An Architecture for Provenance Systems

Authors:	Paul Groth
	Sheng Jiang
	Simon Miles
	Steve Munroe
	Victor Tan
	Sofia Tsasakou
	Luc Moreau
Reviewers:	All project partners
Identifier:	D3.1.1 (Final Architecture)
Type:	Deliverable
Version:	0.6
Version:	October 24, 2006
Status:	public

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.

The PROVENANCE project receives research funding from the European Commission's Sixth Framework Programme

Executive Summary

Provenance Definition

According to the Oxford English Dictionary, provenance is defined as *(i) the fact of coming from some particular source or quarter; origin, derivation. (ii) the history or pedigree of a work of art, manuscript, rare book, etc.; concr., a record of the ultimate derivation and passage of an item through its various owners*.

Provenance is already well understood in the study of fine art where it refers to the trusted, documented history of some art object. Given that documented history, the object attains an authority that allows scholars to understand and appreciate its importance and context relative to other works. Art objects that do not have a trusted, proven history may be treated with some scepticism by those that study and view them. This same concept of provenance may also be applied to data and information generated within computer systems. This being so, one of our primary objectives is to define a representation of provenance that is suitable for computer systems, and the necessary architecture to make use of such a representation. Hence, in this context, we define the *provenance of a piece of data as the process that led to that piece of data*.

Computational Provenance

Generally, in computer systems, applications produce data. Our vision is to transform applications into so called provenance-aware applications, so that when they run, they produce a description of their execution. Such descriptions, which we refer to as *process documentation*, are stored in a *provenance store*, which is a repository for the storage and management of process documentation. Additionally, the provenance store also provides querying facilities to enable services to retrieve the provenance of data items. In support of this vision we have designed a *provenance architecture*, including suitable *data models* and the necessary underpinning *functionality*, with concerns for *scalability* and *security*.

The development of the architecture has been strongly influenced by the service-oriented architectural style, according to which services or *actors* interact with each other by exchanging messages. By enabling actors to make execution-related assertions, or *p-assertions*, we ensure that necessary and sufficient forms of process documentation are captured to be able to give a complete account of any data item's provenance. For example, the p-assertion model allows us to document various aspects of execution, and thus provide descriptions of those parts of an execution that relate to, or impact upon, a given data item. This allows a user to determine the data item's relationships to other data items and processes, such as its dependencies or causal effects and, at the same time, provides a description of the data flow through an application.

The p-assertions within a provenance store are organised in a conceptual structure, called the *p-structure*, based around *interaction records*, each of which is a collection of p-assertions that relate to a single interaction (i.e. an individual message exchange).

The p-structure provides a hierarchical view of process documentation that facilitates the retrieval of p-assertions, independently of the actual technology used in a given application.

Provenance Functionality

From a functional perspective, the provenance store supports two operations: *recording* p-assertions and *queries* over p-assertions.

In order to record p-assertions, the architecture offers a *recording interface* based on the p-assertion recording protocol (PReP). PReP is designed to be *stateless* to allow for asynchronous and out-of-order recording by actors. Furthermore, the provenance store's behaviour is specified to ensure that p-assertions do not become modified or deleted, preserving documentation in its original form, thus reflecting execution as it was originally documented.

Once recorded, documentation is then available for third parties to obtain the provenance of data items, which is achieved via a *process documentation query interface* for the retrieval of p-assertions and their contents, and a *provenance query interface* for the retrieval of a data item's provenance. Querying the provenance of a given data item involves: identification of the data item at a specific point during execution, and scoping of the process of interest to filter causal and functional relationships. The output of queries comes in the form of a collection of p-assertions representing a portion of the data flow graph, which allows a user to understand the provenance of the data item in question up to the specified point in execution.

Non-Functional Considerations

In terms of non-functional requirements, a provenance architecture must address three important considerations: *scalability*, *security* and *management*.

For many applications, extremely large amounts of process documentation can potentially be captured. This presents problems for recording, querying, management and storage of such information. Consequently, there is a need to deal explicitly with such scalability issues and, since the applications that record provenance may be distributed and large scale, the sheer quantity of recorded p-assertions requires a scalable means of storing them. To achieve this, the architecture enables several *recording patterns* that provide flexible ways for recording actors to record p-assertions. For example, one pattern allows different actors to record p-assertions in different stores, even if they refer to the same interaction. Because the documentation of a single process may end up being recorded in several provenance stores, in order to collect all the p-assertions about a process, it is necessary to provide directional *view links* to these provenance stores, where other parts of the documentation may be found.

For some applications, p-assertions may relate to large data sets, such as an actor's state, for example. In such cases, storage capacity problems can arise that are dealt with by allowing p-assertions to reference data that may be stored externally. The replacement of a data item with a reference can be seen as the result of a transformation, and constitutes just one of the possible ways that messages can be transformed using several *documentation styles*, which provide for more flexible ways to make assertions about data, and enable requirements on scalability and security to be met.

Security represents a central concern in many application domains, and it is standard software engineering methodology to integrate security features at the earliest time possible in the development life-cycle. Security concerns, both in relation to the interactions of the internal components of provenance systems and the actors using such systems are addressed, to ensure that appropriate access control for provenance stores is maintained. In addition, it is important that p-assertions can be attributed to the actor responsible for creating them, which is achieved by the inclusion of *assertion signatures*.

Management is not specific to provenance, but should contain functionality that is common to most data management systems, such as notification to users of changes to a provenance store (e.g. the addition or removal of p-assertions) and indexing of a provenance store's contents.

By developing an industrial strength provenance architecture, the EU Provenance project has made possible the capture and exploitation of provenance, and thus greatly facilitates the growth and utility of Grid-based applications by explicitly tackling the problems of trust, accountability, compliance and validation in such open, distributed systems.

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

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, medicine, 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 has 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.

Our overall aim 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 by addressing issues such as concurrency and distribution, system integrity, 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 non-functional requirements of the system such as availability, reliability, performance and scalability.

This document covers the logical and process architectures of provenance systems. Specifically, the logical architecture identifies key roles and their interactions, whereas the process architecture discusses distribution, scalability 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. Despite this technology-independent view, where appropriate we highlight how the architecture can be considered within Service Oriented Architecture and workflow enactment engine scenarios to address the emphasis on these areas expressed in the Technical Annex of the original project proposal.

The development and physical architectures are presented in separate documents, explaining how the architectural design is mapped onto the Web Services stack of standards, and how each individual architecture component is implemented [Ran05, HI05].

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, centred 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 address these issues is discussed in this chapter.

Chapter 5: Scalability Architecture This chapter discusses scalability in the provenance architecture. Architectural scalability addresses how architectural components can be organised and used by implementations to cater for increasingly large loads in terms of such measures as computation, bandwidth and storage. The chapter first 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. It then goes on to explain how the staging of data, references and templates can be integrated into the provenance architecture to address scalability.

Chapter 6: Provenance Modelling This chapter describes the various data models for the information recorded in the provenance store. From the modelling, a system designer can derive how this information can be organised, identified and extended.

Chapter 7: 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 recording, querying and managing capabilities. This presentation is in natural language, informal, and will be used to derive more detailed, technology-specific presentations.

Chapter 8: Actor Behaviour This chapter describes the expectations on actor behaviour so that process documentation can be correctly recorded and provenance questions usefully answered.

Chapter 9: Justification This chapter describes how the software requirements identified by the EU Provenance project for a provenance system are satisfied by the architecture.

Chapter 10: 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.

During the presentation, it is sometimes convenient to refer to specific technologies and explain how the ideas that are currently exposed apply to such technologies. In order to avoid cluttering the presentation with technology-specific comments, we have grouped all of them in Appendix A.

In developing the ideas contained within this document, the requirements of the following stakeholders are considered: End Users, Developers and System Managers. In particular, this involves adopting several views of the architecture as follows:

- The logical architecture, which forms the bulk of the presentation addresses the needs of users by defining the services and interfaces that users can interact with the architecture (cf. Chapters 6 and 7).
- The process architecture address non-functional aspects such as security and distribution allowing system managers to deploy the architecture (cf. Chapters 4 and 5).

- The development architecture provides developers with the means to build upon and adapt the architecture to their needs (cf Chapter 8).

This document is the outcome of a rigorous software engineering process. In order to clearly identify design decisions, and their relationship with captured requirements, design decisions are marked by the symbol †, and a cross-reference to original requirement and analysis appears in the margin. In the online version of this document, the link can simply be followed by clicking on the requirement reference; for the paper version, a page number is also provided for convenience.

1.3 Status of this Document

This report has been a live document 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 have been agreed and clearly documented.

Milestone	Chapters	Draft by	Review by	Final by
Logical architecture frozen:	2, 3	24/6/05	8/7/05	15/7/05
Functional architecture frozen:	4, 5	7/10/05	21/10/05	28/10/05
Requirement analysis:	9	30/11/05	10/12/05	20/12/05
Final architecture frozen:	6, 7, 8, 10	5/2/06	18/2/06	21/2/06
Final Deliverable:	2 to 10	15/09/06	15/10/06	31/10/06

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

Chapter	Revision
Chapter 1	Revision: 1.42
Chapter 2	Revision: 1.48
Chapter 3	Revision: 1.63
Chapter 4	Revision: 1.61
Chapter 5	Revision: 1.68
Chapter 6	Revision: 1.99
Chapter 7	Revision: 1.102
Chapter 8	Revision: 1.87
Chapter 9	Revision: 1.127

1.4 Acknowledgements

This document was reviewed internally by project members. Special thanks to 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, and Steven Willmott for their contributions. We also thank Jim Myers for his comments.

The architectural design presented in this document is the output of research funded in part by the EU Provenance project (IST 511085). 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).

Specifically, this document is inspired by the following EU Provenance publications [And05a, And05b, MCG+05, XBC+05, IHT05, IGM05, Ran05, KS05] and PASOA publications [GLM04b, GLM04a, GLM04c, MGBM06, GMF+05a, GMF+05b, GMM05, WMF+05b, Bra05, MM06].

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. *(1)* Such services are brought together to solve a given problem typically via a *workflow* that specifies their composition. *(2)* In this abstract view, invocations of services take place using *messages* that are constructed in accordance with service *interface* specifications. *(3)* 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. An actor that sends a message is referred to as a *sender*, whereas an actor that receives a message is known as a *receiver*. One message exchanged between a sender and a receiver is termed an *interaction*. Hence, a given interaction comprises two views: the sending of the message and its receiving. The running of an application programmed in a SOA style requires the execution of the workflow, which characterises composition of the services that belong 'to the application. Hence, the execution of a workflow is referred to as a *process*. *(4)* (We note that this use of the term 'process' differs from the one in 'process architecture'.)

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 the actor owning the state. *(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 generally 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 given 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, GLM04c, GLM04b, TGX05, MGBM06, SM03b], the EU Provenance project pre-prototype [XBC+05], its requirements documents [And05a, And05b], 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 6).

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 a *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 algebra 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 thus 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.

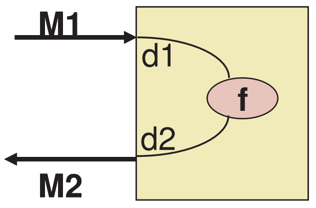
Actors may directly return outputs for the inputs they receive; alternatively, they may invoke other actors in order to obtain intermediate results that help them return their outputs. In both circumstances, the relationship between the outputs and the inputs of the actor is not explicit in the messages 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).

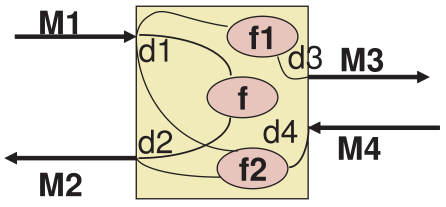
Definition 2.7 (Relationship p-assertion) *A relationship p-assertion is an assertion by an actor that the sending of a message would not be occurring or a data item it is sending would not be as it is (the* effect*), if it had not received other messages or data items had not been as they are (the* causes*), and that this relationship is due to its own action, expressible as the function applied to the causes to produce the effect.* □

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 receives a message and produces a result, but does not invoke subsequent actors, or alternatively, an actor that does not make assertions of the invocations it makes of subsequent 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 that it invokes. 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.



interaction key	p-assertion type	p-assertion content
1	interaction	M1
2	interaction	M2
2	relationship	d2=f(d1)



interaction key	p-assertion type	p-assertion content
1	interaction	M1
2	interaction	M2
3	interaction	M3
4	interaction	M4
2	relationship	d2=f(d1)
3	relationship	d3=f1(d1)
2	relationship	d2=f2(d4,d1)

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. On the right-hand side of Figure 2.1, we see *(8)* a symbolic representation of the p-assertions generated by the actors. Each p-assertion has a type and a content, and is asserted in the context of an interaction identified by a key.

Hence, interaction p-assertions denote data flows *between* actors, whereas relationship p-assertions denote data flows *within* actors. 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 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 [SR-1-6, p. 116] a specific interaction.

Definition 2.8 (Actor State p-assertion) *An actor state p-assertion is an assertion, by an actor, of data received from an (unspecified) internal component of the actor just before, during or just after a message is sent or received. It can, therefore, be viewed as documenting part of the state of the actor at an instant, and may be the cause, but not effect, of other events in a process.* □

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, [SR-1-12, p. 118] 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, with respect to process execution. These different phases identify a *provenance lifecycle*, which we now describe. (We note that such a lifecycle is to be understood in the context of application execution and should be distinguished from a methodology that identifies design steps in order to conceive an application that is provenance aware.)

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 to a specific element, which we refer to as a *provenance store*, the role of making persistent, managing and providing controlled access to such provenance representation. 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 to handle distribution, change management, curation 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 includes information from which a representation of the provenance of the data we are interested in can be derived. *(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 representation 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 on service oriented architectures, used in building open, large scale systems. 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 feed back 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 artifact. We note that either the actuator or the sensor functionality in an actuator/sensor pair may be void.

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 artifact. By querying the provenance of the electronic data, we can therefore obtain an accurate representation of the provenance of the physical artifact, 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 practise, 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

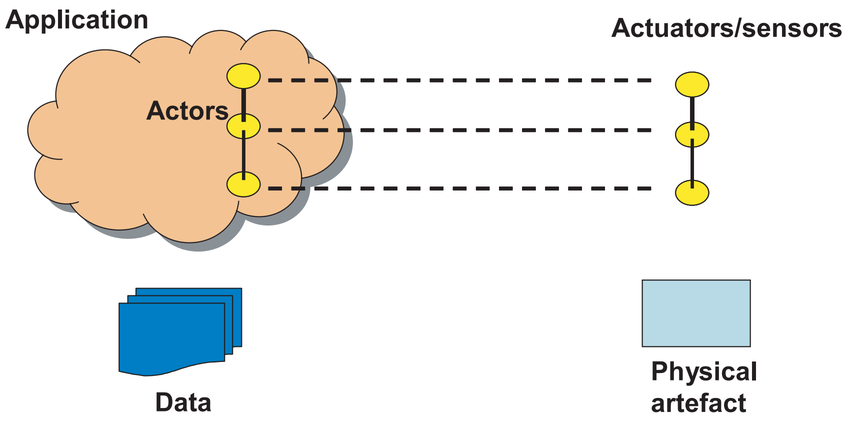


Figure 2.2: Mapping to the Physical World

under the control of the computer application, e.g. in a system where machines are controlled by humans. In such circumstances, the provenance of the electronic data only helps us to derive a partial representation of the provenance of the physical artifact. Such a 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 artifacts. 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 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 a 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 7.2. *(i)* A limit can be set on the length of the relationship chains. *(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 messages 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.

This discussion on provenance queries lead us to enumerate some properties of the computer representation of provenance.

Definition 2.10 (Computer Representation of Provenance) *The computer representation of the provenance of a data item has the following properties:*

1. *it is the result of a query;*
2. *it describes the process that led to the data item;*
3. *it constitutes a DAG.*

□

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 [MGBM06, And05a]. The architecture is summarised in Figure 3.1 (†), which we discuss in the rest of this section. [GR-EHCR.6, p. 131] 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 or other 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 (†). [GR-OTM.3, p. 129] The logical architecture does not prescribe the number of provenance stores, nor their use for a given application, for a given domain, or other. Scalability concerns, network topology, legal and sociological application requirements may all influence the specific deployment to be adopted by application designers.

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 [SR-1-1, p. 115] and internal states, for recording purposes. The recording interface supports the second phase of the provenance lifecycle (storing) and is further specified in Section 7.1. 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 [SR-1-1, p. 115] query capability; the query interfaces support the third phase of the provenance lifecycle (querying) and are specified in Sections 7.2 and 7.3. 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. (A description of its functionality is found in Section 7.4.)

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. [SR-1-9, p. 117] 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); further details can be found in Section 8.7.

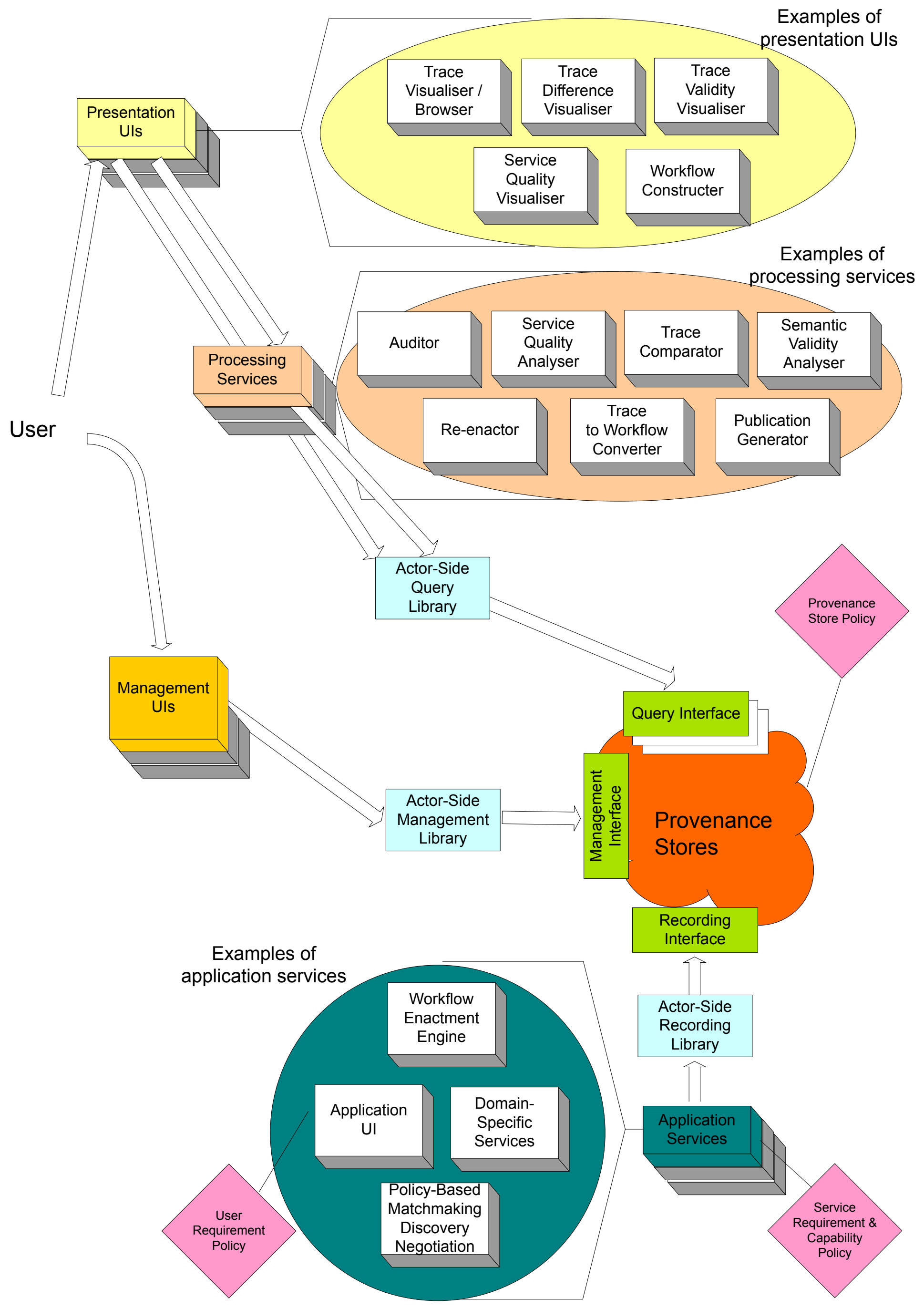

Examples of presentation UIs
Presentation UIs
Trace Visualiser / Browser
Trace Difference Visualiser
Trace Validity Visualiser
Service Quality Visualiser
Workflow Constructer
Examples of processing services
Processing Services
Auditor
Service Quality Analyser
Trace Comparator
Semantic Validity Analyser
Re-enactor
Trace to Workflow Converter
Publication Generator
User
Actor-Side Query Library
Provenance Store Policy
Management UIs
Query Interface
Management Interface
Provenance Stores
Actor-Side Management Library
Recording Interface
Examples of application services
Workflow Enactment Engine
Actor-Side Recording Library
Application UI
Domain-Specific Services
Application Services
Policy-Based Matchmaking Discovery Negotiation
User Requirement Policy
Service Requirement & Capability Policy

Figure 3.1: Architecture of a Provenance-Aware Application

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 implementations 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 *(9)* applies to *domain-specific services*, but also to generic middleware, such as *workflow enactment engines*, *registries* and *application user interfaces*. *(10)*

Once p-assertions have been recorded in a provenance store, process documentation can be used by *processing services* (†) and *presentation user interfaces*(†). The [SR-1-4, p. 116] former provide added-value to the query interfaces by further searching, analysing and [SR-4-2, p. 122] 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; details of such services are discussed in [MGBM06, WMF+05a]. They typically are application specific and therefore cannot be characterised in a generic provenance architecture. For instance, processing services can offer auditing facilities (†), can analyse quality of service based on previous execution, can compare the [SR-1-7, p. 117] processes used to produce several data items, can verify that a given execution was semantically valid [WMF+05a], 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 executions, illustrate execution from a more user-oriented 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 EU Provenance project to deliver these services specifically. The services that are provided by the project are defined and designed in a separate document [Ran05].

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 recording process documentation 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). Policies are further specified in Section 7.5. 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.

Figure 3.1 displays how applications can integrate with the provenance system. It is however important to clarify the scope of the architecture that we are addressing in this document. This is precisely the purpose of Figure 3.2, which introduces a circle around the architectural elements that are discussed in this document. Other components are excluded from further discussion because their behaviours are entirely application-dependent, apart from that specified in provenance-specific policies.

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 6.

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. As it is application actors that make assertions, we place a further obligation on them 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 contextual information, sent along with the interaction's message.* □

In practise, 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

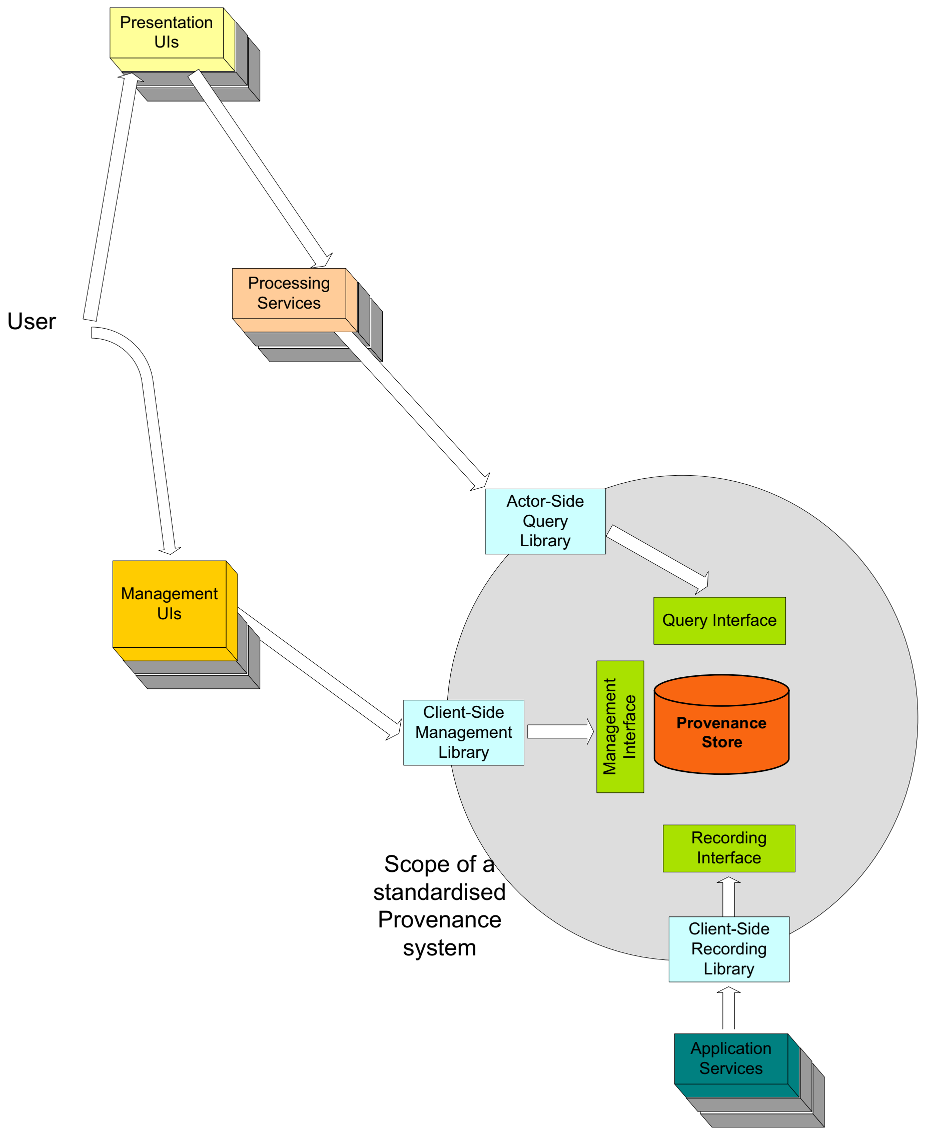


Figure 3.2: Provenance Logical Architecture and its Scope

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 6.

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 role 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. In a *role based access control (RBAC)* system, entities are classified into different roles or groups; each role or group corresponds to differing levels of access clearance to different resources in the system. The roles are generally structured, with higher level roles subsuming the clearance rights of lower level roles beneath them in the hierarchy. Entities can assume several roles, and can move between different roles during the lifetime of the system.

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 9), 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 utilising the information in a p-assertion. This issue is discussed further in Section 6.9.

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 6.10). 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. [OTM-17, p. 131]

4.3.1 Components of Security Architecture

The provenance store exposes three different interfaces (recording, management, query) for different purposes. All of these interfaces can be enhanced with additional security relevant operations or parameters. 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-assertions. [SR-6-5, p. 123]

The identity validator then performs four functions:

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. [OTM-19, p. 132]

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

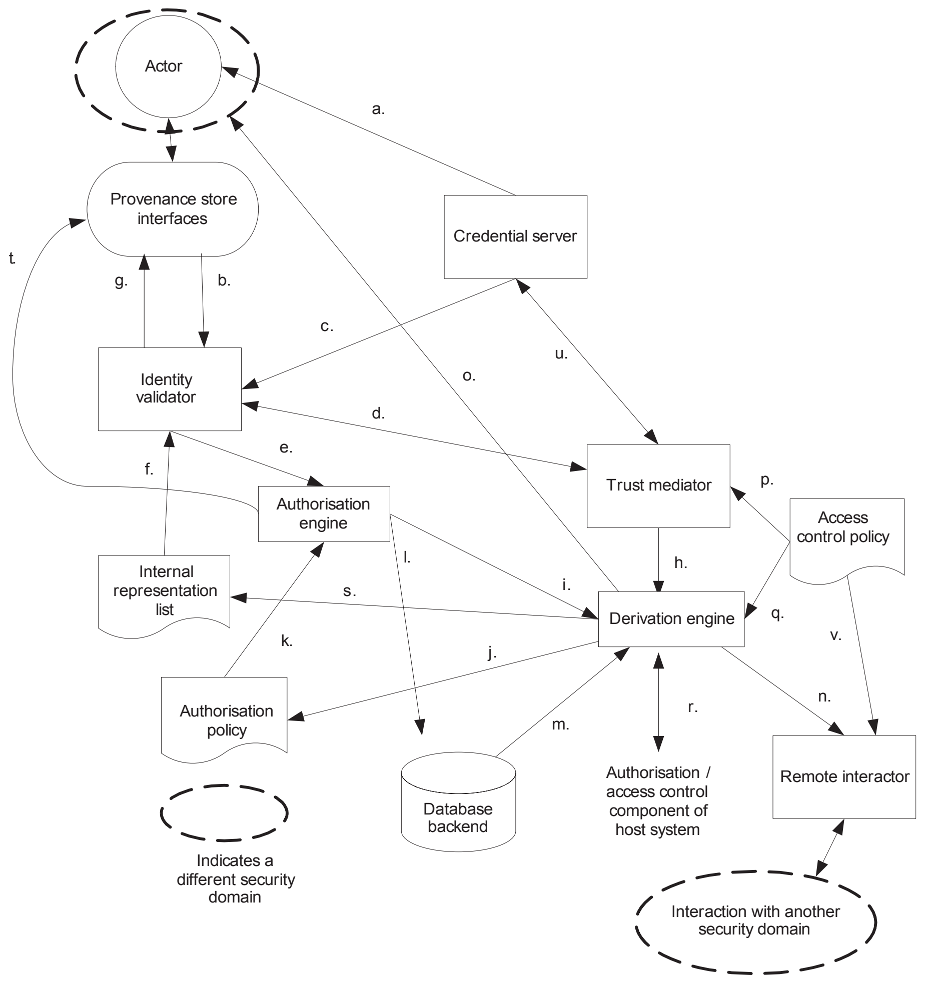


Figure 4.1: Provenance store security architecture

is optional, and correlates with the third security issue in Section 4.2. If the asserter identity is to be utilised 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: [GR-OTM.5, p. 129] [SR-6-1, p. 122] [SR-6-1, p. 122]

- 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 acknowledgement 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 semantics and the granularity of the authorisation assertions within the authorisation policy will determine how fine grained and flexible access control can be in the infrastructure. Ideally, authorisations should be specifiable at the level of individual p-assertions, and could be refined to individual elements within a p-assertion if such need arises. Current authorisation systems in use provide differing levels of semantic expressibility and granularity for the authorisation policies they employ. In terms of semantic expressibility, a possible example could be authorisation systems that allow restriction on access based on additional characteristics such as user attributes, time of day or processing time of job, in contract to simpler systems that only permit user identity in the authorisation expression. If an existing system is to be used as a building block for the provenance store security infrastructure, it should therefore incorporate sufficient expressibility and granularity to satisfy the security requirements of the project (Section 9.1.6).

Additionally, we note that the content-based authorisation described in Section 4.2 would require a higher level policy language to describe, that would then subsequently require translation into the lower level assertions of the authorisation policy (as access control can only be meaningfully performed at this level). This feature is not specified in the software requirements (Section 9.1.6), and merely represents an optional enhancement to the actual security architecture to be implemented.

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. It also describes in a high level manner the configuration of the provenance store from a security viewpoint and the protocols that external entities (such as actors or other provenance stores) need to adhere to in order to communicate with it. The access control policy, authorisation policy and internal representation list would constitute the security policy of the provenance store as a whole (Section 7.6). The *database backend* provides actual physical storage for the p-assertions. [SR-6-1, p. 122]

The *trust mediator* is used to support federation of authentication and/or authorisation for the case where distributed provenance stores (Section 5.2) exist in different security domains. Its role is to obtain security assertions or credentials from other relevant entities in order to support the specific federation methodology employed. Further details on the need for federation, particularly with regards to distributed provenance stores can be found in (Section 5.7).

These entities could be a trusted third party (such as the credential server) in the local or remote security domain, or the trust mediator of another provenance store. The gathered assertions or credentials can then be used locally (for example, to verify other credentials received by the identity validator) or can be passed on to the *remote interactor*. Like the trust mediator, the remote interactor is intended to interact with external entities, but within a more general context rather than a security specific one. The remote interactor uses the credentials provided by the trust mediator for secure communication towards this end. The operational parameters for both the trust mediator and remote interactor can be configured via the access control policy.

The *derivation engine* (†) provides the following functionality: [SR-6-2, p. 123]

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.

The derivation engine may additionally need to interact with other external systems in order to perform specific tasks related to the storage and retrieval of p-assertions. It will perform these via the remote interactor.

4.3.2 Interaction Between Components

We illustrate the interaction between these components using some simple scenarios in a technology independent manner. The flows 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 actor side library functionality (Section 8.7), which may involve encryption or signing of the invocation message. The required credentials can be obtained (if necessary) from the credential server prior to the invocation process. (a.).

2. The identity validator intercepts the message and attempts to verify the submitted credentials; again 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 verified, but there is additional information present to indicate the manner in which they might be verified, an appropriate request is sent off to the trust mediator (d.).

3. Depending on the methodology used for federation purposes (see Section 5.7), the trust mediator may opt to contact a trusted third party (the credential server or another entity) within the current or remote security domain (u.) to obtain the necessary credentials to communicate with the party that is capable of verifying the actor's credentials.

4. The external communication with the designated party is handled by the remote interactor using the credentials obtained by the trust mediator. If the result of this interaction is the verification of the actor's credentials, there may arise a need to add new updates to the local IR or authorisation policy. Such a decision (if required), is made by the derivation engine (h.), which will in turn make the appropriate additions to the internal representation list (s.) and the authorisation policy (j.). The entire operation is mediated using the guidelines specified in the access control policy (q.)

5. If the credentials fail to be verified successfully, 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.

6. 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.

7. 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 policy (j). The p-assertion is now sent onwards to the database backend, along with relevant authorisation information or metadata that it is meant to be stored with (l.).

8. An acknowledgement (as well as other pertinent information) is returned to the derivation engine (m.), which is processed accordingly and another acknowledgement returned to the recording actor (o).

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 authorisation information associated with the request as well as any additional authorisation 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.

Depending on various factors such as the particular deployment of provenance stores, the p-assertions returned and the nature of the request made by the querying actor, there may arise a need for the provenance store to engage in remote interactions with other systems (potentially located in different security domains as well) in order to return a satisfactory result set to the querying actor. This activity is undertaken by the remote interactor (n.), which will procure the necessary security credentials for its interaction from the trust mediator. Again, this operation is coordinated under the guidelines specified in the access control policy (p., q., v.).

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 policy which may include deletion, modification and addition of entries. All of these operations are consequent on the identity validator first recognising that the authenticated managing actor has the role or capability to perform these management type activities (see Section 7.6).

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 policy / 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 Section 5.7.

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 (Section 8.7) 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 actor 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 authorisation 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 authorisation 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 authorised, 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 authorisation 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 authorisation 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 recognise and process this delegated credential accordingly, and infer the identity or associated role of the original user. Subsequent authorisation 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 authorisations 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 authorisation 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, authorisation policies are likely to be duplicated between many presentation services, as it is unlikely that authorisation for a specific user will differ between different services. Accordingly, changes or additions to these authorisation policies 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 authorisation 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 authorisation 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 authorisation policy in the provenance store to reflect the delegation of authorisations 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 authorisation 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 authorisation 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 authorisation engines of both the presentation service and provenance store via their respective remote interactors and trust mediators, if complex authorisation 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. We state where we do not consider an issue to be 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. This issue is entirely within the responsibility of the application domain.

2. Anonymisation of data. Some applications (for example Medical applications using sensitive patient data) 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). This requirement is also 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. There is no explicit requirement for multiple authentication to be handled by the provenance store, although this could be employed in the application domain itself for which it assumes full responsibility for.

4. Specifying policies in an RBAC fashion. It may be useful for authorisation to be performed in a RBAC fashion in the provenance store; the authorisation engine, policy and internal representation list would thereby need to incorporate the necessary semantics to express RBAC-type assertions.

5. Long term storage of process documentation. If a third party database provider is used, then process documentation may need to be encrypted or signed by the remote interactor prior to sending it off for storage. In the event that this documentation 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 several peripheral security issues, of which some may need to be addressed in the development of the security architecture. In the following chapters, we will again discuss security issues (where relevant) with appropriate references to this chapter.

Chapter 5

Scalability Architecture

After introducing a logical architecture for provenance systems in Chapter 3, this chapter now discusses scalability in the architecture. Architectural scalability addresses how architectural components can be organised and used by implementations to cater for increasingly large loads in terms of such measures as computation, bandwidth and storage. The chapter first 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. It then goes on to explain how the staging of data, references and templates can be integrated into the provenance architecture to address scalability. Finally, it discusses the security implications of these architectural concepts. The design presented in this chapter is inspired by previous scalability analyses and use cases [IGM05].

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. These include where to store process documentation, how many provenance stores to deploy, where in the network topology to deploy provenance stores, and how application actors can associate process documentation stored across provenance stores. Therefore, one aim of the distribution architecture is to present a set of *deployment patterns* (†) that address these problems. [SR-2-4, p. 120]

A pattern [AIS77, Ale79, GHJV95] 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. In the context of a provenance architecture, patterns allow us to present a solution that any developer can use to integrate p-assertion recording into their application. We now describe a set of patterns for p-assertion recording adopted from [Gro05]. The format is as follows:

Each section title is 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 labelled 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. A circle above a line denotes information inside the message.

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 SeparateStore Pattern

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 shut down, moved or be under repair.

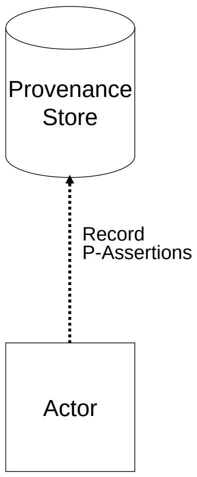


Figure 5.1: SeparateStore Pattern Diagram

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 in a provenance store so that it does not have to retain this information itself. A provenance store should have the following properties:

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 capability 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 ContextPassing Pattern

Diagram See Figure 5.2.

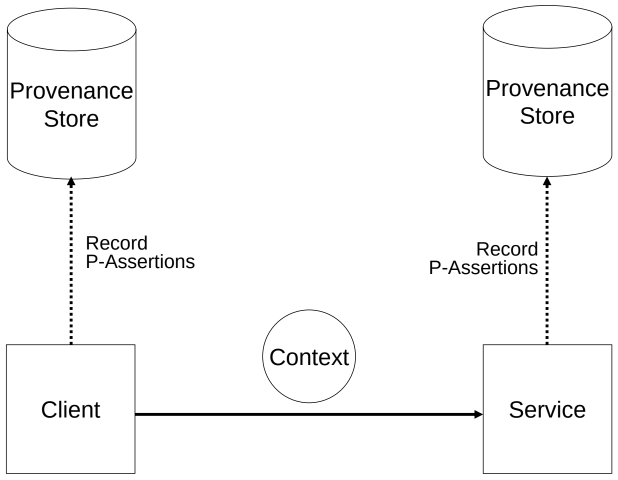


Figure 5.2: ContextPassing Pattern Diagram

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 Separate-Store). 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 and B may want to record actor state p-assertions and relationship p-assertions with respect to a particular 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 sender in the interaction must obtain the appropriate identifiers (IDs) to identify the interaction. It must then pass a *context* containing those IDs to the receiving 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 (cf. matching p-assertions in Chapter 2). 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 context used in this architecture is the p-header, which was defined in Section 6.3. The use of ContextPassing is core to the architecture as discussed in Section 3.4; Chapter 8 discusses further the expected behaviour of actors regarding such contexts. Identifiers and contexts are also discussed in Sections 6.1 and 6.3.

5.1.3 SharedStore Pattern

Diagram See Figure 5.3.

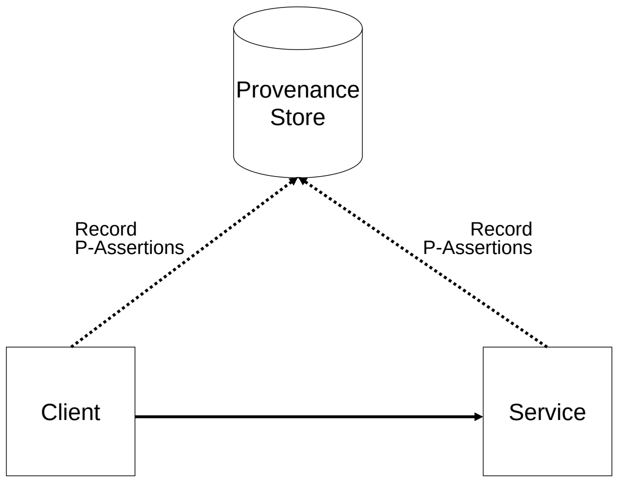


Figure 5.3: SharedStore Pattern Diagram

Context Actors record p-assertions in 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 provenance 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. SharedStore emphasises that actors can record their p-assertions in any store they choose and provenance stores may hold p-assertions from multiple actors. It does not prescribe how many stores there should be and which provenance stores should be shared. It 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.

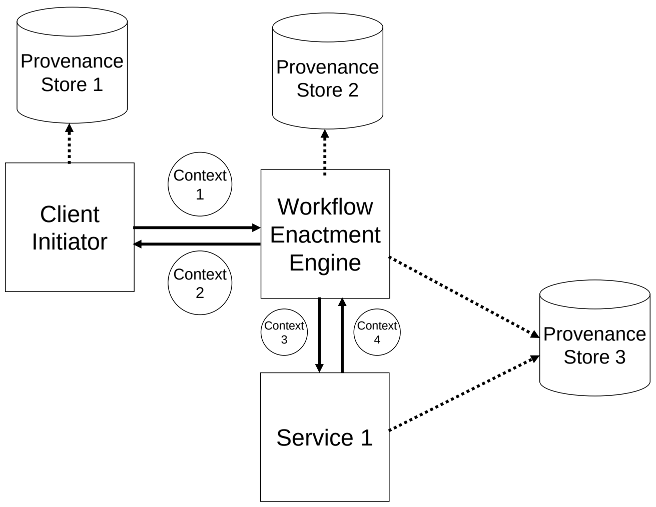


Figure 5.4: A system in which SeparateStore, ContextPassing and SharedStore have been applied multiple times

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. These distribution patterns however do not mandate the number of provenance stores that must be 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. The scalability and flexibility are key to allowing these patterns to be applied to the large scale, open, distributed systems that we consider.

Given that the p-assertions documenting a given execution 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. [GR-OTM.5, p. 129]

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 knowledge is built into A, or it is communicated to A in the course of execution. If it is built into A, then such 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 knowledge can be passed as part of a context, as formalised by the ContextPassing pattern (for instance, when B returns a result to A). Hence, A can assert the link as part of an interaction p-assertion. It is the responsibility of the provenance store to extract View Links from actor state or interaction p-assertions and make them 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 6.) Each triple consists of a subject, a relation and several objects. A relationship p-assertion made by an actor is a directional relation with the subject of the relation referring to the local current interaction p-assertion (or data contained in it).

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 to is stored in; such a link is referred to as an *Object Link*(†). [GR-OTM.5, p. 129] Again, an Object Link only points to the provenance store and not to a particular piece of data in the store.

5.2.3 Linking Summary

Figure 5.5 shows an example of how both Object and View Links are recorded. Actor A sends a message (M2) to actor B as a consequence of message M1. This interaction is identified by the key 2. In the context (shown by the circle) of the message, A puts a link to the provenance store, P_A, that it uses for the interaction with B. Actor B then extracts the link from the context and records it as an interaction p-assertion in the provenance store P_B. As a result, a View Link from P_B to P_A is created (shown by the arch VL 1). Likewise, A knows from its configuration that B always stores its p-assertions in P_B. Hence, A records a link to P_B as an actor state p-assertion in P_A, which creates a corresponding View Link shown by the arc VL 2. Finally, A makes a relationship p-assertion between its interaction with B and a previous interaction identified by the key 2 containing M2. As part of the p-assertion, it adds a link to the provenance store, P_R, where the p-assertions related to the other interaction were stored. It then records the relationship p-assertion in P_A, which creates an Object Link from P_A to P_R shown by the arc OL. Figure 5.6 shows the contents of the provenance stores P_A and P_B after all p-assertions have been recorded.

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).

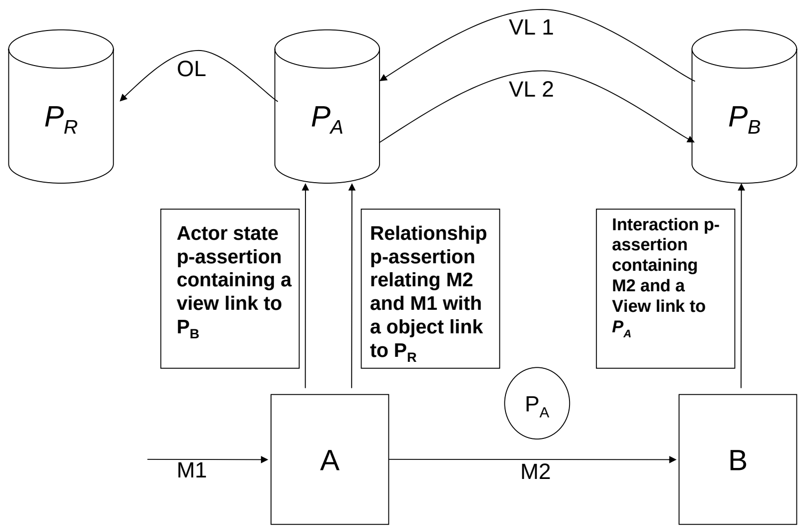

P_R
OL
P_A
VL 1
VL 2
P_B
Actor state p-assertion containing a view link to P_B
Relationship p-assertion relating M2 and M1 with a object link to P_R
Interaction p-assertion containing M2 and a View link to P_A
P_A
M1
A
M2
B

Figure 5.5: An example of linking

Contents of P_A

interaction key	p-assertion type	p-assertion content
1	interaction	M1
2	interaction	M2
2	actor state	View Link to P_B
2	relationship	2 is related to 1, Object Link to P_R

Contents of P_B

interaction key	p-assertion type	p-assertion content
2	interaction	M2, with View Link to P_A

Figure 5.6: Contents of provenance stores

Links provide a scalable solution to the problem of connecting distributed process documentation. Similar to the Web, the unidirectional nature of links prevents the problem of having to synchronise between provenance stores when recording a link. Instead, each recording actor is responsible for recording a link just as each web page author is responsible for adding links to other pages as appropriate. Recording of links is lightweight as the information needed to establish a link is minimal. Furthermore, the link structure provides a structured simple mechanism for querying actors to traverse provenance stores in order to perform queries.

5.3 Data Staging

A provenance store that manages p-assertions over an extended lifecycle requires extensive management and data storage features (see Implementation Recommendations IR-PS-9, p 135 and IR-PS-8, p 134). These features are typically provided within an organisation on managed servers administered centrally. In many cases, such provenance stores may not be close enough (in terms of network topology) for actors to record p-assertions with acceptable performance. An example of this might be where process documentation is being recorded for a workflow running on a computing cluster with high speed networking support between the clustered processors. Slower speed networks connect the cluster to other servers which include a managed provenance store. Communicating directly with the managed provenance store would impact the performance of the clustered workflow. Hence, systems designers may choose to include data staging in their design. *Data staging* is the buffering of data at temporary storage sites until that data can be moved to a final storage site.

Data staging can be done via the creation of local temporary provenance stores (i.e. stores that are deployed close to the recording actor with good connectivity and that exist solely for the purpose of data staging). Such a temporary store could be deployed on the same host as the recording actor itself. Creation of temporary provenance stores should be done through a factory mechanism and configured in such a way that once all recording is finished by the actor/actors using it, the documentation is then transferred to the next provenance store. We note that data may be staged several times as it is transferred to the final set of managed provenance stores. The transfer of data may be accomplished either by a push from one provenance store to the next or pulled by the latter from the former. This is a flexible approach that supports multiple firewall and network topologies. Finally, if the provenance store is temporary, a shutdown mechanism is required that safely archives or discards the staged documentation. The use of temporary stores allows for efficient use of network bandwidth by actors as well as ensuring that staged data can be both protected with appropriate access control mechanisms and queried over.

Whether a provenance store is temporary or permanent, its behaviour as a *data staging* provenance store is determined by a policy. A provenance store can act as both a regular and data staging store for different actors depending on the associated policies. Furthermore, policies are necessary to determine whether provenance stores should push or pull staged data. Policies for data staging are defined in Definition 7.9, page 102.

Data staging implies the need to copy or move data between various provenance stores. A provenance store can copy data to a target provenance store by first querying itself (using the query interface) for the particular p-assertions to be copied and then recording those p-assertions in the target provenance store (using the recording interface). The provenance store is acting as a recording actor in this case and not an asserting actor. The provenance store may use implementation specific functionality for deleting p-assertions if movement of data is necessary. Likewise, implementation specific functionality may be implemented to update linking information (see Section 7.4.2).

The copying and deletion of p-assertions between provenance stores have an effect on querying those p-assertions: they are available for search and retrieval from a different set of stores. For a querying actor expecting to find process documentation in a store from which it has been deleted, there should be a mechanism by which it can find the new location of that documentation. This is a data management issue out of scope of this architecture (we assume that, in general, p-assertions are not deleted from stores), but suggested mechanisms are given in Section 7.4, such as the sending of notifications when p-assertions are moved to another store.

The querying over staged data is challenging. For example, data may be in transit when a query is issued to a provenance store. If the data necessary to answer a query is in transit to another provenance store, is not readily available and the query cannot be answered. One solution to the problem is to inform querying actors when data is finished being transferred and where that data now resides through the use of notifications (Section 7.4). The remote interactor of the security architecture (Section 4.3.1) can be used in the sending of these notifications, as well as supporting any security requirements implicit with such notifications.

5.4 References

In many applications, the size of the messages being transferred between actors may be so large or bandwidth costs so high that the actor cannot record a copy of the message in the provenance store. In other instances, actors may wish to record multiple p-assertions that contain the same data but only record that data one time. Both these problems can be solved through the use of references. References differ from links. Links point to provenance stores whereas references point to data.

5.4.1 By-Value versus By-Reference recording

To address the situation where an actor does not wish to record data by-value in a p-assertion, we introduce the notion of recording a reference to a message outside the provenance system. An actor may store its application data on some persistent storage outside a provenance store, such as a shared file system. If the persistent storage provides some unique identifier for the data (for example, a path), the identifier can be placed inside a p-assertion that is submitted to the provenance store. A querier then must resolve the path in order to obtain the corresponding data. In this case, the lifetime of the application data in the persistent storage must be equivalent to that of the provenance store.

Another solution is for the provenance store to periodically resolve the references downloading the data into a separate logical repository in the provenance store. The provenance store can then inform querying actors that it has a copy of the referenced data that it can provide. However, this solution implies that the provenance store can understand the reference scheme used. Policies identify how provenance stores deal with referenced data. See Definition 7.8, page 101, for a discussion of such a policy.

5.4.2 Record-Once versus Record-Many

Applications, especially those which iterate heavily, may record the same p-assertion content multiple times leading to potentially large amounts of redundant information within the provenance store. To address this we allow references to data items within provenance stores.

A p-assertion has a unique key (cf. Figure 6.2, page 69). If an actor records a subsequent p-assertion that has the same content as a previously recorded p-assertion, it may choose to replace the contents of the new p-assertion with the previous p-assertion's key thereby reducing the new p-assertion's size. If there are multiple p-assertions to be recorded, an actor may repeatedly replace the p-assertion content with the appropriate p-assertion key. Querying the p-assertion can then include a lookup stage that replaces the recorded identifier with the contents of the message that matches the identifier.

Both referencing mechanisms are explained in terms of replacing the whole content of p-assertions or messages. However, if a subpart of a message or p-assertion can be identified that subpart may also be replaced by references. For a further discussion of modelling references, see documentation styles 6.13, page 74 and 6.15, page 74.

5.5 P-Assertion Templates

P-Assertions may often follow the same pattern. They may contain the same structure or data with minor changes in between. For example, during an iterative workflow, p-assertions may be generated that have roughly the same data where one or two parameters in the data may be changed (for instance the state of the loop counter for that iteration). The p-assertions generated by the actors in the workflow follow a template: for every iteration, put Y in position X of the p-assertion content. *P-Assertion Templates*, then, are algorithms that actors use to generate p-assertions.

Typically, such templates are defined locally and used to create p-assertions, which are then recorded in a provenance store. However, it may be the case that the structure of a p-assertion is large or the computation required for its generation is so complex that creation of the actual p-assertion may negatively affect the performance of the actor. Therefore, a provenance store may allow actors to upload templates to it. The actor can then record a p-assertion by sending a template identifier and its parameters to the provenance store. The provenance store can then either generate the p-assertion and place it in its own repository or dynamically generate the p-assertion as and when a query needs it. This reduces the amount of data and processing an actor must record to provide process documentation, thereby, increasing the scalability of the system. P-Assertion Templates must be expressed in a language that both asserting actors and provenance stores understand. The choice of such a language is implementation dependent and can be specified through policies that govern templates (see Definition 7.10, page 102).

Templates are derived from work in mobile code [ACV97]. Extensive literature is available about mobile code/agents. Implementers should consult this literature when implementing this portion of the architecture.

5.6 Large Query Results

Process documentation stored across multiple provenance stores is likely to be massive. Query interfaces such as those defined in Section 7.2 allow querying actors to obtain the process documentation that most precisely fits their provenance questions. However, even with precise queries, the number and size of p-assertions returned to the querying actors may be large. Therefore, we outline the following techniques for dealing with large query results. Each of which allows querying actors to obtain information *as needed* from the provenance store.

To handle large numbers of p-assertions, provenance stores should allow for the buffering of p-assertions. A querier can then retrieve iteratively the p-assertions as needed instead of in one large message. A specification of such an interface can be found in Section 7.2. Furthermore, to support p-assertions that contain large data items (images for example), query interfaces should allow the sending of keys for those p-assertions instead of the p-assertions themselves. In some cases, the content of a p-assertion may also be large. Provenance stores may also wish to support the iterative retrieval of p-assertion content. This notion is supported through the Process Documentation Query interface defined in Section 7.2 as well as Implementation Recommendation IR-PS-7, p 134.

5.7 Security

As discussed earlier, linking provides a mechanism to discover related p-assertions in multiple stores. A potential security issue arises if the querying actor is not recognised or does not have the necessary authorisation 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 can use the links in the available p-assertions to locate other distributed provenance stores which it subsequently needs to query. Thus, the querying functionality queries the distributed provenance stores on behalf of the querying actor.
- The querying functionality can return the available p-assertions to the querying actor, which then itself has to navigate the links and make queries to the relevant distributed provenance stores.

In the first approach, the provenance store to which the original query is directed needs to interact with other provenance stores, possibly in other security domains. This interaction will be handled by the remote interactor using credentials (if necessary) acquired by the trust mediator in the manner previously detailed in Scenario 2 in Section 4.3.2. The remote interactor may also choose to use the delegated credentials of the querying actor in its interaction with the remote provenance store, if so desired.

In the second approach, the querying actor will need to obtain the necessary credentials in order to communicate with the remote provenance stores, possibly through a credential server. Alternatively, depending on the federation methodology employed, the querying actor could attempt to authenticate directly using its own credentials and the remote provenance store would attempt to verify its credentials in the manner outlined in Scenario 1 in Section 4.3.2.

Another security concern revolves around the need to establish liability for erroneous context information. In Section 5.1.2, we 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. Signatures are supported by following Implementation Recommendation IR-ASL-5, p 135.

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 (see Rule 8.11). 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 6.9.

Furthermore, if p-assertions are copied or moved between stores that are located in different security domains (i.e. for the staging of data), the access control restrictions on them in their new destinations needs 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 recognisable in the new provenance store domain. The process of moving p-assertions between different stores also needs 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.

Security considerations can also arise in the use of references, where a p-assertion contains a unique identifier for data stored elsewhere rather than the actual data itself. In this case, both the asserting and querying actor may require some sort of assurance that any data eventually retrieved by resolving the identifier in the p-assertion is actually the same piece of data that was referred to by the identifier at the time of p-assertion creation by the asserting actor. This can be accomplished by having the querying actor include a digest of the data being referred to along with the identifier of that data in the p-assertion. The signature on the p-assertion assures that the digest and the identifier will not be changed. Subsequently, if the referenced data is retrieved later, its digest can be computed and compared against the digest within the p-assertion as a check of its integrity. Digests can be provided through the reference-digest documentation style (Definition 6.14 page 74, Section 6.5).

When references are being used, it may also be possible for the provenance store to resolve the references itself (Section 5.4). Retrieval of the remotely stored data will be achieved by the remote interactor of the security architecture, with the required credentials being obtained by the trust mediator (Section 4.3.1).

5.8 Conclusion

This chapter presented the architecture facets 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 to promote scalability. It then discussed how provenance retrieval could be enabled across a set of distributed provenance stores via linking, which supports the deployment of multiple distributed provenance stores. Data staging was presented as a mechanism for minimising the use of network bandwidth and for allowing p-assertions to be recorded in a timely manner by recording actors. The chapter then presented two notions of recording by reference either by referring to p-assertions already recorded or by referring to data stored at some other location from within a p-assertion. References support scalability by not duplicating data and allowing data to be kept in a single location. P-Assertion Templates were then described as a mechanism to offload the generation of p-assertions to provenance stores reducing the computational load on asserting actors. We then discussed how large query results could be handled. Finally, we addressed relevant security issues that pertain to these scalability solutions.

This chapter has presented architectural solutions to scalability for provenance systems. However, implementations of the architecture can also address the above issues in technology specific ways. For example, provenance stores can use clustering [IGM05] and database management technologies to handle large loads. Likewise, in Web Service implementations, where XML is the common format, binary/compressed data representation should also be supported. These implementation specific scalability solutions are supported by the flexibility and design decisions prescribed in this architecture.

Chapter 6

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.

6.1 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 instant 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.

Definition 6.1 (Interaction Key) *An interaction key is a globally unique identifier for an interaction.* □

In Figure 6.1, 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 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.

Definition 6.2 (Message Source) *A message source is the address from which a message was sent.* □

Definition 6.3 (Message Sink) *A message sink is the address to which a message was sent.* □

Definition 6.4 (Interaction Identifier) *An interaction identifier is a value that is unique for a given message sent from one message source to one message sink.* □

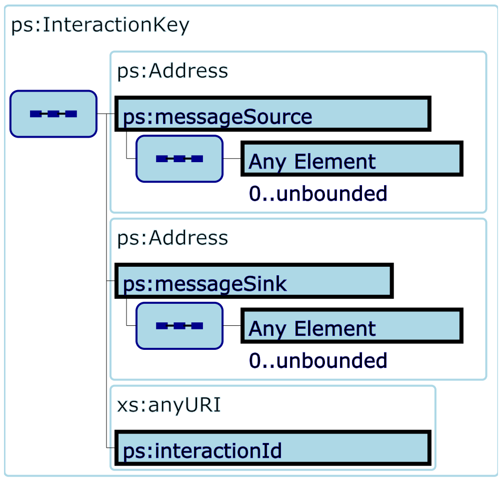


Figure 6.1: Model for identifying an interaction

6.2 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 6.2. A GPAK consists of an interaction key, whether the sender or 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. [AER-6, p. 133]

Definition 6.5 (Local P-Assertion Identifier) *A local p-assertion identifier is a value that is unique for each p-assertion made by one asserting actor about one interaction.* □

Definition 6.6 (View Kind) *A view kind denotes, for a p-assertion, whether the actor making that p-assertion was the sender or the receiver in the interaction to which the p-assertion refers.* □

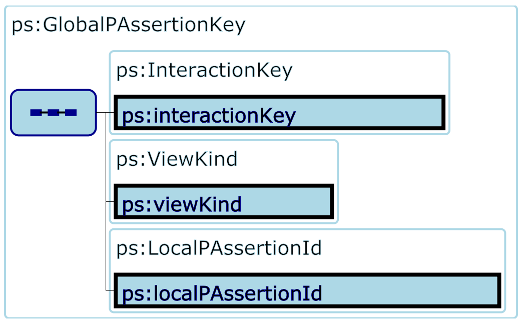


Figure 6.2: Global P-Assertion Key

A *p-assertion data item* is part, or all, of a p-assertion, and can be identified by a *p-assertion data key*. A p-assertion data key optionally extends a global p-assertion key, identifying the p-assertion containing the data item, with a *data accessor*, identifying the location of the data item within the p-assertion. The model for a p-assertion data key is shown in Figure 6.3.

Definition 6.7 (P-Assertion Data Item) *A p-assertion data item is part, or all, of a p-assertion.* □

Definition 6.8 (P-Assertion Data Key) *A p-assertion data key is a globally unique identifier for a p-assertion data item.* □

6.3 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 sender and receiver must use the same interaction key for the same interaction so that their assertions can be matched. Context information can be passed as independently in messages specifically for the purpose or as extra data in existing messages.

In the former case, the context information conforms to the *interaction context* structure shown in Figure 6.4. An interaction context contains an interaction key and

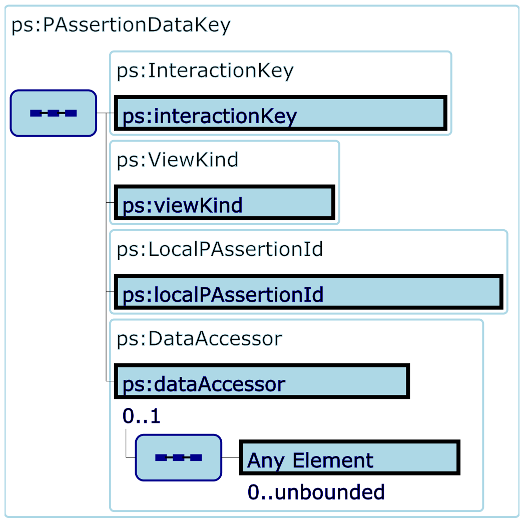


Figure 6.3: P-Assertion Data Key

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 View Links discussed in Section 5.2. The view kind (sender or receiver) to which an interaction's metadata refers to is also included.

Definition 6.9 (Interaction Metadata) *Interaction metadata is provenance-related data about an interaction.* □

Definition 6.10 (Interaction Context) *An interaction context is a set of interaction metadata about the same, identified, interaction.* □

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 6.5 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 sender to inform a receiver 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.

6.4 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

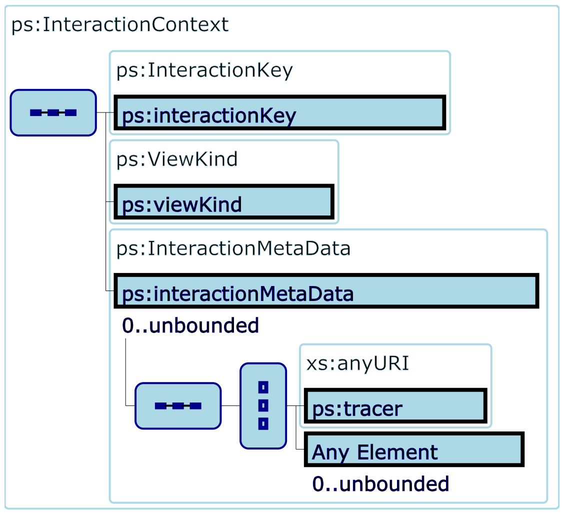


Figure 6.4: Model of an Interaction Context.

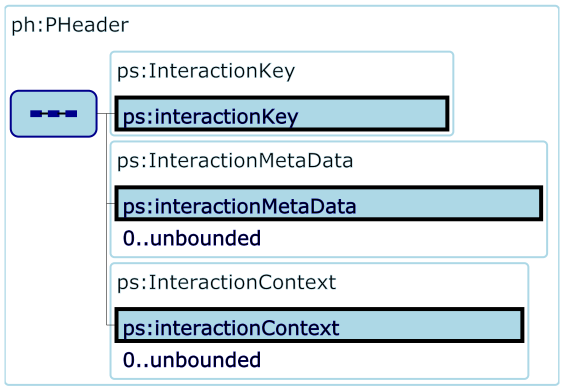


Figure 6.5: Model of the PHeader.

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 'verbatim' documentation style is asserted. A taxonomy of documentation styles is given in Section 6.5. [GR-OTM.1, p. 128]

We define a data type *InteractionPAssertion* to represent any interaction p-assertion and depict its structure in Figure 6.6. A p-assertion consists of three pieces of information: 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 message.

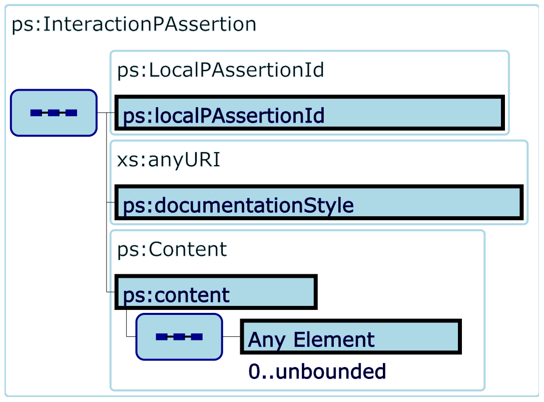


Figure 6.6: Model for an interaction p-assertion

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 by a provenance query about the result produced by that process. The act of constructing a 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 which we agree not to document a computation further: this is precisely the role of documentation style.

6.5 Documentation Style Modelling

When an actor documents an interaction, it constructs an interaction p-assertion (specified in Section 6.4), which states the content of a message received or sent by the asserting actor. We regard the activity of constructing a p-assertion from a message as an atomic transformation, which needs to be qualified by the asserting actor so that querying actors can understand the nature of the transformation that was applied to an application message. This is exactly the purpose of *Documentation Style*, which we now define.

Definition 6.11 (Documentation Style) *Documentation style is a representation of the transformation according to which the content of a message or actor's state is asserted in a p-assertion.* □

Given such an explicit representation of a message transformation, documentation style can be used for different purposes:

1. A documentation style is an explicit representation of the transformations that happened to a message, which enables a querying actor to navigate the contents of an assertion, through the use of corresponding schemas, and determine how the original message was transformed.

2. Once an application designer has identified the type of transformation that needs to be applied to a message, documentation style can be used by the asserting actor in order to construct a p-assertion from a message; such an operation is typically facilitated by an actor-side library (Implementation Recommendation IR-ASL-7, p 135.).

For some particular documentation styles, it is also possible to *reverse* the applied transformation to determine the original message, in which the documentation style of a p-assertion also represents the reverse transformation that can be applied to it. While we do not limit the set of permitted documentation styles, since many will be application-specific, some are essential across applications and worth defining explicitly for interoperability, e.g. tools can be developed to interpret these standard styles. There are several types of documentation style that the provenance architecture standardises upon, each of which have different motivations. These are verbatim, reference, anonymous, security-signing and security-encryption, which we now describe.

Definition 6.12 (Verbatim Documentation Style) *The verbatim documentation style denotes a null transformation applied to the contents of a message.*□

Such a style is the simplest documentation style as no transformation actually happens and the data is copied to the p-assertion in a "verbatim" way.

Section 5.4 introduces the notion of reference in order to provide scalable handling of large p-assertions. Such references are handled by the reference documentation style, which we now define.

Definition 6.13 (Reference Documentation Style) *The reference documentation style denotes a transformation of a message by which a part of (or the whole of) its contents has been replaced by a reference to the location where the actual contents can be found.*□

Section 5.7 discussed security implications of Reference Documentation Style; in order to ensure that the querier of a p-assertion who dereferences a reference accesses the same data as intended by the recorder of the p-assertion, a digest may be associated with the reference.

Definition 6.14 (Reference-Digest Documentation Style) *The reference-digest documentation style denotes a transformation of a message by which a part of (or the whole of) its contents has been replaced by a reference to the actual location where it can be found and a digest of the substituted data.*□

Section 5.4.2 introduced the notion of allowing actors to record references to p-assertions that have already been recorded in a provenance store. This allows actors to avoid recording multiple duplicate p-assertions. To support these references that point internally to a provenance store, we introduce the following documentation style.

Definition 6.15 (Internal Reference Documentation Style) *The internal reference documentation style denotes a transformation of a message by which a part of (or the whole of) its contents has been replaced by a global p-assertion key, which refers to another p-assertion that contains the actual data.*□

This documentation style allows actors to avoid duplicating data unnecessarily. The style is a special case of the reference documentation style.

Section 9.4 introduces the requirement for anonymisation of patient identifiers from the OTM/EHCR application (GR-OTM.1, GR-EHCR.4). These application requirements are handled by anonymous documentation style, which we now define.

Definition 6.16 (Anonymous Documentation Style) *The anonymous documentation style denotes a transformation of a message by which a part of (or the whole of) its contents has been replaced by an "anonymous" identifier. This identifier hides the actual data without losing the link to them.* □

Consider the case of medical data for example, where information for each patient is kept. It may be necessary to hide the patient name from the provenance store due to medical data privacy policies. In order to support this requirement, the anonymous documentation style is used to replace the actual data (patient name in this case) with an ID that in future, and with the appropriate access rights, can be used to access the original data. So the transformation that takes place is the substitution of the actual data with an anonymised ID.

There may sometimes be a need to indicate that certain parts of the message that is about to be transformed into a p-assertion is attributable or linked in some way to certain parties. This goal can be accomplished by attaching the signatures of these parties to the relevant parts of the message. In the simplest case, the required signatures are already present in the original message. For example, two actors may sign parts of the messages that they exchange with each other. In that case, the verbatim documentation style will ensure that they are retained in the created p-assertion (see Rule 8.13, page 110). However, there may be instances where the asserting actor creating the p-assertion may be in possession of delegated credentials of other actors that it may wish to use to sign relevant parts of the message in accordance with some application-dependent protocol. The security-signing documentation style is thus used towards this end.

Definition 6.17 (Security-signing Documentation Style) *The security-signing documentation style denotes a transformation of a message by which a part of (or the whole of) its contents has been signed.*□

Note that the notion of security-signing here differs from that of signing the entire p-assertion (described in Section 6.9), which is what the asserting actor will do with its own private key in order to establish its responsibility and corresponding liability for the contents of the p-assertion. Security-signing however involves the asserting actor using proxy certificates from other entities or actors to sign parts of only the content of the p-assertion. Regardless of whether any security-signing is done, the asserter should always sign the entire p-assertion it creates.

There may be situations where the asserting actor may want to ensure that certain parts of the p-assertion it creates are only accessible to certain parties. In the simplest case, this can be achieved by instituting appropriate access controls on the provenance store through the authorisation policy described in Section 4.3.1. However, once a p-assertion is retrieved from the provenance store, it is very difficult to control which parties it is subsequently propagated to. If the asserting actor shares a secret key with certain parties, it can elect to encrypt parts of the p-assertion so that only those parties are able to view it. This is then the purpose of the security-encryption documentation style, which we now define.

Definition 6.18 (Security-encryption Documentation Style) *The security-encryption documentation style denotes a transformation of a message by which a part of (or the whole of) its contents has been encrypted.*□

In applying the encryption documentation style, the data is replaced with the encrypted data and a set of encryption-related information, such as the name of the encryption algorithm used. This information can later be used to decrypt the data. Note that this encryption documentation style is unrelated to the encryption discussed in 4.3.2. For that case, encryption is applied for the purposes of securing the communication channel between the recording actor and the provenance store.

All previous documentation styles denote an atomic transformation (which is not described in terms of internal steps). We now define the means by which several transformations can be applied to a single message.

Definition 6.19 (Composite Documentation Style) *A composite documentation style denotes that more than one atomic documentation style has been applied to a message.*□

The atomic documentation styles described above can be used in any combination for the same message. This means that several transformations may happen either at different parts of the message or at the same part of it. The resulting message consists of the composition of all the transformations. A composite documentation style denotes that a set of atomic transformations has been applied to a message.

Documentation styles allow actors to express transformations they have performed on messages when asserting interaction p-assertions for those messages. Documentation styles are mandatory for interaction p-assertions because querying actors need to be able to determine whether two interaction p-assertions are matching (see 2.6). Using the documentation styles of the two interaction p-assertions, a querier can generate the original message from both p-assertions and determine whether they are matching, irrespective of how the message was transformed when being asserted. In the case of actor state p-assertions, there is no application-independent notion of "matching" actor state p-assertions. However, we allow the documentation style that has been applied in asserting an actor state p-assertion to be declared along with the state (see below). Relationship p-assertions do not have documentation styles: all the fields in a relationship p-assertion are necessary so that the provenance query functionality can be performed in an application independent manner.

6.6 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 performs an execution that the message has triggered and, similarly, before sending a message, it performs 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 6.7. The p-assertion consists of three pieces of information: a local p-assertion identifier, *localPAssertionId*, an optional *documentation style*, 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.

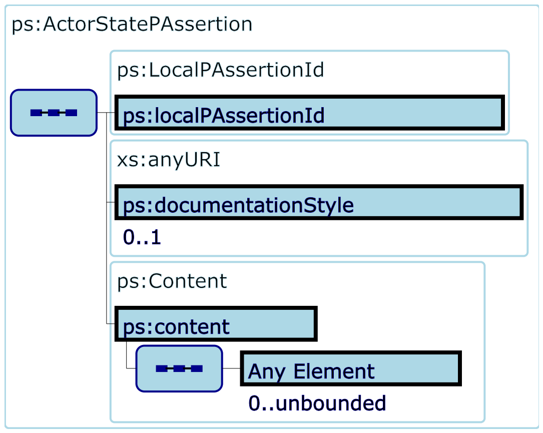


Figure 6.7: Model for an actor state p-assertion

6.7 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, where the domain of a relationship is called the *subject* and the range is the set of *objects*. The triple consists of a subject identifier (subjectId), a relation, and several object identifiers (objectIds). The model for relationship p-assertions is shown in Figure 6.8.

Typically, a relationship p-assertion is expressing a causal relationships, where the subject of the relationship is a data item in a sent message, i.e. an output, and the objects are entities in messages received by the same actor, i.e. inputs, where the inputs had a caused the output to be as it is. 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 data accessor. An objectId also contains a parameter name, 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.

Definition 6.20 (Data Accessor) *A data accessor is a reference to the location of a p-assertion data item within a p-assertion's content.* □

Definition 6.21 (Parameter Name) *A parameter name is an identifier for an entity's role in a relationship, where that entity is documented by a p-assertion data item.* □

There is no bound on the relationships that can be asserted between entities in p-assertions. However, to aid interoperability, we provide a core set of terms essential to find the provenance of entities. The *causal relationships ontology* aims to define causal relations between properties of the system at different instants. We define "C caused E according to A" as A's belief/assertion that E would not have become true if C had not been true, all else being equal. The full ontology, and the namespace used for the terms, is given in [Rel06].

The ontology includes a property, *wasCausedBy*, which is a relation between an effect and its cause, and two classes, *Cause* and *Effect* that can be used to classify causes and effects. Using the causal relationships ontology, we can specify causal relations between the contents of interaction p-assertions (and actor state p-assertions). In particular, *wasCausedBy* can be used as the relation term in a relationship p-assertion, while *Cause* and *Effect* can be used as parameter names. More generally, application-specific relations may sub-class *wasCausedBy* to indicate that they are causal relations (as opposed to structural, temporal or other relations between entities).

6.8 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*(†). [SR-2-1, p. 120]

Definition 6.22 (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 6.9 shows the p-structure. It reflects the models discussed above.

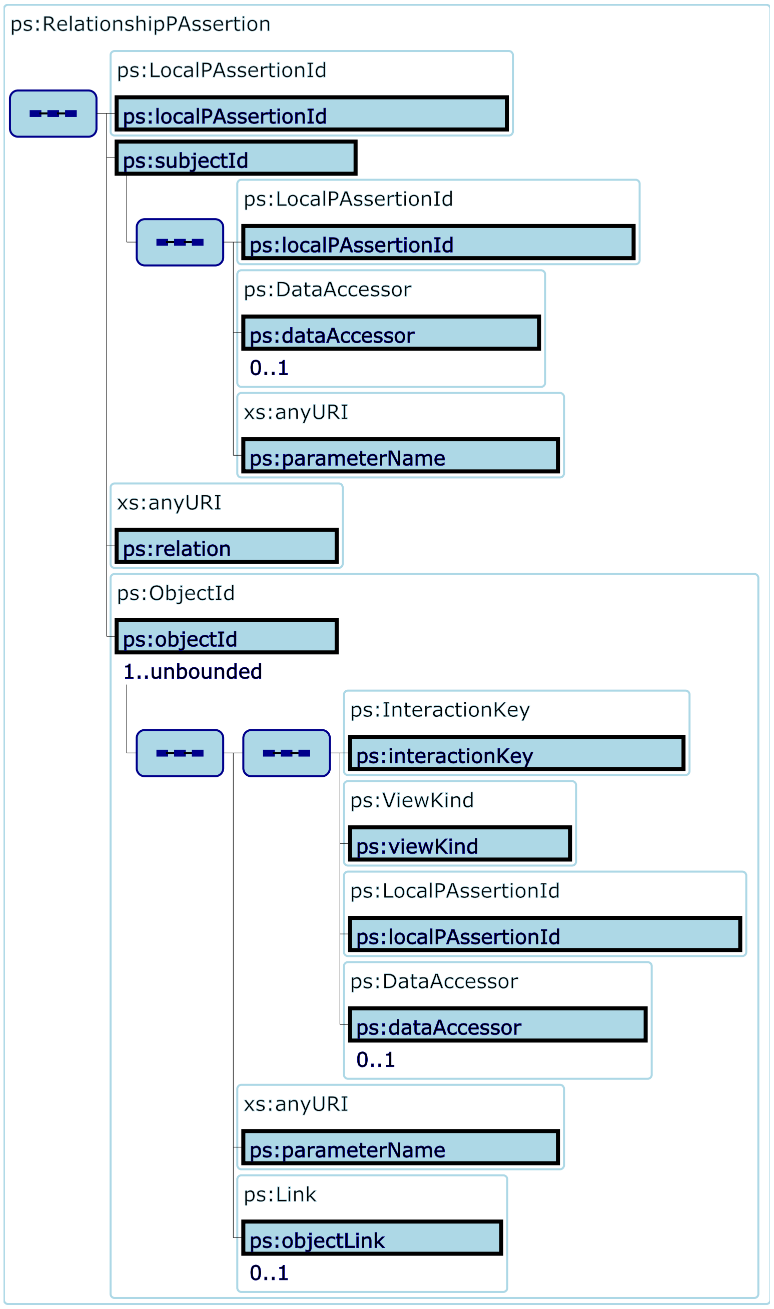


Figure 6.8: Relationship p-assertion model

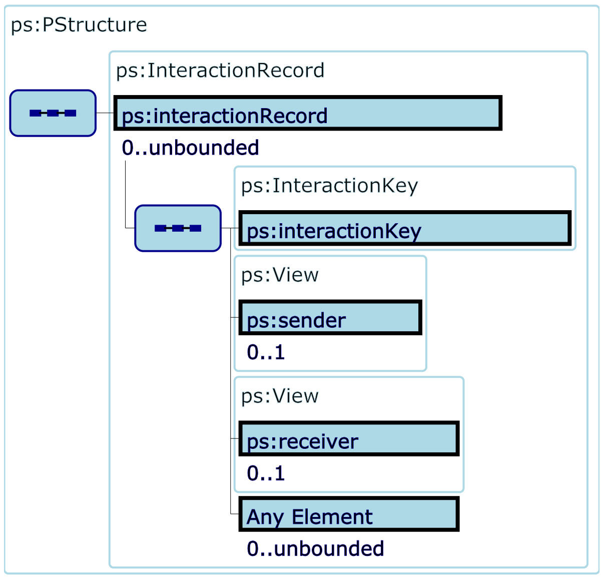


Figure 6.9: The P-Structure

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 an interaction key, as shown in Figure 6.1. The interaction key distinguishes one InteractionRecord from all others and is provided by the asserting actor and not the provenance store. Therefore, no contact with the provenance store 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 sender in the interaction, while the other View contains those from the receiver. A View has the following structure as shown in Figure 6.10: every view has an *asserter* (†), which is the identity of the actor asserting a set of p-assertions, 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, and a set of ExposedInteractionMetada. These components make explicit at a high-level the information contained within the InteractionMetaData contained within the p-assertions stored within the InteractionRecord, and can be used to facilitate the location of p-assertions within the p-structure. A View also contains a set of AnyElements that can be instantiated for specific purposes, such as with ViewLinks for linking different views of an interaction when they are stored in different provenance stores (as described in Chapter 5). [GR-OTM.9, p. 130]

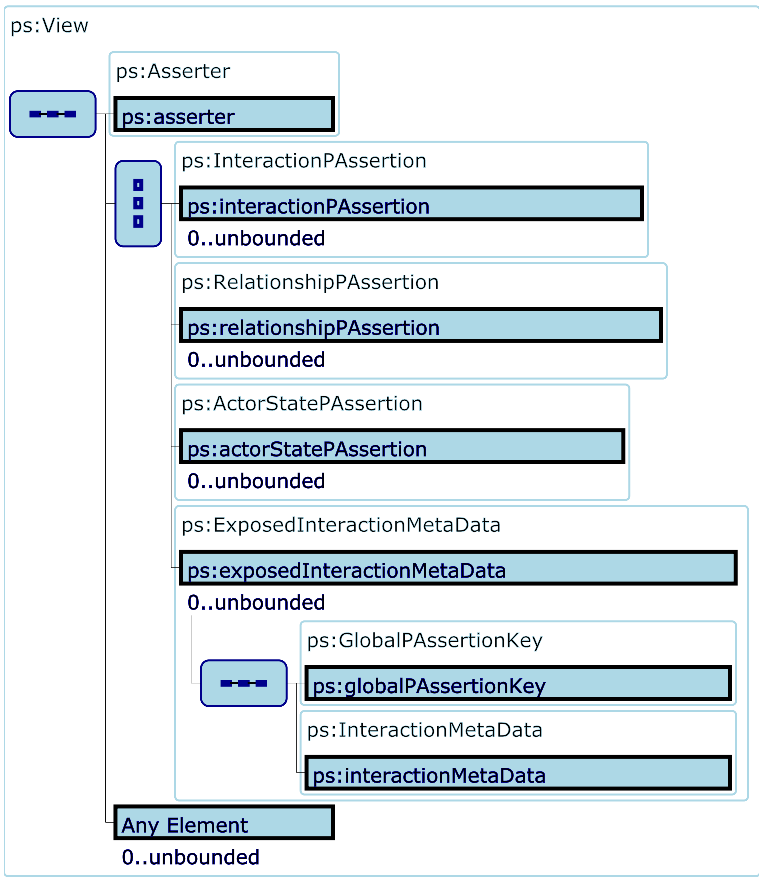


Figure 6.10: A View in the P-Structure

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 . All of these are optional. Each p-assertion is defined by an associated model described above.

Definition 6.23 (View) *A view is the set of p-assertions by one actor about one interaction.* □

Definition 6.24 (Asserter Identity) *An asserter identity is an identifier for an asserting actor.* □

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.

6.9 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* (†) [SR-6-6, p. 123] 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 (Definition 7.13), or if it is not done here, by a querying actor prior to processing the p-assertion in some 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 6.11, 6.12 and 6.13. The signature is shown as 'any' as it is entirely application-dependent and so no generic data structure can be specified for it.

6.10 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 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

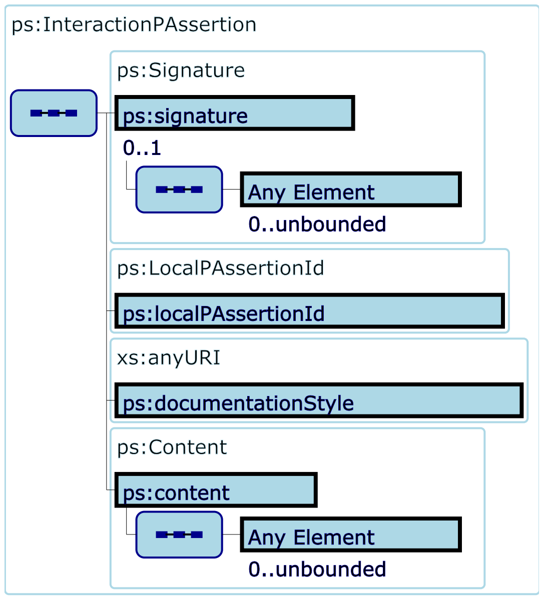


Figure 6.11: Model for a secured interaction p-assertion

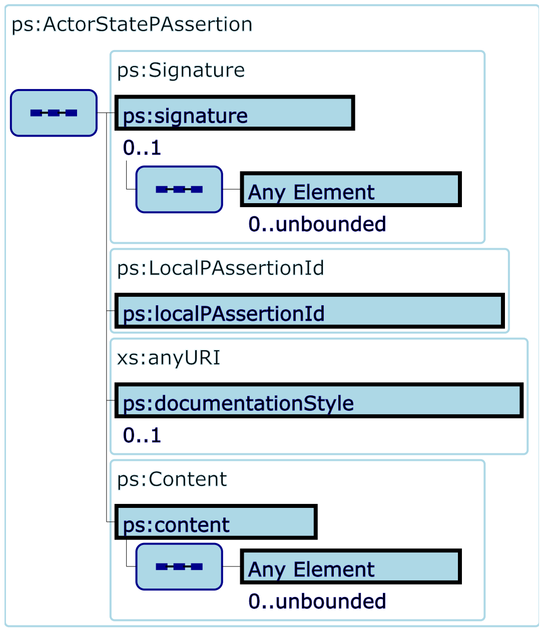


Figure 6.12: Model for a secured actor state p-assertion

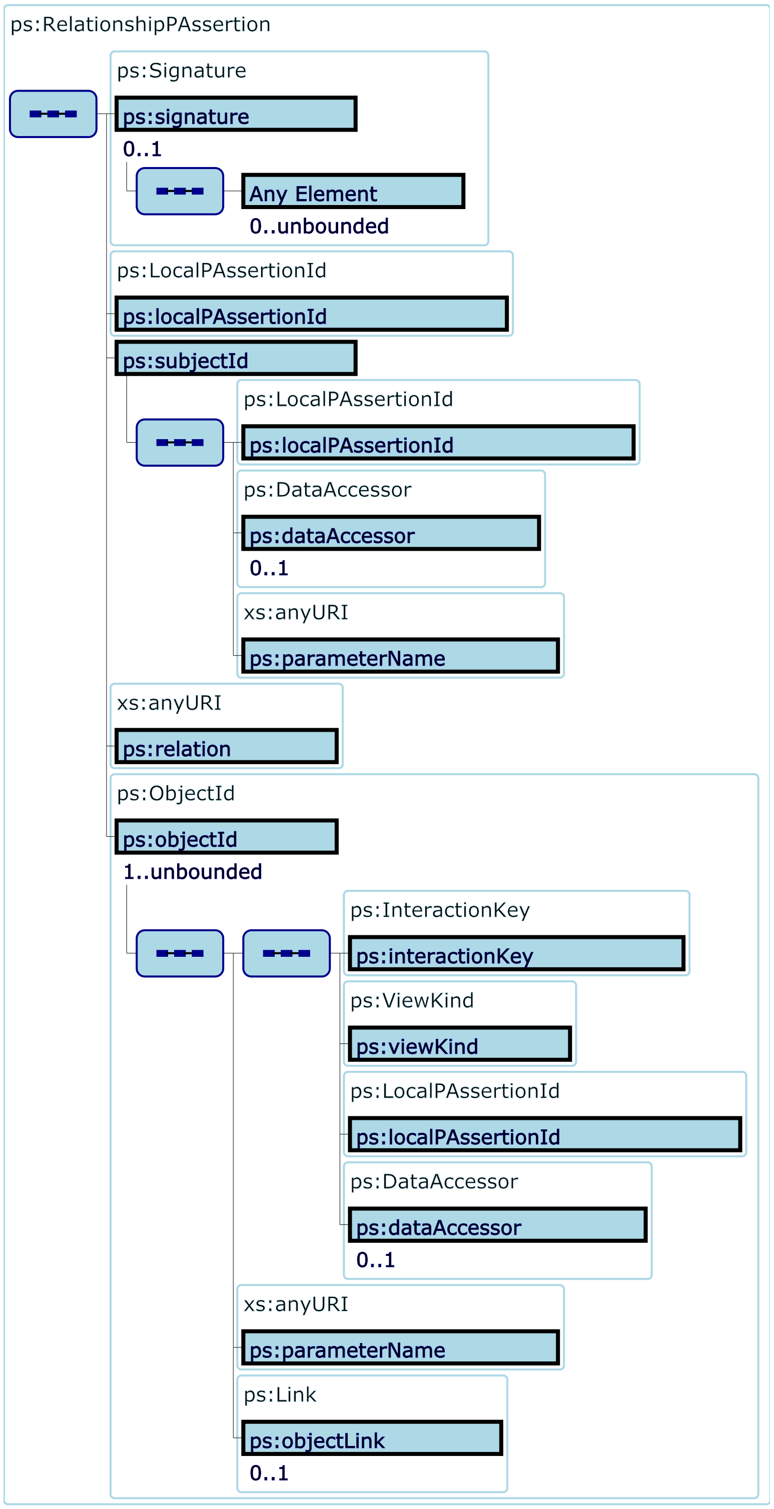


Figure 6.13: Model for a secured relationship p-assertion

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 7

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 detailed specification and service interface specifications, which will be part of a separate document.

7.1 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, GLM04c, GMF+05b, 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. [SR-2-3, p. 120]

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 [GLM04c, GMF+05b]. 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 6.3.

We now discuss the messages that are exchanged by actors with the provenance store to record p-assertions. The messages are shown in Figures 7.1 and 7.2 (†) and constitute the recording functionality of the provenance store. The messages in Figures 7.1 and 7.2 each contain an interaction key, which is the same as the interaction key transferred in the P-Header (see Figure 6.5). Because each message in the protocol contains an interaction key, they 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. [GR-OTM.9, p. 130]

Figure 7.1 shows the record message used to record process documentation into a provenance store. Using these messages, both senders and receivers record their view (†) of an interaction. A record message contains a sequence of *identified content*, which contains an interaction key, a view kind, an asserter to enable querying actors to know what actor is responsible for the particular p-assertion and *content*. Inside content are stored the interaction, actor state and relationship p-assertions that the actor is recording. Additionally, any exposed interaction metadata can also be recorded here. Finally, an integer is used to indicate when the recording submission is finished. There is no bound placed on the number of interaction, actor state or relationship p-assertions that an actor can place in the content of a record message. 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. This means that actors cannot overwrite their previously recorded p-assertions, which preserves the idempotence of the protocol. [SR-1-12, p. 118]

The final message defined by PReP is the *recordAck* message 7.2. This message is sent by the provenance store when it has received a *record* message. The acknowledgement may be sent synchronously or asynchronously depending on the underlying transport mechanism being used between the client and the provenance store. If synchronous, then the provenance store need not include details of the message the acknowledgement is in reply to. If the acknowledgement is asynchronous, the provenance store must make the contents of the message available through the navigation of the p-structure via some query interface. The *recordAck* message contains a *synch_ack* for synchronous acknowledgements and an *ack* for asynchronous acknowledgements, and contains the name of the content that was sent, which can be any of interaction p-assertion, actor state p-assertion, relationship p-assertion, exposed interaction metadata or submission finished. *ack* also includes an interaction key, a view kind, and a local p-assertion id. 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 message layer that supports request-response message patterns, the binding may choose not to include the interaction key, view kind, local p-assertion id

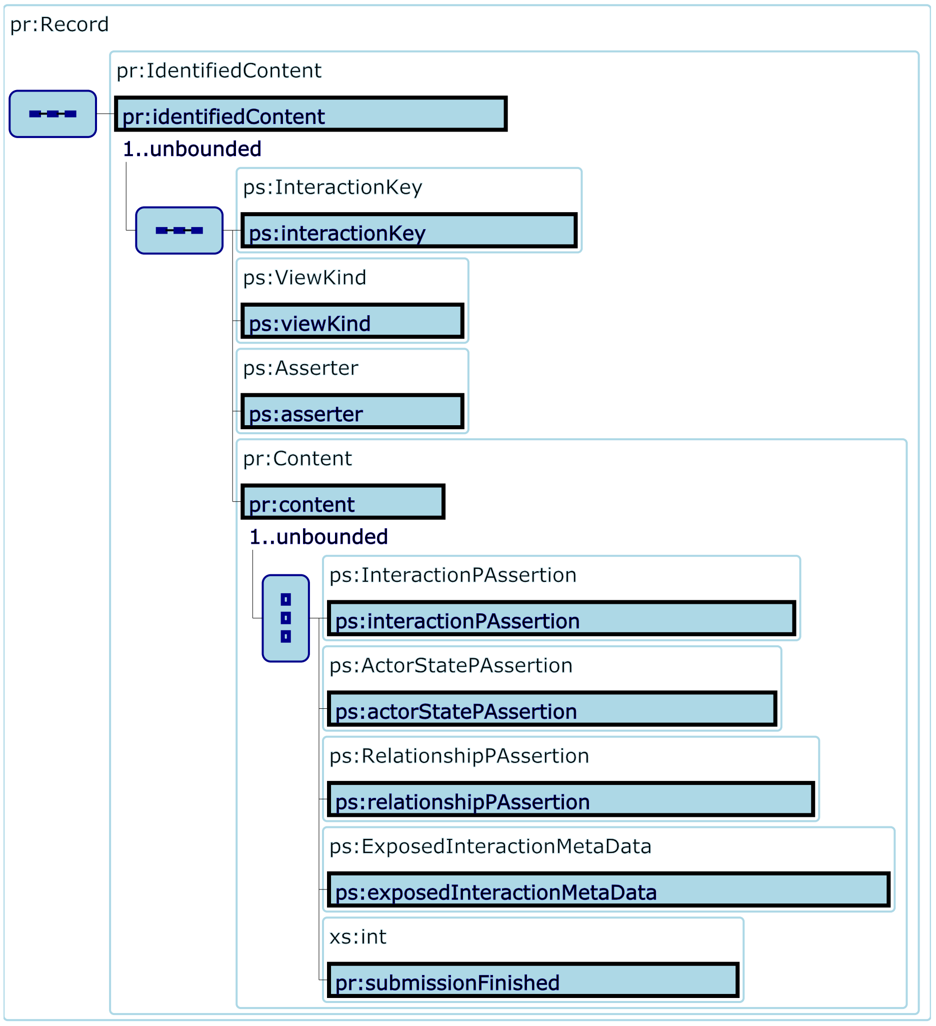


Figure 7.1: Record Message

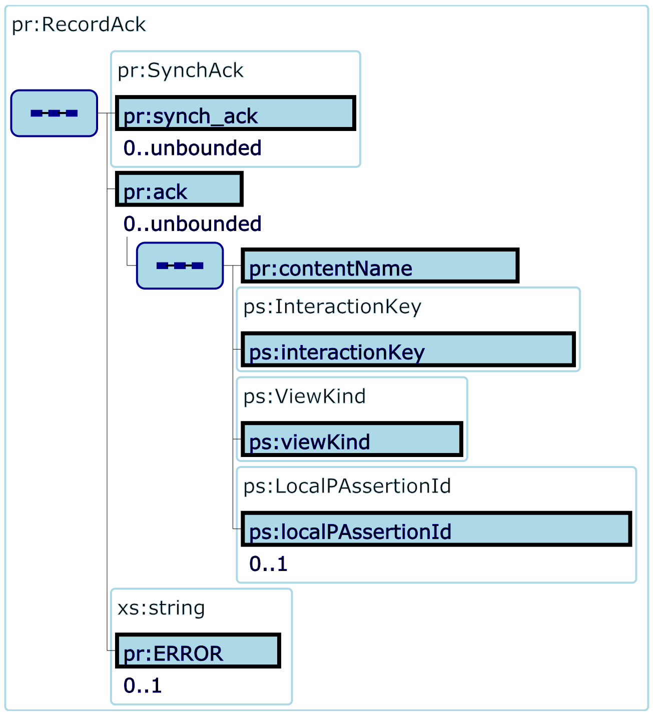


Figure 7.2: Record acknowledgement message

tuple in the *acknowledgement* message because the message layer can identify what request a response relates to. Finally, both *sych_ack* and *ack* also contain *ERROR* to allow application specific error messages to be passed

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.

7.2 Provenance Query Interface

The provenance store supports two query interfaces: a *provenance query interface* which allows querying actors to retrieve the provenance of application entities, and a *process documentation query interface*, by which the content of identified p-assertions can be retrieved. In this section, we specify the functional requirements of the provenance query interface, while the process documentation query interface is specified in the next section.

The provenance query interface (†) accepts a *provenance query request* and responds with *provenance query results*. A provenance query request defines a search for the provenance of an entity at a given instant, and provenance query results represent the provenance of that entity at that instant. The reason that the provenance of [SR-1-2, p. 115] an entity must begin at a specified instant in time is that the entity may be different at different instants in its lifetime, and so the processes by which it came to be in those states (its provenance) are also different. Because p-assertions may be recorded any time after the interaction that the assertion is about, there is no realistic way to define the instant "now": a provenance store or stores can never know whether it has documentation on the latest version of an entity, nor does it necessarily know what what the latest version of an entity is.

A provenance query request is made up of a *query data handle* and a *relationship target filter*, defined below and depicted in Figure 7.3.

Definition 7.1 (Query Data Handle) *A query data handle is a search over the contents of a provenance store in order to find the record of an entity at a given instant that the querying actor wishes to find the provenance of.* □

Definition 7.2 (Relationship Target Filter) *A relationship target filter is a set of criteria by which the querying actor specifies whether any given entity in the process documentation should be included in the query results. By this mechanism, the querying actor can scope the provenance query results.* □

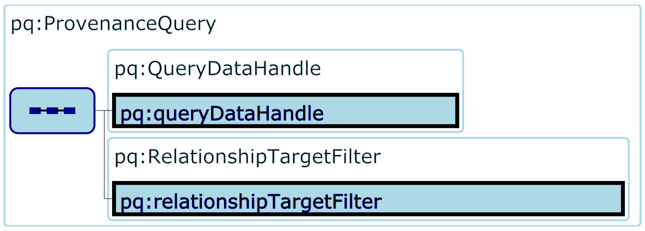


Figure 7.3: Provenance Query Request model

7.2.1 Query Data Handles

In order for a querying actor to ask provenance stores the question "What is the provenance of entity E at instant T?", the actor must identify the entity in a way that the provenance stores can interpret. The identification used is called a *query data handle*. From the application perspective, a query data handle identifies an application entity at a given instant. From the provenance store perspective, a query data handle identifies a search for *p-assertion data items* within the process documentation.

A query data handle is made up of the following two parts:

- A search over the p-structure for instants at which the entity may occur.
- A search over the contents of interaction or actor state p-assertions to retrieve the data items which are documentation of the entity.

Because interaction and actor state p-assertions are also part of the p-structure, these parts can be combined into a single search over the contents of a provenance store, represented by a p-structure. We will look at how the querying actor specifies each part of the search below.

Identifying Instants

Following the formulation of a system as being a collection of distributed interacting actors, described in Chapter 2, there are three ways in which to identify a documented instant in the past:

- The instant at which an actor sent a message.
- The instant at which an actor received a message.
- The instant at which an actor accessed an asserted part of its state, i.e. the instant to which an actor state refers.

These are apparent in the p-structure as the following.

- An interaction p-assertion in the sender's view of an interaction.
- An interaction p-assertion in the receiver's view of an interaction.
- An actor state p-assertion whose content specifies to which instant it applies, e.g. just before or just after sending or receiving a message.

The identities of the actors involved in an interaction are apparent in the message source, message sink and asserter elements of the p-structure.

Identifying Data Items

The entity of which the querying actor wishes to find the provenance must be apparent in the interaction or actor state p-assertions in order for the provenance store to find it. The entity may not always be present as an exact copy of data: it may appear by a reference in a p-assertion or otherwise implied by application-specific structures. For instance, application messages may refer to a data item by the name of the file in which it is contained, but the querying actor wishes to find the provenance of the data, rather than the file.

The query data handle includes a search over the contents of p-assertions to retrieve data items which are documentation of the entity. This search is expressed in a particular *search language*, and the range of search languages supported by any one provenance store may vary.

A search language must assume some format and structure of the documents over which it searches, which we call the *document language*, e.g. the XPath search language assumes the XML document language. However, a provenance store is agnostic to and unaware of the structure of application messages and assertions of state it contains. In fact, application messages may have used different formats, so p-assertions within one provenance store may use different document languages. Therefore, a query data handle may also specify *document language mappings* [ZDF+05] between the document language used for a p-assertion and the document language required for the search.

Definition 7.3 (Document Language Mapping) *A document language mapping is a definition of how to transform documents formatted in one document language into another document language.* □

Composition of Provenance Queries

A query data handle is a search for an entity at an instant within a p-structure. Primarily, this p-structure will be the full contents of a provenance store. However, in some cases a query data handle may be better expressed as a *composition* of provenance queries, i.e. one provenance query is performed and the results are searched over by another provenance query. For example, to find the provenance of the second item added to a set, we can first determine the provenance of the set, and then identify from that the item added second, so the provenance of the item can then be found. In this case, the search that the query data handle specifies is over a p-structure formed from the results of another provenance query.

In general terms, we divide the query data handle into the search to be performed and the p-structure over which it will search. That p-structure, referred to by a *p-structure reference*, can be one of three possibilities (also depicted in Figure 7.4).

- The contents of the provenance store.
- The results of another provenance query, in the form of a p-structure.
- A p-structure given by another p-structure reference but, in addition, including the transitive closure of relationships with a given relation name in that p-structure, i.e. wherever there is a relationship of the specified type from A to B and one from B to C, the transitive closure will also contain a relationship p-assertion from A to C.

Definition 7.4 (P-Structure Reference) *A p-structure reference is a declaration of the p-structure over which a provenance query's entity search will be executed.* □

Model

A model of the query data handle is shown in Figure 7.5. The *search* element specifies a search, in the chosen search language, over the p-structure for data items within p-assertions asserted about sending or receiving messages at given instants. The *pStructureReference* refers to the set of p-assertions that the search will be conducted over,

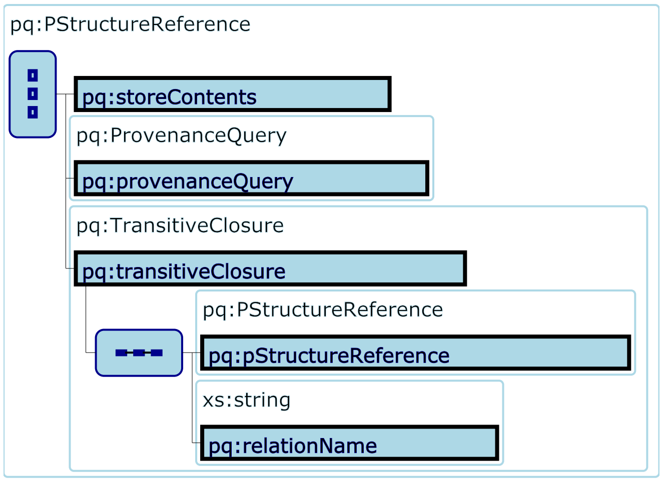


Figure 7.4: P-Structure Reference model

and is one of the options discussed in the section above. The *documentLanguageMappings* specify how p-assertion contents are mapped to the document language required by the search.

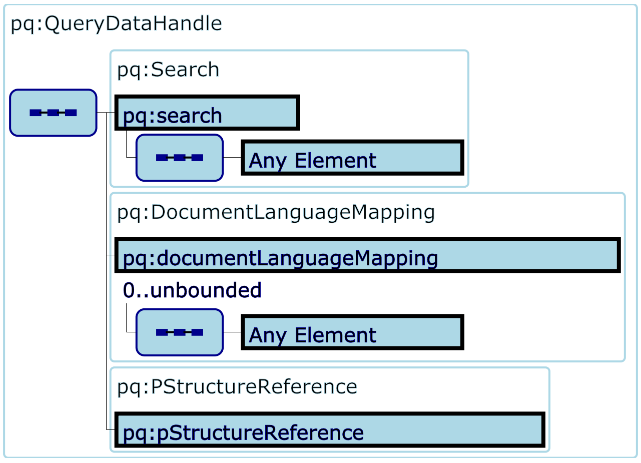


Figure 7.5: Query Data Handle model

7.2.2 Relationship Target Filters

The set of process documentation about entities that ultimately have some causal influence on the entity identified by a query data handle could be vast, and most of it irrelevant to a querying actor for any one purpose. Therefore, we need to allow the querying actor to specify the *scope* of the provenance query, i.e. a definition of what documentation is relevant enough to be part of the results. This is the purpose of the *relationship target filter*.

More concretely, we can say that, given that one p-assertion data item has been identified as part of a provenance query's results, and that that data item is related to a set of other data items (by relationship p-assertions), the relationship target filter specifies which of those related data items (and other entities related to them, iteratively) should also be included in the results. A relationship target filter is specified as a function over a *relationship target*, defined below, returning a boolean value specifying whether the data item that the relationship target represents is within scope.

Relationship Target

A *relationship target* is defined as follows.

Definition 7.5 (Relationship Target) *A relationship target is the full set of information about a p-assertion data item that is the subject or object of a relationship p-assertion.* □

It includes the following (also depicted in Figure 7.6):

- The interaction key, including source and sink, of the interaction in which the data item was exchanged or part of an actor's state.
- Whether the actor that asserted the data item was the sender or receiver in the interaction.
- The local p-assertion ID of the p-assertion in which the data item is contained.
- The parameter name of this data item in the relationship p-assertion.
- The address of the provenance store in which the p-assertion is documented.
- The location of the data item within the p-assertion, given by the data accessor.
- The name of the relation of which this target is a subject or object.
- The identity of the actor that asserted the data item.
- The content of the interaction or actor state p-assertion containing the data item.

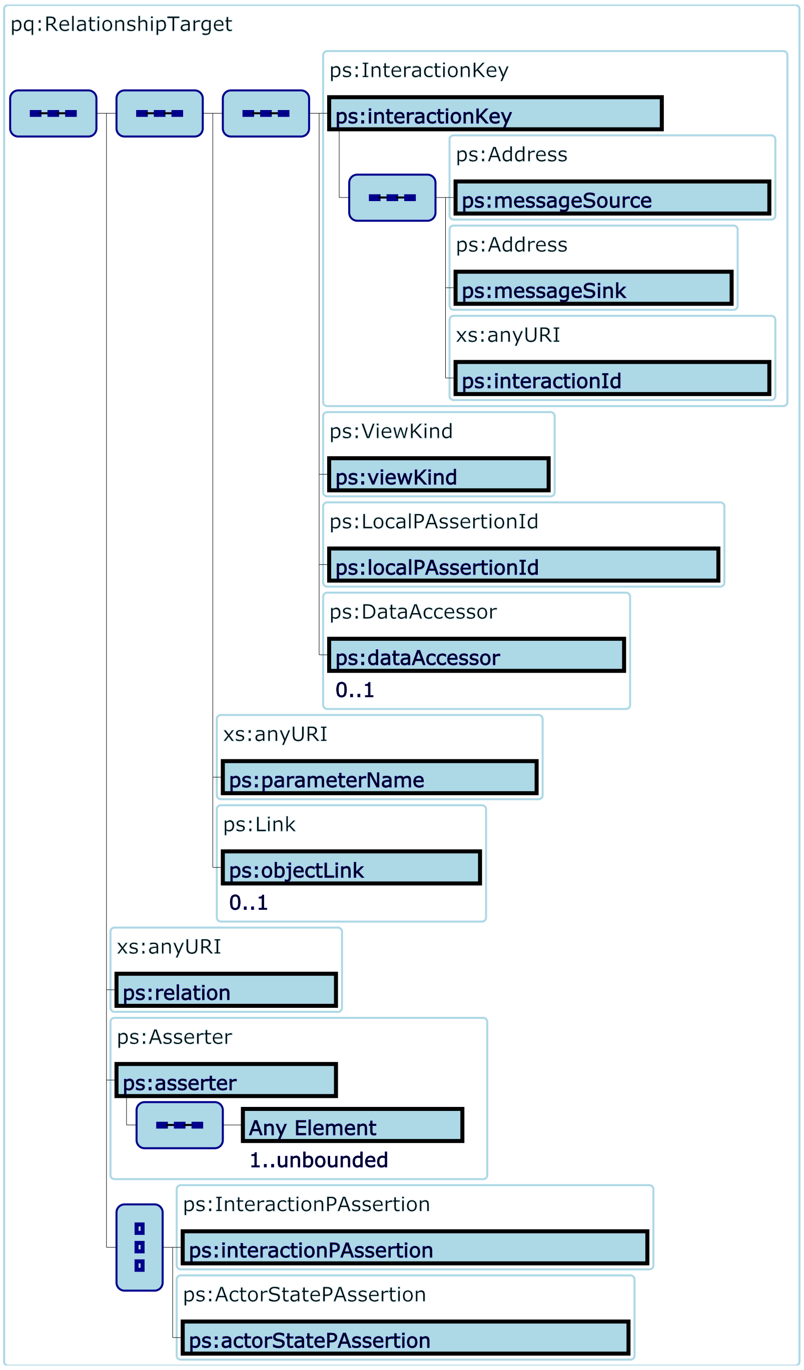


Figure 7.6: Relationship Target model

Checks on P-Assertion Content

In order to check whether a relationship target is in scope, a relationship target filter must specify how to find the relevant data against which to compare criteria. As the data must be about the relationship target, the properties that mark the target as being in scope must be evident in the set of information described above. The relationship target filter can be seen, then, as a *search* over the relationship target for the properties that identify the target as being within scope: if those properties are found then the target is accepted.

One of the pieces of data that makes up a relationship target is the content of the p-assertion containing the data item, and scoping a p-assertion may depend on checking those contents. For example, the relationship target filter may exclude interactions of a particular operation type, and this operation type will be apparent only within the content of an interaction p-assertion. As the content of the p-assertion may be in any application-defined format, we have the problem of defining a search over data of arbitrary structure: the same problem as faced in specifying the query data handle.

Therefore, as with the query data handle, the relationship target filter includes a set of *document language mappings*, to translate p-assertions in different document languages to the one required by the relationship target search.

Model

A model of the relationship target filter is shown in Figure 7.7. The *check* element specifies a function, using a chosen search language, over relationship target returning true or false. The *documentLanguageMappings* specify how p-assertion contents are mapped to the document language required by the search.

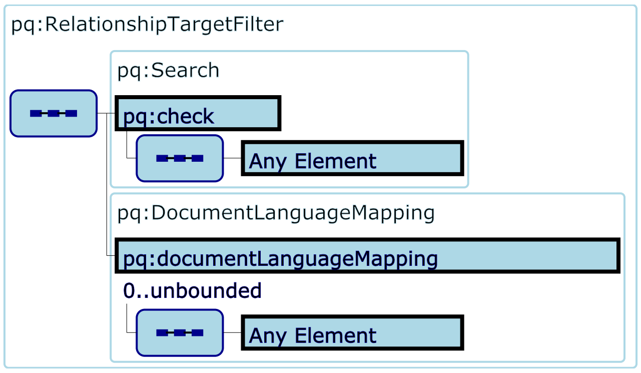

pq:RelationshipTargetFilter
pq:Search
pq:check
Any Element
pq:DocumentLanguageMapping
pq:documentLanguageMapping
0..unbounded
Any Element

Figure 7.7: Relationship Target Filter model

7.2.3 Provenance Query Results

The response to a provenance query is a representation of the provenance of an entity at a given instant, i.e. the one specified by the query data handle. Provenance query results are comprised of *start* p-assertion data keys and a set of *full relationships*, defined below and depicted in Figure 7.8:

Definition 7.6 (Provenance Query Result Start) *A provenance query result start is the p-assertion data key(s) to the process documentation of the entity for which the provenance was found, i.e. the key(s) for the p-assertion data item(s) found by resolving the query data handle.* □

Definition 7.7 (Provenance Query Result Full Relationships) *A provenance query result full relationship is a relationship between two p-assertion data items in the provenance of entity found by the query.* □

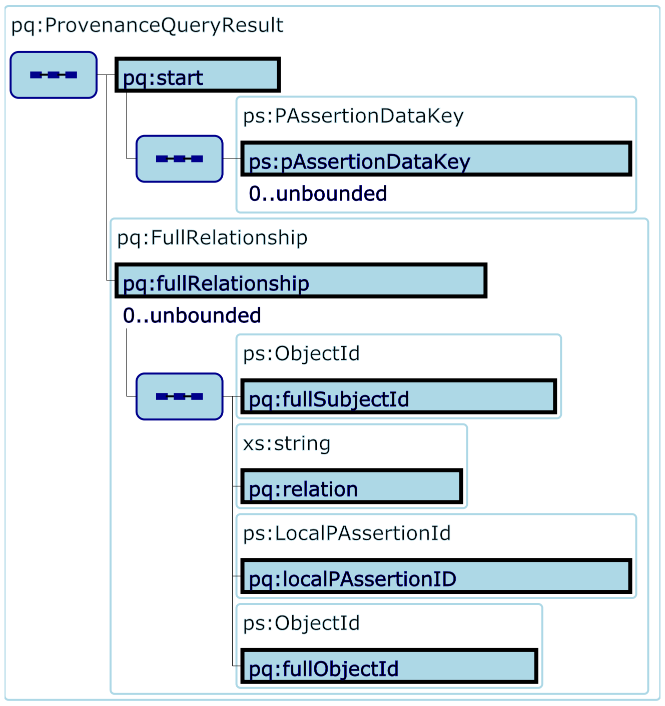


Figure 7.8: Provenance Query Result model

Full relationships are adapted versions of relationship p-assertions in the process documentation, differing in two regards. First, because a relationship p-assertion can have multiple objects, and not every object may be within scope of the provenance query results, a full relationship is between strictly one subject and one object, to indicate that that exact relationship is within scope. If multiple objects of a relationship p-assertion are within scope, there will be one full relationship for each object returned in the provenance query results. The second difference is that a relationship p-assertion is asserted within the context of an interaction, while a full relationship is designed to be independent of any one interaction (as it documents the provenance of an entity, not an interaction), so that the form in which each is specified may differ. In particular, part of the global p-assertion key of the subject of a relationship p-assertion can be determined from the identity of the interaction that the relationship p-assertion makes an assertion about, while this must be made explicit in a full relationship.

The model for the subjects of full relationships is shown in Figure 7.9, and the model for objects is identical.

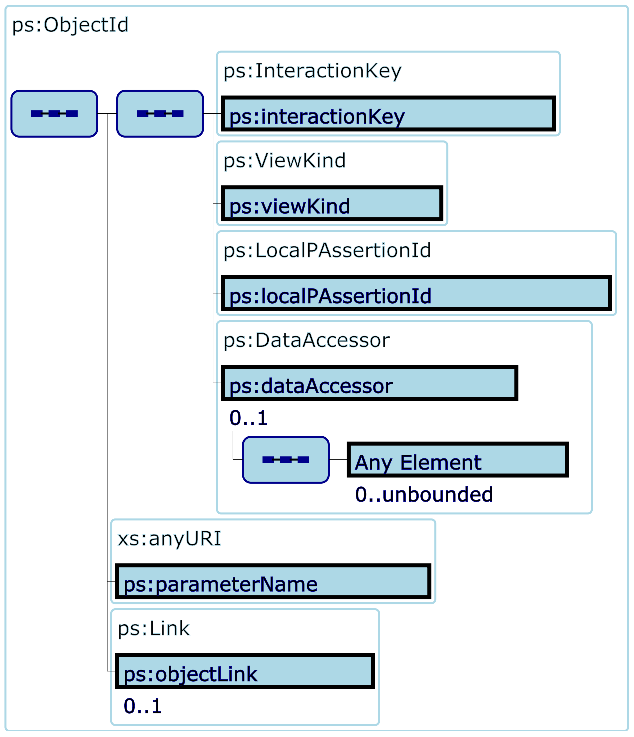


Figure 7.9: Full Relationship Subject model

7.3 Process Documentation Query Interface

Provenance query results include a related p-assertion data keys. To retrieve the actual process documentation that makes up the provenance of an entity at an instant, the querying actor uses the *process documentation query interface*. This interface gives direct access to the process documentation contents, by allowing the querying actor to search over and retrieve parts of the p-structure (†). [TSR-1-4, p. 125]

At minimum, the process documentation query interface must allow the querying actor to perform the following operations (†). [SR-1-2, p. 115]

- Retrieve the contents of a p-assertion with a given global p-assertion key.
- Retrieve all p-assertions asserted about one interaction, identified by an interaction key, by one actor, identified as the sender or receiver or by its identity.
- Retrieve all p-assertions asserted about one interaction, identified by an interaction key.

The actual results of the query depend on the contents of the provenance store to which the query is sent, because the query will only return data contained in that provenance store, and the access control restrictions placed on the querying actor by the store.

As the amount of data returned may be large in volume, the process documentation query interface should allow for the iterative retrieval of query results. By this mechanism, a querying actor should be able to process the results in manageable chunks. (†) [TSR-1-2, p. 125]

Ideally, the process documentation query interface should allow more than the above minimum operations, so that queries can be used to search for and retrieve more p-assertion data meeting different criteria, e.g. to retrieve all interaction p-assertions of a particular type, and possibly to perform transformations on the results before returning, so that the querying actor receives the results in the form they can most easily process.

The process documentation query interface is not more fully specified here because there are a range of query languages already available that can be used to query a set of stored data, and the ideal one will depend on the structure of a particular application's p-assertion content.

7.4 Management Interface

Within a provenance architecture, a management interface is necessary in order to facilitate the administration, reuse and maintenance of provenance stores. Such an interface may provide generic data storage administration capabilities and will not, in itself, be provenance specific. This being so, this section merely suggests some useful functionality that a management interface might have, and we do not make any commitments to a formal specification of the suggested functionality described below.

7.4.1 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. A management interface should provide the following functionality regarding notification.

- Notification

  A management interface should be able to notify subscribed managing actors of record and query operations. (†) [SR-4-2, p. 122]

- Subscription management

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

7.4.2 Provenance Store Utility

- Link Modification

  A management interface could provide functionality to update links. This is useful when process documentation has been moved from one provenance store to another and contains links.

- Deletion

  Ideally, p-assertions are never deleted, but in some circumstances, such as data staging, it may be necessary for an application to delete particular p-assertions that have been moved to a new provenance store. A management interface may provide this deletion capability.

- 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. A 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.

7.5 Policies

Policies describe the capabilities, requirements and general characteristics of components in service oriented systems. In the Provenance architecture, they are important for providing interoperability, enabling users to identify services and provenance stores that provide the required functionality. In Chapter 3, policies identified for the Provenance architecture were clarified into the three distinct areas shown below.

- Service requirement and capability policies.

- Provenance store policies.
- User requirement policies.

Although we list service and user requirement policies, these are, in effect, subsets of the policies describing service and provenance store capabilities. For example, a service may require that a provenance store can cope with high levels of throughput, which may be necessary in situations of data staging where large numbers of p-assertions are being moved around from store to store. However, such policies can be seen as a subset of the policies expressing the capabilities of provenance stores, and as such we avoid repetition by only describing policies that specify capabilities.

7.5.1 Provenance Store Capability Policies

Policies describing the capabilities of provenance stores are necessary to enable services that want to record p-assertions to find provenance stores that offer the required capabilities. In this section, we provide a description of several policies that are necessary to ensure provenance store interoperability. Note that policies for service capabilities are described in Chapter 8, Section 8.6.

In certain situations, a recording service may need to store the data about which it makes assertions in some persistent storage location external to the provenance store. In these cases, the data is referenced in the associated p-assertions to enable querying actors to dereference the data when the p-assertions are obtained from the provenance store. An alternative method is for the provenance store itself to periodically resolve the references contained in p-assertions, retrieve the data and store it locally (see Chapter 5, Section 5.4 for a discussion of reference recording). Provenance stores may also perform this functionality more or less frequently. To be useful, this capability of the provenance store must be advertised by an appropriate policy so that querying services can discover it.

A data upload policy therefore is defined as follows.

Definition 7.8 (Data Upload Policy) *A data upload policy identifies if and how frequently a provenance store has to resolve a reference contained in a p-assertion.* □

When applications are running with high throughput, and large amounts of p-assertions are being recorded, it may sometimes be necessary to create a temporary provenance store close to the application actors (in terms of network location) in order to cut down on communication costs (this technique is called data staging and a more detailed discussion can be found in Chapter 5, Section 5.3, and see Chapter 6, Section 6.5 for a discussion on the reference documentation style). When the process for which p-assertions are being recorded finishes, the temporary provenance store can query itself to obtain all of its p-assertion content and transfer this to another provenance store. In terms of policy, several criteria need to be specified.

- A temporary provenance store must indicate in its data staging policy the fact that it is temporary.
- It must describe where (i.e. which other provenance stores) p-assertions can be migrated to.
- In some cases, a provenance store may act as a persistent store for one asserting actor but perform data staging for another asserting actor. The different forms of data storage performed by a provenance store for different asserting actors should be specified.
- The above kinds of data staging represent cases where a data staging provenance store 'pushes' the data over to another provenance store. However, there may be cases where it is more desirable to have provenance stores 'pull' assertions from other data staging stores. Both functionalities are catered for by a provenance store's ability to *query* and *record* p-assertions (see Section 7.2 and Section 7.1 respectively).

In summary, the definition for a data staging policy must capture these criteria and we present such a definition below.

Definition 7.9 (Data Staging Policy) *A data staging policy must specify whether or not a provenance store is capable of data staging and, if so, to which other provenance stores it can send its contents, its data staging capabilities for different classes of recording actors, and whether it is capable of both recording p-assertions into other provenance stores (push-based data staging) or querying other provenance stores (pull-based staging), or both.* □

When an asserting actor is recording many p-assertions iteratively, templates can be used to cut down on computational costs (see Chapter 5, Section 5.5 for a discussion). In essence, templates can generate p-assertions given a correct set of parameters by an asserting service. In some cases, the provenance store may allow a recording actor to submit a template which can subsequently be sent parameters by a recording actor leading the template to generate the appropriate p-assertions. For this to be practical, both recording actors and provenance stores must share the same template language for producing p-assertions.

A p-assertion template policy is defined in the following manner.

Definition 7.10 (P-Assertion Template Policy) *A provenance store template policy must indicate the ability of the store to accept templates from asserting actors, what languages for producing p-assertions the store supports and from which recording actors it can accept templates from.* □

When querying a provenance store, a querying service must know what search language is supported by the provenance store for queries (Section 7.2 describes the approach taken in the architecture for provenance store querying). Several search languages may be supported by any given store (such as XPath in the case of an XML document language), and the query languages supported should be advertised by a policy.

The definition of a search language policy is given below.

Definition 7.11 (Search Language Policy) *A search language policy states the kinds of search languages the provenance store supports.* □

A provenance store can potentially hold very large amounts of p-assertions, and to manage these it may be necessary to perform indexing of the contents of provenance stores to facilitate search and retrieval (see section 7.4.2). The different indexing schemes offered by a provenance store (e.g. by time, trace etc.) should be made available to querying services by appropriate policies.

The index management policy definition is as follows.

Definition 7.12 (Index Management Policy) *The index management policy must state that a provenance store can or cannot perform indexing and, if it can, the different kinds of indexing offered by the store must be enumerated.* □

As noted in Section 6.9, the asserting actor signs the p-assertion it creates; this signature can then be subsequently verified by either the provenance store or the querying actor that retrieves this p-assertion at a later stage. If the provenance store is responsible for the verification process, then this occurs at the time the p-assertion is submitted for storage by the recording actor. If the signature does not verify, an appropriate error message should be generated. To indicate that a given provenance store can indeed validate the signature on a p-assertion and determine its corresponding asserter identity, we require a suitable policy to state such capability, which we define below.

Definition 7.13 (Security Signature Checking Policy) *A security signature checking policy states whether or not a provenance store has the capability to examine and validate the signatures placed in a p-assertion by an asserting actor, as well stating what action is to be taken if a conflict is detected (e.g. sending an error message).* □

It is important to note that the validity of the signature on a submitted p-assertion is independent of whether the recording actor has the appropriate access rights to record that p-assertion to the provenance store in the first instance. For example, a recording actor may have access rights to store p-assertions in a provenance store, but may be refused if the p-assertions it submits do not have valid signatures. Conversely, a recording actor may possess correctly signed p-assertions, but not have the authorisations to record them to a specific provenance store.

Given such policies, services can understand the behaviour of a given provenance store. By allowing services without prior knowledge of a provenance store to discover its policies, they can thus understand how to interact with and what to expect from it, and this ensures interoperability.

7.6 Security

The security policy of the provenance store essentially encompasses the internal representation list, authorisation policy 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 policy.

The access control policy also defines the manner or protocols used by the remote interactor in any secure interaction with other entities, for example other actors, provenance stores or trusted third parties. Advertisement of this portion of the policy to other entities interested in interacting with the provenance store supports interoperability as far as secure communication is concerned.

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 policy. 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 (see Section 7.4).

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 5.7.

7.7 Conclusion

This chapter presented a more detailed, informal description of functionality to be supported by the provenance store. Ultimately, it will lead to separate specification documents that will define how the interfaces are instantiated in specific infrastructures.

Chapter 8

Actor Behaviour

8.1 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 p-assertion 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 must 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. These behavioural rules are aimed at designers of provenance-aware applications: the reference semantics of the provenance architecture is only defined in the case where behavioural rules are followed. In this chapter, we use the words 'should' and 'must' to indicate requirements levels as specified in [Bra97].

8.2 Architectural Rules

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

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

Rule 8.1 (Unique Interaction Key Rule) *A sender asserting actor (i.e. a sender in the role of an asserting actor) must assign a unique interaction key to an interaction.*□

There are many ways actors can obtain interaction keys. For example, an actor could generate an interaction key itself or obtain an interaction key from a naming service. The interaction key assigned to an interaction by a sender must be passed to the receiver in an interaction so that the receiver may also record p-assertions about the same interaction. Therefore, we introduce the interaction key transmission rule.

Rule 8.2 (Interaction Key Transmission Rule) *A sender asserting actor must transmit the interaction key it assigns to an interaction to the receiver 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 8.3 (Appropriate Interaction Rule) *An asserting actor must use the interaction key associated with an interaction, I, when asserting p-assertions about I.*□

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

Rule 8.4 (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.

Furthermore, if p-assertions are recorded across multiple provenance store, actors need to record links between provenance stores so that the provenance of a data item can be found. Therefore, we introduce the link recording rule.

Rule 8.5 (Link Recording Rule) *An actor must record a view link (see Section 5.2.1) to the provenance store that contains the corresponding actors view of the interaction if the view is in another provenance store. Moreover, if an actor makes a relationship p-assertion where objects of the relationship are in other provenance stores, actors must create object links (see Section 5.2.2) for them.* □

As we can see from these rule definitions, it is difficult to conceive a protocol that would enforce their execution. Should such a protocol be designed, it would require multiple extra message exchanges between actors, which would impact on application performance. Instead, the rules are aimed at designers, who we expect will implement them in their provenance-aware applications. To help them, the actor side library discussed in Section 8.7 provides some support to this end.

8.3 Tracers

Section 3.4 introduced tracers as a mechanism for demarcating processes. Tracers (†) [AER-9, p. 133] are tokens that are passed between actors based on the actor's internal knowledge and the semantics of each tracer. The semantics of a tracer are defined by the rules used by actors to determine when to generate and/or propagate it. These rules are defined by the tracer's type. This section describes a particular type of tracer and its semantics. If an actor follows the rules placed on it by a tracer's semantics, 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 which 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.

Given this assumption, an actor, A, is said to be *inferior* to another actor, B, when A received a request from B and A returned a response back to B. Likewise, an actor B is said to be *superior* to an actor, A, when it sent a request to A and receives a response back from A. Using these definitions, we can now present a particular type of tracer.

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 type has 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.

8.3.1 Session Tracer

We now define a tracer type that is useful for grouping together interactions that belong to the same workflow. It shows how interactions can be part of sub-workflows whose results are used in a larger workflow. Tracers of this type are termed *session tracers* and follow the subsequent three rules.

Rule 8.6 (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 8.7 (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 8.8 (Propagation Rule: to superior) *An inferior actor must add the session tracers supplied by its superior to its response to the superior.* □

Figure 8.1 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 tracers a and b. The enactment engine passes along the tracer according to propagation rule 8.7 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 8.8. 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.

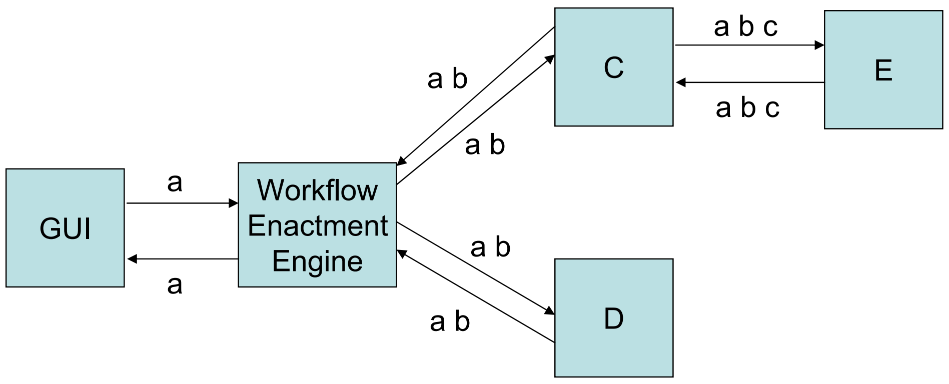


Figure 8.1: Diagram showing session tracers

8.3.2 Other Tracers

Other types of tracers are conceivable. They may be generic or application specific. These tracer types may also use a messaging model other than the client-service model presented here. For example, an application specific tracer in the medical domain could be used to identify what interactions belong to a particular patient's case.

8.4 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-repudiatable fashion. For the case of actors, we can introduce two rules:

Rule 8.9 (Signature Rule) *Asserting actors must digitally sign the p-assertions they create.*□

The designated locations for the signatures are shown in Figure 6.11, 6.12 and 6.13. This rule ensures that accountability for the information contained within a p-assertion can be traced back to its creator.

Rule 8.10 (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 8.11 (Secure Behaviour Rule) *Actors should use the necessary security functionality and credentials required to interact with each other or the provenance store in a secure manner as dictated by the appropriate policies. This functionality and credentials should ideally be provided by the environment in which the actors operate in.*□

An example of such security functionality could be a signature generating algorithm, while credentials such as certificates could be provided through key stores.

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 8.12 (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 sender application actor invoking a receiver 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 sender and receiver.

Rule 8.13 (Tokens Rule) *If actors utilise security tokens (such as signatures) in specific portions of messages exchanged between them, 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 verbatim documentation style is employed (Definition 6.12).

8.5 Documentation Style Driven Message Transformation

As already defined in Definition 6.11, documentation style is a representation of the transformation according to which the content of a message is asserted in an interaction p-assertion. A number of constraints need to be satisfied in order for a documentation style driven message transformation to be correct. The purpose of this section is to specify the rules to be followed. First, we illustrate the relationship between the different entities involved in such a transformation in Figure 8.2.

We consider an *original message* (d1) as sent between actors in an application. At runtime, the asserter actor applies to this message one or more documentation styles transformations which result into a *transformed message* (d2). The message transformation is an operation performed at execution time by the asserting actor. Such a transformation can be described by a *message transformation description*; we are expecting such a message transformation description to be declarative, so that it can characterise both a tranformation and its reverse transformation (when the transformation is reversible).

Messages are always expected to be structured according to some schema. Hence, the original message (d1) should satisfy its *original schema* (S1), whereas the transformed message (d2) should satisfy a *transformed schema* (S2). In the most general case, schemas S1 adn S2 may be different: for example, in the reference documentation style (Definition 6.13), we replace an element with a reference, which means that the corresponding schema should change accordingly to reflect that replacement. Hence, the original schema S1 is also transformed to schema S2 according to a schema transformation, itself described by *schema transformation description*. Both the message transformation description and the schema transformation description have to be related, in accordance to the documentation style.

It is recommended that both schemas, as well as their transformations, should be predefined by the application designer at design time. We note that, is some cases, the schema of the transformed message could only be defined at runtime, as a result of a message transformation. This has some challenging implications not only from a performance view point, but also from a manageability view point, because the new

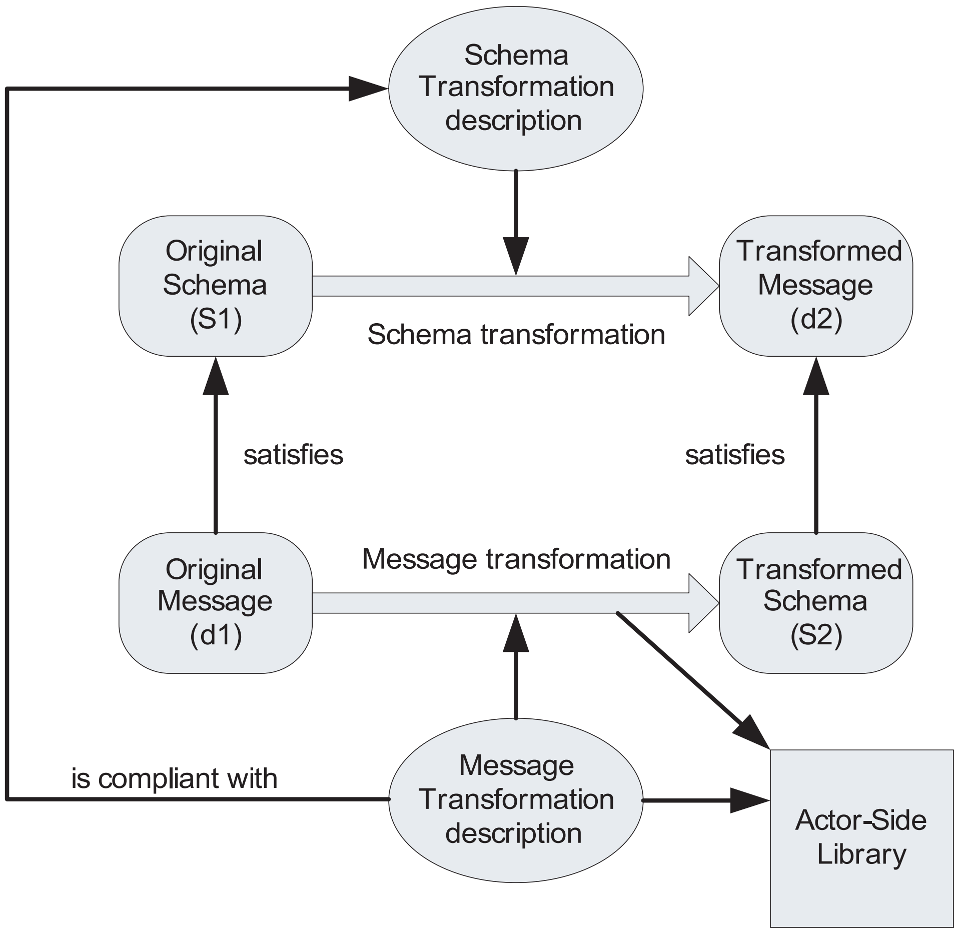


Figure 8.2: Documentation Style Driven Message Transformation

schema has to be published and shared so that querying actors can make use of it. To avoid such problems, we recommend that designers identify at design time the schema S2 of the transformed message.

Such constraints on transformations can be formalised in the following behavioural rules. When a documentation styles are used, transformations are applied in accordance with the Rules 8.14, 8.15 and 8.16.

Rule 8.14 (Message Transformation Rule) *A message must be transformed according to a message transformation description as defined by the documentation style being used.* □

Rule 8.15 (Schema Transformation Rule) *The schema of a message being transformed must also be transformed according to a schema transformation description as defined by the documentation style being used.* □

Rule 8.16 (Transformation Description Compliance Rule) *The message transformation description of a given documentation style must be compliant with the schema transformation description of that documentation style.* □

The *actor side library* provides some support for the above mechanisms (see Section 8.7). Specifically, the message transformation description that happens at runtime is part of the actor side library implementation.

8.6 Actor Capability Policies

In order for users of a provenance system to be able to record p-assertions, they must be able to discover services capable of asserting and recording process documentation. By advertising their capabilities in policies, such services allow potential users to find them. In this section, we describe the various service capability policies that ensure that services can be found and understood by users, and by doing so we ensure interoperability.

The standard capabilities of a provenance-aware service include *record* and *query*. For each, there are several policies that are necessary to specify.

8.6.1 Recording

Asserting actors should advertise the kinds of p-assertions that they can assert. Thus, for example, an application service actor may be designed to document all, some, or none of its internal state as actor state p-assertions; the different levels of documentation detail available should be stated in a policy. Similarly, asserting actors may have the ability to document certain relationships between interaction or actor state p-assertions, which may be of interest to a user and, thus, must advertise such capabilities in order that users can discover which asserting actors offer the right kinds of documentation capabilities.

Definition 8.1 (Assertion Category policy) *An assertion category policy should specify what categories of p-assertions a service can record.* □

Services may offer all, or only a subset of documentation styles (see Chapter 6, Section 6.5 for a discussion of documentation styles). Users must be able to choose services offering the desired documentation style and this capability should be expressed by an appropriate policy.

Definition 8.2 (Documentation Style Policy) *A documentation style policy should describe the various different kinds of documentation style a recording service offers.* □

Different provenance stores may have different restrictions on access as dictated by their respective access control and authorisation policies (Section 4.3.1); recording services must specify which provenance stores they are able to record to and under what restrictions.

Definition 8.3 (Recipient Store Policy) *A recipient store policy should specify which provenance stores a service can record to.* □

8.6.2 Querying

Just as in the case of recording, different provenance stores have different restrictions on query operations as dictated by their respective access control and authorisation policies (Section 4.3.1); querying services must specify which provenance stores they are able to query and under what restrictions.

Definition 8.4 (Query Store Policy) *A query store policy states the which provenance stores a querying service has access to.* □

8.6.3 Service Requirement Policies

Service requirement policies describe any requirements that a service may have on the provenance store such as, for example, that a store must have a high level of throughput, which may be necessary in situations of data staging where large numbers of p-assertions are being moved around from store to store, and will be specified by the service administrator. However, such requirements policies can be seen as a subset of the policies described elsewhere for provenance store capabilities (see Chapter 7, Section 7.5). Services with requirements must match these to the capabilities of the available provenance stores using the advertised policies.

8.7 Actor Side Library

In this section, we introduce the actor side library and its usage. The actor side library (ASL) is a collection of functions, which allow provenance-aware applications to communicate with provenance store services. It also provides functionality to help application developers enforce the above-mentioned rules.

Although the actor side library is an important part of functionality of a provenance system, much of its detail is implementation specific. Many implementation recommendations have been identified in Section 9.6.3. A separate document [JGM+06] has also been provided referring to our specific implementation of actor side library. In this section, however, we only give a brief account of the role of such a library.

An implementation of the actor side library should contain at least one of actor side query library, actor side record library and actor side management library. It should also contain a number of facilities, each of which enforces certain rule(s).

An application developer can use the actor side library to communicate with provenance store services. He can also ensure actor behaviour rules are suitably enforced in his provenance-aware applications by invoking these corresponding facilities. After integration with applications, the actor side library becomes a part of provenance-aware applications and should be distributed with the applications.

8.8 Conclusion

This chapter has defined a number of expectations on actor behaviour. These expectations include a set of architectural rules that inform system developers about actor behaviour with respect to the identification of interactions and the recording of links. We also provide a description of tracers in terms of propagation rules. Security rules are also introduced. These rules define actor behaviour for recording p-assertions in a secure manner. Furthermore, we discuss how actors are expected to transform p-assertions when using documentation styles. Policies are then defined for provenance-aware actors. Finally, actor side libraries are introduced as the architectural component that can be used by system developers to make sure their actors adhere to the rules and expectations defined in the chapter. Actors must follow the expectations outlined here in order for process documentation to be correctly recorded, managed, and queried.

Chapter 9

Justification

In this chapter, we study the impact of various requirements on the architecture and how these are addressed (or otherwise) by features of the architecture. In some specific cases, the requirements lead to implementation recommendations, which we enumerate in Section 9.6. The requirements we study are those from the software requirements document [And05b], from the security [IHT05], scalability [IGM05] and tools [Ran05] workpackages, and additional requirements from the two target application domain areas (organ transplant management [VSWTV05] and aerospace engineering [KS05]).

All requirements enumerated in this chapter are quoted verbatim from the original documents.

9.1 Software Requirements Document

This section explains how the software requirements for provenance system are satisfied by the functionality provided by the architecture described in this document. These requirements are sourced from the software requirements document [And05b] and are listed in the remaining sections of this chapter along with the corresponding feature of the provenance architecture that addresses it.

9.1.1 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 recording and querying interfaces which can be used respectively to store and retrieve p-assertions to/from the provenance store (cf. section 3.3, p. 27 and section 3.3, p. 27)). □

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 (cf. section 3.3, p. 27) with multiple levels of query capability that support the required granularity of trace to be retrieved. The navigation capability allows p-assertions to be retrieved from the provenance store (cf. section 7.3, p. 99), whereas the provenance-specific query capability allows for more process-oriented results to be retrieved (cf. section 7.2, p. 89). □

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 the querying interface (cf. section 7.3, p. 99); the specific functionality of archival is outside the domain of the logical architecture, but would require the use of a long-term persistent storage capability. □

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 (cf. section 7.3, p. 99). The queries can be generated in the appropriate format to allow accessibility at the desired level. Comparisons are not performed by the querying interface, but are the responsibility of processing services (cf. section 3.3, p. 29). (Implementation Recommendation IR-P-1, p 135.) □

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 by the query interface (cf. section 3.3, p. 27), and can be subsequently stored for future use. The provenance architecture however does not manage any storage space (except in the internal implementation of the provenance store itself); therefore, managing the storage of queries and results is outside the scope of the architecture. □

SR-1-6 The provenance architecture should allow a user to access a Provenance Record based on the time and date (calendric information) at which the Record was stored. □

Design Feature: The recording interface (cf. section 3.3, p. 27) does not mandate *when* the provenance store actually stores p-assertions in persistent storage. This design decision allows implementers to choose efficient implementation techniques that do not impact performance of the store, such as asynchronous storing. Thus, it is unclear what the semantics of such a time information would actually be, given that storing time is not specified by the architecture. Alternatively, we could request provenance stores to add time information when an p-assertion is submitted through the recording interface. Given the asynchronous nature of the recording protocol, this moment is not clearly specified either. Ultimately, other forms of time information may be of use to some applications: time a p-assertion is backed up, time a p-assertion is replicated, ... We see such information as a form of metadata relating to the lifecycle of p-assertions within provenance stores. The provenance architecture however does not specify such a lifecycle because it may constrain implementers of provenance stores unduly. Instead, several alternate approaches are proposed to support this requirement. First, asserters of p-assertions can use actor state p-assertions (cf. section 2.4, p. 20) to assert the time at which interactions occur, or can define types of p-assertions that contain suitable time information. Second, a generic metadata facility can be used to annotate (with time information or other) p-assertions recorded in the provenance store; such metadata facility is generic and does not belong to the provenance architecture. (Implementation Recommendation IR-PS-1, p 134.) Third, the principles described in this architecture can also been applied for the interaction(s) between an asserter and a provenance store. Both can document their execution in a provenance store, and in particular the time at which they send and receive a record message, which can be expressed as actor state p-assertion. (Implementation Recommendation IR-PS-2, p 134.) □

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 analyser are examples (cf. section 3.3, p. 29) of the processing services group (cf. section 3.3, p. 29), introduced by the logical architecture. Such services need to be implemented and have to make use of the query interface (cf. section 3.3, p. 27) to retrieve information from the provenance store. □

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 (cf. section 3.3, p. 29). □

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: It is the responsibility of tools developers to provide libraries similar to actor-side libraries that expose the provenance store functionality (cf. section 3.3, p. 27). □

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 provenance store to be used can be specified as part of a configuration decision by the service or a user. The actor-side libraries (cf. section 3.3, p. 27) should be configurable and allow the identity of provenance stores to be specified. (Implementation Recommendation IR-ASL-1, p 135.) □

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 interfaces (cf. section 3.3, p. 27); however, storage and retrieval across multiple repositories is a storage layer issue that is not addressed by the architecture, but is left to specific implementations of provenance stores. (Implementation Recommendation IR-PS-3, p 134.) □

SR-1-12 The system should support the recording of different provenance information views related to an event or an entity. □

Design Feature: The recording protocol supports different views of interactions (senders and receivers) (cf. section 7.1, p. 87) and allows different assertions to be made about interactions, states and relationships (cf. section 2.4, p. 20). □

SR-1-13 The provenance architecture should support the migration of provenance data among provenance stores. □

Design Feature: Migration of p-assertions is composed of querying of the source provenance store, followed by recording into the target provenance store, and finally deletion of the p-assertions in the source provenance store. The recording and querying interfaces (cf. section 3.3, p. 27, section 3.3, p. 27) can be used to that end. On the other hand, the provenance store does not provide any deletion or update capability in its APIs so as to offer immutability. Such functionality may instead be offered by the storage layer. The issue of migration of p-assertions is also discussed in Section 5.3. (Implementation Recommendation IR-PS-5, p 134.) □

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 enforced by the logical architecture. This is essentially a quality of service element that has to be provided by the host of a provenance store. In addition, implementations of provenance store should consider the expiry of cryptographic public keys. (Implementation Recommendation IR-PS-10, p 135.) □

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: The architecture is open and allows for analysis and reasoning services to be implemented as processing services (cf. section 3.3, p. 29) that interact with the provenance through its query interface (cf. section 3.3, p. 27). As indicated in SR-1-5, the actual storage functionality is beyond the remit of the provenance architecture. □

SR-1-16 The provenance architecture should provide support for maximum automation of the provenance recording mechanism. □

Design Feature: The recording interface (cf. section 3.3, p. 27) provides for the recording of one or more p-assertions, and as such offers no automation. Such a feature has to be offered by the actor-side libraries (cf. section 3.3, p. 27). Two different approaches may be adopted by these: policies allow users to specify in a declarative manner what information and when it should be recorded. (Implementation Recommendation IR-ASL-2, p 135.) Alternatively, automation support may also be provided with knowledge of the actor hosting environment, for instance to capture invoked messages, or to recover information from logs. (Implementation Recommendation IR-ASL-3, p 135.) □

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 (cf. section 3.3, p. 27) allow integration into the existing system, whilst the UIs, processing (cf. section 3.3, p. 29) and presentation services allow the provenance architecture to be used as an external third-party service. The derivation engine (cf. section 4.3.1, p. 42) of the provenance security architecture allows policies of the client domain to be reflected into the provenance access control functionality. □

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. Capability to handle errors as suggested by this requirement (or others) may be offered by actor side libraries (cf. section 3.3, p. 27). (Implementation Recommendation IR-ASL-4, p 135.) □

9.1.2 Performance Requirements

SR-2-1 The additional execution overhead for an application recording provenance information should be kept to a minimum. □

Design Feature: Execution overhead is implementation-specific and cannot be enforced or specified at architectural level. The architecture however is designed in such a way that it allows asynchronous recording and has ways of dealing with large data structures allowing impact of p-assertion recording to be minimised. □

SR-2-2 : Storage space requirements of the provenance architecture for provenance information recording should be kept at a reasonably low level. □

Design Feature: The p-structure (cf. section 6.8, p. 78) is a well-defined data structure whose space requirement can be predicted at design time. A trade-off exists between the amount of storage required to document execution and its level of detail. Since storage requirements can be predicted, the designer of a provenance-aware application may therefore take decisions to record (or not) aspects of execution that are relevant (or not) to the provenance queries to be supported by the application. □

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 architecture is agnostic about the transport layer used to deliver record messages to the provenance store. Protocols that offer the guarantee of once-and-once-only delivery may be used in applications that have such a requirement. We note that the provenance store has the idempotence property (cf. section 2, p. 86), which allows a same message (with same local identified) to be submitted multiple times, with the guarantee that it affects the provenance store at most once. □

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 architecture provides several ways of dealing with large amounts of data. On the one hand, appropriate recording patterns (cf. section 5.1, p. 51) allow for deployment of provenance stores that are local to the asserter, and therefore avoid transfer of large amounts of data over long distance network. On the other hand, assertions (or part of them) may be passed by reference, and can be used to prevent the submission of large data sets; such a facility is offered by documentation style (cf. section 6.4, p. 72). □

9.1.3 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: SR-3-1-1, SR-3-1-2 and SR-3-1-3 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. □

SR-3-1-4 The provenance architecture should support an XML-based API format for provenance data. □

Design Feature: The architecture described in this document is format and technology neutral. Its implementation by EU Provenance will be Web Services based, and therefore, will be offering WSDL specifications to describe service interfaces. □

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. □

Design Feature: The architecture described in this document is format and technology neutral. Its implementation by EU Provenance will be Web Services based, and its query functionality will return XML representations of p-assertions. □

9.1.4 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: Once retrieved from the provenance store, any p-assertion or derived data can be viewed or rendered by a user interface. Such data can be updated locally, but by its immutability property, documentation of execution cannot be changed in the provenance store. (Implementation Requirement IR-PS-5, p 134.) □

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: A processing service, such as the provenance store visualiser (cf. section 3.3, p. 29) may poll the provenance store, by repeatedly issuing queries, or alternatively can subscribe to notification of change (cf. section 7.4.1, p. 100) in the provenance store. We note that the architecture does not guarantee any real-time constraint, and therefore the timeliness of such notifications is not specified. □

SR-4-3 The update frequency of provenance information displayed by the system on a HCI should be configurable based on policies. □

Design Feature: Following requirement SR-4-3, such policies are specific to the provenance store visualiser (cf. section 3.3, p. 29). □

SR-4-4 Human-computer interfaces presented by the provenance tools should be designed to allow multilingual support. □

Design Feature: (Implementation Requirement IR-P-2, p 135.) □

9.1.5 Documentation Requirements

SR-5-1 Detailed documentation of the provenance architecture public interfaces should be produced both for APIs and HCIs. □

Design Feature: This document provides concepts, definitions and abstract descriptions, which should be referred to by the documentation of specific components. □

SR-5-2 A detailed description of the administrative interface of the provenance architecture should be produced. □

Design Feature: This document provides concepts, definitions and abstract descriptions, which should be referred to by the documentation of specific components. □

9.1.6 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 authorisation engine (cf. section 4.3.1, p. 40) provides access control functionality, which is configurable through the access control policies in the security architecture (cf. section 4.3.1, p. 40, section 4.3.1, p. 41). The access control functionality could be configured to reflect the functionality of other access control systems. (Implementation Recommendation IR-PS-6, p 134.) □

SR-6-2 The provenance architecture should allow both automated and manual determination of access control rights. □

Design Feature: The access control policies (cf. section 4.3.1, p. 41) in the security architecture are manually determined by the system administrator . The derivation engine (cf. section 4.3.1, p. 42) 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: The user can determine the required level of security to be used in the recording process from the access control policy (cf. section 4.3.1, p. 41) 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 (cf. section 4.3.1, p. 42) of the provenance security architecture allows policies / access control rights of the embedding system to be reflected into the provenance access control functionality. (Implementation Recommendation IR-PS-6, p 134.) □

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 minimised. □

Design Feature: The derivation engine (cf. section 4.3.1, p. 42) 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 (cf. section 4.3.1, p. 38) 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: Process documentation will not be overwritten, modified or deleted by any operation in the supplied APIs. (Implementation Requirement IR-PS-5, p 134.) Non-repudiation can be accomplished using signatures, and the P-structure supports the signing of a p-assertion by an asserting actor (cf. section 6.9, p. 82). Signing functionality can be provided by actor side libraries. (Implementation recommendation IR-ASL-5.) □

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: Asserters of p-assertions can use actor state p-assertions (cf. section 2.4, p. 20) to assert the time at which interactions occur, or can define types of p-assertions that contain suitable time information. Second, a generic metadata facility can be used to annotate (with time information or other) p-assertions recorded in the provenance store. (Implementation Requirement IR-PS-1, p 134.) In either case, access can be provided to a trusted third party time stamping service. (Implementation Requirement IR-APP-4, p 137.) We also refer the reader to SR-1-6 for an analysis of timestamping by the provenance store. □

9.1.7 Other Requirements

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

Design Feature: The Provenance system is not a logging system and therefore comparison is difficult. Cost efficiency and robustness are deployment and implementation properties. □

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-1-16, SR-1-17, SR-6-4, SR-6-5. □

9.2 Tools Requirements

In this section, we discuss some of the requirements of the tools workpackage (as detailed in the tools workpackage deliverable D6.1.1 [Ran05]) and their impact on the architecture. Note that some requirements marked TSR-* in the tools document do not have corresponding requirements on the architecture (most after TSR-4-2 are requirements on the tools rather than the provenance architecture), so are not included here.

TSR-1-1 Provenance Architecture should support a Query Interface and a Submission/Recording Interface to the provenance store. Tools may make use of this interface to retrieve one or more p-assertions from the provenance store. □

Design Feature: See the Design Feature for SR-1-1. □

TSR-1-2 Provenance Architecture should provide support for storing results generated from a Query locally at the provenance store. These results can then be read from a temporary storage area by the Tools. □

Design Feature: The query interface temporarily stores results from a query, for iterative retrieval (cf. section 7.3, p. 99). □

TSR-1-3 Provenance Architecture should not impose constraints on access to the store via the Query or Management Interfaces. Tools may be able to read all or part of the PS directly. □

Design Feature: The query interface (cf. section 3.3, p. 27) can provide clients with direct access to all or part of the process documentation contained in a provenance store, depending on access control policies. The provenance architecture cannot prevent access to the back-end storage of process documentation by means other than the prescribed interfaces. □

TSR-1-4 The Query interface at the PS should allow search to be carried out on a data attribute or value. For instance, the tools may generate an Xpath query that needs to be processed by the PS, and this query would be defined with reference to one or more namespaces. □

Design Feature: The navigation capability of the query interface (cf. section 7.3, p. 98) is based on evaluating expressions against a hierarchical view of the documentation in provenance stores. □

TSR-1-6 The Recording interface at the PS may attach a timestamp with each p-assertion recorded. This timestamp may be in addition to one generated by the application submitting the p-assertion. Generally, no timestamp is produced by the tools. □

Design Feature: See the Design Features for SR-1-6 and SR-6-7. According to the SR-6-7, the tools may have to invoke the necessary timestamping functionality. □

TSR-1-8 The Query interface at the PS should provide some indication to the Tools about the list of queries that are waiting to be answered by the PS. Essentially, if a message queue exists at the PS, the contents and size of this query should be accessible via the Management interface at the PS. □

Design Feature: See the Design Feature for SR-1-8. A processing service which provides a scheduling/buffering facility can also provide a message queue for interrogation. The management interface of the provenance store provides functionality for managing process documentation. □

TSR-1-9 The Provenance Architecture should support a registry service to enable the software interfaces for all Tools to be visible. This is particularly true for any visualisation tools that may be application specific. Information about new tools, or updated versions of existing tools should be accessible through such a registry. This would be associated with 'Processing Services' in the Logical Architecture. □

Design Feature: The provenance architecture does not contain a registry, but one may be deployed as part of an application and used to register provenance architecture components. □

TSR-1-10 The Management interface at the PS should enable the identity of the PS to be queried. □

Design Feature: The obtaining of certificates and public keys for services can be achieved by many mechanisms, and is out of scope for the provenance architecture. □

TSR-1-17 The Provenance Architecture should not assume the existence of a particular set of services in the Tools Suite. □

Design Feature: No particular set of processing services, UIs or other tools are assumed or required in the provenance architecture. □

TSR-1-18 The Provenance Architecture should not necessitate the submission of a p-assertion from a particular type of client. No requirements on synchronous or asynchronous behaviour should be assumed. □

Design Feature: See the Design Feature for SR-1-18. □

TSR-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. □

Design Feature: See the Design Feature for SR-3-1-1. □

TSR-4-1 The Provenance Architecture should make no assumption about the types or modes of user interfaces being supported. □

Design Feature: The provenance architecture does not make any assumptions about the types or modes of user interfaces being supported. □

TSR-6-2 The Provenance Architecture should allow such access control to be supported (and configured) through the Management interface of the PS, and a specialist Credentials Management service (or similar). □

Design Feature: See the Design Feature of SR-6-2 for the ways in which the architecture supports configuration of access control . □

TSR-6-6 The Provenance Architecture should make no assumption about the recording format used. □

Design Feature: The provenance architecture is format and technology-independent. □

9.3 Scalability Requirements

In this section, we discuss some of the requirements of the scalability workpackage (as detailed in the scalability workpackage deliverable D5.1.1 [IGM05]) and their impact on the logical architecture.

SL-1 The provenance query interface should support the retrieval of large result sets. □

Design Feature: The query interface allows actors to iterate over query results (cf. 7.3). (Implementation Recommendation IR-PS-7, p 134.) □

SL-2 The provenance recording interface should support references to data within a p-assertion. □

Design Feature: The architecture supports storing data reference in the provenance store instead of the actual data through the use of documentation style (cf. section 6.4, p. 72). (Implementation Recommendation IR-APP-2, p 136.) □

SL-3 The provenance recording interface should support the recording of repeated p-assertions. □

Design Feature: An actor can make a p-assertion that refers to an already recorded p-assertion by using the global p-assertion key of the previously recorded p-assertion. (cf. 6.2 p. 60). □

SL-4 The provenance architecture should support handling of large p-assertion messages. □

Design Feature: This is addressed by the documentation style through reference as well as the asynchronous nature of the recording protocol. An actor can record a reference to a p-assertion instead of the entire message itself. Likewise, an actor could split up the content of the large p-assertion into several smaller p-assertions and record those separately (cf. section 6.4, p. 72). (Implementation Recommendation IR-APP-2, p 136.) Also see Design Feature for SR-2-4. □

SL-5 Provenance stores should be able to exchange their contents. □

Design Feature: See the Design Feature for SR-1-13. □

SL-6 The provenance store recording interface should support the SOAP MTOM recommendations for non-XML data. □

Design Feature: The architecture is logical and does not specify specific transport protocols. This is an implementation concern. □

SL-7 The provenance store should support multiple actors recording, querying it simultaneously. □

Design Feature: The recoding, querying interfaces place no limit on the number of actors that can use these interfaces. (Implementation Recommendation IR-PS-9, p 135.) □

SL-8 The provenance store implementation should be able to manage large amount of p-assertions. □

Design Feature: The management interface places no limit on the number of p-assertions that can be managed. (Implementation Recommendation IR-PS-8, p 134.) □

SL-9 Provenance actors should have no affinity to a provenance store. □

Design Feature: The logical architecture does not prescribe or enforce what provenance store an actor can use. □

9.4 Requirements from the OTM/EHCR Application

In this section, we discuss the requirements of the Organ Transplant Management application on the provenance architecture. We make reference to the General Rules (and sometimes section numbers) in deliverable D8.1.1 [VSWTV05]. Not all General Rules listed in that document are included as requirements here: General Rules OTM 4 and 10–23 and EHCR 7–17 are design decisions regarding the intended use of the provenance architecture (and compatible with it) rather than requirements.

GR-OTM.1 Anonymisation of patient identifiers must be carried out before any data is stored in Provenance stores. A system wide anonymisation mechanism is required. □

Design Feature: Asserting actors can replace patient identifiers with anonymous IDs in interaction p-assertions. Documentation style (cf. section 6.4, p. 72) can then be used to ensure that querying actors know this replacement has occurred. (Implementation Recommendation IR-APP-1, p 136.) □

GR-OTM.2 Source medical data is never stored in Provenance stores but only referenced therein. A system wide medical data referencing scheme is required. □

Design Feature: Asserting actors can replace sensitive data with references to other data stores in interaction p-assertions. Documentation style (cf. section 6.4, p. 72) can then be used to ensure that querying actors know this replacement has occurred. (Implementation Recommendation IR-APP-2, p 136.) □

GR-OTM.3 Each application component (actor/service) is shadowed by its own local Provenance store. □

Design Feature: Multiple provenance stores can be deployed within a single application (cf. section 3.3, p. 27). □

GR-OTM.5 Provenance stores are interlinked and communicate with one another, they are considered to be in one single-sign-on domain for security purposes, even though the application components will generally not be. □

Design Feature: Provenance stores can contain *view links* (cf. section 5.2.1, p. 57) and *object links* (cf. section 5.2.2, p. 58) which reference provenance stores containing related documentation. Querying actors can then follow these links to obtain the full provenance of an application entity. Single sign-on is possible with the security architecture because the credential server may be in an independent security domain from of any provenance store or application actor (cf. section 4.3, p. 38). (Implementation Recommendation IR-PS-4, p 134.) □

GR-OTM.6 All decision, result and edit (and possibly consult) actions/events are recorded in the system by the actor responsible for carrying them out. □

Design Feature: This matches the requirements on actor behaviour and so is compatible with the architecture. (Implementation Recommendation IR-APP-3, p 137.) □

GR-OTM.7 ...process documentation is expected to be available indefinitely, however the underlying medical data may be removed over time due to the application of data retention policies. □

Design Feature: This matches the intended use of the provenance architecture. Process documentation will not be overwritten, modified or deleted by any operation in the supplied APIs. (Implementation Recommendation IR-PS-5, p 134.) Any subsequent dangling links must be dealt with at the application level. □

GR-OTM.8 Messages stored in the Provenance system MAY NOT be the complete messages originally sent, but a reduced form removing sensitive medical data. □

Design Feature: Asserting actors can replace sensitive data with references to other data stores in interaction p-assertions. Documentation style (cf. section 6.4, p. 72) can then be used to ensure that querying actors know this replacement has occurred. (Implementation Recommendation IR-APP-2, p 136.) □

GR-OTM.9 All messages sent in the system are stored by BOTH the sender and the receiver. □

Design Feature: It is possible to distinguish which actor in an interaction is asserting documentation using the record data structure (cf. section 7.1, p. 87) and for a querying actor to distinguish which actor asserted each p-assertion in a provenance store (cf. section 6.8, p. 80). □

GR-EHCR.1 In order to assembly the full EHCR of the patient, the EHCR application uses the Provenance information returned from the Provenance store. □

Design Feature: If the provenance of the patient contains all information to be included in the EHCR, then the query interface supports this action (cf. section 3.3, p. 27). □

GR-EHCR.2 The EHCR application will use the one Provenance store per actor/service/data-store approach to store p-assertions. □

Design Feature: Multiple provenance stores can be deployed within a single application (cf. section 3.3, p. 27). With this, deployers can associate one provenance store to one application actor if they wish. □

GR-EHCR.3 The Provenance system connects the distributed Provenance stores and answers Provenance questions as if the distributed Provenance stores were logically a single centralised Provenance store. □

Design Feature: This functionality is an illustration of a processing service (cf. section 3.3, p. 29), which can query multiple distributed stores and return combined results. □

GR-EHCR.4 Patient information is stored in the Provenance system only through references using the Global Medical Patient ID (GMPID). □

Design Feature: Asserting actors can replace patient identifiers with anonymous IDs in interaction p-assertions. Documentation style (cf. section 6.4, p. 72) can then be used to ensure that querying actors know this replacement has occurred. (Implementation Recommendation IR-APP-1, p 136.) □

GR-EHCR.5 Health care data are stored in the Provenance system only through references using the system wide medical referencing scheme. □

Design Feature: Asserting actors can replace sensitive data with references to other data stores in interaction p-assertions. Documentation style (cf. section 6.4, p. 72) can then be used to ensure that querying actors know this replacement has occurred. (Implementation Recommendation IR-APP-2, p 136.) □

GR-EHCR.6 Provenance activities related to the assembly of the full EHCR is in the control of the EHCR store and is hidden from medical applications. □

Design Feature: Processing services and querying actors can be totally independent from application actors following the provenance architecture (cf. section 3.3, p. 27). □

OTM-15 Provenance stores are... linked together to answer queries in a form of overlay infrastructure spanning all relevant OTM/EHCR services" (Section 4.1, page 48 of D8.1.1). □

Design Feature: Provenance stores can contain *view links* (cf. section 5.2.1, p. 57) and *object links* (cf. section 5.2.2, p. 58) which reference provenance stores containing related documentation. Querying actors can then follow these links to obtain the full provenance of an application entity. □

OTM-16 [By using the provenance architecture in the application, we aim to] provide a coherent way of tracking/locating interim results related to a case across all records / documents / reports" (Section 4.1.1, page 48 of D8.1.1). □

Design Feature: Interim results and metadata are communicated between application actors or are part of their state, so will be apparent in interaction and actor state p-assertions. Relationship p-assertions allow these to be associated with a particular patient or outcome (cf. section 2.4, p. 20). □

OTM-17 [By using the provenance architecture in the application, we aim to] provide (without additional security clearance) a small amount of anonymised, low security risk meta-data to characterise the result types, decisions and outcomes without being exposed to the details of each decision." (Section 4.1.1, page 48 of D8.1.1). □

Design Feature: The provenance security architecture ensures secure access to process documentation (cf. section 4.3, p. 38). □

OTM-18 [By using the provenance architecture in the application, we aim to] provide a skeleton / framework for a more detailed probe which is able to apply security clearance to retrieve detailed records as needed in conjunction with Provenance meta-data." (Section 4.1.1, page 48 of D8.1.1). □

Design Feature: Replacing data with references in interaction p-assertions, where the references point to databases in other security domains, allows multiple steps of security clearance. Querying actors can be aware of, and so follow, the references by use of documentation style (cf. section 6.4, p. 72). (Implementation Recommendation IR-APP-2, p 136.) □

OTM-19 In general, access to Provenance stores will be available to persons across multiple hospitals / entities whereas internal data stores, generally are not accessible to anybody but local teams. If absolutely necessary however in certain cases, data could be included. Such needs will be evaluated on a case-by-case basis." (Section 4.1.1, page 49 of D8.1.1). □

Design Feature: The provenance store security architecture makes provision for interaction between different security domains through the trust mediator (cf. section 1, p. 38). □

9.5 Requirements from the Aerospace Engineering Application

In this section, we discuss requirements of the Aerospace Engineering application (as detailed in the workpackage deliverable D7.1.1 [KS05]) and their impact on the logical architecture.

AER-1 The need for physical distribution is not expected to be necessary considering the estimated amount of provenance data (Section 4.1 of D7.1.1). □

Design Feature: The provenance store is designed to be standalone in the initial instance; and is meant to accommodate distribution when circumstances require it (cf. section 3.3, p. 27). □

AER-2 Data sets are referenced from the provenance store using URIs to the data server or similar pointers (Section 4.1 of D7.1.1). □

Design Feature: Asserting actors can place references to data sets in the data server p-assertions. Documentation style (cf. section 6.4, p. 72) can then be used to ensure that querying actors know this. (Implementation Recommendation IR-APP-2, p 136.) □

AER-3 For practical reasons, it should be possible to provide a matching (or similar) access control list (ACL) in the provenance system to the ACL used for the data server (Section 4.2.1. of D7.1.1). □

Design Feature: The access control list (cf. section 4.3.1, p. 41) of the provenance system should be able to express the same access control semantics and restrictions as the access control list of the embedding system (the data server), using the derivation engine (cf. section 4.3.1, p. 42) of the security architecture if necessary. □

AER-4 The provenance data should be exportable into files and capable of being moved along with the simulations data (Section 4.2.1. of D7.1.1). □

Design Feature: The provenance store provides a querying interface (cf. section 3.3, p. 27) that allows (a subset of) p-assertions to be retrieved from the provenance store, and a recording interface (cf. section 3.3, p. 27) that allows p-assertions to be recorded (i.e., the provenance store offers both import and export facilities). □

AER-5 Data integrity between the provenance store and the linked information on the data server (Section 4.2.1. of D7.1.1). □

Design Feature: It is the responsibility of the application to keep track of data movements and make corresponding updates to the provenance store. □

AER-6 All p-assertions need to be uniquely identifiable by an assertion ID (Section 5.1 of D7.1.1). □

Design Feature: Provenance modelling introduces the notion of a global p-assertion key (GPAK), which uniquely identifies a p-assertion (cf. section 6.2, p. 68). □

AER-7 It would be useful to provide a way to generate unique IDs in a provenance store client-side library (Section 4.2.2.1 of D7.1.1). □

Design Feature: Actor side libraries can provide functionality for generation of unique IDs (Implementation recommendation IR-ASL-6, p 135.) □

AER-8 It would be useful for a user to supply an intuitive name for all interactions between particular actors in a given workflow execution. □

Design Feature: This requirement can be modelled as metadata about interactions in actor state p-assertions (cf. section 2.4, p. 20). □

AER-9 The user interface needs to be intuitive and easy to use as the results may be very large and quite complex during a targeted typical simulation of a weeks duration. □

Design Feature: User interface functionality is to be provided appropriately by the presentation UIs in the provenance architecture (cf. section 3.3, p. 29). □

AER-10 All p-assertions need to be time stamped by the provenance store. □

Design Feature: This requirement is analysed as SR-1-6. □

AER-11 A distinction between process defining and run time p-assertions has to be made. □

Design Feature: This can be supported by using different tracers for the two p-assertions types (cf. section 8.3, p. 107). Another option would be to model the distinction as metadata about interactions in the actor state p-assertions (cf. section 2.4, p. 20). □

9.6 Implementation Recommendations

This section contains a series of implementation recommendations that follow our analysis of technical requirements. Such recommendations suggest features that may be implemented by specific components of the architecture, namely the provenance store, processing and UI services, and actor-side libraries, and also by applications making use of the provenance architecture.

9.6.1 Provenance Store

IR-PS-1 A provenance store implementation may use a metadata infrastructure such as RDF to annotate p-assertions with metadata information about time at which p-assertions are received through the recording interface or stored in persistent storage. □

IR-PS-2 A provenance store implementation may document its interactions with asserters by storing p-assertions in some provenance store (itself or another one). This allows the provenance to document back ups, replication, time, etc. □

IR-PS-3 A provenance store implementation may use a storage layer that offers replication of p-assertions in order to be fault-tolerant. □

IR-PS-4 Provenance stores for an application can be deployed in security domains to which all querying actors have credentials to access. Tools can provide single-sign on by storing credentials and using as appropriate for each provenance store. □

IR-PS-5 Process documentation cannot be removed from the data storage using the provenance store APIs; instead, it may be removed, following data retention policies, by making direct access to the storage layer, using curation methodology typical of long term storage. □

IR-PS-6 The access control functionality of the provenance store may provide a way to model the expression of access control policies and rules so that they are functionally and / or semantically equivalent to the access control policies of other data stores □

IR-PS-7 Scalability of query results can be implemented by caching of the results within the provenance store. A set of interface operations allows actors to iterate and retrieve the cached results in response to a query request. □

IR-PS-8 Management of large number of p-assertions can be addressed by implementing the provenance store persistent store component using a proprietary or open source database management system. □

IR-PS-9 Implementations of the provenance recording, query and management interfaces should be clusterable. □

IR-PS-10 Implementations of provenance store should consider long term storage of p-assertions and the necessity to keep a long term copy of public keys (which may expire) to verify signed assertions. □

9.6.2 Processing and UI Services

IR-P-1 A processing service should allow for comparison of p-assertions. □

IR-P-2 UI services presented by the provenance system should be designed to allow multilingual support. □

9.6.3 Actor-Side Libraries

IR-ASL-1 Actor side libraries should allow for identification of the provenance store to be used. However, it should also ensure that all p-assertions pertaining to one interaction from a particular actor must be recorded in the same provenance store, Rule 8.4. □

IR-ASL-2 Actor side libraries may provide re-usable functionality to communicate and interact with the assigned provenance store. □

IR-ASL-3 Actor side libraries may provide functionality to mutually authenticate with the assigned provenance store. This also helps actors in obeying Rule 8.10. □

IR-ASL-4 An actor may provide functionality to record a view link to the provenance store that contains the corresponding actors view of the interaction if the view is in another provenance store. This also helps actors in obeying Rule 8.5. □

IR-ASL-5 Actor side libraries may provide functionality for generation of unique IDs. This helps actors in obeying Rule 8.1. □

IR-ASL-6 Actor side libraries may provide functionality to add assigned Interaction Key into its asserting message. This also helps actors in obeying Rule 8.2. □

IR-ASL-7 Actor side libraries may provide functionality for generation of a new session tracer and add it into task message. This helps actors in obeying Rule 8.6. □

IR-ASL-8 Actor side libraries may provide functionality to 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. This helps actors in obeying Rule 8.7. □

IR-ASL-9 Actor side libraries may provide functionality to add the session tracers supplied by its superior to its response to the superior. This helps actors in obeying Rule 8.8. □

IR-ASL-10 Actor side libraries may support declarative policy specification that specify what information needs to be recorded and when. This helps actors in obeying rules in Section 8.6. □

IR-ASL-11 Actor side libraries may be customised for specific actor hosting environment to capture information automatically from existing logs or from runtime environment. It may also use security functionality and credentials provided by the hosting environment. This helps actors in obeying Rule 8.11. □

IR-ASL-12 Actor side libraries may provide a range of error handlers and other facilities to desynchronise application execution from execution documentation recording. This helps actors in obeying Rule 8.12. □

IR-ASL-13 Actor side libraries may provide the necessary access to cryptographic functionality and material (such as key stores) in order to accomplish functionality such as signing or encrypting. This helps actors in obeying Rule 8.9. □

IR-ASL-14 Actor side libraries may provide functionality to transform messages according to particular documentation styles. This helps actors in obeying rules in Section 8.5. □

9.6.4 Application Use of Provenance Architecture

IR-APP-1 A documentation style can be created that marks a message as having been *anonymised*. Patient IDs can then be anonymised in an application message before asserting its content in an interaction p-assertion, and the *anonymised* documentation style used. □

IR-APP-2 A documentation style can be created that marks data as having been replaced by a *reference*. Sensitive data or large data sets can then be replaced with references in an application message before asserting its content in an interaction p-assertion, and the *reference* documentation style used. □

IR-APP-3 Decisions and events are triggered by information being received and produce results (information being sent). Therefore, they can be modelled as relationships between interactions. Additional information about an event can be supplied as actor state relating to those interactions. □

IR-APP-4 Support and access to a trusted third party time stamping facility can be supported. □

9.7 Conclusion

All requirements have been analysed from an architectural viewpoint. Either requirements were addressed by some specific design features of the architecture, which we explicitly referred to in the analysis, or they were beyond the remit of the architecture, and may have led to implementation recommendations for the different components identified by the architecture.

Chapter 10

Related Work

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 trails, 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.

10.1 Fine Granularity Provenance Systems

The following systems record the documentation of a 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 they achieve, where granularity of documentation means how detailed the documentation of a 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 documentation at a finer granularity. 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 an interesting case for determining the bounds on process documentation granularity.

Another technique for capturing fine granularity process documentation is the *sub-pushdown algorithm* [Mar01]. However, 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, and allows the provenance of an array in ArrayDB to be retrieved. A more comprehensive system is *audit facilities* designed for the S language [BJMC88]. S is an interactive system for statistical analysis. The result of users commands 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 re-execute a series of commands from the audit file.

A similar fine granularity technique for recording the documentation of 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 its inputs/outputs [Tan04]. Although interaction tracing is not concerned with the provenance of a result, it presents a novel notion for recording process documentation.

10.2 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). Knowing the provenance of map products is critical in GIS applications because it allows one to determine their quality. Lanter [Lan91b] 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, while the second 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 is 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 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. As a workflow is run inside ESSW, the results of each metadata described step is stored in the lab notebook service. ESSW is noteworthy because it emphasises the need to have immutable process documentation.

In chemistry, the CMCS project has developed a system for managing metadata in 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-silico 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 [ZGG+03]. Using this personal repository, the provenance of bioinformatics results can be determined. The focus of myGrid is about personalising the way provenance is presented to the user, and highlights the need for provenance in the bioinformatics domain.

The needs of particular domains have 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, however, are designed for one particular domain and are not general in nature, but they do provide insight into 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.

10.2.1 Current Practises of Document Management Systems

Provenance has been formally and explicitly defined document management systems in museums, libraries and archival management.

In terms of this document, two of the most related works are as follows. First, the International Standard for Archival Description [ISA00] 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 the archival history of an item and the immediate source of acquisition, i.e. from where the item was last obtained.

Second, the Encoded Archival Description DTD [EAD02] is a standard for encoding archival finding aids using SGML or XML. It includes two pieces of information: the provenance of the material being described, meaning the chain of ownership of the materials, and information about the immediate source of acquisition.

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).

10.3 Provenance in Database Systems

Provenance in database systems has focused on the data lineage problem [CWW00], which involves determining the set of source data used to produce a given item. In [WS97] Woodruff *et al.* attempt to solve this problem using *weak inversion*, which is a technique that, given some output data and a weak inversion function f^{-w}, attempts to lazily recreate the input data used to generate the output. Unfortunately, this requires the user who creates a new database view to also define an inversion function for that view. This technique has been used to improve database visualisation [Woo98]. In [CWW00] Cui *et al.* formalise the data lineage problem and present 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 examines 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 which is termed a schema transformation pathway. From these transformation pathways, the 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.

Buneman *et al.* [BKT01] redefine the data lineage problem as "why-provenance" and define a new type of provenance for databases, namely, "where-provenance". "Why-provenance" refers to why a piece of data is in the database, i.e. what data sets (tuples) contributed to a data item, whereas, "where-provenance" addresses the location of a data element in the source data. Based on this terminology, a formal model of provenance is developed applying to both relational and XML databases. In other work [BKKT02], Buneman argues for a time-stamped based archiving mechanism for change tracking in contrast to diff-based mechanisms which, it is argued, 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, and the time-stamped approach is advocated.

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. Moreover, they deal only with closed systems. However, in open, distributed systems the number of transformations can be infinite, therefore, work is needed in developing systems that can handle any kind of transformation in such open systems. Cui *et al.* [CW03] is a first step toward addressing this problem, but it only pertains to one particular context, the data warehouse.

10.4 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.

In [RXBR04] Ruth et al. 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 participating in a process. To enable support of trust and credential tracking, each node in a DAG must be digitally signed by the node's creator. This system is noteworthy 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 by 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, but instead acts as an open repository. However, this lack of structure means that retrieving the provenance of a result in its entirety can 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 can 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 by analysing the transformation descriptions without having to understand the underlying data. Process documentation in Chimera is stored in the Provenance Transformation Catalog (PTC). However, 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, however, noteworthy 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 represents information about the invocation of various Web Services specified by the executing workflow script. This system is interesting from the perspective of the work presented in this document since it is both Web Services based, and was designed with Grid applications in mind. The drawbacks of this system, however, are that it relies completely on a workflow enactment engine, and it lacks demonstration in any large scale system.

10.5 Conclusions

In this chapter we have reviewed relevant work carried out in other areas. We have discussed the state of the art in a number of provenance related computational systems from those covering fine granularity provenance to a number of domain specific systems, including document management systems. Additionally, we examined how provenance related work is carried out in database systems and we discussed several middleware provenance systems. In terms of their relation to the work in this document, the finer granularity systems are not designed specifically for determining the provenance of data items, and they may carry excessive overheads for the kinds of large scale systems that we consider. In addition our approach is to design a general purpose provenance architecture that overcomes the limitations of the domain specific provenance systems described above. We have also addressed the kinds of multiple transformations to data items that can occur in open, distributed systems that current database systems do not handle, due to their closed world assumptions.

In comparison to current systems available, the architecture presented in this document provides an *implementation independent*, *general specification* for industrial strength Grid-based applications that overcomes many of the limitations of previous provenance systems.

Chapter 11

Conclusion

11.1 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, and 7, we provide further detail about the functionality of the provenance architecture. Chapter 6 describes the underpinning data model of the provenance architecture. In Chapter 7, we detail the functionality available through the three interfaces of a provenance store: *recording*, *query* and *management*. In Chapter 8, 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 9 and 10 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 10 we discuss related work.

Throughout this document, we highlight ways that the technology-independent architecture presented herein can be applied to Service Oriented Architectures and scientific workflows. For example, the architecture defines workflows in Chapter 2 and identifies workflow enactment engines as a potential example of application services in Figure 3.1). Chapter 5 discusses data staging in relation to workflows, and Appendix A includes a note, which discusses how workflow languages and enactment engines are related to the production of p-assertions. Other references to workflows can be found in the index provided.

Other work produced by the EU Provenance project provides a detailed methodology that enables application designers to incorporate the provenance architecture described in this document[MMT+06].

A set of associated documents are available (see list below) that describe other work performed by the EU provenance project and can be found at the project's public web site: `www.gridprovenance.org`.

- A set of open specifications of the architecture
- An instantiation of the architecture for the Web Services stack.
- A design for implementation of the 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.

Each of these have contributes to fulfilling the projects obligations for the three sets of stakeholders identified in Chapter 1. For example, the *methodology* addresses the needs of *users* and *developers*. The *open specifications* address the needs of developers and the d*esign implementation* addresses the needs of *system managers* who must deploy a realized vision of the architecture in their domain.

Appendix A

Notes

This appendix refers to annotations introduced in the margin of the 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 libraries 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. □

Note 10: In Chapter 2, we have defined a workflow as the specification of a given service composition. In the context of Web Services, such a service composition can be expressed in a workflow language, such as WS-BPEL or other, which can be executed (or enacted) by a component usually referred to as workflow enactment engine. The workflow enactment engine is just one of the actors that is involved in an application and is therefore expected to contribute p-assertions. We anticipate that for a given workflow language, many p-assertions may be derived automatically from the workflow script itself. For instance, interaction p-assertions can be produced for each service invocation; likewise, relationship p-assertions can be derived from so-called data links. □

Appendix B

Abbreviations

This appendix contains abbreviations used in this document.

CA	Certificate Authority
GPAK	Global P-Assertion Key
GPID	Global P-Assertion Identifier
GUI	Graphical User Interface
IR	Internal Representation
PReP	P-Assertion Recording Protocol
RBAC	Role Based Access Control
RDF	Resource Description Framework
UI	User Interface
XML	Extensible Markup Language

Appendix C

XML Schema Diagrams

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

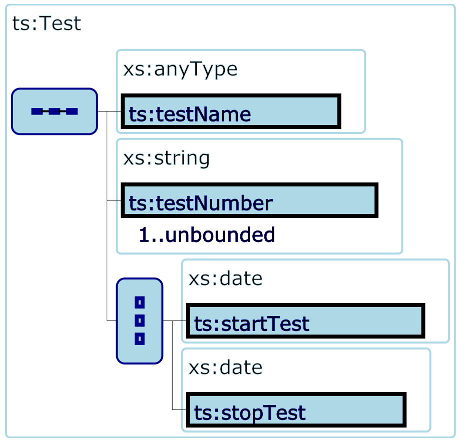
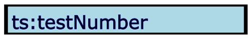


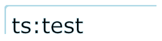
Figure C.1: An example XML Schema diagram

Figure C.1 defines the structure of type ts:Test. The type Test contains a sequence of elements, which we now detail. One element in the sequence is ts:testName, which can be any type and must occur once and only once in an instance of ts:Test. ts:Name is followed by element ts:testNumber, which must contain a string. The ts:testNumber element must occur at least once and can occur as many times as needed. This is denoted by the "1..unbounded" under the element. Finally, the sequence contains a choice between two elements, ts:startTest and ts:stopTes, either of which must contain a date.

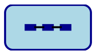
Below is a simple of description of each of the parts of the XML Schema diagram format.



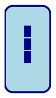
An element (instance) is represented by the qualified name of the element in the box. By default an element must occur once and only once. Where this restriction does not hold, the text "1..unbounded", "0..unbounded", "0..N", "1..N" (where N is an integer) appears under the element box. The left hand number is the minimum occurrences of the element at the position in the XML document, the right hand number is the maximum (with "unbounded" for no maximum).



A complex type is denoted by a lightly marked box with the qualified name of the type at the top left. The structure of the type is given by the elements, types and control structures within the box.



A horizontal sequence of dots represents a sequence of elements or control structures, that must appear in an element conforming to the type in the surrounding type box.



A vertical sequence of dots represents a choice between elements or control structures, that must appear in an element conforming to the type in the surrounding type box.

Index

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.

[ACV97] Gian Pietro Picco Antonio Carzaniga and Giovanni Vigna. Designing distributed applications with mobile code paradigms. In *Proceedings of the 19th international conference on Software engineering*, pages 22–32, Boston, Massachusetts, May 1997.

[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.

[And05a] Árpád Andics (ed.). D2.1.1: User requirements document. Technical report, MTA SZTAKI, February 2005.

[And05b] Árpád Andics (ed.). D2.2.1: Software 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.

[Bra97] S. Bradner. Rfc 2119 — key words for use in rfcs to indicate requirement levels. http://www.faqs.org/rfcs/rfc2119.html, March 1997.

[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.

[EAD02] Encoded archival description. http://www.loc.gov/ead/, 2002.

[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 Teruo Higashino, editor, *Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS'04)*, volume Lecture Notes in Computer Science, pages 124–139, Grenoble, France, December 2004. Springer-Verlag.

[GLM04c] 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.

[GMF$^+$05a] 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.

[GMF$^+$05b] 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)*, 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.

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

[HI05] Neil Hardman and John Ibbotson. D9.3.1: Functional prototype. Technical report, IBM, September 2005.

[IGM05] John Ibbotson, Paul Groth, and Simon Miles. D5.1.1: Scalability requirements. Technical report, IBM Hursley, September 2005.

[IHT05] John Ibbotson, Neil Hardman, and Victor Tan. D4.1.1: Security requirements. Technical report, IBM Hursley, September 2005.

[ISA00] General international standard archival description (isad(g)). http://www.icacds.org.uk/eng/ISAD(G).pdf, 2000.

[JGM+06] Sheng Jiang, Paul Groth, Simon Miles, Victor Tan, Steve Munroe, Sofia Tsasakou, and Luc Moreau. Client side library design and implementation. Technical report, 2006.

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

[KS05] Guy K. Kloss and Andreas Schreiber. D7.1.1: Application 1: Aerospace engineering. specification of mapping to provenance architecture, and domain specific provenance handling. Technical report, German Aerospace (DLR), September 2005.

[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.

[MGBM06] Simon Miles, Paul Groth, Miguel Branco, and Luc Moreau. The requirements of recording and using provenance in e-science experiments. *Journal of Grid Computing*, 2006.

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

[MM06] Simon Miles and Luc Moreau. Querying the provenance of electronic and physical entities. Technical report, University of Southampton, January 2006.

[MMT$^+$06] Steve Munroe, Simon Miles, Victor Tan, Paul Groth, Sheng Jiang, Luc Moreau, , John Ibbotson, and Javier Vázquez-Salceda. PrIMe: A methodology for developing provenance-aware applications. Technical report, University of Southampton, 2006.

[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.

[Ran05] Omer F. Rana. D6.1.1: Tools description document. Technical report, University of Cardiff, December 2005.

[Rel06] Causal relationships ontology. http://twiki.pasoa.ecs.soton.ac.uk/pub/PASOA/Ontologies/causal.owl, 2006.

[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 *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.

[VSWTV05] Javier Vzquez-Salceda, Steve Willmott, Kifor Tams, and Lszl Zs. Varga. D8.1.1: Application 2: Organ transplant management. specification of mapping to provenance architecture, and domain specific provenance handling. Technical report, UPC, September 2005.

[WMF$^+$05a] Sylvia C. Wong, Simon Miles, Weijian Fang, Paul Groth, and Luc Moreau. Provenance-based validation of e-science experiments. In *Proceedings of 4th Internation Semantic Web Conference (ISWC'05)*, volume 3729 of *Lecture Notes in Computer Science*, pages 801–815, Galway, Ireland, November 2005. Springer-Verlag.

[WMF$^+$05b] 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.

[ZDF$^+$05] Yong Zhao, Jed Dobson, Ian Foster, Luc Moreau, and Michael Wilde. A notation and system for expressing and executing cleanly typed workflows on messy scientific data. *Sigmod Record*, 34(3), September 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.