the internal representation list (f.) If the credentials cannot be immediately resolved but there is additional information to indicate the domain in which it might be recognised, an appropriate request is sent off to the trust mediator (d.).

3. The mediator interacts with its counterpart in the relevant domain using appropriate assertions as part of a security protocol, and then formulates a new security assertion based on the access control policy (p.). This assertion is then sent off to the derivation engine (h.). The recognition of the new IR is achieved by appropriate additions to the internal representation list (s.) and the authorisation list (j.) by the derivation engine, based on the access control policy (q.)

4. If the credentials fail to resolve successfully either in the current domain or the domain that the mediator attempts to interact with, an appropriate security exception is returned via a fault (g.). Otherwise, the validator converts the store request into an appropriate format and sends it off to the authorisation engine (e) along with the role and accompanying security attributes.

5. The authorisation engine first needs to ascertain whether the store request is valid for the specified role based on the authorisations specified in the access control policy (k.). If it is not, an appropriate security exception is again returned via a fault (t.). Otherwise, new authorisation information for the p-assertion to be stored needs to be determined. The authorisation engine formulates an appropriate statement which is then sent off to the derivation engine (i.). For the case of the third security issue mentioned in Section 4.2, the submitted p-assertion will also contain accompanying

authorisation information; this will be also be taken into account in the formulated statement of the authorisation engine.

6. The derivation engine subsequently creates new authorisation information from the formulated statement based on the rules prescribed in the access control policy (q). For purposes of maximising performance, this new authorisation information could have been created by the recording actor doing the submission so that the derivation engine uses it directly without any further processing. The new authorisation for the p-assertion to be stored is added to the authorisation list (j). The p-assertion is now sent onwards to the storage pre-processor, along with relevant authorisation information or metadata that it is meant to be stored with (l.). Here, the p-assertion may be encrypted or signed using the private key of the provenance store domain (in the case that the database backend is in a different domain) and then formatted to correspond to the recording interface schema of the backend.

7. The p-assertion is then sent over a secure, mutually authenticated link to the database backend in an analogous manner to the way that the original submission by the recording actor to the provenance store was accomplished (o). The database backend, if hosted by a third party provider, could be exposed for remote access in a variety of ways with corresponding authentication and access control mechanisms; the storage pre-processor will need to be aware of these possibilities and cater to them accordingly by acquiring and generating the relevant security credentials. The acknowledgement sent back from the database backend (n.) is processed accordingly and sent back to the recording actor (m.)

8. A session connection can be established on a secure link so that future submissions no longer require the submission of authentication credentials that need to be verified. This requires that the implementation support such a secure persistent session connection.

Scenario 2 : Retrieval of a p-assertion by a querying actor

The sequence of interactions is nearly identical to that for the case of storage. The primary difference arises from the fact that there is no need for the derivation engine to generate new authorisation information as there is no new p-assertion to be stored. However, when the requested p-assertion is returned (m.), further transformations may be performed on it, in accordance to the initial authorization information associated with the request as well as any additional authorization information stored and associated with the p-assertion itself. The transformations which are undertaken by the derivation engine, may take the form of filtering out portions of the p-assertion or transforming the information in the p-assertion in

some specific manner.

Scenario 3: Management of the provenance store by a managing actor

Management operations on the stored p-assertions are achieved in an identical manner to that for scenario 1 and 2. Submission of a management operation request is treated like the submission of a p-assertion to be stored, with the difference that the management request is not stored but rather processed by the derivation engine and the appropriate functionality then enacted. This may require retrieval of p-assertions, if so, these are then returned to the managing actor in a similar manner to that in Scenario 2. There may also be modifications of internal representation list/authorisation list which may include deletion, modification and addition of entries. All of these operations are consequent on the identity validator first recognizing that the authenticated managing actor has the role or capability to perform these management type activities.

Scenario 4 : Integrating authorisations of the provenance store and the host system

This can be accomplished by providing a link / interface between the access control/authorisation components of the host system and the derivation engine. If the provenance store architecture is tightly integrated with its host system, this link may not need to be secured as all communications between the architectures are internal within the operating system, rather than through an exposed network medium. Changes that need to be made to the authorisation list / internal representation list can then be propagated through the derivation engine (r).

An implementation of the provenance architecture may require distributed provenance stores for reasons such as scalability, as will be discussed in Chapter 5. In such an instance, p-assertions related to a specific workflow or sequence of execution may be stored in multiple provenance stores by the responsible recording actors. Consequently, a query to retrieve a group of related p-assertions may potentially require a series of queries to the various provenance stores holding the desired p-assertions. We discuss the security implications of this requirement in 5.4.

4.4 Security in Other Architecture Components

In the previous subsection, we presented and described the functioning of a security architecture to protect the provenance store, a key component of the logical architecture. Here, we study the security considerations underlying interactions involving other components of the logical architecture.

4.4.1 Between other components and the provenance store

The other components in the logical architecture that interact directly with the provenance store will now require corresponding security functionality as well in order to ensure their interactions are secured properly. We describe the nature of the required functionality below for application services, management UIs and processing services.

1. A facility is required for accessing credentials that are to be submitted to the identity validator in the provenance store. This can be provided as additional libraries in the corresponding actor side libraries or as interfaces that permit interoperation with external third party applications that provide credential generating functionality. A straightforward example would be a keystore manager application that generates, archives keys and certificates and obtains approval for these certificates from a CA.

2. If a keystore or some other facility for storing cryptographically generated material is to be used by the client side libraries, it has to be secured appropriately (e.g. located in a secure account, encrypted and contents accessible only by the provision of a username/password combination).

3. A facility is required for accessing specific security mechanisms such as signing or time stamping. This is necessary, for example, when the asserting actor needs to sign the p-assertion it created (see related security issue 2).

4. For the case where authorisation information is desired to be submitted alongside p-assertions, an interface must be provided as part of the domain specific services that allows the retrieval of this information from the appropriate locations (such as a local database). This interface should be congruent with the specific format in which the authorisation information can be expressed in.

4.4.2 Intermediate components

By intermediate components, we refer to components that are not directly accessible by the user. Such components may themselves be invoked or accessed by other components rather than by the user, and may interact directly with the provenance store. For example, a user may use a presentation UI to access a presentation service which in turn accesses the provenance store. In the application domain, a user may access an application UI that in turn invokes a chain of other application services before a final invocation is made to the provenance store. In such cases, the intermediate component may require authentication of incoming requests to it. It is possible to reuse the security architecture developed for the provenance store for this particular component as well. The primary differences would be, with reference to Fig. 4.1, are:

1. As the incoming request is to the intermediate component, it is unlikely to be a p-assertion, rather a generic data item (which may contain a p-assertion) submitted in accordance with the schema of the interface to this intermediate component.

2. The derivation engine will not be used to create new authorization information as the submitted data item is not intended to be stored. However it may be used in performing some security-related functionality on the data item, for example encrypting or filtering out a certain portion of it. This will be accomplished in conjunction with security policy dictating the operation of this intermediate component.

3. Once the request is approved by the authorization engine, it is sent off (l) to some internal function of the intermediate component for further processing, rather than to a database backend (as is the case for the provenance store). Once this processing is complete, a result is returned to the invoking actor (m) and / or a further invocation is made to another component.

4.4.3 Delegation of identity or access control

The need to delegate access control may arise if the intermediate component described previously exists in a separate security domain from both the user and the provenance store. Consider again the logical architecture in Fig. 4.1 and assume that a user is performing a query on the provenance store through the presentation UI and a processing service. Assume now three separate security domains: one containing the user and the presentation UI, another the processing service, and the third encapsulating the provenance store.

When the presentation UI under the users control sends a request to the presentation service, an appropriate credential is submitted by the user for purposes of authentication. If the request is authorized, the presentation service will then decide the type and number of provenance store queries that need to be made in order to satisfy the request. When making these queries, the presentation service needs to present suitable authentication credentials to the provenance store. There are essentially two ways to proceed here:

- Authenticate to the provenance store using the credentials of the presentation service, whereupon subsequent authorization decisions will be based on the identity or associated role of the presentation service. This approach requires the presentation service to be trusted and known to the provenance store security administrators, and that it has the appropriate authorization to access a wide enough pool of p-assertions to satisfy requests from all potential users (or at least users that are known within the security domain of the presentation service).

- Authenticate to the provenance store on behalf of the original user. This approach requires that a form of delegated identity or access control credential be created by the presentation service, possibly in negotiation with the presentation UI. The identity validator of the provenance store must then be able to recognize and process this delegated credential accordingly, and infer the identity or associated role of the original user. Subsequent authorization decisions are then on the basis of the users identity, and may also need to take into account additional constraints specified in the delegated credential itself.

The first approach is suitable if all potential users making queries can ever only do so through the medium of a presentation service. Here, the responsibility of checking authorizations for the actual users is effectively offloaded from the provenance store to the various presentation services in the system. If the number of presentation services known within the provenance store security domain is significantly smaller than the potential number of users, then the overhead of authorization is equivalently reduced as there is now only a need to check on these presentation services.

There are some drawbacks however with this approach however. Firstly, authorization lists are likely to be duplicated between many presentation services, as it is unlikely that authorization for a specific user will differ between different services. Accordingly, changes or additions to these authorization lists must then also be propagated between the different copies on all services. Lastly, application services storing p-assertions through the recording interface must now provide authorization information pertaining to presentation services rather than specific users. This may necessitate additional overhead in communication between application services and presentation services.

The second approach therefore appears to be a more feasible one. There will however be an overhead associated with communication between the presentation UI and the presentation service in order to create an appropriate delegation credential. Depending on the delegation act itself, there may be a need also for further communication between the security architecture of the provenance store and the user / presentation UI during the authentication or authorization process in the security architecture of the provenance store. This might happen, for example, when delegating access control is expressed through the modification of the authorization list in the provenance store to reflect the delegation of authorizations between the security domains of the user and the provenance store.

Even when credential delegation is used, the presentation service may also have an installed security policy that dictates the nature of the results to be returned to the user / presentation UI. For example, assume that a request from the user to the presentation service results in several corresponding query requests being sent in turn to the provenance store along with a delegated credential.

P-assertions pertaining to the authorization associated with this credential are then returned to the presentation service. At this point, the security policy of the presentation service as pertaining to the user in question may dictate further processing of the results (such as transforming or filtering it in some way) before finally returning it to the user. In this case, filtering or transforming of the returned results based on authorization considerations happens at two stages: once at the provenance store, and then subsequently at the presentation service. In both stages, it is performed by the derivation engine of the respective security architecture. There may also be need to communicate between the authorization engines of both the presentation service and provenance store via their respective trust mediators, if complex authorization decisions are to be affected.

The description in this subsection is equally applicable to intermediate components in other places in the logical architecture, for example with an application service that is located in a different security domain from the actual application service that makes the final submission of p-assertions to the provenance store. Similarly, delegation of identity or access control can also occur multiple times if there is an invocation of a chain of application services (such as that might occur in a workflow), with all these services located in different security domains. In cases like this, it is necessary to ensure that the delegation mechanism being used (for example, proxy certificates) can support multiple acts of delegation.

4.5 Additional security issues

While this chapter discusses security considerations for all components of the logical architecture, the primary focus is on the security architecture for the provenance store as we have established it as the core component in the logical architecture. The construction and implementation of this architecture will therefore take precedence over security considerations for other components. In particular, if the provenance system is to be integrated into an independent application domain of which the developers of the provenance system have no control over, then it is assumed that some, if not all, of the security issues relating to the application services have already been addressed adequately. Such issues include the need for delegation of access control, which was already discussed at the end of the previous section. In this section, we describe a few more of these types of security issues, which also do not completely come under the purview of the security work to be achieved for the provenance architecture.

1. Mutual authentication and secure transport of p-assertions between two application actors. Both activities have to be handled or negotiated between the two actors involved in the production of interaction p-assertions.

2. Anonymisation of data. Some medical-based applications require the exchange of patient-related information during the interaction of services.

Legal restrictions mandate that data of this nature is anonymised (patient identity is removed) and depersonalised (i.e. the identity of the patient cannot be traced based on other information in the record). Again, this requirement remains outside the context of the security architecture.

3. Support for multiple authentication schemes. To enhance security in some application scenarios, authentication requires a combination of security credentials in order to be successful. The identity validator of the component in question must then be able to support the use of multiple security credentials.

4. Establishing the access control polices for a RBAC system. Authorisation in this system is very much policy-driven; specifying the nature of these policies within the particular context of RBAC is vital to the correct and efficient functioning of the architecture. Policy related issues are addressed in Chapter 8.

5. Long term storage of provenance information. If a third party database provider is used, then provenance information may need to be encrypted or signed by the storage pre-processor prior to sending it off for storage. In the event that this provenance is intended to be stored for a relatively long period (e.g. 100 years), a situation likely to arise is one where the original cryptographic keys and / or algorithms become outdated or expire. Such issues must be catered for in some way, for example, by having a key archival facility and re-signing / re-encrypting provenance information periodically over the intended storage duration.

6. Expiry of certificates. For workflows that run over a relatively long period, it is possible that certificates could expire in the middle of a workflow run. If an actor uses a certificate as part of the authentication process to the provenance store, then expiry of this certificate would mean that submission invocations that were once accepted within the context of this workflow have now become invalid. To avoid situations like this, proper management of certificates and keys at the actor end is called for (i.e. workflow duration is estimated against certificate life time prior to commencing a workflow). Alternatively, the provenance store security policy could be articulated appropriately to avoid this situation. For example, the authorisation component could keep track of all invocations from a given actor within the context of a specific activity and allow remaining invocations to proceed in that activity as long as the initial invocations were signed with a valid certificate.

4.6 Conclusion

In this chapter, we discussed security issues that were relevant in the context of provenance. The security architecture for the provenance store is then presented along with an explanation of the functionality of its constituent components. This is followed by an illustration of the interaction of these various components in for some standard interactions with the provenance store. We then discuss security issues pertaining to other components of the architecture. Finally, we outline some of the security issues that we do not address or are out of scope of the proposed security architecture. In the following chapters, we will again discuss security issues (where relevant) with appropriate references to this chapter .

Chapter 5

Distribution Architecture

After introducing a logical architecture for provenance systems in Chapter 3, this chapter now discusses distribution in the architecture. First, it presents a set of recording patterns that identify communications between key architecture roles; second, it explains how the data organisation adopted by the provenance store allows for data that is geographically distributed; finally, it describes how deployments of core architecture components can cater for high load.

5.1 Recording Patterns

To be able to cope with the documentation of a single execution, provenance stores may have to be distributed since there can be a large quantity of data, in a large amount of assertions, recorded by a high number of actors deployed in many organisations, each with their own security domain, privacy requirements, etc. The requirement for recording process documentation in distributed provenance stores, such that all documentation related to an execution can be retrieved again, presents a developer with several deployment problems. Therefore, one aim of the distribution architecture is to present a set of deployment patterns that address these problems.

A pattern-based approach began with Alexander in the field of architecture [AIS77, Ale79] and was latter promoted as useful for software systems by Gamma *et al.* [GHJV95]. According to Alexander [AIS77], a *pattern* describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that one can use this solution a million times over, without ever doing it the same way twice.

A pattern then describes a solution to a common design problem; the solution described must strike a balance between being concrete enough to be applicable and abstract enough so that it can be applied to a range of similar problem situations. Patterns allow us to present a solution that any developer can use to integrate p-assertion recording into their application. We now describe each

pattern for p-assertion recording in turn. The adopted format is as follows:

Name A short name of the pattern that reflects the solution.

Diagram A diagram that shows the pattern visually. Diagrams have a common visual appearance. Provenance Stores are labeled and denoted by a 3D cylinder. Actors are denoted by boxes. A single message exchange is denoted by a line with an arrow head. The arrow denotes the direction of the message flow. Dotted lines follow the same convention but denote multiple message exchanges.

Context The situation in which the pattern applies and why this pattern exists.

Problem Describes the problem that the pattern solves providing more detail as to when the pattern should be applied.

Solution A description of how to apply the pattern including the interactions between actors and any properties an actor is expected to have in order to function in the pattern.

5.1.1 Separate Store

Name SeparateStore.

Diagram See Figure 5.1.

Context Application actors want to make available information about their interactions and associated state. This pattern exists because querying actors want to know how application actors have interacted in the past in order to produce a piece of data. To know how application actors performed, these application actors must make available information about their actions.

Problem An application actor, A, may be involved in a large number of interactions over its lifetime and cannot retain all the process documentation itself. Likewise, querying actors would like to access information about A's previous message exchanges and states, even when A is not available. For example, A may have been shutdown, moved or be under repair.

Solution A separately deployed store is introduced to retain information about an application actor's interactions and states, which we referred to as *provenance store* in Chapter 2. An actor records p-assertions into a provenance store so that it does not have to retain this information itself. A provenance store should have the following properties:

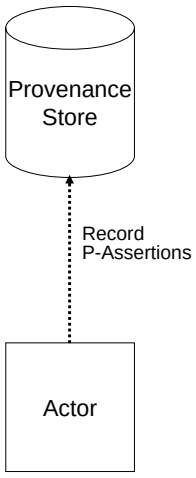


Figure 5.1: SeparateStore Pattern Diagram

1. It should be available in a long term manner in comparison to the application actors that submit p-assertions to it. This property allows p-assertions recorded by an application actor to be accessed after the application actor has become unavailable.

2. It should provide a well defined interface for the recording of p-assertions by an application actor.

3. It should provide a query mechanism to retrieve p-assertions, which makes the p-assertions available to querying actors.

4. It should provide a management mechanism to manage the stored p-assertions.

5.1.2 Context Passing

Name ContextPassing.

Diagram See Figure 5.2.

Context Two application actors, A and B, exchange a message. A and B record p-assertions about this interaction in two provenance stores (see the pattern SeparateStore). Both actors record these interaction p-assertions because they want their view of that interaction to be documented. This allows other actors to determine if A's and B's views of the interaction concur. Likewise, A

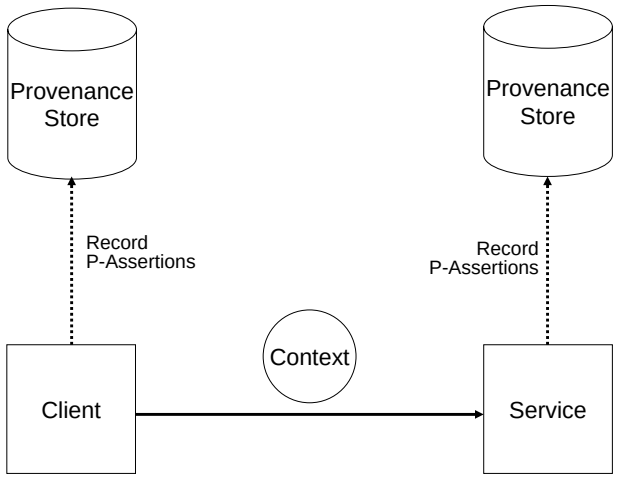


Figure 5.2: ContextPassing Pattern Diagram

and B may want to record actor state p-assertions and relationship p-assertions within the context of the interaction.

Problem The p-assertions that A and B record need to be identified as being the documentation for the same interaction. Otherwise, the actors' views of the interaction cannot be associated with one another; it then becomes difficult to determine if the recorded p-assertions are documenting the same interaction.

Solution The client actor in the interaction must generate the appropriate identifiers (IDs) to identify the interaction. It must then pass a *context* containing those IDs to the service actor. Both actors use these IDs to record their p-assertions in their respective provenance stores. The p-assertions for the interaction can be matched by the IDs generated by the client actor. A method of passing this context is by attaching it to the application message exchanged by the client and service actors. Application actors may use any other appropriate method to pass such context information. Beyond passing IDs, application actors may use a context to pass other information relevant to provenance. The use of ContextPassing is core to the architecture as discussed in Section 3.4; Chapter 9 discusses further the expected behaviour of actors regarding such contexts. Identifiers and contexts are also discussed in Sections 7.1 and 7.3.

5.1.3 Shared Store

Name SharedStore.

Diagram See Figure 5.3.

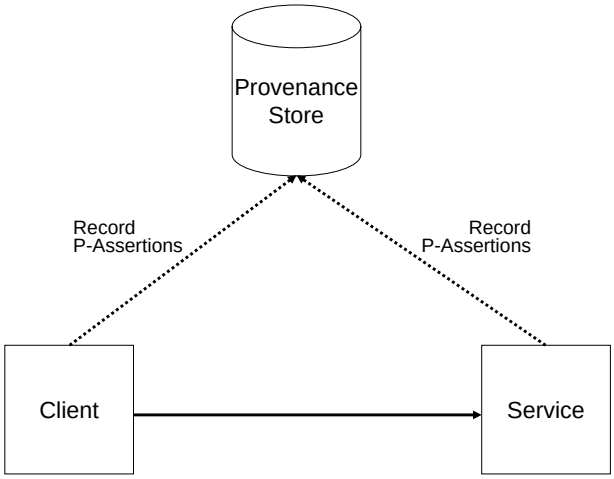


Figure 5.3: SharedStore Pattern Diagram

Context Actors record p-assertions into provenance stores following the SeparateStore and ContextPassing patterns.

Problem The SeparateStore and ContextPassing patterns may lead developers to believe that for every application actor there is a corresponding store. However, developers may not want to deploy a provenance store for every application actor, especially when the number of application actors is large. Also, in order to retrieve the provenance of a result each provenance store must be contacted resulting in slower query performance.

Solution Application actors are allowed to record p-assertions in a shared provenance store.

The SharedStore pattern clarifies the way in which SeparateStore and ContextPassing can be applied. Both SeparateStore and ContextPassing are agnostic as to what provenance store an actor may use to record its p-assertions. The pattern emphasizes that actors can record their p-assertions in any store they choose

and provenance stores may hold p-assertions from multiple actors. SharedStore does not prescribe how many stores there should be and which provenance stores should be shared. This is left to the developer applying the pattern. SharedStore allows developers to determine the distribution of provenance stores that fits their application.

5.1.4 Pattern Application

The patterns that we have introduced show how p-assertions can be recorded in provenance stores by actors. The documentation of process can be recorded for an entire system by applying a selection of these patterns to every actor and every interaction in a system. Hence, a given actor may use different provenance stores for recording p-assertions pertaining to different interactions. For example, Figure 5.4 shows a system with a client initiator, a workflow enactor and a service all recording p-assertions about their request and response interactions. SeparateStore, ContextPassing and SharedStore have all been applied multiple times in this case.

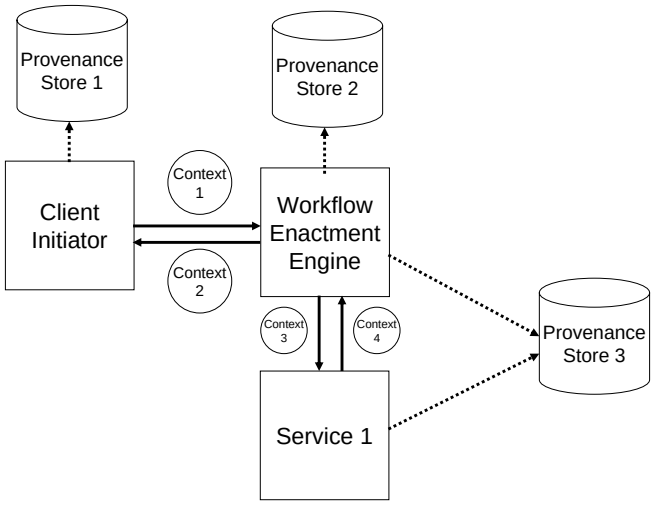


Figure 5.4: A system in which SeparateStore, ContextPassing and SharedStore have been applied multiple times

These recording patterns allow for the flexible deployment of provenance stores to aid scalability. The patterns can be applied to any number of interacting actors using any number of provenance stores in order to record p-assertions; the distributed architecture however does not mandate the number of provenance

stores that must used in a given application, nor the way they must be shared. This is left to the application designer to make those decisions.

5.2 Linking

By the definition of the p-assertion recording patterns, an actor is allowed to record its p-assertions in any provenance store. This means that the documentation of process that led to a result can exist across any number of provenance stores. There are several benefits in allowing documentation to be recorded across multiple stores: the elimination of a central point of failure, the spreading of demand across multiple services and the ability for provenance stores to exist in different network areas (for example, one provenance store may be behind a firewall whereas another is not). In general, allowing p-assertions to be recorded across multiple stores increases the flexibility and scalability of systems recording p-assertions. This scalability and flexibility are key to allowing these patterns to be applied in the large scale, open distributed systems that we consider.

Given that the p-assertions representing the provenance of a result may be spread across multiple stores, there must be some mechanism to retrieve these p-assertions in order to validate, visualise or replay the represented process. To facilitate such a retrieval mechanism, we introduce the notion of a link, which intuitively is a pointer to a provenance store.

Definition 5.1 (Link) *A link is a reference to a provenance store.* □

We note that links are necessarily *unidirectional*: a link always points to a remote provenance store location. Links are used in two instances, which we now describe.

5.2.1 View Links

The first use of a link deals with the situation where a client's view of an interaction and a service's view of the same interaction as identified by a shared interaction ID are stored in two different provenance stores. It is necessary for each actor to record a link, which we refer to as a *View Link*, that points to the provenance store where the opposite party recorded their p-assertions. Hence, the client in an interaction records a link to the provenance store that the service used to record p-assertions for the given interaction, and vice-versa. This allows querying actors to navigate from one provenance store to the other in order to retrieve both views of an interaction. We note that View Links point to provenance stores only, not to particular pieces of data in a provenance store; the actual data of interest can be found by the ID identifying the interactions uniquely.

If an actor A interacting with actor B has to assert, in provenance store P_A, that B is recording its view of the interaction in another provenance store, then

actor A has to become aware that the store used by B is P_B. Either such a knowledge is built in A, or it is communicated to A in the course of execution. If it is built in A, then such a knowledge is part of A's state, and can be asserted by A as an actor state p-assertion. Alternatively, if it is to be communicated to A, then such a knowledge can be passed as part a context, as formalised by the ContextPattern (for instance, when B returns a result to A). It is up to the provenance store to extract such linking information from actor state or interaction p-assertions and make it readily available to the querier.

5.2.2 Object Links

Relationship p-assertions allow relationships between both messages and data to be expressed. We model relationship p-assertions as one-to-many triples between data or messages. (Details about the modelling can be found in Chapter 7.) Each triple consists of a subject, a relation and several objects. A relationship p-assertion made by an actor is a directional relation and the subject is referring to the local current interaction p-assertion (or a data contained in it).

Alternatively, the object of a relation p-assertion may be local or remote. Hence, we introduce a second usage of links to cater for the situation in which an application actor records a relationship p-assertion between a local p-assertion and a remote p-assertion. In this case, the application actor needs to record which provenance store the remote p-assertion being related is stored in; such a kind of link is being referred to as *Object Link*. Again, an Object Link only points to the provenance store and not a particular piece of data in the store, because the data of relevance can be found using an interaction record key.

5.2.3 Linking Summary

Both View Links and Object Links allow data and p-assertions stored across provenance stores to be retrieved by querying actors. View and Object Links can be contrasted as follows. A View Link points to another store that contains a piece of data written by another actor (which is providing a different view on a same interaction). An Object Link points to another store containing a piece of data asserted by the same actor (which is making assertions about another interaction).

5.3 Scalability Issues

This section is a placeholder for more discussion on scalability issues based on the scalability deliverable.

5.4 Security

As discussed earlier, linking provides a mechanism to discover related p-assertions in multiple stores. A potential security issue will arise if the querying actor is not recognized or does not have the necessary authorization to retrieve the relevant p-assertions in the security domains of all these stores. Consider a querying actor submitting a complex query to the provenance store, whereupon the querying functionality determines the necessary p-assertions required to satisfy the query and attempts to retrieve them from the provenance store. In the event that not all of the required p-assertions can be found locally, there are two possible ways to proceed:

- The querying functionality itself uses the links in the available p-assertions to locate other distributed provenance stores which it will subsequently need to query. Thus, the querying functionality queries the distributed provenance stores on behalf of the querying actor, and hence this approach will require delegation of the querying actor's identity or access rights to the querying functionality.
- The querying functionality returns the available p-assertions to the querying actor, which will then itself have to navigate the links and make queries to the relevant distributed provenance stores.

In either case, the identity (delegated or otherwise) of the querying actor may not be recognized or may not have the required authorisation in all the provenance stores that need to be queried. There is therefore a need for additional interaction between the trust mediator of the provenance stores that do not recognize this identity with the trust mediators of other provenance stores that do, in a similar manner to that outlined in Scenario 1 in Section 4.3.2. The credentials (delegated or otherwise) submitted by the querying actor must provide enough information for such interaction between these different trust mediators to take place.

Another security concern revolves around the need to establish liability for erroneous context information. In Section 5.1.2, we had discussed the use of a context containing IDs to identify an interaction between two actors. Since p-assertions in the provenance store are matched on the basis of these IDs, it may be useful to provide non-repudiation on such generated IDs in application domains where the need to identify matching p-assertions correctly is critical. As an example, if a client records X for its ID in its own p-assertion and sends a service Y as an ID instead, the client's signature on Y would establish its liability for the subsequent inability to obtain a match between both IDs. The signature permits us to rule out other possible mitigating factors for the discrepancy, such as transmission errors or an error on the part of the service actor.

The security policy in such an instance should therefore dictate that the client actor sign the IDs it generates (or the context containing such IDs) to the service

actor. This signature activity could be encompassed within the mandate of the protocol governing a secure interaction between both the client and service actors. On a similar note, we observe that since the view link is crucial in locating a related p-assertion for a given interaction, its non-repudiation can also be achieved by ensuring that the recorded link is signed as well. If the recorded link is part of the contents of a recorded p-assertion, then a signature on the entire content will suffice. Signatures on p-assertions are discussed in Section 7.8.

5.5 Conclusion

This chapter presented a distribution architecture that addresses problems of scalability in provenance systems. The chapter presented three deployment patterns, which identify communication between key architecture roles and that can be applied by developers to deploy their provenance system in a distributed manner. It then discussed how provenance retrieval could be enabled across a set of distributed provenance stores via linking. We also discuss some relevant security issues.

Chapter 6

Data Identification

As stated in Chapter 2, the provenance of a data item is the process that led to that data item. A querying actor may ask for the provenance of a data item at any time after that item has been created. Asking for the provenance of that data requires: (1) determining what process that data was the product of and (2) obtaining all documentation about that process.

There are several factors that make asking the provenance question difficult: a data item may not have a unique identifier, may be moved from where it was initially stored after being produced, etc. These factors mean that identifying data items to determine their provenance is a non-trivial task. In this chapter, we develop a set of solutions for identifying arbitrary data items, and the relationships between them, in order to discover their provenance. The solutions place requirements on the querying functionality of the provenance architecture, discussed in Chapter 8.

The chapter is structured as follows. Section 6.1 examines what documentation is produced from executing a process. The algorithm for retrieving the provenance given a *query data handle* for a data item is specified in Section 6.2. Section 6.3 enumerates the different ways in which data can be identified for the purposes of retrieving the provenance of that data: *global p-assertion key*, *unique data* and *relational query data handles*, with some important examples of the latter. Finally, Section 6.4 examines the difficulties presented when documentation is moved around between provenance stores.

6.1 Data Item Reference Creation

Application data items may be given explicit references (names) by an application process. We call this action *naming* of the data items. In this section, we illustrate what documentation is contained within a provenance store after a process has completed. This allows us to later show how such documentation can be searched in order to identify p-assertions relevant to the provenance of a data item. We

emphasise that the mechanism for naming data items is outside the scope of this project, and is application-specific, so we do not prescribe how it should be done, but examples include file paths or database keys.

For each output data item, D, of the application, there was an application actor that sent D to a *data store* where a querying actor can later retrieve it. In order to retrieve the data from the data store, the data is given a *reference*, which is, at minimum, unique locally to the data store. The structure of a data store, and its mapping from references to data items is shown in Figure 6.1.

Storage Reference 1	D 1
Storage Reference 2	D 2
...	...
Storage Reference N	D N

Table 6.1: Data store

In order to clarify what the contents of a provenance store are after an application process has completed, we consider four possible ways in which D could be named in the sections below. We assume that recording actors record interaction p-assertions regarding all the application interactions: this may not be true in practice, but allows us to show the most complete contents of the provenance store. Actor state and relationship p-assertions are also assumed to be recorded where required to provide enough information so that a querying actor can determine which data was stored and the references that were used for them in the data store. The documented relationship between data items and their storage references can be used to discover the provenance of the data item, as discussed in Section 6.3.4.

Scenario 1 The data item may be produced by one application actor, SA, and named by another, SB, with the latter application actor specifying the reference. Figure 6.1 illustrates this scenario. After the process has been documented by recording actors recording p-assertions for each interaction, the provenance store(s) will contain the data schematically described in Table 6.2. Each interaction ID in the table corresponds to the interaction numbered in Figure 6.1 and the message contents are shown to include the information passed as shown in the figure. Because the interaction p-assertions replicate the interaction message, D and the Reference 1 are also contained in the provenance store(s) as shown.

Interaction ID	Sender	Receiver	Interaction PA	Actor State PA
1	SA	SB	...	
2	SB	SA	... D ...	
3	SA	Store A	... D to be stored at Reference 1 ...	Storage operation

Table 6.2: Contents of provenance store after scenario 1

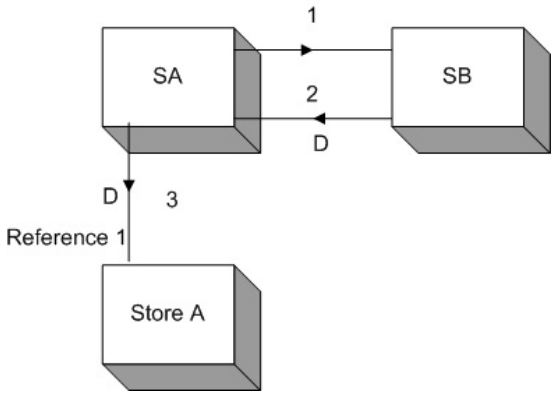


Figure 6.1: Scenario 1

Scenario 2 The data item may be produced by one application actor, and named by another, with the data store specifying the reference. Figure 6.2 illustrates this scenario. After the process has been documented by recording actors recording p-assertions for each interaction, the provenance store(s) will contain the data schematically described in Table 6.3.

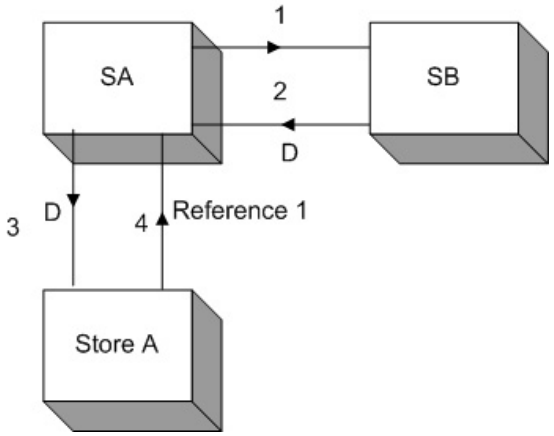


Figure 6.2: Scenario 2

Interaction ID	Sender	Receiver	Interaction PA	Relationship PA
1	SA	SB	...	
2	SB	SA	... D ...	
3	SA	Store A	... D ...	
4	Store A	SA	... Reference 1 ...	Key resulting from storing D in interaction 3

Table 6.3: Contents of provenance store after scenario 2

Scenario 3 The data item may be produced and named by one application actor, with an application actor other than the data store specifying the reference. Figure 6.3 illustrates this scenario. After the process has been documented

by recording actors recording p-assertions for each interaction, the provenance store(s) will contain the data schematically described in Table 6.4.

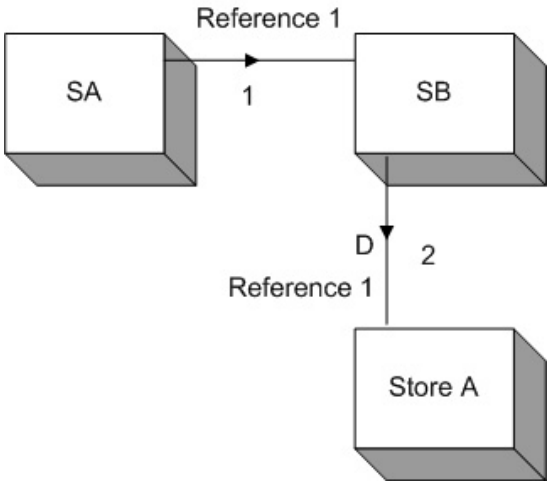


Figure 6.3: Scenario 3

Interaction ID	Sender	Receiver	Interaction PA	Actor State PA
1	SA	SB	... Reference 1 ...	
2	SB	Store A	... D to be stored at Reference 1 ...	Storage operation

Table 6.4: Contents of provenance store after scenario 3

Scenario 4 The data item may be produced and named by one application actor, with the data store specifying the reference. Figure 6.4 illustrates this scenario. After the process has been documented by recording actors recording p-assertions for each interaction, the provenance store(s) will contain the data schematically described in Table 6.5.

Interaction ID	Sender	Receiver	Interaction PA	Relationship PA
1	SA	SB	...	
2	SB	SA	... Reference 1 ...	
3	SB	Store A	... D ...	
4	Store A	SB	... Reference 1 ...	Key resulting from storing D in interaction 3

Table 6.5: Contents of provenance store after scenario 4

6.2 Provenance Query

There are two facts we can state about a querying actor, Q, querying for the provenance of data item D.

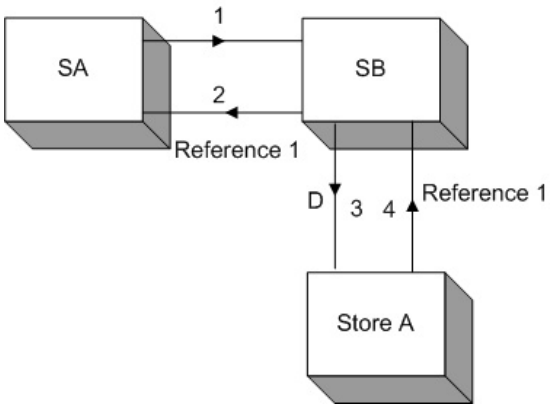


Figure 6.4: Scenario 4

- Q must have received D or some information that identifies D from somewhere in order to ask the provenance question ("What is the provenance of D?").
- Q must know the locations of the provenance stores containing the provenance of D and how to query those stores.

Ultimately, Q's source of D is an application actor that was aware of D from the application, i.e. an application actor in the experiment which received or created D at some point.

We define a *query data handle* of data item D as a data item that can be used to discover the process that led to D. For each query data handle, there is a method by which that name is processed to retrieve a *global p-assertion key* which is part of the documentation of the process that led to D: we call this *resolving* the query data handle. A global p-assertion key is a globally unique identifier for a p-assertion. By passing a global p-assertion key to the provenance store containing the p-assertion to which the identifier refers, a querying actor can retrieve the p-assertion. The general algorithm for discovering the provenance of D given query data handleI is as follows.

- Retrieve an interaction p-assertion regarding part of the process that led to D by resolving I.
- Retrieve, from the provenance stores, the identity of the processes that the interaction p-assertion is marked as being part of.
- Search the provenance stores for all interaction p-assertions marked as belonging to that process, and the associated actor state and relationship p-assertions.

The method by which query data handleI is resolved to a p-assertion depends on the form of I. We discuss each form below.

6.3 Query Data Handles

A query data handlecan have one of the following forms. For each form, we discuss its content, how querying actors may obtain a query data handleof that form and how it can be resolved to find a p-assertion for the process that led to a data item. It should be emphasised that these are a set of options and only a subset will apply in the case of a particular data item in an application. Also, a single data item may have multiple query data handles.

6.3.1 Provenance Store Identification

As was illustrated in Section 6.1, the data occurs at least once in the provenance stores. This means that one way to identify a data item is as a component of the provenance documentation. For convenience, we define a *p-assertion identifier of D* as a global p-assertion key for a p-assertion that is part of the documentation of the process that led to D obtained by means other than querying a provenance store. For example, if a single actor recorded a p-assertion in the process that led to D because they were involved in that process, then they must have a global p-assertion key of D, because supply all the data making up that key on recording. A global p-assertion key of D is a query data handlefor D.

Every recording actor that records an interaction p-assertion about a message in which D (or a reference to D) is sent or received, must have had a p-assertion identifier of D in order for documentation about that interaction to be recorded: the global p-assertion key must be provided in the message from the recording actor to the provenance store on recording.

To resolve a global p-assertion key, it is sent to the provenance store containing the corresponding p-assertion, and the p-assertion is returned.

6.3.2 Unique Data Identification

If the data is unique within the application, e.g. if it includes a universally unique ID as part of its content, then the data itself is its own query data handle.

Any application actor or querying actor that has access to data that is unique has this query data handle.

To resolve unique data D to a p-assertion, search the provenance stores for a p-assertion documenting an interaction returning D.

6.3.3 Relational Identification

A data item may be referred to uniquely by its relation to another data item. This may be an explicit reference that can be resolved by a known service, for example a path to a file in a file system. Alternatively, it may be an intensional

definition, e.g. D may be the second element of list L. These are examples of *relational query data handles*.

A relational query data handleis a query data handlethat can be expressed in the form: subject S is related to data item D by property P. A querying actor with a relational query data handleknows S and P. The resolution of a relational query data handledepends on the type of property P.

A few types of relational query data handlesare common across applications, so we provide these as concrete examples below.

6.3.4 References

Data items may be referred to by a unique identifier, a *reference* within an application. This is relational query data handlewhere the subject is the reference and the property defines a reference.

One instance of a reference is a storage location. Given the data store location, L, at which a data item, D, was stored and store reference, S, that can be used to retrieve the data item from L, L and S uniquely identify D. This is a relational query data handlewhere the relation defines an application-specific operation: the 'storage' operation in which D was stored in L.

As shown in the tables in Section 6.1, at least the application actor that stores a data item and the data store itself know this query data handle.

Relating Operations

If an application-specific operation linked two data items then, to find a p-assertion regarding one of those data items, we can search the provenance stores for an interaction involving that operation and the other data item. For example, if the subject of a relational query data handleis a collection and the property is the containment relation, then searching for the 'add' operations on the collection will find p-assertions documenting the process that led to the contents of the collection, i.e. the data item we want to find the provenance of.

Movement of Data Objects

When D is moved or copied between data stores or within a data store, a new data item with a new store reference is created. That is, D was previously at Reference 1 in Data Store 1 and now D' is in Reference 2 in Data Store 2 (and D may have been deleted from Data Store 1). We consider the movement or copying or D to be equivalent to another experiment in which the data item is D'. The provenance of D' is the documentation of a process whose data item was D' and whose inputs include Reference 1 and Data Store 1. Therefore, any querying actor able to determine the provenance of D' using the algorithm above, will

have the storage identification of D and, so, can also determine the provenance of D.

Therefore D' is relational query data handlefor D where the property defines an application-specific operation: move or copy operation.

6.4 Movement of Process Documentation

The p-assertion that a p-assertion identifier refers to can be moved or copied between provenance stores. That is, the p-assertion identifier was previously resolvable by Provenance Store 1 to retrieve a p-assertion and it is now resolvable by Provenance Store 2. After the move, Provenance Store 1 may no longer exist.

The p-assertion identifier itself remains the same when the p-assertion it refers to is transferred between provenance stores. This means that, unless informed, querying actors no longer have provenance store query data handle. Even if the movement of the p-assertions was itself recorded in a provenance store, Provenance Store 3, this will not help because the querying actors may not know to look in Provenance Store 3.

We identify three alternative solutions.

1. If there are an adequately small number of provenance stores that the p-assertion could have been moved to then a querying actor may be able to discover Provenance Store 2 by attempting to resolve the p-assertion in each provenance store.

2. The move could be recorded in a location that all previously aware querying actor would know to look in, i.e. all querying actors know to look in Provenance Store 3.

3. Each interested querying actor may be informed of the move of the provenance data their p-assertion identifier refers to when it happens. This latter option could be achieved by subscription to notifications of provenance data movement (possibly sent by the Provenance Store 1).

6.5 Security

If p-assertions are copied or moved between stores that are located in different security domains, the access control restrictions on them in their new destinations will need to be defined. In the simplest case, the newly moved or copied p-assertions retain the same access control restrictions that were associated with them in their original domain. These restrictions can be provided as authorisation information along with the p-assertions as they are recorded to their new destinations, where they can be processed by the derivation engine in the manner described in Section 4.3.1.

If the authorisation information involves identities from the originating domain that are currently unknown in the destination domain, then this identity information needs to be communicated between the trust mediators of both domains. The communication can be performed when the p-assertions are initially recorded, or at a later time when a request is made to the provenance store from an entity that is not recognizable in the new provenance store domain. The process of moving p-assertions between different stores will also need to ensure that the transfer medium is secure (if such a requirement is present), and that both stores are properly authenticated to each other prior to the movement.

6.6 Conclusion

In this chapter, we have presented different ways in which to identify application data items, and thereby find their provenance. We suggested a general algorithm for discovering the provenance of D given query data handleI, repeated below.

- Retrieve a p-assertion regarding part of the process that led to D by resolving I.
- Retrieve, from the provenance stores, the identity of the process that the interaction is marked as being part of.
- Search the provenance stores for all interaction p-assertions marked as belonging to that process, and the associated actor state p-assertions.

In summary, we provided three ways in which data items could be identified: by the identifier of a p-assertion that documents part of its provenance, by its own unique content of which there will be copies in the interaction p-assertions in the provenance store, or by relation to something that can itself be used to identified the provenance. Examples of the former, relational, means of identification were given: a reference identifier, the storage location and the data items produced by processes that used another data item as input.

For each data item for which a querying actor may want to discover the provenance, application developers should decide the solution or solutions to apply. The algorithms for resolving query data handlepresented in the chapter place requirements on the query interface of provenance stores. In particular, the query interface must have the following functionality.

- Retrieve a p-assertion given its p-assertion ID.
- Retrieve the processes that a p-assertion is marked as belonging to.
- Return all p-assertions marked as belonging to a process.
- Search for interaction p-assertions containing given data.

A data item that has been exchanged in an application message can be referred to even after it has been deleted from the data store in which it was stored, because a global p-assertion key identifies a p-assertion that contains it. A *p-structure data identifier* is an extension of a global p-assertion key that refers to a data item contained within a structured p-assertion. The structures and contents of global p-assertion keys and p-structure data identifiers are defined in Chapter 7.

Chapter 7

Provenance Modelling

In Section 2.4, we discussed the representation of provenance and introduced the concept of a p-assertion as an assertion by an actor about a process. We identified three types of p-assertion: interaction, relationship and actor state p-assertions. In this section, we identify how each type of p-assertion can be *modelled*, i.e. we define the data structure used to represent each type of p-assertion. Based on these models, we introduce a common structure according to which p-assertions are structured in the provenance store.

We note that the p-assertion models presented could be instantiated in different languages, such as XML, RDF or even application-specific binary formats. The choice is specific to the application that will make use of the process documentation and the infrastructure on which it is run. To depict the models in the sections below, we adopt a graphical representation of XML Schema, but this should not be interpreted as prescribing the method of encoding. A description of the graphical representation can be found in Appendix C.

7.1 Identifying Interactions

Every p-assertion is made in the context of an interaction: an interaction p-assertion asserts the content of a message sent or received in an interaction, an actor state p-assertion asserts the state of an actor at a specific instance during an interaction and a relationship p-assertion relates an interaction to other interactions. Therefore, in order to model p-assertions, we must provide a way to identify an interaction.

In Figure 7.1, we specify our model for referring to a single interaction: the *interaction key*. This key is made up of three parts: the address from which the message came, the *messageSource*, the exact address to which the message was sent, the *messageSink* and an identifier that specifies a particular interaction between these two addresses, the *interactionId*. An instance of this data structure must be sent along with every p-assertion when the p-assertion is recorded in a

provenance store, so that the store is aware to which interaction the p-assertion pertains.

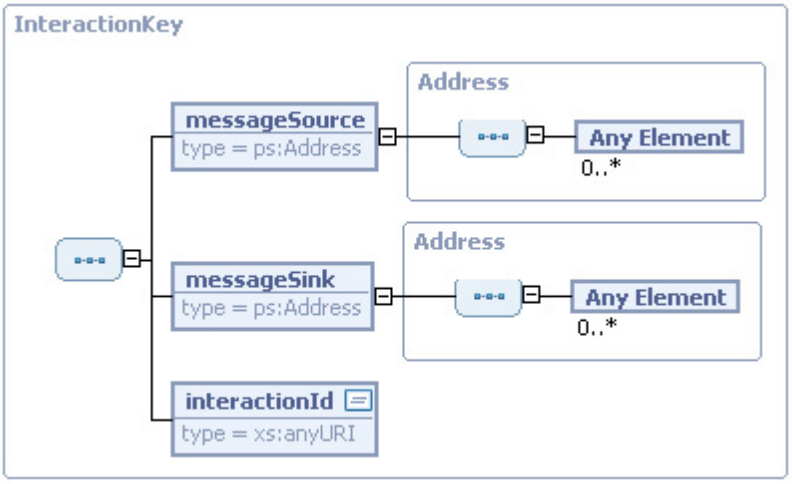


Figure 7.1: Model for identifying an interaction

7.2 Identifying P-Assertions and Data

In addition to identifying the interaction about which an assertion is being made, every p-assertion has its own identifier: the *local p-assertion identifier*. Each p-assertion made by one asserting actor about one interaction must have a different local p-assertion identifier. With both interaction identifiers and local p-assertion identifiers, we can construct a global p-assertion key (GPAK) as shown in Figure 7.2. A GPAK consists of an interaction key, whether the client (sender) or service (receiver) in the interaction made the assertion (the *view kind*) and the local p-assertion id. A GPAK uniquely identifies a p-assertion whether that p-assertion is stored in a provenance store or not.

We can extend the GPAK to also allow data to be uniquely identified within a p-assertion. This is done by adding a *dataAccessor* to a GPAK to form what we term a P-Structure Data Identifier. This dataAccessor is p-assertion specific and identifies uniquely a piece of data within the particular p-assertion referenced by the GPAK that is extended.

7.3 Interaction Contexts and the P-Header

As described in Chapter 3, application actors must exchange provenance-specific context information related to particular interactions for the process documentation to be usable by query actors. For example, both client and service must

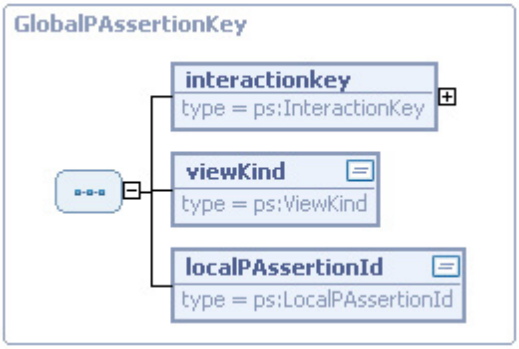


Figure 7.2: Global P-Assertion Key

use the same interaction key for the same interaction. Context information can be passed as extra data in existing messages or independently in messages specifically for the purpose. In the latter case, the context information conforms to the *interaction context* structure shown in Figure 7.3. An interaction context contains an interaction key and any number of items of *interaction metadata*, which are contextual information regarding the identified interaction. Examples of interaction metadata include tracers and addresses of provenance stores, which are used to create ViewLinks discussed in Section 5.2.

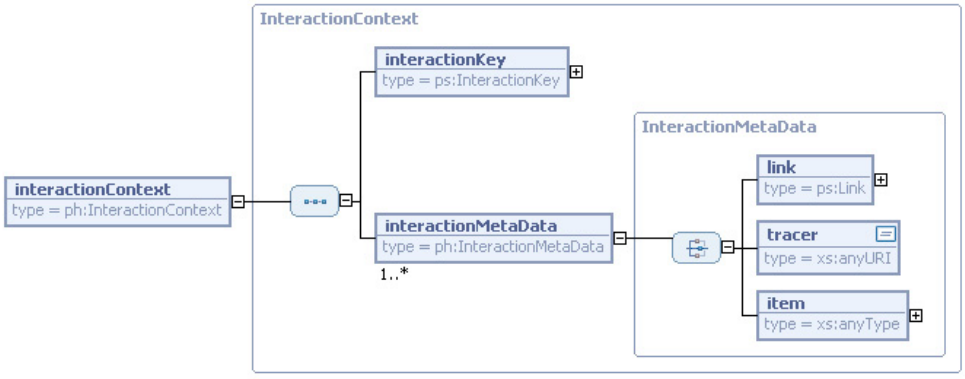


Figure 7.3: Model of an Interaction Context.

If information contexts are exchanged via the header of an application message, according to the ContextPassing pattern, then the *p-header* structure, shown in Figure 7.4 is used. Apart from potentially including a set of interaction contexts about other interactions, the p-header provides context information about the message to which it is attached. It includes an interaction key that the message sender is stating should be used to denote the interaction to which

the p-header is attached. This can be used, for example, for a client to inform a service of the interaction key for a given interaction. Additionally, a set of interaction metadata can be provided about the message to which the p-header is attached.

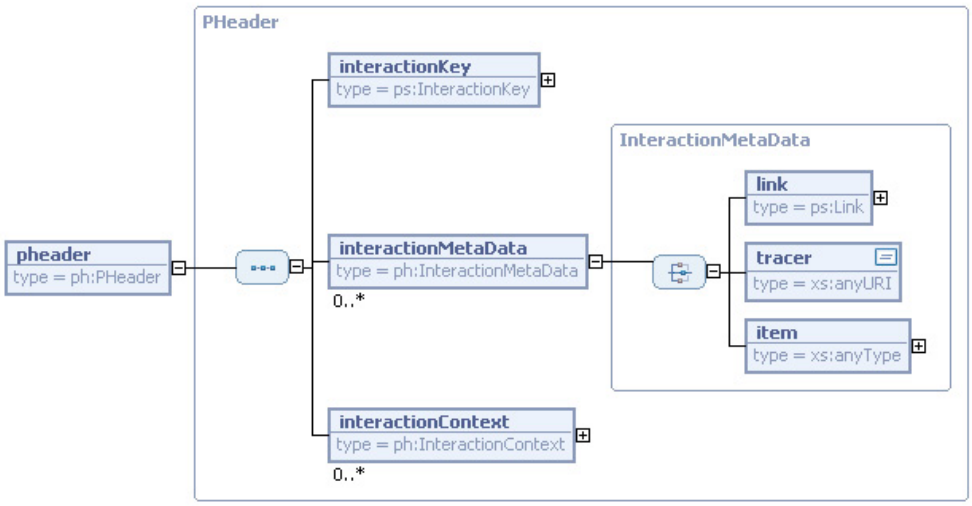


Figure 7.4: Model of the PHeader.

When View Links are sent in interaction context messages, not a p-header, the receiving actor will have to determine to which view it refers, i.e. if an actor that was a client in an interaction sends a View Link for that interaction to the actor which was the service in the same interaction, the service will determine that the link refers to the store containing the client view even though this information is not explicitly stated in the interaction context.

7.4 Interaction P-Assertion Modelling

Interaction p-assertions, as defined in Definition 2.6, state the content of a message received or sent by the asserting actor. There may be different ways according to which the content of a message may be asserted: for instance, the message content may be asserted verbatim as the asserting actor received/sent it, or an altered description may be asserted in which, for example, sensitive or large data items within the message are replaced with references to those copies of the data items stored elsewhere, or are replaced with references and a digest of the data. Therefore, in modelling an interaction p-assertion, we need a data structure in which asserting actors can declare not only the content of the message but also the *documentation style* that has been applied to it. If no change has been made

between the message content sent/received and that asserted in the p-assertion, a 'direct copy' documentation style is asserted.

We define a data type *InteractionPAssertion* to represent any interaction p-assertion and depict its structure in Figure 7.5. The p-assertion consists of three pieces of information: the local p-assertion identifier, *localPAssertionId*; an identifier specifying the documentation style applied to the message content, *documentation style*; and the message content itself, shown as 'any' as it is entirely application-dependent and so no generic data structure can be specified for it. For example, the message content may be a SOAP or a CORBA meessage.

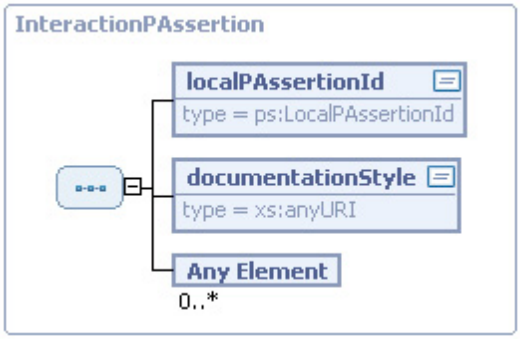


Figure 7.5: Model for an interaction p-assertion

We remind the reader that the purpose of an interaction p-assertion is to describe an interaction that took place in a process; ultimately, that p-assertion may be returned as part by a provenance query about the result produced by that process. The act of constructing an documentation item, i.e. a p-assertion, is itself a computation. Our rationale for introducing the concept of documentation style is that such a computation is so small or trivial (e.g. verbatim copy of a message) that it simply can be described by documentation style. If an application needs to perform a complex computation involving multiple actors in order to produce a p-assertion, we expect such a complex computation to be documented fully by using the p-assertion model of documentation introduced in this chapter. It is up to the application designer to decide the level of documentation that is required for a given computation: if the process of creating p-assertions needs to be documented by p-assertions itself, we must have a base case according to we agree not to document a computation further: this is precisely the role of documentation style.

7.5 Actor State P-Assertion Modelling

Actor state p-assertions, as defined in Definition 2.8, are assertions made by an actor about its internal state in the context of a specific interaction. Each actor

in an interaction sends or receives a message, so an actor state p-assertion asserts something about the state of the actor just before or just after it sent or received the message. For example, a service with an incoming message buffer may assert the state of its buffer just before and after receiving a message. Often, after an actor receives a message it will perform an execution that the message has triggered and, similarly, before sending a message it will perform an execution that resulted in that message. Therefore, a common subset of actor state p-assertions give details of the *execution* that took place just after receiving or just before sending a message. For example, a service may assert the computational resources allocated to an execution. For example, the actor state may name the workflow that the interaction occurred as part of.

We define a data type *ActorStatePAssertion* to represent any actor state p-assertion and depict its structure in Figure 7.6. The p-assertion consists of two pieces of information: the local p-assertion identifier; *localPAssertionId*, and the actor state document content itself, shown as 'any' as it is entirely application-dependent and so no generic data structure can be specified for it.

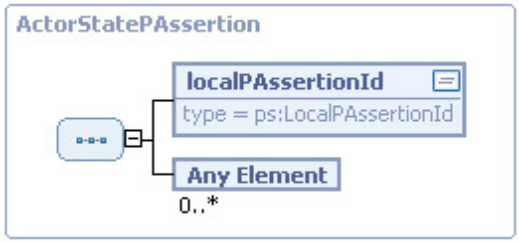


Figure 7.6: Model for an actor state p-assertion

In Appendix D, we provide suggested models for common types of actor state, which can be followed to allow greater interoperability.

7.6 Relationship P-Assertion Modelling

Relationship p-assertions allow uni-directional relationships between both messages and data to be expressed. We model relationship p-assertions as one-to-many triples between data or messages. The triple consists of a subject identifier (subjectId), a relation, and several object identifiers (objectIds).

A subjectId identifies a data item or message within the asserting actor's view of an interaction. Therefore, we limit the subjectId to identifying one message or data item within the context of the particular interaction. An objectId identifies any data item or message. It accomplishes this by referring to the interaction, the view in that interaction, the local p-assertion identifier, and if referring to a data item an additional dataAccessor. An objectId also contains a parameterName,

which specifies which particular input the object was used as in the operation that transformed the objects of the relation into the subject, e.g. in a 'division' operation the parameter name may a 'dividend' or a 'divisor' concept. Similarly, the subjectId can have a parameter name specifying which output of the operation the subject refers to. Finally, the objectId optionally contains an *object link* giving the address of the provenance store in which the p-assertion is kept.

The relation of the triple is a URI that has some semantics that can be understood by querying actors.

Examples of possible relationships are as follows: interaction A *is before* interaction B; interaction B *is after* interaction A; data item C *was zipped to produce* data item D; data item D *is a combination of* data item C and data item B; data item D *is a product of multiplying x and y* data item C is x and data item B is y; interaction A *is in reply to* interaction B; interaction A *causes* interaction B. Since, we do not limit what relationships can be expressed, this allows asserting actors to express application specific relationships.

7.7 The P-Structure

Up to this point the architecture has assumed that a provenance store contains a collection of p-assertions. However, if this collection has no structure several problems may arise: the inability to identify one p-assertion from another, the inability to tell the interaction context of an actor state p-assertion, the inability to causally relate one p-assertion to another, and therefore the inability for a querier to retrieve the provenance of a piece of data. To solve these problems, we introduce the notion of a *p-structure*.

Definition 7.1 (p-structure) *The p-structure is a common logical structure of the provenance store shared by all actors including asserting, recording, querying and managing actors.* □

The p-structure is designed specifically for organising p-assertions in a manner that allows the provenance of a piece of data to be retrieved. We now detail the p-structure itself and then show how parts of the p-structure, including p-assertions, can be identified. Figure 7.8 shows the p-structure. It reflects the models discussed above.

The p-structure is organised as a hierarchy. At the top level of the hierarchy are *InteractionRecords*. Each record encapsulates all the p-assertions and identifiers related to one interaction. The choice of interaction records as the chief items in the p-structure comes from the idea that interactions are the core actions of a process. Each InteractionRecord is identified by a set of an interaction key, as shown in Figure 7.1. The interaction key distinguishes one InteractionRecord from all others and is provided by the asserting actor and not the provenance

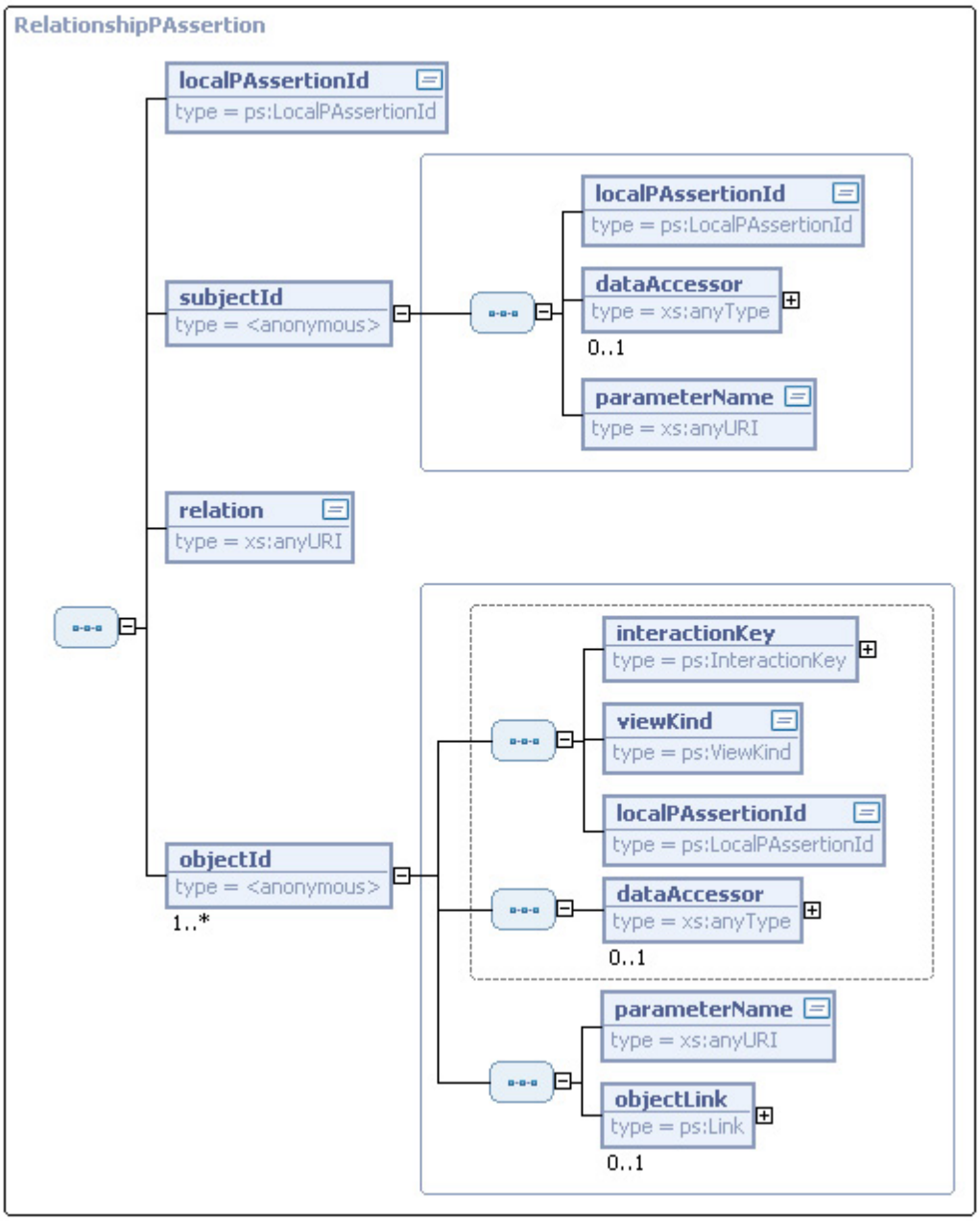


Figure 7.7: Relationship p-assertion model

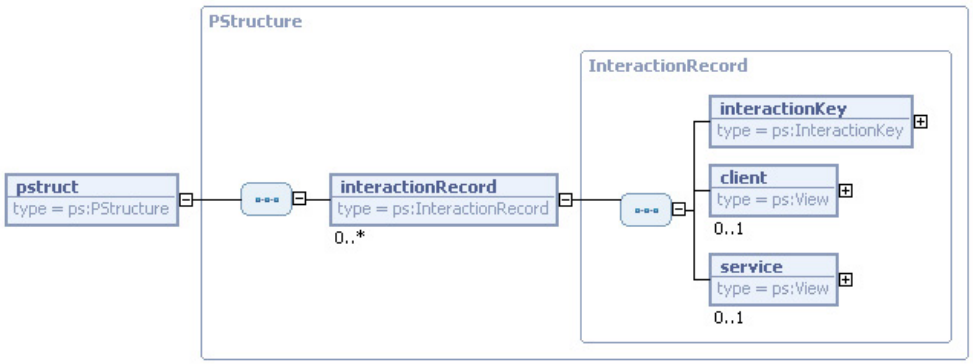


Figure 7.8: The P-Structure

store. Therefore, no contact with the provenance is required in order to create p-assertions.

In the p-structure hierarchy, we find two *Views* under the InteractionRecord. One View contains the p-assertions from the client in the interaction, while the other View contains those from the service. A View has the following structure as shown in Figure 7.9: it can contain several interaction p-assertions (where there are more than one, we would expect different document styles to be used for the same message), several actor state p-assertions, several relationship p-assertions, the number of p-assertions that the store should expect to receive in this view (which can be asserted by the recording actor as part of the recording protocol) and a View Link (as described in Chapter 5). All of these are optional. Each p-assertion is defined by an associated model described above.

We note that the definition of p-structure does not dictate its internal implementation. Instead, the p-structure facilitates the asserting, recording, querying and management of p-assertions by allowing actors to address and create p-assertions with a common knowledge about how p-assertions are logically associated with one another.

7.8 Security

As discussed in Section 4.2, integrity and non-repudiation of p-assertions as well as the need to verify asserter identity in a p-assertion are important security requirements. Both these requirements can be fulfilled by having an asserting actor sign the created p-assertion. A subsequent check can then be done to determine whether the identity associated with the signature corresponds with the identity of the asserter as recorded in the p-assertion. This can be accomplished either by the provenance store prior to storing a submitted p-assertion, or if it is not done here, by a querying actor prior to processing the p-assertion in some

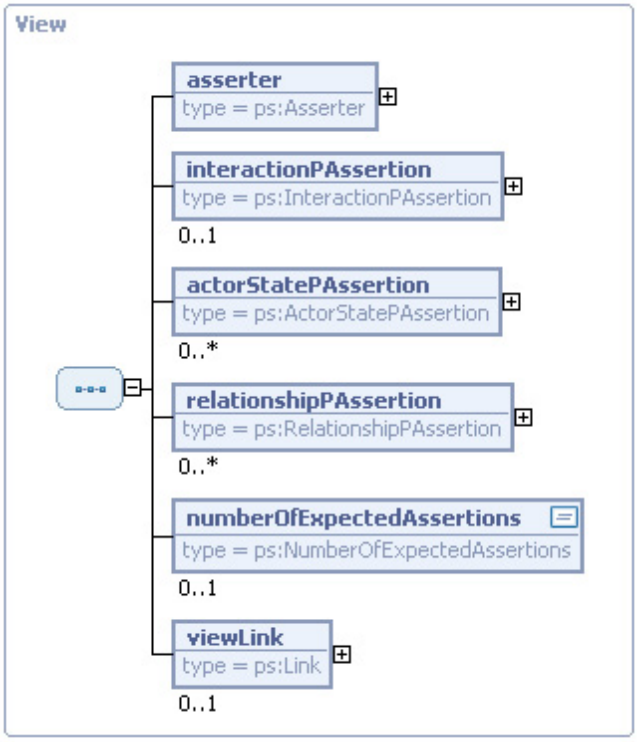


Figure 7.9: A View in the P-Structure

manner.

To provide for this, the model should state where in the p-structure the signatures as well as the certificates necessary to verify them are to be found. Therefore, we augment the three p-assertion models to include an optional fourth element: the signature applied to that p-assertion in recording. These models, with the additional *signature* element, are shown in Figures 7.10, 7.11 and 7.12.

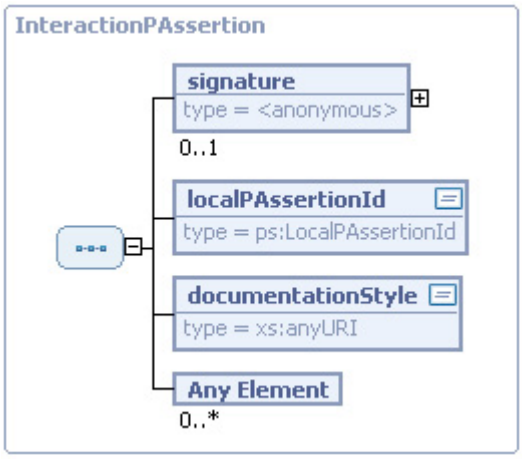


Figure 7.10: Model for a secured interaction p-assertion

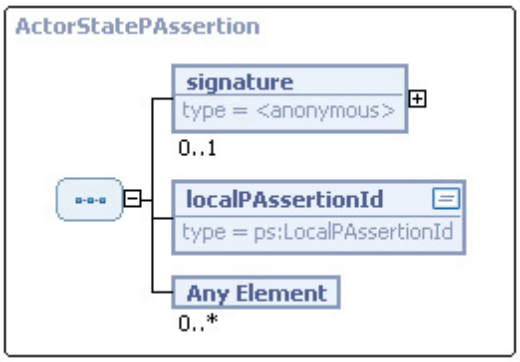


Figure 7.11: Model for a secured actor state p-assertion

7.9 Conclusion

In this chapter, we discussed how p-assertions may be modelled within an application. We presented a model for how interactions, p-assertions and the data

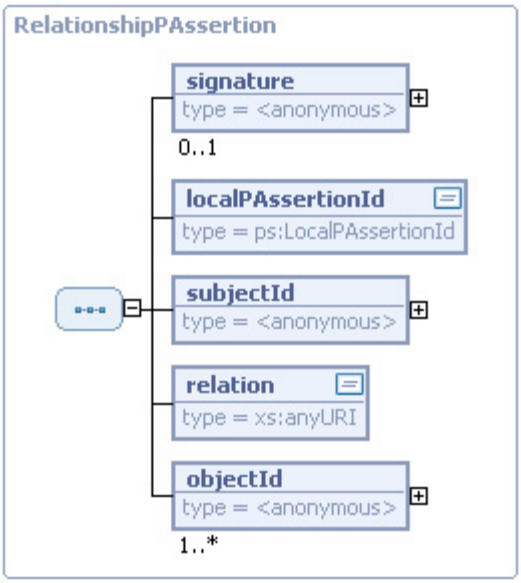


Figure 7.12: Model for a secured relationship p-assertion

contained in p-assertions can be identified, as well as a model of an interaction context and the p-header in which this context information can be passed between actors. We then provided more detailed models for interaction p-assertions, actor state p-assertions, and relationship p-assertions. The p-assertion models were designed to be extensible and uniquely identifiable. Combining the above models, we defined the p-structure as a common logical structure of the provenance store shared by all actors. In the next chapter, we describe the functionality of the provenance store, which utilises the p-structure to accomplish its task.

Chapter 8

Functionality

This chapter discusses the functionality of the provenance store: for each of the recording, querying and management interfaces, it provides an informal description, in English, of the functionality that is supported by the provenance store. Such informal presentation will serve at the derivation of a more formal specification in UML and service interface specifications, which will be part of a separate document.

8.1 Recording Interface

Provenance stores provide a recording interface based on the P-assertion Recording Protocol (PReP), which defines the messages that actors can exchange with the provenance store in order to record p-assertions [GLM04a, GLM04b, Gro05, GMM05]. The protocol is designed to be asynchronous so that p-assertions can be recorded at any time to the selected provenance store. The protocol is also designed to have the following properties:

1. A protocol is *stateless* when an actor can understand a message without relying on any previous or subsequent messages. By supporting this property, the provenance store can record any p-assertion with only the information in the protocol messages it receives. This allows for out of order message delivery and for unfinished interaction records to be present in the provenance store.

2. *Idempotence* is the quality of something that has the same effect if used multiple times as it does if used only once. With this property, once a p-assertion has been submitted to a provenance store then that p-assertion cannot be overwritten or modified. In other words, an actor cannot retract its assertion.

3. The protocol *terminates*. This means that an actor will not be indefinitely recording p-assertions for one interaction.

These properties are discussed in greater detail in other documents [GLM04b, Gro05]. PReP assumes a model in which application actors send application messages to one another. To record p-assertions, we augment the standard *application message* with an additional parameter, the P-Header, as described in Sections 3.4 and 7.3.

We now discuss the messages that are exchanged by actors with the provenance store to record p-assertions. The messages are shown in Figure 8.1 and constitute the recording functionality of the provenance store. Each message in Figure 8.1 contains an interaction key, which is the same as the interaction key transferred in the P-Header (see Figure 7.4). Because every message in the protocol contains an interaction key, all the messages are self-contained and can be understood without reference to any other messages. Therefore, the protocol is stateless. The protocol is asynchronous. This allows actors to record p-assertions about an interaction when it is most convenient to them. We now discuss how actors use these messages to record p-assertions.

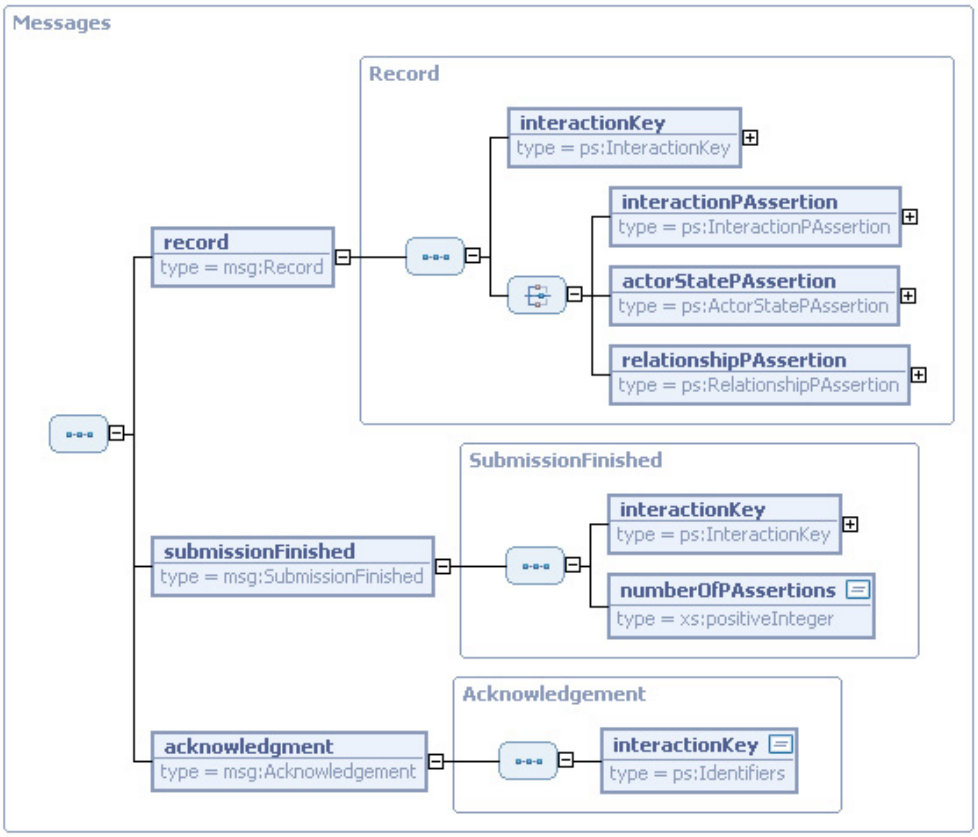


Figure 8.1: Protocol messages

Both clients and services record their view of an interaction in the provenance store. Both actors record their interaction p-assertions using the *record* message.

Interaction p-assertions can only be recorded once per message exchange. Any subsequent interaction p-assertions received for a particular interaction will be discarded. This is to preserve the idempotent nature of the protocol for interaction p-assertions. Therefore, the provenance store will contain only one view of the interactions that took place in an activity from a particular actor.

Beyond interaction p-assertions, an actor can record multiple actor state p-assertions using the same *record* message. Actors record assertions about their state in the context of an interaction. We note that any number of these p-assertions may be recorded. Relationship p-assertions can also be recorded. Again, there is no bound on the number of relationship p-assertions an actor can record. We note that if an actor attempts to record a p-assertion and uses a local p-assertion identifier that is already assigned to a previously recorded p-assertion in that interaction context then the p-assertion will not be recorded.

The second message specified in Figure 8.1 is the *submission finished* message. This informs the provenance store of the total number of p-assertions the store can expect from a recording actor. In this manner, a store can determine when an actor has finished submitting p-assertions for a particular interaction identified by an interaction key.

The final message defined by PReP is the *acknowledgement* message. This message is sent by the provenance store when it has received either a *record* or *submission finished* message. The *acknowledgement* message contains an interaction key. This allows an actor to determine if all the messages it sent pertaining to a particular interaction were received by the provenance store. In a binding of PReP to a system where requests and responses are synchronous, the binding may choose not to include the interaction key in the *acknowledgement* message.

The recording functionality of the provenance store lets actors record p-assertions about their interactions, state and relationships between them. These p-assertions are uniquely identifiable. The protocol terminates and its messages are idempotent and stateless. These properties are important in the context of the large scale distributed applications.

8.2 Query Interface

Once documentation of a process has been recorded, the query interface provides a way to navigate, search and retrieve it for further processing. The navigation capability is based on evaluating expressions against a hierarchical view of the documentation in provenance stores. We then envisage other query capabilities being provided as layers over the navigation capability, to ensure that queries can be expressed in a way convenient to the processing services. In particular, we consider below two fundamental types of query: discovering the provenance of some data and asking questions about past processses in the physical world.

8.2.1 Navigating Documentation of Process

As the p-structure demonstrates, p-assertions recorded when an application executes can be arranged into a hierarchy: every p-assertion is with regard to one interaction, for each interaction there can be differing views from the sender and the receiver of the message, and each view consists of three sets of p-assertions, one for each of three types: interaction, actor state and relationship.

The navigation capability should allow querying actors to refer to the following.

- An interaction with a given key.
- The sender or receiver's view of that interaction.
- One p-assertion about the interaction in one actor's view.

8.2.2 Querying for Provenance

In the conceptual model defined in Chapter 2, the provenance of a data item is the process that led to that data item. At very least, therefore, the query interface should allow querying actors to express queries that do the following.

1. Identify the interaction that finally produced the data item within the provenance store's documentation, i.e. within the p-structure.
2. Follow the asserted relationships from that interaction to related interactions according to the semantics of the relationships.
3. Scope the related interactions by some criteria of relevance, *interaction filters*.

The first problem above, identifying the interaction that produced a data item, is the one tackled by applying the data identification algorithms in Chapter 6. We assume that, from applying these algorithms, a querying actor identifies a single interaction p-assertion. In the second step, we identify the interactions that are *related* to the one finally producing the data item by querying the relationship p-assertions. In the final step, we filter interactions to determine where to bound the related interactions.

Therefore, the provenance query interface should allow querying actors to perform the following general functions.

- Retrieve the related interactions for a given interaction p-assertion.
- Scope related interaction p-assertions according to some criteria.

We have currently identified four interaction p-assertion filters to limit the scope of query results.

Execution Filter Exclude or include all p-assertions that are marked as part of an execution by a single actor.

Relationship Filter Include or exclude p-assertions that are related by a given relationship.

Message Filter Exclude or include all interaction p-assertions that have a given content, e.g. requests of a given type.

Input Filter When an execution takes as input a data item of a given type, include or exclude the interaction in which that data item was provided.

Distance Filter Include or exclude p-assertions who are related to the final interaction via a number of relationships greater than a given amount.

8.2.3 Querying Physical World Processes

As discussed in Section 2.6, we would like to be able to query processes involving actors who cannot directly document the processes they are involved in, particularly physical actors such as people and machines. To support this, a provenance query interface should be able to:

- Express queries regarding application actors that are hidden but who have interacted with visible application actors.
- Allow for interactions between hidden actors to be inferred from the interaction and actor state p-assertions between visible actors.

8.3 Management Interface

A management interface is defined to facilitate the administration, reuse and maintenance of provenance stores. While such an interface may provide generic data storage administration capabilities, we focus on provenance-specific management functionality. The functionality of this interface is described below.

8.3.1 Notification of Provenance Store Use

Managing actors might like to be informed when operations are performed on a provenance store. For example, they might like to know when a p-assertion has been recorded. The management interface should provide the following functionality regarding notification.

- Notification

  The management interface should be able to notify subscribed managing actors of record and query operations.

- Subscription management

  The management interface should allow actors to manage their subscription information e.g. where notifications are sent to.

8.3.2 Provenance Store Utility

- Import and export

  The management interface should provide functionalities to *import* and *export* p-assertions. Importing, in this context, means adding process documentation to the provenance store that has been taken from storage elsewhere and possibly in a different format. Similarly, exporting means retrieving documentation without regard to the fact that it is process documentation and possibly not in the same format, so that it can be stored elsewhere.

- Setup and management of indexes

  Provenance stores hold a large amount of p-assertions. No matter how these p-assertions are organised, some storage structures may be suitable for some query operations and not suitable for others. The management interface should provide a mechanism to setup and manage indexes in terms of time, tracer or other criteria, so that p-assertions can be organised into multiple views and structures, thus facilitating querying.

8.3.3 P-Assertion Lifetime Management

A number of operations can be perfomed on recorded p-assertions to ensure the provenance stores maintain useful process documentation over the long term. The management interface should provide the following functionalities regarding p-assertion lifetime management.

- Movement

  A provenance store may automatically export the p-assertions it contains to another location under given conditions, e.g. after a given period of time, when the store contents becomes too large etc. This operation allows a managing actor to control the automatic transfer of p-assertions.

- Deletion

  In some circumstances, when application data is deleted, the p-assertions about the process that led to that data may also need to be deleted. The deletion of p-assertions should be strictly controlled, and should only happen when specific asserting actors make an explicit signed request. The management interface should provide functionality to check and verify an

actor's deletion authority and request. It should also allow an actor to specify which p-assertions are deleted.

8.4 Policy

This section is a placeholder for description of policy configurations that should be supported by the provenance architecture.

8.5 Security

The security policy of the provenance store essentially encompasses the internal representation list, authorisation list and access control policy components of the provenance store security architecture as discussed in Section 4.3. The information in all these components are provided by the system administrator of the provenance store prior to its deployment; for example, the potential users of the store may interact with the administrator who subsequently classifies them into roles and assigns authorisations accordingly in the authorisation list.

During the operation of the provenance store, changes may need to be made to the security policy. This could entail, for example, adding or removing identities from the internal representation list and creating or modifying existing authorisations in the authorisation list. These changes can be achieved through a security specific section in the management interface. Correspondingly, the security policy should be initially set to ensure that only trusted managing actors are authorised to make changes of this nature. The same comment is equally applicable to other non-security functionality exposed by the management interface that we have already described, such as provenance store utility or p-assertion lifetime management.

Moving p-assertions between different provenance stores has implications on the authorisations associated with these p-assertions; this issue has already been discussed in Section 6.5.

8.6 Conclusion

This chapter presented a more detailed, informal description of functionality to be supported by the provenance store. Such functionality needs to be further refined. Ultimately, it will lead to UML specification documents and interface specifications, which will be part of a separate document.

Chapter 9

Actor Behaviour

Note: this is the very first draft of this chapter.

9.1 Introduction

So far, we have defined an architecture and a data model for provenance systems. The architecture identifies different roles and functional interfaces that characterise the behaviour of actors to some extent. However, the architectural framework does not (and cannot!) enforce allowed actor behaviours (because provenance recording is a *voluntary* activity by applications and because enforcing very specific behaviour would make the system excessively inefficient).

Instead, this chapter describes the behavioural constraints that actors should follow so that process documentation can correctly be recorded; if such behaviour is followed, querying actors can have the expectation that their provenance questions will be usefully answered. These constraints provide bounds for actor behaviour in provenance-aware systems.

To be systematic, behavioural constraints are expressed as named *architectural rules*, which express a behaviour that an actor *must* follow. The reference semantics of the provenance architecture is only defined in the case behavioural rules are followed.

9.2 Architectural Rules

This section introduces rules that actors playing a particular role must follow in order for process documentation to be both recorded, managed and queried correctly.

Fundamental to the provenance architecture is the ability to uniquely identify interactions from one another. To ensure that interactions are uniquely identified, we introduce the unique interaction key rule.

Rule 9.1 (Unique Interaction Key rule) *A client asserting actor (i.e. a client in the role of an asserting actor) must generate a unique interaction key for an interaction.*□

The interaction key generated by the client must be passed to the service in an interaction so that the service may record p-assertions about the interaction. Therefore, we introduce the interaction key transmission rule.

Rule 9.2 (Interaction Key Transmission Rule) *A client asserting actor must transmit the interaction key it generated for an interaction to the service in that interaction.*□

In order for the provenance of a piece of data to be retrieved, p-assertions must be associated with a particular interaction. The appropriate interaction rule governs how p-assertions should be associated with a particular interaction.

Rule 9.3 (Appropriate Interaction Rule) *A recording actor must use the interaction key associated with an interaction, I, when recording p-assertions about I.*□

Given that an actor can record p-assertions in multiple provenance store, we introduce the following recording consistency rule.

Rule 9.4 (Recording Consistency Rule) *All p-assertions pertaining to one interaction from a particular actor must be recorded in the same provenance store.* □

The recording consistency rule implies that the documentation of an interaction from a given actor's viewpoint is kept in a single place, which allows for efficient storage, fast query processing and easy consistency checks.

9.3 Tracers

Section 3.4 introduced tracers as a mechanism for demarcating processes. Tracers are tokens that are passed between actors based on the actor's internal knowledge and the semantics of the tracer. The semantics of the tracer are the rules used by actors to determine when to generate and/or propagate tracers. This section describes several tracers and their semantics. If an actor follows the expectations placed on it by a tracer, more detailed process documentation can be recorded.

We begin by defining the notion of a computational activity. This definition is derived from the definition of an activity in WS-Context [LNP04]. A computational activity is a conceptual grouping of actors cooperating to perform some work. It represents the execution of a series of related interactions between a set

of actors; these interactions are explicitly related via a tracer. This notion can be used to scope processes.

For the purpose of this discussion, we assume a pure client-server model, in messages are categorised into requests and responses, and in which all requests to an actor are followed by a response from that actor to the originator of the request. In the future, we shall relax this assumption in order to deal with non client-server interactions.

Given this assumption, an actor, A, is said to be inferior to another actor, B, if A received a request from B. Likewise, an actor B is said to be superior to an actor, A, when it sent a request to A. Using these definitions, we can now present several tracers.

Additionally, we define the notion of a task. A task is a independent computation within an actor that has a defined start point and end point. A request-response pair defines the start point and end point of a task, respectively.

Each tracer is defined by a generation rule and several propagation rules. A generation rule defines when an actor should create a tracer. A propagation rule details when an actor should propagate a tracer that it has received in a request.

9.3.1 Session Tracer

A session tracer is defined by the following three rules.

Rule 9.5 (Generation Rule) *An actor must generate a new session tracer at the start of each task and add the tracer to all requests within that task.* □

Rule 9.6 (Propagation Rule: to inferior) *An actor must add any session tracers received from a superior actor to all requests it makes to inferiors within the task started by the superior's request.* □

Rule 9.7 (Propagation Rule: to superior) *An inferior actor must add the session tracers supplied by its superior to its response to the superior.* □

Figure 9.1 shows an example of how session tracers are generated and propagated. Each actor is a box. Tracers are labelled by a lower case letter. In this example, a GUI invokes a workflow enactment engine with a tracer a. The enactment engine invokes the actors C and D with the tracer a and b. The enactment engine passes along the tracer according to propagation rule 1 and adds its own tracer to the request according to the generation rule. Actor C invokes the actor E within the task started by the request from the enactment engine. C passes the tracers it receives along with its own tracer to E, which returns a response to C. The response contains the tracers a, b, and c per propagation rule 2. Each actor responds to its superior until the GUI receives a result from the enactment engine and finishes its task. In this example, a computational activity (the execution of a workflow) is defined by the tracer a.

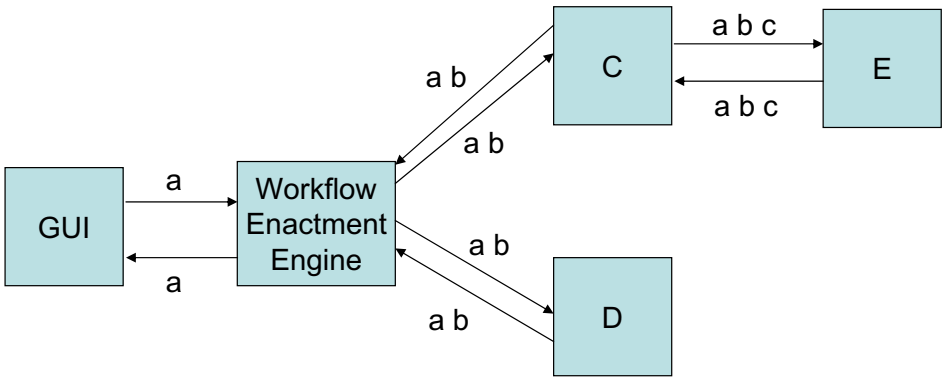


Figure 9.1: Diagram showing session tracers

9.4 Security

Here, we discuss the constraints on the behaviour of the actors as well the provenance system as a whole in order that process documentation be recorded securely and in a non-repudiable fashion. For the case of actors, we can introduce two rules:

Rule 9.8 (Signature Rule) *Asserting actors must digitally sign the p-assertions they create.*□

The designated locations for the signatures are shown in Figure 7.10, 7.11 and 7.12. This rule ensures that liability for the information contained within a p-assertion can be traced back to its creator.

Rule 9.9 (Mutual Authentication Rule) *Recording actors must mutually authenticate with the provenance store that they record their p-assertions to.*□

This rule ensures that the identity of the recording actor can be extracted for access control purposes as detailed in Chapter 4.3.1. Conversely, the recording actor needs to ensure that it is submitting its p-assertion to the intended provenance store. In order to allow these two constraints on actor behaviour to be enforced, an additional constraint may be required on the provenance system as a whole. We describe this as a rule as well:

Rule 9.10 (Security Environment Rule) *The provenance environment should provide the necessary security functionality and credentials that actors require in order to behave in a secure manner, as well as the information required to use these credentials properly.* □

An example of such security functionality could be a signature generating algorithm, while credentials such as certificates could be provided through key stores. Information would include components like a security policy that actors can consult in deciding the manner in which they interact securely between themselves or with the provenance store.

Independently of the p-assertions they assert and record to provenance stores, application actors may also choose to interact with each other in a secure manner as well. The rules constraining their behaviour in this instance will be entirely dependent on the requirements of the application domain they operate in, and so is outside the scope of discussion of this document. However, the nature of the security-specific interactions that they engage in may need to be reflected in the process documentation asserted pertaining to their interaction as a whole. Below, we provide some constraining rules on the types of p-assertions that need to be produced by the participating actors as a result of a security-specific interaction.

Rule 9.11 (Error Message Rule) *Security-specific error messages and exceptions exchanged between actors should be asserted as p-assertions in the same manner as normal interactions.* □

As an example, a client application actor invoking a service actor may not be authenticated properly or does not have the appropriate access rights corresponding to its request on that service. The service actor will return an appropriate error message indicating the appropriate fault; this message should be documented as a p-assertion in the normal manner by both client and service.

Rule 9.12 (Relationship Rule) *When security interactions utilize credentials that reflect relationships between several identities, these relationships should be asserted appropriately as a relationship p-assertion.* □

As an example, in Section 4.4.3, we note the need for delegation of access control or identity could potentially arise amongst application actors if the various actors involved in a chain of invocations are located in separate security domains. If a delegated credential is used by a client when authenticating to a service during an interaction, then an appropriate relationship p-assertion reflecting this delegated relationship should be created in addition to the standard interaction p-assertion.

Rule 9.13 (Tokens Rule) *If security tokens (such as signatures) are used in specific portions of messages exchanged between actors, these tokens should also be included with the messages when they are recorded as p-assertions.*□

As an example, the protocol dictating interaction between two application actors may dictate that certain parameters in the exchanged messages be signed for purposes of non-repudiation. In that case, both actors must ensure that

these tokens are recorded appropriately in the message content portion of the interaction p-assertion if a direct copy documentation style is employed (Figure 7.5).

9.5 Conclusion

This chapter has defined a number of expectations on actor behaviour. This expectations include a set of architectural rules, a description of tracers, along with a set of security considerations. Actors must follow these expectations in order for process documentation to be correctly recorded, managed, and queried.

Chapter 10

Justification

This chapter briefly describes how the software requirements for provenance system are satisfied by the functionality provided by the logical architecture described in Chapter 3. These requirements are sourced from the software requirements document [AV05a] and are listed in the remaining sections of this chapter along with the corresponding feature of the logical architecture that addresses it. The notation (NA) is used to indicate that the software requirement refers to a specific functionality which is not applicable at the level of the logical architecture. The classification of requirements in the following sections mirrors those in the original software requirements document.

10.1 Functional Requirements

SR-1-1: The provenance architecture should provide for the recording and querying of interaction and actor provenance.
Design Feature: The architecture includes submission, query and management interfaces which can be used to retrieve one or more p-assertions from the Provenance Store.

SR-1-2: The provenance architecture should allow the retrieval of a provenance trace from the Provenance Store. Either a complete trace or a subset may be retrieved.
Design Feature: The architecture provides a query interface with multiple levels of query capability that supports the required granularity of trace to be retrieved. Intermediate results can be stored locally before being returned.

SR-1-3: The provenance architecture should allow the back-up of a Provenance Store to be taken. This will generally include an archiving facility that allows data within a Provenance Store to be saved for future use.
Design Feature: Retrieval of data for purposes of archival is supported through

appropriate use of query and management interface functionality; the specific functionality of archival is outside the domain of the logical architecture.

SR-1-4: The provenance architecture should allow comparisons to be made across Provenance Records within a Provenance Store with reference to particular data attributes within a Provenance Record.
Design Feature: The architecture provides a query interface with multiple levels of query capability and granularity. For example, at the level of a single interaction or actor state p-assertion, or the specific items within each of these. The queries can be generated in the appropriate format to allow accessibility at the desired level.

SR-1-5: The provenance architecture should allow the results of a query to the Provenance Store to be captured for future use. A query in this context must be specified with reference to the structure of the Provenance Store.
Design Feature: The results of a query are returned in an appropriate format that reflects the structure of the Provenance Store, and this result can be subsequently stored for future use. The nature of this storage is outside the bounds of the logical architecture.

SR-1-6: The provenance architecture should allow a user to access a Provenance Record based on the time and date (calendrical information) at which the Record was stored.
Design Feature: The Recording interface at the Provenance Store should attach a timestamp with each p-assertion sent to the Provenance Store. This timestamp may be in addition to one generated by the application submitting the p-assertion.

SR-1-7: The provenance architecture should allow a user to verify the contents of a Provenance Store against a specified set of rules. Verification in this context means that the contents of the Provenance Store meet the set of constraints expressed by the set of rules.
Design Feature: The trace comparison service or semantic validity analyzer (or some other equivalent service) that form part of the processing services group can be used here.

SR-1-8: The provenance architecture should allow a user to specify a time period in the future at which a provenance query may be submitted to a Provenance Store. A scheduler will be made available that allows queries to be stored to disk, and dispatched to the store in the future.
Design Feature: The scheduling functionality can be achieved by a service that forms part of the processing services group. A notification service could be provided to alert the user when the Provenance Store is ready to accept queries.

SR-1-9: The provenance architecture should allow capabilities provided by the tools to be accessible as an API. This is to allow such capabilities to be embedded within an existing application.
Design Feature: These APIs can be exposed at the actor side query, management and submission libraries respectively. The provenance architecture also provides the possibility of exposing the software interfaces for all functionality through a registry service, as part of the processing services group.

SR-1-10: As part of the initialisation of the provenance recording process, the provenance architecture should allow a service or user to specify the identity of the Provenance Store to which data should be recorded.
Design Feature: The specification of a specific Provenance Store to be used can be specified as part of a policy configuration decision by the service or a user, or as negotiation between two actors in the P-Assertion Recording Protocol (PReP). The interfaces should support the acquisition of appropriate identity information for a Provenance Store.

(NA) *SR-1-11*: The system should support the multiple storage of a provenance record, i.e. the system should provide a way to store copies of a provenance record in more than one repository.
Design Feature: Copies of provenance records can be obtained via the query / management interfaces; however, storage and retrieval across multiple repositories is a storage layer issue that cannot be addressed by the logical architecture.

SR-1-12: The system should support the recording of different provenance information views related to an event or an entity.
Design Feature: PReP provides the facility of different views of interaction p-assertions.

SR-1-13: The provenance architecture should support the migration of provenance data among Provenance Stores.
Design Feature: Query, management and submission interfaces allow the extraction of process documentation and transferring it to a different store. In addition, multiple physical Provenance Stores corresponding to a single logical Provenance Store is supported; this will also include support for migration of process documentation among stores.

(NA) *SR-1-14*: The system should support the storage of recorded provenance data for an indefinite period of time.
Design Feature: This is a non-functional requirement that cannot be addressed by the logical architecture.

(NA) *SR-1-15*: The provenance architecture should support the storage of results

of analysis and reasoning operations performed on the provenance data by tools that are not part of the generic architecture (3rd party tools on the application layer).
Design Feature: Such storage functionality is beyond the domain of the provenance architecture.

SR-1-16: The provenance architecture should provide support for maximum automation of the provenance recording mechanism.
Design Feature: The management and application UIs, presentation and processing services provide a high level of automation of the entire activity of provenance storage and processing.

SR-1-17: The provenance architecture should be deployable as an integrated part of a system, as a service within the same administrative domain as the client system and as a 3rd (external) party operated service, too.
Design Feature: The APIs of the query, submission and management libraries allow integration into the existing system, whilst the UIs, processing and presentation services allow the provenance architecture to be used as a external 3rd party service. The derivation engine of the provenance security architecture allows policies of the client domain to be reflected into the provenance access control functionality.

(NA) *SR-1-18*: Client side components of the provenance architecture should not block an executing workflow if any provenance services are unavailable.
Design Feature: No requirements on synchronous or asynchronous behaviour should be assumed. This is an implementation issue and hence not applicable within the context of a logical architecture.

10.2 Performance Requirements

(NA) *SR-2-1*: The additional execution overhead for an application recording provenance information should be kept to a minimum.
(NA) *SR-2-2*: Storage space requirements of the provenance architecture for provenance information recording should be kept at a reasonably low level.
SR-2-3: The provenance architecture should guarantee reliable once-and-once-only delivery of provenance information to and from a Provenance Store.
Design Feature: The logical architecture supports the use of recording protocols for interacting with the Provenance Store that guarantees this property.

(NA) *SR-2-4*: The provenance architecture should be capable of handling large amounts of provenance data submitted frequently by user applications. The

provenance architecture should not be the cause of any bottlenecks in the overall system due to the processing of provenance data.
Design Feature: The protocol used for submission in the logical architecture provides a way for dealing with large amounts of data being transferred through appropriate recording patterns.

10.3 Interface Requirements

SR-3-1-1: All of the functions of the provenance architecture should be accessible through its API so it can be used as an embedded component in a system.
SR-3-1-2: The provenance architecture should support a rich set of published, generic application programming interfaces (APIs) that allow application specific analysis and reasoning tools to be built upon.
SR-3-1-3: The provenance architecture should provide a programmatic interface for the administration of the system.
Design Feature: These are all catered for via the management and provenance querying actor side libraries. The Provenance Architecture should make no assumptions about the contents of this API, or the particular protocol that is used to access services made available through this API.

(NA) *SR-3-1-4*: The provenance architecture should support an XML-based API format for provenance data.
(NA) *SR-3-2-1*: Export formats for provenance data should be non-proprietary to allow tools and applications to be built without violating IPR rules. A format based on an existing data representation standard (with special focus on XML defined by XML Schema) would be highly preferred.

10.4 Operational Requirements

SR-4-1: Provenance information displayed by the provenance architecture on a human computer interface (HCI) should be updatable on user request.
Design Feature: The Provenance Architecture should make no assumption about the types or modes of user interfaces being supported.

SR-4-2: HCIs presented by the provenance architecture for displaying the contents of a Provenance Store should support continuous monitoring, i.e. the displayed information should be updated automatically on every change as soon as possible.
Design Feature: These can be handled by a suitably developed functionality in

the trace visualizer / browser.

SR-4-3: The update frequency of provenance information displayed by the system on a HCI should be configurable based on policies.
Design Feature: A policy for presentation/processing services akin to those for Provenance Store and user requirements can be used.

SR-4-4: Human-computer interfaces presented by the provenance tools should be designed to allow multilingual support.
Design Feature: This can be handled by a suitably developed functionality in the trace visualizer / browser.

10.5 Documentation Requirements

(NA) *SR-5-1*: Detailed documentation of the provenance architecture public interfaces should be produced both for APIs and HCIs.
(NA) *SR-5-2*: A detailed description of the administrative interface of the provenance architecture should be produced.

10.6 Security Requirements

SR-6-1: The provenance architecture should have a configurable access control system over the resources it provides, with a granularity that is sufficient to protect these resources.
Design Feature: The authorization engine and list are configurable through suitable access control policies in the security architecture. The granularity of protection will be determined on the basis of the granularity of the provenance information being stored (i.e. p-assertions).

SR-6-2: The provenance architecture should allow both automated and manual determination of access control rights.
Design Feature: The access control policies in the security architecture are manually determined through the management interface. The trust derivation engine allows automated generation of access control rights on the basis of access control statements accompanying provenance information that is meant to be stored.

SR-6-3: The provenance architecture should allow a service or user to request the level of security they wish to be associated with the recording process. The level of security can range from no security through encrypted data transfer to

more complex security mechanisms.
Design Feature: This can be specified through the user requirement policy on the user side and through the access control policy in the security architecture of the provenance system.

SR-6-4: The provenance architecture should provide a way to map access rights information of embedding systems into its security subsystem.
Design Feature: The derivation engine of the provenance security architecture allows policies / access control rights of the embedding system to be reflected into the provenance access control functionality.

SR-6-5: Security related procedures for accessing the provenance system should be subsumed under the existing security related procedures for the embedding system if possible, so that changes or additions to the existing procedures are minimized.
Design Feature: The derivation engine of the provenance security architecture allows policies / access control rights of the embedding system to be reflected into the provenance access control functionality. In addition, the identity validator can accept a multiplicity of security credentials, including the ones used for authenticating to the embedding system.

SR-6-6: The provenance architecture should provide a mechanism for recording provenance data in an unmodifiable form and also ensuring that the party responsible for the recording process cannot deny having recorded that provenance data.
Design Feature: Provenance submission interface supports the submission of signed data items, and the domain specific services for the appliaction should include functionality for accessing certificates in keystores and signing submitted p-assertions. No assumptions should be made about the recording format used.

SR-6-7: The provenance architecture should provide a mechanism for the authentic timestamping of provenance records. Authenticity should be guaranteed by the mechanism on a level that is enough even for the use in legal procedures.
Design Feature: The application domain specific services in the architecture provide an interface to external third parties or applications that can provide this functionality, or may implement the functionality itself.

10.7 Other Requirements

(NA) *SR-7-1*: The provenance architecture should have the properties of cost efficiency and robustness versus an in-application hand-engineered logging system.

SR-7-2: The provenance architecture should be loosely coupled and independent from the applications as much as possible. Integration costs for existing systems should be minimal, ideally existing system components should remain unaffected. *Design Feature*: Addressed by features for *SR-1-17, 6-4, 6-5.*

10.8 Conclusion

The vast majority of software requirements appear to have been addressed adequately by the logical architecture. Those that have not are either implementation or storage layer requirements that can only be fulfilled at a later stage of the development life-cycle.

Chapter 11

Related Work

This chapter gives a literature review about provenance and provenance systems in different areas.

11.1 Current Practices of Provenance in Museum, Library and Archival Management Systems

Apart from as an English word, provenance has been formally and explicitly defined and used in the several domains, in particular, in museum, library and archival management systems. In Dublin Core Application Profile (DCAP), provenance is defined as a statement of any changes in the ownership and custody of the resource that are significant for its authenticity, integrity or interpretation. In the Dublin Core-based DSpace metadata schema for DSpace, a digital library system to capture, store, index, preserve, and redistribute the intellectual output of a university's research faculty in digital formats, provenance is defined as The history of custody of the item since its creation, including any changes successive custodians made to it. Two most related works are as follows. The International Standard for Archival Description (ISAD(G)), Second Edition (2000) is a descriptive standard for archival records. It can be applied to units of description at any level from the collection to the individual item, and includes two pieces of related information.

- Archival history

  Purpose: To provide information on the history of the unit of description that is significant for its authenticity, integrity and interpretation.

  Rules: Record the successive transfers of ownership, responsibility and/or custody of the unit of description and indicate those actions, such as history of the arrangement, production of contemporary finding aids, re-use of the

records for other purposes or software migrations, that have contributed to its present structure and arrangement.

- Immediate source of acquisition

  Purpose: To identify the immediate source of acquisition or transfer. Rules: Record the source from which the unit of description was acquired and the date and/or method of acquisition if any or all of this information is not confidential.

The Encoded Archival Description (EAD) DTD is a standard for encoding archival finding aids using SGML or XML. It includes two pieces of information:

- Provenance

  Information about the chain of ownership of the materials being described, before they reached the immediate source of acquisition. Both physical possession and intellectual ownership can be described, providing details of changes of ownership and/or custody that may be significant in terms of authority, integrity, and interpretation.

- Acquisition Information

  The immediate source of the materials being described and the circumstances under which they were received. Includes donations, transfers, purchases, and deposits.

As can be seen from the above provenance definitions, standards and use context, provenance in these domains is mainly referred to the acquisition and creation information, and the history of the ownership and custody of a resource (description or data).

11.2 The State of Art of Provenance Systems

At the beginning of this report, we defined provenance as the process that leads to a result. Prior research has referred to this concept using several other terms including audit trail, lineage [Lan91b], and dataset dependence [AH97]. We use these terms interchangeably to refer to the process that leads to a result. Much of the literature considers what we term a provenance system, i.e. a system that records the documentation of a process and allows a representation of the provenance of a result to be retrieved. The literature can be divided into four categories: fine granularity provenance systems, domain specific provenance systems, provenance in database systems, and middleware provenance systems. This review gives a brief summary and analysis of the literature pertaining to each of these categories.

11.2.1 Fine Granularity Provenance Systems

The following systems record the documentation of script or program execution thereby allowing a representation of the provenance of a script/program's result to be retrieved. These systems differ from workflow centric systems because of the finer granularity of process documentation these systems achieve. By granularity of documentation, we mean how detailed the documentation of process is. If a system records all the instructions in a program whereas another system records the name of the program being run, the first system will record a finer granularity of documentation. With finer granularity documentation the corresponding representation of provenance for a result can be more detailed.

One example is the Transparent Result Caching (TREC) prototype [VA98]. TREC uses the Solaris UNIX proc system to intercept various UNIX system calls in order to build a dependency map. Using this map, a trace of a program's execution can be transparently captured, which can be used to keep Web page caches current and to provide an 'unmake' function. Although TREC has several limitations, including high overhead, it is interesting case for deteriming the bounds on process documentation granularity.

Another technique for capturing fine granularity process documentation is [Mar01] sub-pushdown algorithm. This algorithm can only be used to record documentation of array operations in the Array Manipulation Language and was implemented in a prototype database system, ArrayDB. In this system the provenance of an array in ArrayDB can be retrieved. A more comprehensive system is the audit facilities designed for the S language [BJMC88]. S is an interactive system for statistical analysis. The result of users command are automatically recorded in an audit file. These results include the modification or creation of data objects as well as the commands themselves. The AUDIT utility can then be used to analyse the audit file to retrieve the provenance of a statistical analysis. This utility can also create a script to reexecute a series of commands from the audit file.

A similar fine granularity technique for recording the documentation a process has been used in security for mobile agent systems. Using a technique called interaction tracing, a user sending a mobile agent can verify that it has correctly executed on the host platform. Interaction tracing treats a mobile agent as a black box, recording all the inputs/outputs of the executing agent [Tan04]. Although interaction tracing is not concerned with the provenance of a result it presents a novel notion for recording process documentation.

11.2.2 Domain Specific Provenance Systems

Much of the research into provenance has come in the context of domain specific applications. Some of the first research in provenance was in the area of geographic information systems (GIS)[Lan91b]. Knowing the provenance of map

products is critical in GIS applications because it allows one to determine the quality of those derived map products [Lan91b]. Lanter developed two systems for recording process documentation and retrieving the provenance of map products in a GIS. The first system was a meta-database for recording documentation of a GIS process. The second system was for tracking Arc/Info GIS operations from a graphical user interface with a command line [Lan91a, LE91]. The workflow centric user interface was integrated into a software product called Geolineus, which was one of the few lineage systems to be incorporated into a commercial software product [Bos02]. Another GIS system that includes process tracking is Geo-Opera, which is based on a non-domain specific software [AH97]. Many of the ideas in Geo-Opera are extended from GOOSE, which uses data attributes to point to the latest inputs/outputs of a data transformation. All inputs/outputs must be stored in GOOSE and data transformations are programs or scripts [AA93]. Both GOOSE and Geo-Opera are workflow based systems.

Another domain where provenance is of interest is satellite image processing. The Earth System Science Workbench (ESSW) is designed for processing satellite imagery locally. It provides a lab notebook service for tracking processing steps and a No-Duplicate Write Once Read Many storage service for storing files. Essentially, as a workflow is run inside ESSW the results of each metadata described step is stored in the lab notebook service. ESSW is interesting because it emphasises the need to have immutable process documentation. In chemistry, the CMCS project has developed a system for managing metadata in a multi-scale chemistry collaborations [MPL+03]. The CMCS project is based on the Scientific Application Middleware project [MCE+03], which we discuss in greater detail later in this review. Another domain where provenance tools are being developed is bioinformatics. The myGrid project has implemented a system for recording the documentation of process in the context of in-silco experiments represented as workflows aggregating Web Services [GGS+03]. In myGrid, documentation is recorded about workflow execution and stored in the user's personal repository along with any other metadata that might be of interest to the scientist [ZGG+03]. Using this personal repository the provenance of bioinformatics results can be determined. The focus of myGrid is personalising the way provenance is presented to the user. MyGrid highlights the need for provenance in the bioinformatics domain.

The needs of particular domains has led to the development of specific systems for recording domain dependent process documentation, which allow the provenance of results to be retrieved. The majority of the systems are designed for one particular domain and are not general in nature. However, they do provide insight on how a general provenance system might be designed to meet the needs of a variety of domains. For example, the systems tend to use a workflow centric approach. Also, the systems highlight the need to both record the set of transformations as well as the data used in a process.

11.2.3 Provenance in Database Systems

Provenance in database systems has focused on the data lineage problem [CWW00]. This problem can be summarised as given a data item, determine the source data used to produce that item. [WS97] look at solving this problem through the use of the technique of weak inversion. Given some output data and a weak inversion function f^{-w}, f^{-w} attempts to lazily recreate the input data used to generate the output. Unfortunately, this requires that a user who creates a new database view must also define an inversion function for that view. This technique has been used to improve database visualization [Woo98]. [CWW00] formalises the data lineage problem and presents algorithms to generate lineage data in relational databases. The generation algorithms are similar to automatically creating weak inversion functions for every new view in a database, which allows users to "drill through" the lineage of a data item seeing the source data (tuples) that contributed to the given data item [CW00]. This work was also extended to deal with general transformations of data sets inside a data warehouse [CW03]. Another system that looks at the data lineage problem in a data warehouse context is AutoMed [FP03]. Process documentation is recorded in AutoMed by recording schema transformations. A series of schema transformations is termed a schema transformations pathway. From these transformation pathways, the data lineage of a data item can be retrieved. The granularity of this approach depends on the granularity and number of schemas defined in the system.

[BKT01] redefines the data lineage problem as "why-provenance" and defines a new type of provenance for databases, namely, "where-provenance". "Why-provenance" is why a piece of data is in the database, i.e. what data sets (tuples) contributed to a data item, whereas, "where-provenance" is the location of a data element in the source data. Based on this terminology a formal model of provenance was developed applying to both relational and XML databases. In other work, [BKKT02] argue for a time-stamped based archiving mechanism for change tracking in contrast to diff-based mechanisms. Diff-based mechanisms may not capture the complete process of database modification because there may be multiple changes between each archive of the database. Therefore, a diff-based mechanism is not a reliable approach for the development of a general provenance system.

The research into provenance in database systems is well grounded, systematic, and formal. Because databases have a well defined and fixed set of transformations, the community has focused mainly on the data lineage problem. However, in distributed systems the number of transformations is infinite, therefore, work is needed in developing systems that can handle any kind of transformation. [CW03] is a first step toward addressing this problem but it only pertains to one particular context, the data warehouse.

11.2.4 Middleware Provenance Systems

Several middleware systems have been developed to provide provenance support to applications. These systems aim to provide a general mechanism for recording process documentation and retrieving provenance for use with multiple applications across domains and beyond the confines of a local machine.

[RXBR04] presents a system based around the concept of an e-notebook. Each user is required to have an individual e-notebook which can record data and transformations either through connections directly to instruments or via direct input from the user. Data stored in an e-notebook is represented as a DAG and can be shared with other e-notebooks via a peer-to-peer mechanism. A DAG may span multiple e-notebooks to take in account multiple individuals participation in a process. To enable support of trust views and credential tracking each node in a DAG must be digitally signed by the node's creator. This system is interesting because of its use of the notebook metaphor and its approach to trust of the stored process documentation. However, there are questions as to whether this approach is appropriate for large scale distributed systems in terms of scalability.

Another system supporting provenance is Scientific Application Middleware (SAM) [MCE+03]. SAM provides facilities for storing and managing records, metadata and semantic relationships and is built on the WebDav standard. Support for provenance is provided through adding metadata to files stored in a SAM repository. SAM is of interest because it does not specify the format of the data or metadata that it handles instead it acts as an open repository. However, this lack of structure means that retrieving the provenance of a result in its entirety may be difficult.

The Chimera Virtual Data System is a virtual data catalogue, which is defined by a virtual data schema and accessed via a query language [FVWY02]. The schema is divided into three parts a transformation: a derivation and a data object. A transformation represents an executable, a derivation represents the execution of a particular executable, and a data object is the input or output of a derivation. The virtual data language provided by Chimera is used to both describe schema elements and query the data catalogue. Using the virtual data language a user could query the catalogue to retrieve the DAG of transformations that led to a result. The benefit of using a common description language is that relationships between entities can be extracted without understanding the underlying data by analysing the transformation descriptions. Process documentation in Chimera is stored in the Provenance Transformation Catalog (PTC). The data stored in the PTC is limited to a defined schema and does not allow for arbitrary information. Likewise, there is currently no support for associating data in the PTC making determining data lineage difficult. Chimera is of interest because it is specifically designed to work with large scale distributed systems.

[SM03a] argued for infrastructure support for recording process documentation in Grids and presented a trial implementation of an architecture that was

used to demonstrate several mechanisms for handling the documentation after it was recorded. The system is based around a workflow enactment engine submitting data to a provenance service. The data submitted is information about the invocation of various Web Services specified by the executing workflow script. This system is of interest because it is both Web Services based and was designed with Grid applications in mind. The drawbacks of this system include it relies completely on a workflow enactment engine and it lacks a demonstration in any large scale system.

Chapter 12

Conclusion

12.1 Summary

In this document, we have presented a logical architecture for provenance systems and the accompanying details required to understand how such an architecture functions. We have proposed a definition of provenance suitable for representation in a computational system, in Chapter 2, as "The provenance of an entity in a given state is the process that led to that entity being in that state.", and a concrete representation of this concept, *process documentation*, in terms of interactions between actors and states of those actors during interaction. In Chapter 3, we present the logical architecture itself which defines the components of a system for the recording, maintaining, visualising, reasoning and analysis of process documentation.

Chapters 4 and 5 address the security and scalability aspects of the architecture respectively. A security architecture, complementary to the logical architecture, is presented that provides secure transmission and access control to provenance stores, and a series of scenarios is given to illustrate how different modes of interaction with the secured system will take place. For scalability, the need for distribution of provenance stores is emphasised, and a set of deployment patterns for recording of process documentation to distributed stores is given.

In Chapters 6, 7, and 8, we provide more detail about the functionality of the provenance architecture. Chapter 6 presents solutions to the problem of discovering the provenance of a given entity: in particular, how an entity should be identified in queries to provenance stores. Chapter 7 describes the underpinning data model of the provenance architecture. In Chapter 8, we detail the functionality available through the three interfaces of a provenance store: *recording*, *query* and *management*. In Chapter 9, we enumerate the constraints that application actors have to satisfy in order to successfully record documentation of execution and issue queries about the provenance of data.

Finally, Chapters 10 and 11 place the work in context. In the former chapter,

we compare the logical architecture to the software requirements captured in the EU Provenance project, showing how each is addressed by our design, and in Chapter 11 we discuss related work.

12.2 Future Work

It should be emphasised that this document continues to evolve. Chapters will be frozen following the timetable in Section 1.3. Along with this work, a set of associated documents will be produced by the EU Provenance project.

- An instantiation of the logical architecture for the Web Services stack.
- A standardisation proposal for the logical architecture.
- A design for implementation of the architecture.
- A methodology explaining how to deploy the provenance architecture.
- An application of the methodology and architecture to an organ transplant management system.
- An application of the methodology and architecture to an aerospace engineering system.

Appendix A

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). It has a beginning, and, if it is finite, it has an end.

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: Should the actors involved in the process be the only one to document it? The answer is yes. Indeed, if actors are not involved in the process, then no message has been sent to them. Hence, they cannot be aware of the process, and therefore could not possible provide any documentation relevant to this specific execution.

Note 7: In a grid application based on command line executables, an interaction p-assertion can include the executable fully qualified name, its inputs and its outputs, whereas in a Web Services based approach, interactions documentation can 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 potentially includes not only the SOAP message body, but also its envelope, containing valuable information such as security, addressing, resource or coordination contexts.

Note 8: We note that capturing such data dependencies in large scale databases is the focus of research on data provenance in databases; techniques developed in such a context may have to be integrated with the proposed approach.

Note 9: 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, and extended with some hand written convenience functions.

Appendix B

Abbreviations

This appendix contains abbreviations used in this document.

CA	Certificate Authority
GPAK	Global P-Assertion Key
GPID	Global P-Assertion Identifier
IR	Internal Representation
PReP	P-Assertion Recording Protocol
RBAC	Role Based Access Control
UI	User Interface

Appendix C

XML Schema diagram format description

Figure C.1 gives an example of a small XML Schema file displayed as a diagram. We will now explain the format of the diagram with reference to this figure.

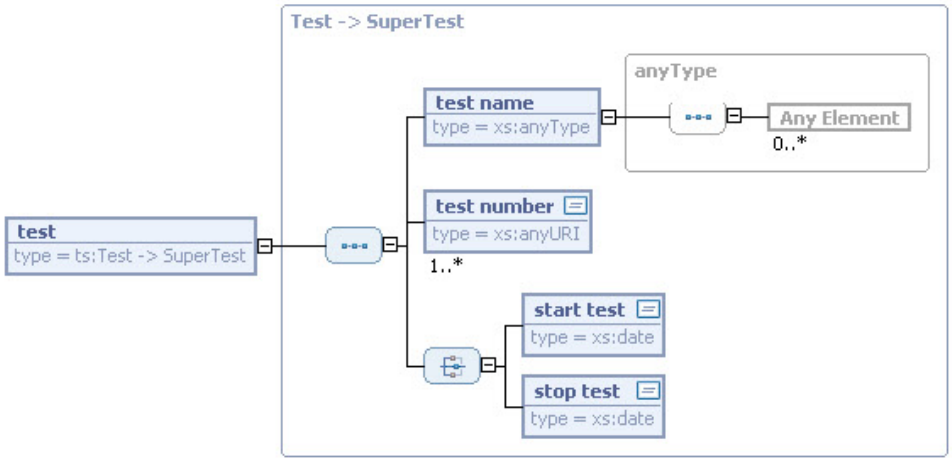
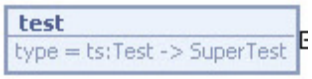


Figure C.1: An example XML Schema file diagram

Figure C.1 shows an element (i.e. an instance) named test that is a type of ts:Test. The element definition is in the box to the left of the picture. The type description is to the right of the element. The type Test extends the type SuperTest as denoted by the arrow notation. The type Test contains a sequence of elements, which we now detail. One element in the sequence is a test name which can be any type and must occur once and only once in an instance of Test. We note that the definition of the type anyType is in a lighter tone than the definition of the Test type. This is because the type anyType is defined in another schema and is referred to by this schema. Test also contains an element

test number which is a simple type as denoted by the box next to the name. The test number element must occur once and can occur as many times as needed. This is denoted by the 1..* under the element. The sequence also contains a choice between two elements; start test and stop test.

Below is a simple of description of each of the parts of the XML Schema diagram format.



Represents an element (instance). The type of the element is defined underneath the element name.

By default an element must occur once and only once. This can be changed and is denoted by the 1..*, 0..*, 0..1 under the element box. If the element is a simple type a small box is put to the right of the element's name.

Represents a type. A type is denoted by a box with a name at the top left. All the graphical elements within the box are a part of that type except for other types.

Represents a sequence of elements.

Represents a choice between elements.

Appendix D

The Actor State Model

Different applications might need roughly the same types of information to represent an actor's state but could use different terminology. To enhance interoperability and use of provenance systems, it is essential for applications to assert actor states based on as much as possible the commonly agreed information items as well as their models. We have analysed provenance user requirements [AV05b] and identified a number of common information that, once recorded through actor state p-assertions, can satisfy provenance use cases. The information is described below.

- The Profile, which provides basic information on an asserting actor such as contact information, actor's attributes, status, version, etc.
- The Site, which provides location information about an asserting actor. A site can hosts zero or more actors.
- The Cluster, which provides information about the set of subclusters or hosts that offer an execution environment for an asserting actor.
- The ComputingElement, which provides information about the characteristics, resource set and policies of the underlying scheduling system in which an asserting actor is running.
- The InvocationMetadata and ResultMetadata, which provide metadata about an asserting actor's invocation and result messages. They include message receiving and sending time, inputs/outputs metadata and their storage information, and their semantics.
- The Functionality, which provides information about the algorithms, executables and resources an asserting actor uses to realise its functionality.
- The WorkflowInfo, which provides information about the workflow in which an asserting actor is invoked and executed.

- The WasBeforeInteraction, which specifies the temporal relation between an actor state p-assertions and its corresponding interaction.

We define data types for the above information items. Given that the Grid Laboratory Uniform Environment, i.e., the GLUE schema [Glu], has been commonly accepted as an information model for grid resources and already incorporated into Globus toolkits, we have reused some data types from the GLUE schema. We also draw ideas from the ongoing work on the semantic rich Grimoires service registry [gri04] and the research results in semantic web service, in particular, the OWL-S service ontology. Our design rationale is to make data models for describing actor states compatible with available or forthcoming standards or endorsed specifications. The benefit is that an open standards-based model enables the provenance infrastructure to coexist with other service-oriented infrastructure so that provenance systems can interact and integrate with other systems and application seamlessly.

To provide design guidance and a template for creating actor state p-assertions, an abstract, generic model for actor state p-assertions, called ActorStatePAssertion, is defined in Section 7.7. Any application-specific actor state p-assertions model should be derived from this model to guarantee consistency and extensibility. Furthermore, we have designed an example actor state p-assertions model, which is derived from the generic ActorStatePAssertion model and named as ActorStateModel. The ActorStateModel consists of all the information we identified above, as depicted in Figure D.1.

We recommend using ActorStateModel as a general actor state p-assertion model. However, we do realise that different domains and applications have their own characteristics and may need to capture and record specific or proprietary data. Therefore, we do not mandate any actor state data model to contain the identified information nor do we limit it for adding any extra information. The idea is that we encourage applications to adopt the general actor state p-assertion model so as to achieve some degree of interoperability. However, applications have the flexibility to add their own data types and even to develop their own actor state p-assertions model.

An application can design its own actor state p-assertions model in the following three ways.

- Derive the actor state p-assertion model from the ActorStateModelType by extension or restriction, define and add its own application-specific data types into the model to meet its specific provenance needs.

- Derive the actor state p-assertion model from the ActorStatePAssertion data type, reuse part (or all) of these pre-defined data types as described above, define its own application-specific data types if necessary, and use them in the model.

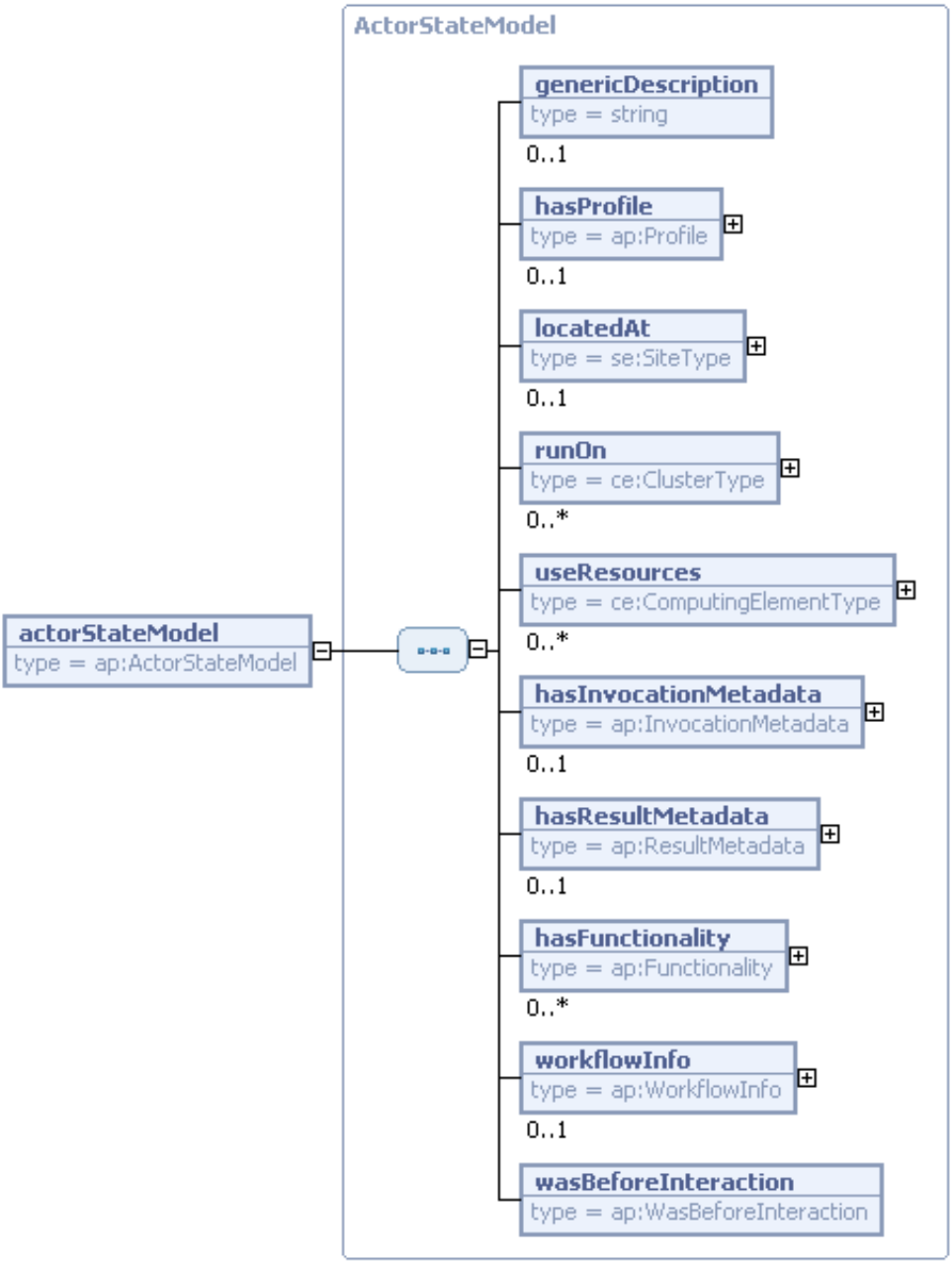


Figure D.1: The actor state p-assertion model in XML Schema

- Derive the actor state p-assertion model from the ActorStatePAssertion data type, define its own data types and use them in the model.

Bibliography

[AA93] G. Alonso and A. El Abbadi. Goose: Geographic object oriented support environment. In *Proc. of the ACM workshop on Advances in Geographic Information Systems*, pages 38–49, Arlington, Virginia, November 1993.

[AH97] G. Alonso and C. Hagen. Geo-opera: Workflow concepts for spatial processes. In *Proc. 5th Intl. Symposium on Spatial Databases (SSD '97)*, Berlin, Germany, June 1997.

[AIS77] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. *A Pattern Language*. Oxford University Press, 1977.

[Ale79] Christopher Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.

[AV05a] Árpád Andics and Laszlo Varga. Software requirements document. Technical report, MTA SZTAKI, February 2005.

[AV05b] Árpád Andics and Laszlo Varga. User requirements document. Technical report, MTA SZTAKI, February 2005.

[BJMC88] R. A. Becker and J. M. J. M. Chambers. Auditing of data analyses. *SIAM Journal of Scientific and Statistical Computing*, 9(4):747–760, 1988.

[BKKT02] P. Buneman, S. Khanna, K.Tajima, and W.C. Tan. Archiving scientific data. In *Proc. of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 1–12. ACM Press, 2002.

[BKT01] P. Buneman, S. Khanna, and W.C. Tan. Why and where: A characterization of data provenance. In *Int. Conf. on Databases Theory (ICDT)*, 2001.

[Bos02] R. Bose. A conceptual framework for composing and managing scientific data lineage. In *Proceedings of the 14th International Conference on Scientific and Statistical Database Management*, pages 15–19, Edinburgh, Scotland, July 2002.

[Bra05] Miguel S. Branco. A provenance infrastructure for the atlas experiment at cern. 9 month report, University of Southampton; Faculty of Engineering, Science and Mathematics; School of Electronics and Computer Science, 2005.

[Bur00] Steve Burbeck. The tao of e-business services. Technical report, Emerging Technologies, IBM Software Group, October 2000.

[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.

[CW00] Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *Proceedings of the 16th International Conference on Data Engineering (ICDE'00)*, San Diego, California, February 2000.

[CW03] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. *The VLDB Journal*, 12(1):41–58, 2003.

[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.

[FP03] H. Fan and A. Poulovassilis. Tracing data lineage using schema transformation pathways. In B. Omelayenko and M. Klein, editors, *Knowledge transformation for the Semantic Web*, pages 64–79. IOS Press, 2003.

[FVWY02] 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.

[GGS+03] 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 Simon J Cox, editor, *Proc. UK e-Science All Hands Meeting 2003*, pages 223–226, September 2003.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley Professional, 1995.

[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)*, volume 3544 of *Lecture Notes in Computer Science*, pages 124–139, Grenoble, France, December 2004. Springer-Verlag.

[Glu] Grid laboratory uniform environment. `http://www.cnaf.infn.it/~sergio/datatag/glue/index.htm`

[GMF+05] Paul Groth, Simon Miles, Weijian Fang, Sylvia C. Wong, Klaus-Peter Zauner, and Luc Moreau. Recording and using provenance in a protein compressibility experiment. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC'05)*, July 2005.

[GMM05] Paul Groth, Simon Miles, and Luc Moreau. Preserv: Provenance recording for services. In *Proceedings of the UK OST e-Science second All Hands Meeting 2005 (AHM'05)*, Nottingham,UK, September 2005.

[gri04] Grid registry with metadata oriented interface: Robustness, efficiency, security. `www.grimoires.org`, 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.

[Gro05] Paul T. Groth. On the record: Provenance in large scale, open distributed systems. Transfer thesis, University of Southampton; Faculty of Engineering, Science and Mathematics; School of Electronics and Computer Science, July 2005.

[Kru95] Philippe Kruchten. Architectural blueprints — the "4+1" view. model of software architecture. *IEEE Software*, 12(6), November 1995.

[Lan91a] D.P. Lanter. Design of a lineage-based meta-data base for gis. *Cartography and Geographic Information Systems*, 18(4):255–261, 1991.

[Lan91b] D.P. 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.

[LE91] D.P. Lanter and R. Essinger. User-centered graphical user interface design for gis. Technical Report 91-6, National Center for Geographic Information and Analysis (NCGIA). UCSB, 1991.

[LNP04] Mark Little, Eric Newcomer, and Greg Pavlik (Editors). Web services context specification committee draft version 0.8. Committee draft version 0.8, Arjuna Technologies, Fujitsu, IONA Technologies, Oracle and Sun, November 2004.

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

[Mar01] A. P. Marathe. Tracing lineage of array data. *J. Intell. Inf. Syst.*, 17(2-3):193–214, 2001.

[MCE+03] J.D. Myers, A.R. Chappell, M. Elder, A. Geist, and J. Schwidder. Re-integrating the research record. *IEEE Computing in Science & Engineering*, pages 44–50, 2003.

[MCG+05] Luc Moreau, Liming Chen, Paul Groth, John Ibbotson, Michael Luck, Simon Miles, Omer Rana, Victor Tan, Willmott, and Fenglian Xu. Logical architecture strawman for provenance systems. Technical report, University of Southampton, 2005.

[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.

[MPL+03] J. D. Myers, C. Pancerella, C. Lansing, K. L. Schuchardt, and B. Didier. Multi-scale science: supporting emerging practice with semantically derived provenance. In *ISWC 2003 Workshop: Semantic Web Technologies for Searching and Retrieving Scientific Data*, Sanibel Island, Florida, USA, October 2003.

[RXBR04] 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.

[SH05] Munindar P. Singh and Michael N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley & Sons, Ltd., 2005.

[SM03a] 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.

[SM03b] 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.

[Tan04] V. H. K. Tan. *Interaction tracing for mobile agent security*. PhD thesis, University of Southampton, 2004.

[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.

[VA98] A. Vahdat and T. Anderson. Transparent result caching. In *Proc. of the 1998 USENIX Technical Conference*, New Orleans, Louisiana, June 1998.

[WMF+05] Sylvia C. Wong, Simon Miles, Weijian Fang, Paul Groth, and Luc Moreau. Validation of e-science experiments using a provenance-based approach. In *Proceedings of Fourth All Hands Meeting (AHM'05)*, Nottingham, September 2005.

[Woo98] Allison Gyle Woodruff. *Data Lineage and Information Density in Database Visualization*. PhD thesis, University of California at Berkeley, 1998.

[WS97] A. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *Proc. of the 13th International Conference on Data Engineering*, pages 91–102, Birmingham, England, April 1997.

[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.

[ZGG+03] J. Zhao, C. Goble, M. Greenwood, C. Wroe, and R. Stevens. Annotating, linking and browsing provenance logs for e-science. In *Proc. of the Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data*, October 2003.