PROVENANCE
Enabling and Supporting Provenance in Grids for Complex Problems
Contract Number: 511085

The Open Provenance Specification

Authors: Paul Groth, Simon Miles, Steve Munroe, Sheng Jiang, Victor Tan
John Ibbotson and Luc Moreau

Reviewers: All project partners
Identifier: D3.2.1 (The Open Specification)
Type: Deliverable
Version: 1
Version: November 23, 2006
Status: public

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

Abstract

The Open Provenance Specification is composed of the following documents: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. In this deliverable, we have concatenated 11 distinct documents for the convenience of the reader. Each of these has its own numbering (in the footer) and bibliography. A global numbering (in the header) has been introduced for the purpose of the table of contents:
page 6 document [1]
page 21 document [2]
page 53 document [3]
page 68 document [4]
page 90 document [5]
page 102 document [6]
page 119 document [7]
page 131 document [8]
page 154 document [9]
page 163 document [10]
page 173 document [11]

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

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

Open Specification Documents

[1] Steve Munroe, Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. The provenance standardisation vision. Technical report, University of Southampton, November 2006. http://eprints.ecs.soton.ac.uk/13055/.

[2] Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, Victor Tan, and Luc Moreau. Data model for process documentation. Technical report, University of Southampton, November 2006. http://eprints.ecs.soton.ac.uk/13047/.

[3] Paul Groth, Victor Tan, Steve Munroe, Sheng Jiang, Simon Miles, and Luc Moreau. Process documentation recording protocol. Technical report, University of Southampton, November 2006. http://eprints.ecs.soton.ac.uk/13053/.

[4] Simon Miles, Luc Moreau, Paul Groth, Victor Tan, Steve Munroe, and Sheng Jiang. Provenance query protocol. Technical report, University of Southampton, November 2006. http://eprints.ecs.soton.ac.uk/13050/.

[5] Simon Miles, Steve Munroe, Paul Groth, Sheng Jiang, Victor Tan, John Ibbotson, and Luc Moreau. Process documentation query protocol. Technical report, University of Southampton, November 2006. http://eprints.ecs.soton.ac.uk/13052/.

[6] Victor Tan, Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. A profile for non-repudiable process documentation. Technical report, University of Southampton, November 2006. http://eprints.ecs.soton.ac.uk/13054/.

[7] Steve Munroe, Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. A WS-addressing profile for distributed process documentation. Technical report, University of Southampton, November 2006. http://eprints.ecs.soton.ac.uk/13057/.

[8] Victor Tan, Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. Basic transformation profile for documentation style. Technical report, University of Southampton, November 2006. http://eprints.ecs.soton.ac.uk/13049/.

[9] Simon Miles, Luc Moreau, Paul Groth, Victor Tan, Steve Munroe, and Sheng Jiang. Xpath profile for the provenance query protocol. Technical report, University of Southampton, November 2006. http://eprints.ecs.soton.ac.uk/13051/.

[10] Steve Munroe, Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. A soap binding for process documentation. Technical report, University of Southampton, November 2006. http://eprints.ecs.soton.ac.uk/13056/.

[11] Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, and Luc Moreau. WS provenance glossary. Technical report, University of Southampton, November 2006. http://eprints.ecs.soton.ac.uk/13048/.

ws-prov-overview

Authors:
Steve Munroe, U. Southampton
Paul Groth, U. Southampton
Sheng Jiang, U. Southampton
Simon Miles, U. Southampton
Victor Tan, U. Southampton
John Ibbotson, IBM
Luc Moreau, U. Southampton

November 23, 2006

Overview of the Provenance Specification Effort

Abstract

This document provides an overview of a model of provenance along with a description of a family of specification documents that support the model. Important aspects of the model are specified within these documents in a detailed and clear manner that provides an unambiguous reference for developers.

Contents

1 Introduction

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.

Two common sense definitions consider provenance to be the derivation from a particular source to a specific state of an item. They are taken from the Oxford English Dictionary, and the Merriam-Webster Online Dictionary respectively and are presented below.

Definition 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.* □

Definition 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 1 and 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 have identified a process in a service oriented architecture (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, GLM04b, GLM04b, MGBM05, SM03], we propose the following definition of provenance, which makes explicit the notion of process.

Definition 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 3 is concerned with provenance as a concept. Ultimately, our aim is to specify a computer-based *representation* of provenance that allows us to perform useful analysis and reasoning to support a wide variety of applications.

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.

In this document, we introduce a framework for computational provenance; a set of nine technical specifications that define the normative description of the provenance framework in terms of a SOA model and related XML definitions. These technical specifications, summarised in Figure 1, define the means by which:

- a computational representation of process documentation can be realised;
- process documentation can be recorded;
- process documentation can be queried;
- the recording and querying of process documentation can be made secure;
- process documentation can be recorded in distributed systems.

The family of documents comprise a set of 2 support documents, four documents that introduce and specify the core framework, four generic profiles that extend the basic framework and one example of a technology specific binding.

2 Overview of the Provenance Model

In this section we present a intuitive, non-technical view of provenance and the provenance model. In it we provide some context to the model before describing definitions of the different kinds of process documentation that go to form the basis of the model.

2.1 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 [MH05], 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. Specifically, the following are all considered as "services" because they

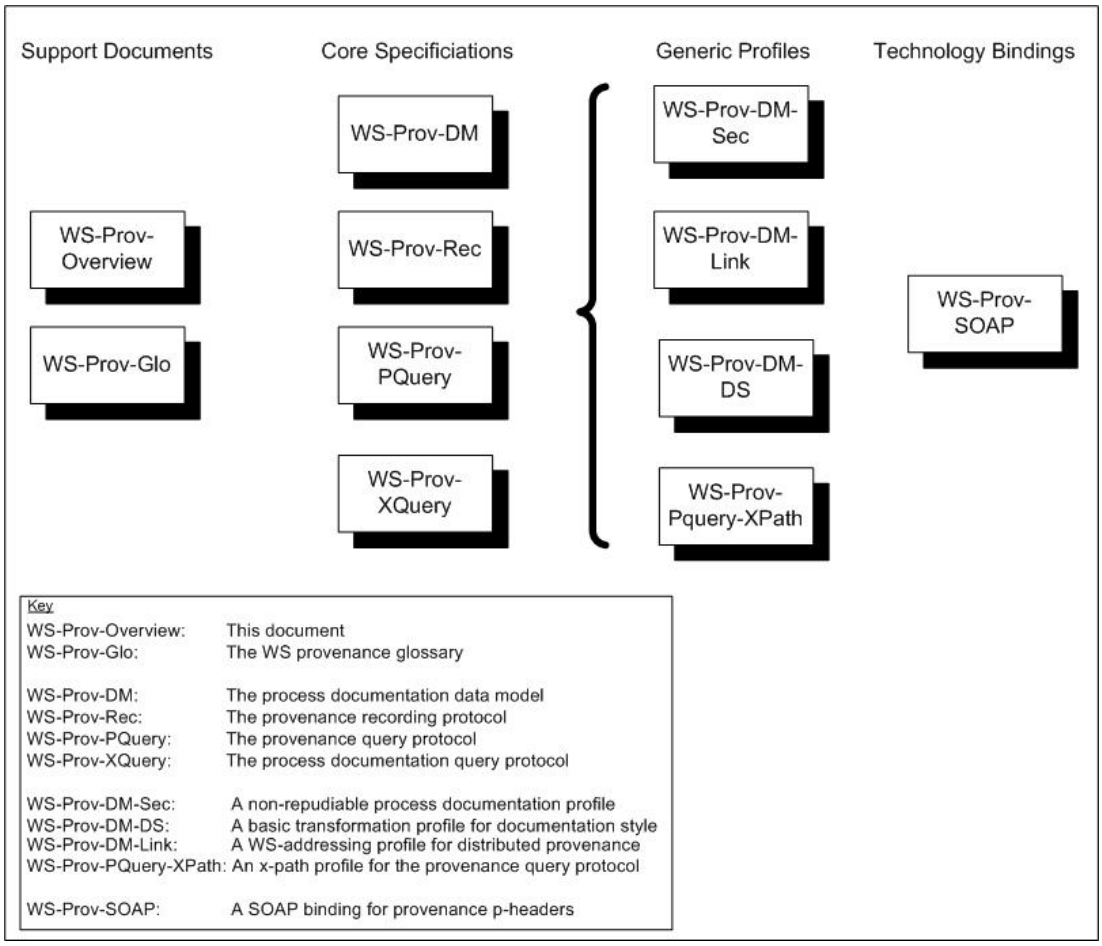


Figure 1: The family of specification documents

all take some inputs and produce some outputs: Web Service, CORBA or RMI objects, command line program.

Such services are brought together to solve a given problem typically via a *workflow* that specifies their composition. 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.

In this abstract view, invocations of services take place using *messages* that are constructed in accordance with service *interface* specifications. 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.

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*. 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.

At this stage of the specification, we do not make the distinction between resource and service [SRB06] 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.

2.2 Representation of Provenance

In this section, we introduce the key elements that form the representation of provenance in a SOA.

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

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 possibly provide any documentation relevant to this specific execution.

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 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.

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.

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).

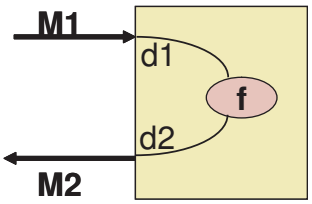
Definition 7 (Relationship p-assertion) *A relationship p-assertion is an assertion by an actor that the sending of a message would not be occurring or a data item it is sending would not be as it is (the* effect*), if it had not received other messages or data items had not been as they are (the* causes*), 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.

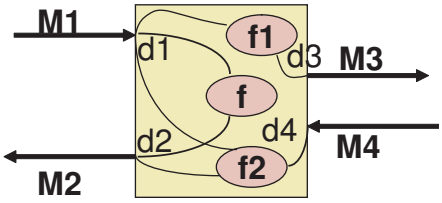
Figure 2 illustrates two actors. The first is a primitive actor, i.e., one that receives a message and produces a result, but does not invoke subsequent actors, or alternatively, an actor that does not make assertions of the invocations it makes of subsequent actors (say, for privacy reasons). In order 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 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.

Figure 2 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, we see 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.



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: Data flow assertions by opaque and transparent actors

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 a specific interaction.

Definition 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, relationship p-assertions and actor state p-assertions. We note that p-assertions are independent of the actual service technology used to implement applications.

2.3 Provenance Lifecycle

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. 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. 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.

3 The Family of Specification Documents

In this section we describe each of the specification documents and supporting documents listed in Figure 1. The content is split into four groupings. First, we discuss two supporting documents: the overview and the glossary. The core specifications come next, which define the key aspects of the provenance model. After this we have a set of generic profiles that describe non-core aspects of the model. Finally, we provide an example of one specific technology binding – a tieing in of an aspect of the model to an implementation technology.

3.1 Support Documents

WS-Prov-Overview This document.

WS-Prov-Glo: The Provenance Glossary The WS-Prov-Glo document [TGJ+06] provides a glossary that defines a set of terms used in the draft provenance specification documents. The terms described are intended to be implementation and technology independent, with the intent that they can be analysed and applied to as many contexts as possible.

3.2 Core Specifications

WS-Prov-DM: The Process Documentation Data Model The WS-Prov-DM document [MGJ+06] presents a specification of the data model for process documentation. The approach is top down in nature, and starts by describing the p-structure — the logical organisation of process documentation, before drilling down into the models of the different forms of p-assertions. The identification of p-assertions and data items is then described, followed by a description of a model of context that allows information about related interactions to be passed between actors.

WS-Prov-Rec: The Provenance Recording Protocol Every provenance store supplies a Web Service interface for recording process documentation. It has a single operation, *record*, that takes Record document as input and returns an acknowledgement as result. The WS-Prov-Rec document [GTM+06] defines the schema for the record request and acknowledgement messages.

WS-Prov-Query: Provenance Queries In the WS-Prov-Query document [MMG+06b], a protocol is specified by which a querying actor and provenance store can communicate in performing a provenance query. This protocol primarily takes the form of an abstract WSDL interface defining messages to be accepted and produced by a provenance store. This document defines the schema for a provenance query request, the behaviour expected in processing that request and the resulting response.

WS-Prov-XQuery The process documentation data model defines schemas to be used for documentation about the execution of a process, *process documentation*, and introduces a *provenance store* — a type of Web Service with the capability for storing and giving access to process documentation. In particular, process documentation has a defined schema, the *p-structure*, which clients of a provenance store can navigate in queries to extract particular pieces of process documentation. In this document, a protocol is specified by which a querying actor and provenance store can communicate in performing a process documentation query. This primarily takes the form of an abstract WSDL interface defining messages to be accepted and produced by a provenance store.

3.3 Generic Profiles

WS-Prov-DM-Sec: Secure Provenance The data model for process documentation [MGJ+06] describes p-assertions as individual units for documenting process. These p-assertions can be signed by asserting actors in order to establish accountability for their creation. The WS-Prov-DM-Sec document [TMG+06a] extends on the data model for the basic p-assertions (relationship, actor-state and interaction) to include support for signatures.

WS-Prov-DM-Link: Distributed Provenance Process documentation can often be distributed across different provenance stores. To enable the discovery of related process documentation, a mechanism is required to link disparate but related process documentation to enable the effective collection of such documentation to answer provenance queries. The WS-Prov-DM-Link document [MTG+06b] represents a WS-Addressing profile on distributed process documentation that provides mechanisms to solve this problem.

WS-Prov-DM-DS: Transforming Process Documentation The activity of constructing an interaction p-assertion from a message can be considered as a single atomic transformation, which needs to be qualified by the actor creating that p-assertion in order for actors that retrieve that p-assertion from the provenance store to understand the exact nature of the transformation applied. This is equally true for an actor state p-assertion. Documentation styles are essentially descriptions of the types of transformations that can be applied to a message or to the internal state of an actor. The WS-Prov-DM-DS document [TMG+06b] presents a profile of several basic documentation style transformations that are likely to be useful in application domains that use process documentation. It is not intended to be exhaustive; other profiles may be provided of alternative documentation style transformations which may be generic or more specific in nature.

WS-Prov-PQuery-XPath The provenance query protocol [MMG+06a] has been defined, and includes the request for, algorithms to execute and result from a provenance query, as executed by a provenance query engine. Many parts of the request document are unspecified, being dependent on the provenance query engine implementation. This document defines an XPath-based profile by which provenance queries can be fully specified against process documentation that is in, or can be mapped to, XML format.

3.4 Technology Bindings

WS-Prov-SOAP: Provenance and SOAP Technology 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, a further obligation is placed on them to pass context information between each other regarding the process being executed. This would often be achieved by putting the context information in the header of an application message (such as a SOAP message). The WS-Prov-SOAP document [MTG+06a] describes a specification of the p-header in the context of SOAP [Mit03] messages.

4 Conclusion

In this document we have presented an overview of the provenance model, the provenance lifecycle and the set of supporting specification documents that describe the model in detail. 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. For these functions we have provided detailed models in the family of specification documents, which also specify models for transforming documentation, distributed provenance, security and a technology specific binding for SOAP messages. A glossary of all the terms found in this document and the other specification documents is also available.

References

[Bur00] S. Burbeck. The tao of e-business services. Technical report, IBM Software Group, 2000.

[GLM04a] P. Groth, M. Luck, and L. Moreau. Formalising a protocol for recording provenance in grids. In *Proc. 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.

[GTM+06] Paul Groth, Victor Tan, Steve Munroe, Sheng Jiang, Simon Miles, and Luc Moreau. Process Documentation Recording Protocol. Technical report, University of Southampton, June 2006.

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

[MGJ+06] Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, Victor Tan, and Luc Moreau. Data model for Process Documentation. Technical report, University of Southampton, June 2006.

[MH05] M.P.Singh. and M.N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. Wiley, 2005.

[Mit03] N. Mitra. Soap version 1.2 part 0: Primer. http://www.w3.org/TR/soap12-part0/, 2003.

[MMG+06a] Simon Miles, Luc Moreau, Paul Groth, Victor Tan, Steve Munroe, and Sheng Jiang. Provenance Query Protocol. Technical report, University of Southampton, June 2006.

[MMG+06b] Simon Miles, Steve Munroe, Paul Groth, Sheng Jiang, Victor Tan, John Ibbotson, and Luc Moreau. Process Documentation Query Protocol. Technical report, University of Southampton, June 2006.

[MTG+06a] Steve Munroe, Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. A SOAP Binding For Process Documentation. Technical report, University of Southampton, June 2006.

[MTG+06b] Steve Munroe, Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. WSRF Data Model Profile for Distributed Provenance. Technical report, University of Southampton, June 2006.

[SM03] 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.

[SRB06] David Snelling, Ian Robinson, and Tim Banks. Web Services Resource Framework v1.2 OASIS Standard, 1st April 2006. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf, 2006.

[TGJ+06] Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, and Luc Moreau. WS Provenance Glossary. Technical report, Electronics and Computer Science, University of Southampton, 2006.

[TMG+06a] Victor Tan, Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. A Profile for Non-Repudiable Process Documentation. Technical report, University of Southampton, June 2006.

[TMG+06b] Victor Tan, Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. Basic Transformation Profile for Documentation Style. Technical report, University of Southampton, June 2006.

ws-prov-dm

Authors:
Steve Munroe, U. Southampton
Paul Groth, U. Southampton
Sheng Jiang, U. Southampton
Simon Miles, U. Southampton
Victor Tan, U. Southampton
Luc Moreau, U. Southampton
John Ibbotson, IBM
Javier Vazquez, UPC

November 23, 2006

Data Model for Process Documentation

Status of this Memo

This document provides information to the community regarding the specification of a data model for process documentation used to describe the provenance of data and has the status of a working draft. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

This document describes the data model for *process documentation*; information describing process. It starts by describing the logical organisation of process documentation, before drilling down into the models of the different forms of process documentation. It then describes how individual pieces of process documentation and data items can be identified. Finally, a model of context is provided.

Contents

1 Introduction

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, a primary objective here is to define a representation of provenance that is suitable for computer systems where, in this context, provenance is defined as *the process that led to that piece of data.*

Applications produce data and to obtain the provenance of such data they must be transformed into so called provenance-aware applications, so that when they run they produce a description of their execution, called *process documentation.*

This document presents a specification of the data model for process documentation. The approach is top down in nature, and starts by describing the p-structure — the logical organisation of process documentation, before drilling down into the models of the different forms of process documentation, or p-assertions. The identification of p-assertions and data items is then described, followed by a description of a model of context.

A full overview document is (soon to be) available that describes the vision for the standardisation effort [TMG+06c].

1.1 Goals and Requirements

The goal of this document is to define an open, interoperable data model for process documentation.

1.1.1 Requirements

In meeting this goal, this document must address the following requirements:

- Define the data items necessary for process documentation and their logical organisation.
- Provide the basis for an open, interoperable set of standards.
- Provide extensibility for more sophisticated and/or currently unanticipated scenarios.

1.1.2 Non-Requirements

This document does not intend to meet the following requirements:

- Supply definitions and scope of data provenance. This is covered in [TMG+06c].
- Supply a model for the transformation of process documentation — called documentation styles. This aspect of data provenance is covered in [TMG+06b].
- Supply a model of storing and retrieving process documentation. This aspect of data provenance is described in [MMG+06].
- Supply a model for provenance security. This aspect of data provenance is described in [MMG+06].
- Supply a model for provenance scalability and distribution. This aspect of data provenance is described in [MTG+06].

2 Terminology and Notation

All definitions for the concepts and structures found within this document can be found in [TGJ+06].

2.1 XML Namespaces

The XML Namespace URI that MUST be used by implementations of this specification is: `http://www.pasoa.org/schemas/version023s1/PStruct.xsd`

Table 1 lists XML namespaces that are used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

Prefix	XML Namespace	Specification(s)
ps	`http://www.pasoa.org/schemas/version023s1/PStruct.xsd`	[P-Structure]
ph	`http://www.pasoa.org/schemas/version023s1/PHeader.xsd`	[P-Header]
wsa	`http://schemas.xmlsoap.org/ws/2004/08/addressing`	[WS-Addressing]
xs	`http://www.w3.org/2001/XMLSchema`	[XMLSchema]

Table 1: Prefixes and XML Namespaces used in this specification

2.2 Notational Conventions

The keywords "MUST ", "MUSTNOT ", "REQUIRED ", "SHALL ", "SHALLNOT ", "SHOULD ", "SHOULDNOT ", "RECOMMENDED ", "MAY ", and "OPTIONAL " in this document are to be interpreted as described in [Bra97].

2.3 XML Schema Diagrams

This documents adopts a graphical notation to describe XML Schema. Figure 1 gives an example of a small XML Schema displayed as a diagram, which is now explained with reference to the figure.

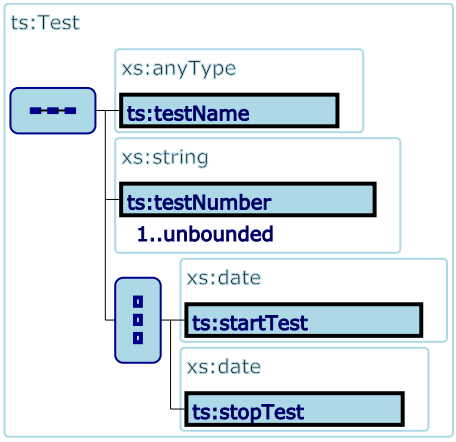


Figure 1: An example XML Schema diagram

Figure 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:stopTest`, 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.

2.4 XPath notation

In addition to the XML Schema diagrams, an XPath notation [W3C99] is used to identify each individual element in the specification along with its context, in order to describe precisely its meaning and use.

3 The Process Documentation Model

The data model presented here represents all the necessary data structures required to represent and organise process documentation. In the model, process documentation is organised via collections of *interaction records* that collectively make up the *p-structure* data type. Within each interaction record is the process documentation that relates to the interaction being recorded. Individual items of process documentation are referred to as *p-assertions* , and there are three different forms that p-assertions can take:

- interaction p-assertions,
- actor state p-assertions and,
- relationship p-assertions.

The combination of these three forms of process documentation provide the necessary means to record process documentation. In what follows, a top-down approach is adopted to describing the data model. Starting at the level of the p-structure it is shown how the organisation of process documentation can be expressed. Then, the representation of each of the p-assertions is described before a description of how p-assertions and their contents can be identified. Finally, the model of *context* is described. Context refers to extra information that may be included along with any p-assertions made about a specific interaction. This model of process documentation is taken from the EU Provenance project [GJ+06].

3.1 The P-Structure

Process documentation is recorded in order to answer subsequent provenance queries. To facilitate this aim, a structural organisation of process documentation is given — referred to as a *p-structure*. The p-structure itself is now described. (Note that the complete schema definition of the p-structure can be found in Appendix A.) Figure 2 shows a graphical representation of the p-structure.

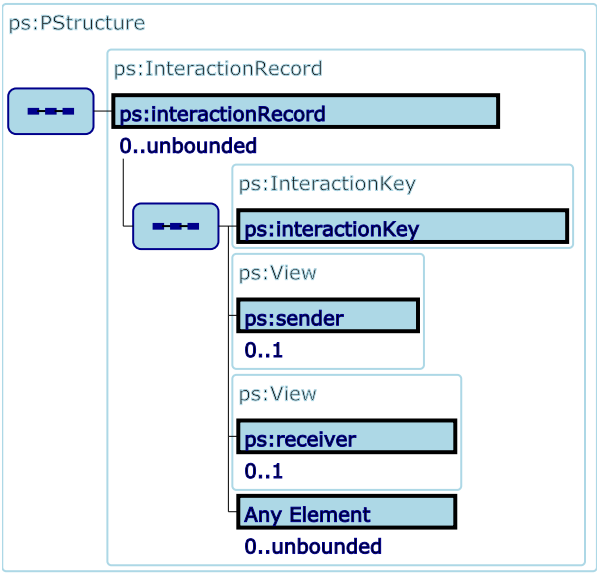


Figure 2: The P-Structure

The p-structure is organised as a hierarchy in which, at the top level, is a collection of *interaction records*. Each record encapsulates all the p-assertions and identifiers related to one interaction. The choice of interaction record as the chief item in the p-structure is derived from the idea that interactions are the core actions of a process. Each interaction record is identified by an interaction key, as shown in Figure 5. The interaction key distinguishes one interaction record from all others and is provided by the sending actor.

The p-structure and its contents are further described as follows:

`/ps:pstruct`

> This element provides a common logical structure for process documentation. All subsequent elements, with the exception of the p-header, are sub-elements of this element. The structure imposed upon process documentation is given by an unbounded sequence of interactionRecords.

`/ps:pstruct/ps:interactionRecord`

The intent of this component is to hold all p-assertions and identifiers relating to one interaction, i.e. the passing of one message from one actor to another. The p-assertions contained within an interaction record are grouped according to whether it was the sender or the receiver of the message that asserted the p-assertion, where this is represented by the type `View`. The full definition of View is given Section 3.2.

`/ps:pstruct/ps:interactionRecord/ps:interactionKey`

The intent of this component is to uniquely identify the interaction contained in an `interactionRecord`, and thus also identifying the `interactionRecord`. Its full definition is given in Section 3.4.

`/ps:pstruct/ps:interactionRecord/ps:sender`

The intent of this element is to contain the information relating to the sender's view of an interaction, where this is of type `ps:View`. The full definition of `ps:View` is given Section 3.2.

`/ps:pstruct/ps:interactionRecord/ps:receiver`

The intent of this element is to contain the information relating the receiver's view of an interaction, where this is of type `ps:View`. The full definition of `ps:View` is given Section 3.2.

`/ps:pstruct/ps:interactionRecord/xs:any`

The intent of this component is to provide extensibility by allowing application dependent data formats to be introduced.

The general policy for extensibility is to allow any type to be extended by other schemas, except where explicitly restricted. Where we explicitly expect extension, we add the `any` element at the end of a sequence (as above). The `any` element MUST use namespace="##other". All extensions MUST be done in another namespace.

3.2 Interaction Views

In the p-structure hierarchy, there are two `ps:View` elements under the `ps:InteractionRecord` type. One `ps:View` contains the p-assertions from the *sender* in an interaction, while the other contains those from the *receiver*. `ps:View` has the structure shown in Figure 3: it has an *asserter*, which is the identity of the actor asserting a set of p-assertions, it can contain several interaction p-assertions, several actor state p-assertions, several relationship p-assertions, and some exposed interaction metadata. This is information that has been exposed from within the stored p-assertions. The rationale for this element lies with requirements imposed when process documentation is distributed. The data model

for distributed provenance is described in detail in the profile [MTG+06]. All of these are OPTIONAL. Each p-assertion is defined by an associated model described in Sections 3.3, 3.5 and 3.6.

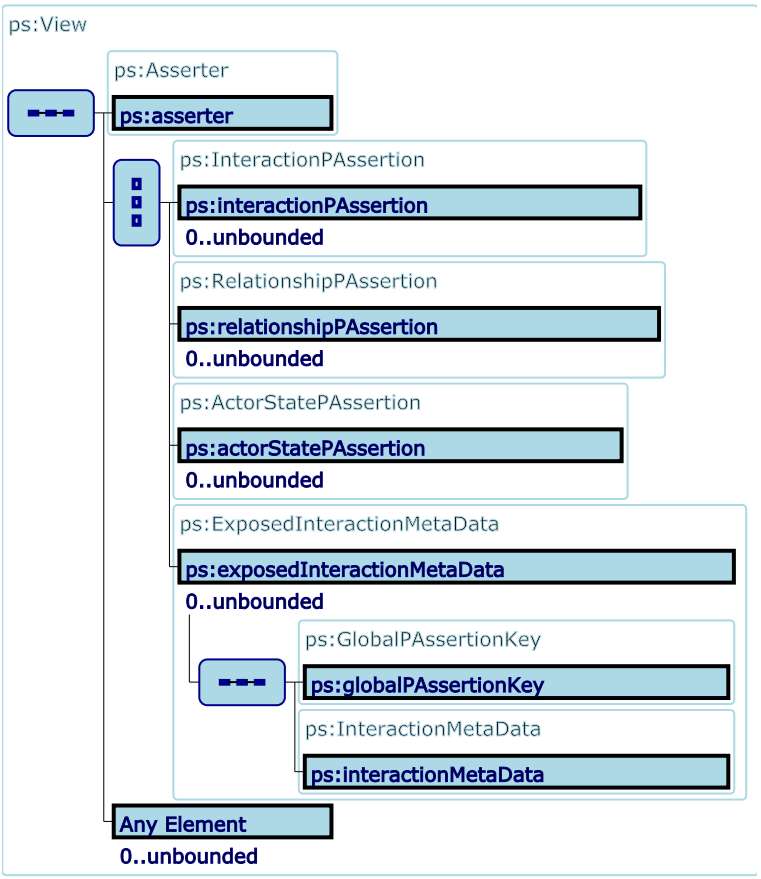


Figure 3: A View in the P-Structure

The contents of View are further described as follows:

`/ps:view`

This element contains information relating to a view of an interaction from an actor involved in that interaction (either as the sender or the receiver). It contains a sequence where each element in the sequence contains the identity of the asserting actor, a choice of one of three possible p-assertion types that the asserting actor asserts, and an OPTIONAL `any Element` to provide any application dependent information.

`/ps:view/ps:asserter`

The intent of this component is uniquely identify the actor to which this view belongs. This will be accomplished via the security architecture of the

application. Since the way actors are identified is application specific, the format used MUST be accompanied by its own namespace.

`/ps:view/ps:interactionPAssertion`

The intent of this component is to hold all interaction p-assertions asserted by the actor who owns this view for this interaction. There can be an unbounded number of occurrences of this element.

`/ps:view/ps:relationshipPAssertion`

The intent of this component is to to hold all relationship p-assertions asserted by the actor who owns this view for this interaction. There can be an unbounded number of occurrences of this element.

`/ps:view/ps:actorStatePAssertion`

The intent of this component is to to hold all actor state p-assertions asserted by the actor who owns this view for this interaction. There can be an unbounded number of occurrences of this element.

`/ps:view/ps:exposedInteractionMetaData`

The intent of this component is to enable actors to make metadata about the interaction easily available to queriers. It does not contain any information that cannot already be found inside p-assertions about this interaction.

`/ps:view/ps:exposedInteractionMetaData/ps:globalPAssertionKey`

The intent of this component is to enable the p-assertion that contributed this metadata for this interaction to be locatable. It holds the globally unique identifier for a p-assertion. The component's full definition is given below in Section 3.7.

`/ps:view/ps:exposedInteractionMetaData/ps:interactionMetaData`

The intent of this component is to hold metadata about this interaction. It contains a sequence of choices between two components: a *tracer* or an `Any Element`. (Tracers are defined in Section 3.9.)

`/ps:view/xs:Any`

The intent of this component is to provide extensibility by allowing application dependent data formats to be introduced.

3.3 Interaction P-Assertion Modelling

Interaction p-assertions record the content of a message received or sent by the asserting actor. There may be different ways according to which the content of a message may be asserted: for instance, the message content may be asserted verbatim as the asserting actor received/sent it, or an altered description may be asserted in which, for example, sensitive or large data items within the message are replaced with references to those copies of the data items stored elsewhere, or are replaced with references and a digest of the data. Therefore, in modelling an interaction p-assertion, a data structure is required 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.

The data type *InteractionPAssertion* represents any interaction p-assertion and is depicted in Figure 4. 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.

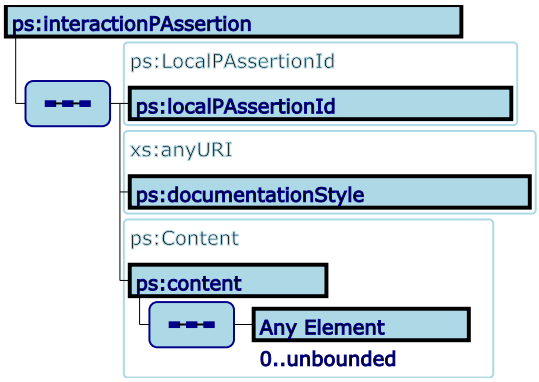


Figure 4: Model for an interaction p-assertion

The model of interaction p-assertions is further described as follows:

`/ps:interactionPAssertion`

> The root element of an interaction p-assertion. It contains elements to identify the p-assertion, describe the kinds of transformations made to the interaction's message content — or the documentation style, and the message content itself.

`/ps:interactionPAssertion/ps:localPAssertionId`

The intent of this component is to uniquely identify the interaction p-assertion within the context of a given interaction. The value of this component MUST be either an `integer`, `string` or `URI`.

`/ps:interactionPAssertion/ps:documentationStyle`

The intent of this component is to uniquely identify the *Documentation style* used to transform the content of the message to which this p-assertion refers, and is of the XML Schema type `anyURI`. Thus, for any documentation style a unique URI MUST be provided to identify it.

`/ps:interactionPAssertion/ps:content`

The intent of this component is to contain the, possibly modified, contents of the message that this interaction p-assertion is documenting, where modifications are made according to one or more documentation styles. In order to accommodate application dependent message content styles, this component includes an unbounded sequence of `any Element`.

`/ps:interactionPAssertion/ps:content/xs:any`

The intent of this component is to provide extensibility by allowing application dependent data formats to be introduced.

3.4 Identifying Interactions

Every p-assertion is made in the context of an interaction, thus an interaction p-assertion documents the receipt or sending of the message constituting the 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 one interaction to other interactions. In order to discover the provenance of some piece of information, p-assertions associated to that information must be extracted from the p-structure. Since p-assertions are organised in terms of interactions, it is necessary to be able to identify which interactions contain the required p-assertions.

In Figure 5, the model for referring to a single interaction using an `interactionKey` is shown. 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`.

The model of interaction keys is further described as follows:

`/ps:interactionKey`

This is the root element of the `ps:InteractionKey` type. Within this element is a sequence of three components that together uniquely identify an interaction.

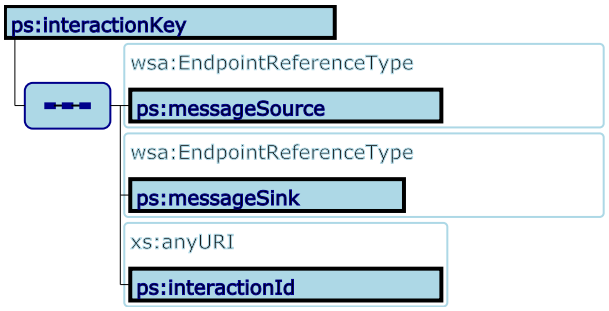


Figure 5: Model for identifying an interaction

`/ps:interactionKey/ps:messageSource`

The intent of this component is to identify from which location (e.g. port address) the message constituting this interaction came from. The type of this component is an `wsa:EndPointReference` type defined in [WS-Addressing].

`/ps:interactionKey/ps:messageSink`

The intent of this component is to identify to which location (e.g. port address) the message constituting this interaction has been sent to. The type of this component is an `wsa:EndPointReference` type defined in [WS-Addressing].

`/ps:interactionKey/ps:interactionId`

The intent of this component is to uniquely identify an interaction between the above given `messageSource` and `messageSink` using the [XMLSchema] type `anyURI`.

3.5 Actor State P-Assertion Modelling

Actor state p-assertions 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 given message, say $M1$, it executes a process (that M1 has triggered) and, similarly, *before* sending a given message, say $M2$, it executes some process (that resulted in M2). 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, or may assert the computational resources allocated to an execution.

For example, the actor state may name the workflow within which the interaction occurred.

The data type `ActorStatePAssertion` is defined to represent any actor state p-assertion and depict its structure in Figure 6. 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.

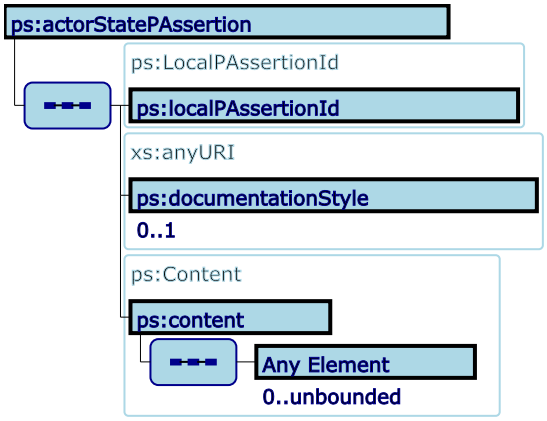


Figure 6: Model for an actor state p-assertion

The model of actor state p-assertion is further described as follows:

`/ps:actorStatePAssertion`

> The root element of an actor state p-assertion. It contains elements to identify the p-assertion, describe the kinds of transformations made to the actor's state — or the documentation style, and the actor state content itself.

`/ps:actorStatePAssertion/ps:localPAssertionId`

> The intent of this component is to uniquely identify the actor state p-assertion within the context of a given interaction. The value of this component MUST be either an `integer`, `string` or `URI`.

`/ps:actorStatePAssertion/ps:documentationStyle`

> The intent of this component is to uniquely identify the *Documentation style* used to transform the representation of the actor's state to which this p-assertion refers, and is of the XML Schema type `anyURI`. Thus, for any documentation style a unique URI MUST be provided to identify it.

`/ps:actorStatePAssertion/ps:content`

The intent of this component is to contain a, possibly modified, representation parts of the actor's state that this actor state p-assertion is documenting, where modifications are made according to one or more documentation styles. In order to accommodate application dependent actor state content, this component include an unbounded sequence of `any Element`.

`/ps:actorStatePAssertion/ps:content/xs:any`

The intent of this component is to provide extensibility by allowing application dependent actor state data formats to be introduced.

3.6 Relationship P-Assertion Modelling

Relationship p-assertions allow uni-directional relationships between both messages and data to be expressed. Relationship p-assertions are modelled 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 (`objectId`s). The model for relationship p-assertions is shown in Figure 7.

Typically, a relationship p-assertion is expressing a causal relationship, 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 or data items in its state, 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, the `subjectId` is limited to identifying one message or data item within the context of the particular interaction. An `objectId` identifies any data item, message or actor state. 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 identify a 'dividend' or a 'divisor' concept. Similarly, the `subjectId` can have a parameter name specifying which output of the operation the subject refers to.

The model of relationship p-assertion is further described as follows:

`/ps:relationshipPAssertion`

The root element of a relationship p-assertion. It contains a sequence of four components that together provide the information necessary to represent a relationship p-assertion: the local p-assertion id for this relationship p-assertion, a subject identifier, a relation, and a sequence of object identifiers.

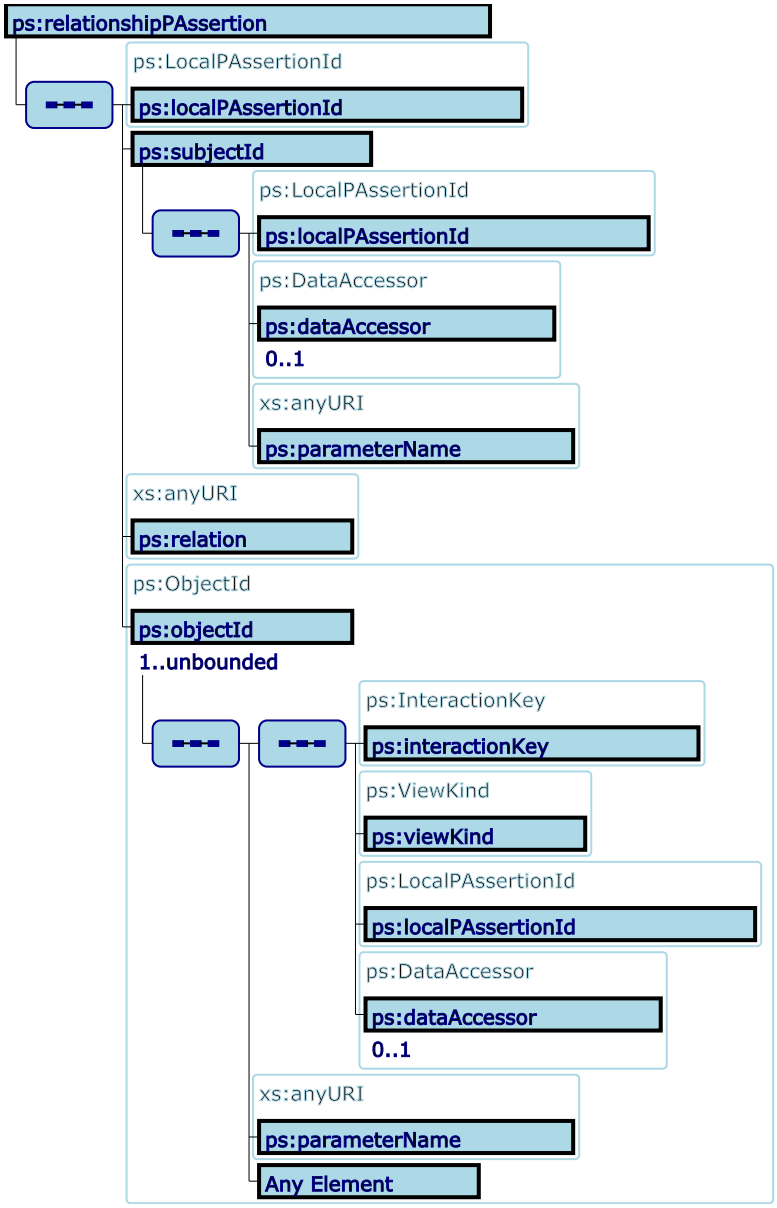


Figure 7: Relationship p-assertion model

`/ps:relationshipPAssertion/ps:localPAssertionId`

The intent of this component is to uniquely identify the relationship p-assertion within the context of a given interaction. The value of this component MUST be either an `integer`, `string` or `URI`.

`/ps:relationshipPAssertion/ps:subjectId`

The intent of this component is to provide a unique means of identifying a data item or message acting as the *subject* of the asserted relationship p-assertion, i.e. the output of the relation that this relationship p-assertion is documenting. It contains three components, which together provide the necessary information to identify the data item or message that acts as the subject within this relationship p-assertion.

`/ps:relationshipPAssertion/ps:subjectId/ps:localPAssertionId`

The intent of this component is to uniquely identify, within an interaction record, the p-assertion within which the data item or message that represents the subject of this relationship p-assertion resides. The value of this component MUST be either an `integer`, `string` or `URI`.

`/ps:relationshipPAssertion/ps:subjectId/ps:dataAccessor`

The intent of this component is to provide an application dependent mechanism for identifying the subject of a relation within the above identified p-assertion, if that subject is a data item. Thus, this component is OPTIONAL, and its use is dependent upon the subject of the relationship p-assertion being a data item.

`/ps:relationshipPAssertion/ps:subjectId/ps:parameterName`

The intent of this component is to identify the subject role that a data item or message plays in this relationship p-assertion. The names of subject roles MUST be referred to via URI's, and thus this component is of type `AnyURI`.

`/ps:relationshipPAssertion/ps:relation`

The intent of this component is to provide a means to name the relation of this relationship p-assertion, where this takes the type `AnyURI`.

`/ps:relationshipPAssertion/ps:objectId`

The intent of this component is to identify the object(s) of this relationship p-assertion, i.e. the inputs to the above identified relation. It contains a sequence of six components described below.

`/ps:relationshipPAssertion/ps:objectId/ps:interactionKey`

The intent of this component is to uniquely identify the interaction within which the this relation object resides. The full definition of `interactionKey` is given above in Section 3.4.

`/ps:relationshipPAssertion/ps:objectId/ps:viewKind`

The intent of this component is to identify the view that the actor asserting this relationship p-assertion has on the above identified interaction within which the object of this relation resides. This view can be either `SenderViewKind` or `ReceiverViewKind`, where these are extensions to the abstract type `ViewKind` (see Section 3.7).

`/ps:relationshipPAssertion/ps:objectId/ps:localPAssertionId`

The intent of this component is to uniquely identify the p-assertion within which the data item or message that represents an object of this relationship p-assertion resides, given the above identified interaction. The value of this component MUST be either an `integer`, `string` or `URI`.

`/ps:relationshipPAssertion/ps:objectId/ps:dataAccessor`

The intent of this component is to provide an application dependent mechanism for identifying an object of the above relation within the above identified p-assertion, if that object is a data item. Thus, this component is OPTIONAL, and its use is dependent upon the object of the relationship p-assertion being a data item.

`/ps:relationshipPAssertion/ps:objectId/ps:parameterName`

The intent of this component is to identify the object role that a data item or message plays in this relationship p-assertion. The names of object roles MUST be referred to via URI's, and thus this component is of type `AnyURI`.

`/ps:relationshipPAssertion/ps:objectId/xs:any`

The intent of this component is to allow application dependent data formats to be introduced.

3.7 Identifying P-Assertions

Earlier it was shown how to identify the interaction about which an assertion is being made, additionally it was also shown how every p-assertion has its own identifier: the `localPassertionIidentifier`. 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 it is possible to construct a global p-assertion key as shown in Figure 8. A global

p-assertion key consists of an interaction key, whether the sender or receiver in the interaction made the assertion (the `viewKind`) and the local p-assertion id. A global p-assertion key uniquely identifies a p-assertion.

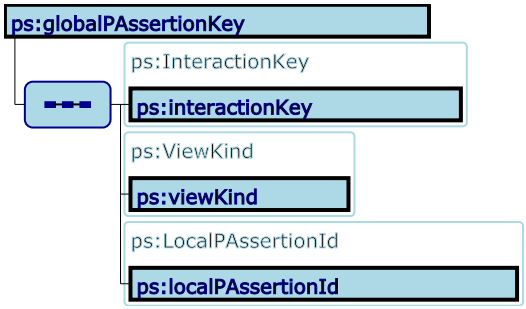


Figure 8: Global P-Assertion Key

The model of a global p-assertion key is further described as follows:

`/ps:globalPAssertionKey`

The root element of a global p-assertion key. This element contains a sequence of three components that together uniquely identify a p-assertion: an interaction key, a view kind and a local p-assertion id.

`/ps:globalPAssertionKey/ps:interactionKey`

The intent of this component is to uniquely identify the interaction within which this p-assertion resides. The full definition of `interactionKey` is given in Section 3.4.

`/ps:globalPAssertionKey/ps:viewKind`

The intent of this component is to identify the view that the actor that asserted this p-assertion had on the above identified interaction. This view can be either `SenderViewKind` or `ReceiverViewKind`.

`/ps:globalPAssertionKey/ps:localPAssertionId`

The intent of this component is to uniquely identify the p-assertion within the context of the above identified interaction. The value of this component MUST be either an `integer`, `string` or `URI`.

3.8 Identifying Data Items

In addition to identifying p-assertions, it is necessary to be able to identify individual data items within those p-assertions in order to answer provenance queries. A *p-assertion data item* is part, or all, of a p-assertion, and can be identified by a

`pAssertionDataKey`. A `pAssertionDataKey` extends a `globalPAssertionKey`, identifying the p-assertion containing the data item, with a `dataAccessor`, identifying the location of the data item within the p-assertion. The model for a p-assertion data key is shown in Figure 9.

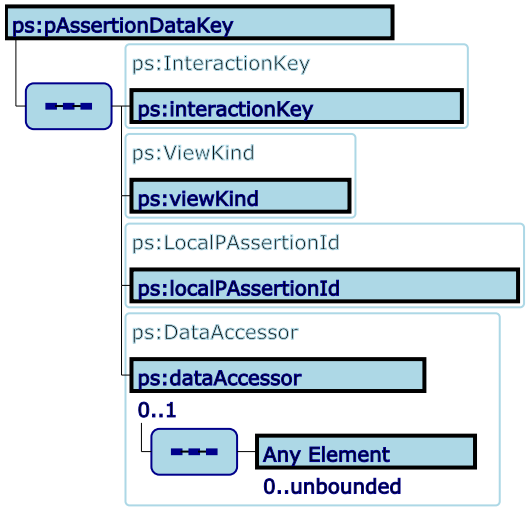


Figure 9: P-Assertion Data Key

The model of p-assertion data key is further described as follows:

`/ps:pAssertionDataKey`

The root element of a p-assertion data key. It contains four components that together uniquely identify a data item within a p-assertion: an interaction key, a view kind, a local p-assertion key and a data accessor. The element extends the `ps:globalPAssertionKey`.

`/ps:pAssertionDataKey/ps:interactionKey`

The intent of this component is to uniquely identify the interaction within which the p-assertion that contains the data item resides. The full definition of `interactionKey` is given in Section 3.4.

`/ps:pAssertionDataKey/ps:viewKind`

The intent of this component is to identify the view that the actor that asserted the p-assertion containing the data item had on the above identified interaction. This view can be either `SenderViewKind` or `ReceiverViewKind`, where these are extensions to the abstract type `ViewKind` (see Section 3.2).

`/ps:pAssertionDataKey/ps:localPAssertionId`

The intent of this component is to uniquely identify the p-assertion containing the data item within the context of the above identified interaction. The value of this component MUST be either an `integer`, `string` or `URI`.

`/ps:pAssertionDataKey/ps:dataAccessor`

The intent of this component is to provide an application dependent mechanism for identifying a data item within the above identified p-assertion.

`/ps:pAssertionDataKey/ps:dataAccessor/xs:any`

The intent of this component is to provide extensibility by allowing application dependent data formats to be introduced.

3.9 Interaction Contexts and the P-Header

Application actors must exchange provenance-specific context information related to particular interactions for the process documentation to be usable by querying 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 independently in messages created specifically for passing such information, or as extra data in existing messages.

In the former case, the context information conforms to the *interaction context* structure shown in Figure 10. An interaction context contains an interaction key, a `viewKind` to indicate the particular view the asserter of this information has on the interaction, and any number of items of *interaction metadata*, which are contextual information regarding the identified interaction. Examples of interaction metadata include tracers and possibly links to other provenance stores in the case of distributed provenance (see the profile document [MTG+06] for a specific way of representing distributed provenance).

The model of an interaction context is further described as follows:

`/ps:interactionContext`

The root element of interaction context. It contains a sequence of two components that together provide information about the context of an interaction: the interaction key and a set of `interactionMetadata`.

`/ps:interactionContext/ps:interactionKey`

The intent of this component is to uniquely identify the interaction to which this context applies. The full definition of `interactionKey` is given above in Section 3.4.

`/ps:interactionContext/ps:viewKind`

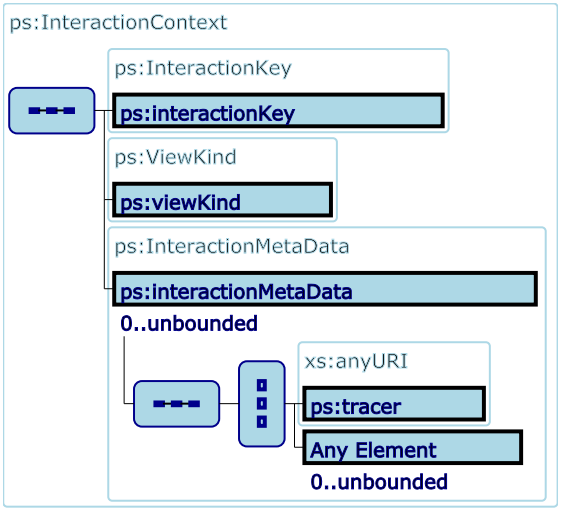


Figure 10: Model of an Interaction Context.

The intent of this component is to identify the view that the actor asserting this information has on the above identified interaction. This view can be either `SenderViewKind` or `ReceiverViewKind`, where these are extensions to the abstract type `ViewKind` (see Section 3.7).

`/ps:interactionContext/ps:interactionMetaData`

The intent of this component is to hold metadata about the above identified interaction. It contains a sequence of choices between two components: a *tracer* or an `AnyElement`.

`/ps:interactionContext/ps:interactionMetaData/ps:tracer`

The intent of this component is to provide a means of associating an interaction with other, related interactions using shared information. All interactions that are related share the same unique piece of information — referred to as a *tracer*, represented by the XML Schema type `anyURI`.

`/ps:interactionContext/ps:interactionMetaData/xs:any`

The intent of this component is to provide extensibility by allowing an unbounded number of application dependent data formats to be introduced.

4 Conclusion

In this document, a data model for process documentation is described, and a model for how process documentation can be organised and how individual

pieces of process documentation — that is, p-assertions, and their contents can be expressed is presented. Also provided is a model of interaction contexts and the p-header, in which such context information can be passed between actors. Combining these models together, a complete specification for an open process documentation data model is provided. This document exists as part of a series of related documents that together form the basis of a proposal towards a standardisation process for data provenance. Other documents in the series present other aspects of data provenance such as documentation styles [TMG+06b], security [TMG+06a], scalability and distribution [MTG+06], provenance queries [MMG+06] and recording [GTM+06] and overall vision [TMG+06c].

Appendix A

The following XML document describes the p-structure types and elements used in this document.

```
 <?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:ps="http://www.pasoa.org/schemas/version023s1/PStruct.xsd"
                    xmlns:xs="http://www.w3.org/2001/XMLSchema"
                    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
                    attributeFormDefault="unqualified" elementFormDefault="qualified"
                    targetNamespace="http://www.pasoa.org/schemas/version023s1/PStruct.xsd">

       <xs:import
        namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"
        schemaLocation="./wsaddressing.xsd"/>

       <xs:annotation>
        <xs:documentation>
            The P-Structure. This is a logical view of the contents of a provenance store.
            The P-Structure contains a set of interaction records that document interactions
            between actors.

            Author: Paul Groth


                  Copyright (c) 2006 University of Southampton
                  See pasoalicense.txt for license information.
                  http://www.opensource.org/licenses/mit-license.php

             </xs:documentation>
    </xs:annotation>

      <!-- We define the global elements of the p-struture here so that
         They can be referenced by external schemas. Below we define
         the types of the p-structure. The prefix ps: refers to this
         document. -->
    <xs:element name="pstruct" type="ps:PStructure">
        <xs:annotation>
            <xs:documentation>
                (Root Element Start Here) An instance of the
                p-structure. Each instance of the p-structure contains a
                set of interaction records.
            </xs:documentation>
        </xs:annotation>
    </xs:element>

    <xs:element name="interactionRecord" type="ps:InteractionRecord">
        <xs:annotation>
            <xs:documentation>
                An interaction record describes the client and service
                view of a particular interaction. An interaction record
                is identified by an interaction key.
            </xs:documentation>
        </xs:annotation>
    </xs:element>

    <xs:element name="view" type="ps:View">
        <xs:annotation>
            <xs:documentation>
                     A view of an interaction by an actor.
            </xs:documentation>
        </xs:annotation>
    </xs:element>
```

```
<xs:element name="interactionKey" type="ps:InteractionKey">
    <xs:annotation>
        <xs:documentation>
            An interaction key contains a message source,
            a message sink, and an interaction id.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="messageSource" type="wsa:EndpointReferenceType">
    <xs:annotation>
        <xs:documentation>
            The source of the message within the sender.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="messageSink" type="wsa:EndpointReferenceType">
    <xs:annotation>
        <xs:documentation>
            The sink of the message within the receiver.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="interactionId" type="xs:anyURI">
    <xs:annotation>
        <xs:documentation>
            A URI that uniquely identifies this interaction .
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="asserter" type="ps:Asserter">
    <xs:annotation>
        <xs:documentation>
            Each view (either client or service) comes from a
            particular actor.  The actor that asserts p-assertion
            in a particular view is termed the asserter. The identity
            of the asserter is documented in the corresponding view inside
            the interaction record.
        </xs:documentation>
    </xs:annotation>
</xs:element>

 <xs:element name="numberOfExpectedAssertions" type="ps:NumberOfExpectedAssertions">
    <xs:annotation>
        <xs:documentation>
            The number of expected p-assertions to be contained
            within a view as documented by the asserting actor.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<!-- The following elements define the three types of p-assertions. -->
<xs:element name="interactionPAssertion" type="ps:InteractionPAssertion">
    <xs:annotation>
        <xs:documentation>
            Assertion as to the content of an interaction.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="actorStatePAssertion" type="ps:ActorStatePAssertion">
```

```
    <xs:annotation>
        <xs:documentation>
            Information supplied by an actor about its state in the
            context of this interaction . Examples include the
            script that was used in running a service or the time
            when an invocation was sent/received.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="relationshipPAssertion" type="ps:RelationshipPAssertion">
    <xs:annotation>
        <xs:documentation>
            Describes a relationship between a p-assertion recorded
            in this view and another p-assertion made by the
            asserting actor. This can be seen as a triple: subject
            identifier, relation, object identifier.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<!-- End P-assertion defintions -->

<xs:element name="viewKind" type="ps:ViewKind">
    <xs:annotation>
        <xs:documentation>
            Whether a view is from the sender or receiver.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="localPAssertionId" type="ps:LocalPAssertionId">
    <xs:annotation>
        <xs:documentation>
            Uniquely identifies a p-assertion within a view.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="dataAccessor" type="ps:DataAccessor">
    <xs:annotation>
        <xs:documentation>
            An application dependent mechanism for referencing a
            piece of data within a p-assertion.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="parameterName" type="xs:anyURI">
    <xs:annotation>
        <xs:documentation>
            The parameter name of a data item referenced
            in a relationship p-assertion.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="documentationStyle" type="xs:anyURI">
    <xs:annotation>
        <xs:documentation>
            The style of documentation used when recording
            an interaction p-assertion.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="pAssertionDataKey" type="ps:PAssertionDataKey"/>
```

```
<xs:element name="objectId" type="ps:ObjectId"/>

<xs:element name="relation" type="xs:anyURI"/>

<xs:element name="globalPAssertionKey" type="ps:GlobalPAssertionKey"/>

<xs:element name="interactionMetaData" type="ps:InteractionMetaData"/>

<xs:element name="interactionContext" type="ps:InteractionContext"/>

<xs:element name="senderViewKind" type="ps:SenderViewKind"/>
<xs:element name="exposedInteractionMetaData" type="ps:ExposedInteractionMetaData"/>

<!-- Type Definitions -->
<xs:complexType name="InteractionKey">
    <xs:sequence>
        <xs:element ref="ps:messageSource"/>
        <xs:element ref="ps:messageSink"/>
        <xs:element ref="ps:interactionId"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="PStructure">
    <xs:sequence>
        <xs:element maxOccurs="unbounded" minOccurs="0" ref="ps:interactionRecord"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="InteractionRecord">
    <xs:sequence>
        <xs:element ref="ps:interactionKey" />
        <xs:element minOccurs="0" name="sender" type="ps:View">
            <xs:annotation>
                <xs:documentation>
                    The senders's view of the interaction .
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element minOccurs="0" name="receiver" type="ps:View">
            <xs:annotation>
                <xs:documentation>
                    The receiver's view of the interaction.
                    WARNING!!! In future revisions the receiver view
                    may not be allowed to include relationship
                    p-assertions. If you have an example of the
                    usage of relationship p-assertions in this view,
                    please contact the authors of the schema.
                    Thanks!

                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:any namespace="##other" processContents="lax"
            minOccurs="0" maxOccurs="unbounded" />

    </xs:sequence>
</xs:complexType>

<xs:complexType name="Asserter">
    <xs:sequence>
        <xs:any namespace="##other" maxOccurs="unbounded"
            minOccurs="0" />
    </xs:sequence>
</xs:complexType>

<xs:complexType name="Asserter">
```

```
    <xs:sequence>
        <xs:any namespace="##other" maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>

<xs:simpleType name="NumberOfExpectedAssertions">
    <xs:restriction base="xs:positiveInteger"/>
</xs:simpleType>



<!-- Following the WS-Security spec, we allow any type of identiification -->


    <xs:complexType name="View">
    <xs:sequence>
        <xs:element ref="ps:asserter" />
        <xs:choice maxOccurs="unbounded" minOccurs="0">
            <xs:element maxOccurs="unbounded" minOccurs="0"
                ref="ps:interactionPAssertion" />
            <xs:element maxOccurs="unbounded" minOccurs="0"
                ref="ps:relationshipPAssertion" />
            <xs:element maxOccurs="unbounded" minOccurs="0"
                ref="ps:actorStatePAssertion" />
            <xs:element maxOccurs="unbounded" minOccurs="0"
               ref="ps:exposedInteractionMetaData"/>
        </xs:choice>

        <xs:any namespace="##other" processContents="lax"
            minOccurs="0" maxOccurs="unbounded" />

    </xs:sequence>
</xs:complexType>

<xs:complexType name="RelationshipPAssertion">
    <xs:sequence>
        <xs:element ref="ps:localPAssertionId"/>
        <xs:element name="subjectId">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="ps:localPAssertionId"/>
                    <xs:element ref="ps:dataAccessor" minOccurs="0" />
                    <xs:element ref="ps:parameterName"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element ref="ps:relation"/>
        <xs:element maxOccurs="unbounded" ref="ps:objectId"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="GlobalPAssertionKey">
    <xs:sequence>
        <xs:element ref="ps:interactionKey"/>
        <xs:element ref="ps:viewKind"/>
        <xs:element ref="ps:localPAssertionId"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="PAssertionDataKey">
    <xs:complexContent>
        <xs:extension base="ps:GlobalPAssertionKey">
            <xs:sequence>
                <xs:element minOccurs="0" ref="ps:dataAccessor"/>
            </xs:sequence>
```

```
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="InteractionPAssertion">
    <xs:sequence>
        <xs:element maxOccurs="1" minOccurs="1" ref="ps:localPAssertionId"/>
        <xs:element maxOccurs="1" minOccurs="1" ref="ps:documentationStyle"/>
        <xs:element maxOccurs="1" minOccurs="1" name="content" type="ps:Content"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="ActorStatePAssertion">
    <xs:sequence>
        <xs:element maxOccurs="1" minOccurs="1" ref="ps:localPAssertionId"/>
        <xs:element maxOccurs="1" minOccurs="0" ref="ps:documentationStyle"/>
        <xs:element maxOccurs="1" minOccurs="1" name="content" type="ps:Content"/>
    </xs:sequence>
</xs:complexType>

<xs:simpleType name="LocalPAssertionId">
    <xs:union memberTypes="xs:long xs:string xs:anyURI"/>
</xs:simpleType>


<xs:complexType name="ViewKind" abstract="true">
    <xs:annotation>
        <xs:documentation>
               Instance documents must select something that is derived
        </xs:documentation>
    </xs:annotation>
</xs:complexType>

<xs:complexType name="SenderViewKind">
    <xs:complexContent>
        <xs:restriction base="ps:ViewKind"></xs:restriction>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="ReceiverViewKind">
    <xs:complexContent>
        <xs:restriction base="ps:ViewKind"></xs:restriction>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="ObjectId">
    <xs:complexContent>
        <xs:extension base="ps:PAssertionDataKey">
            <xs:sequence>
                <xs:element ref="ps:parameterName"/>
                <xs:any namespace="##other" processContents="lax"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="DataAccessor">
    <xs:sequence>
        <xs:any maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="Content">
    <xs:sequence>
        <xs:any namespace="##any" maxOccurs="unbounded" minOccurs="0"/>
```

```
        </xs:sequence>
    </xs:complexType>


    <xs:complexType name="InteractionMetaData">
        <xs:sequence>
           <xs:choice maxOccurs="unbounded">
                <xs:element name="tracer" type="xs:anyURI"/>
                <xs:any namespace="##other" maxOccurs="unbounded" minOccurs="0"/>
           </xs:choice>
       </xs:sequence>
    </xs:complexType>

<xs:complexType name="InteractionContext">
        <xs:sequence>
            <xs:element ref="ps:interactionKey" />
            <xs:element ref="ps:viewKind"/> <!-- View Kind of the actor who created the metadata -->
            <xs:element ref="ps:interactionMetaData"
                maxOccurs="unbounded" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>

     <xs:complexType name="ExposedInteractionMetaData">
         <xs:sequence>
              <xs:element ref="ps:globalPAssertionKey"/>
               <xs:element ref="ps:interactionMetaData"/>
         </xs:sequence>
    </xs:complexType>

</xs:schema>
```

Appendix B

The following illustrates the p-header types and elements used in this document.

```
<?xml version="1.0" encoding="UTF-8"?> <xs:schema
targetNamespace="http://www.pasoa.org/schemas/version023s1/PHeader.xsd"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:ph="http://www.pasoa.org/schemas/version023s1/PHeader.xsd"
           xmlns:ps="http://www.pasoa.org/schemas/version023s1/PStruct.xsd">

   <xs:annotation>
        <xs:documentation>
           The PHeader schema
           Author: Paul Groth


           Copyright (c) 2006 University of Southampton
           See pasoalicense.txt for license information.
           http://www.opensource.org/licenses/mit-license.php

        </xs:documentation>
    </xs:annotation>


    <xs:import namespace="http://www.pasoa.org/schemas/version023s1/PStruct.xsd"
                    schemaLocation="./PStruct.xsd"/>

    <xs:element name="pheader" type="ph:PHeader"/>

    <xs:complexType name="PHeader">
        <xs:annotation>
            <xs:documentation>Provenance Specific Header Information</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="ps:interactionKey" maxOccurs="1" minOccurs="1" />
            <xs:element ref="ps:interactionMetaData" maxOccurs="unbounded" minOccurs="0"/>
            <xs:element ref="ps:interactionContext" maxOccurs="unbounded" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

References

[Bra97] Scott Bradner. Key words for use in RFCs to indicate requirement levels. http://www.ietf.org/rfc/rfc2119.txt, 1997.

[GJ+06] Paul Groth, Sheng Jiang, , Simon Miles, Steve Munroe, Victor Tan, Sofia Tsasakou, and Luc Moreau. An Architecture for Provenance Systems. Technical report, Electronics and Computer Science, University of Southampton, 2006.

[GTM+06] Paul Groth, Victor Tan, Steve Munroe, Sheng Jiang, Simon Miles, and Luc Moreau. Process Documentation Recording Protocol. Technical report, University of Southampton, June 2006.

[MMG+06] Simon Miles, Steve Munroe, Paul Groth, Sheng Jiang, Victor Tan, John Ibbotson, and Luc Moreau. Process Documentation Query Protocol. Technical report, University of Southampton, June 2006.

[MTG+06] Steve Munroe, Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. WSRF Data Model Profile for Distributed Provenance. Technical report, University of Southampton, June 2006.

[TGJ+06] Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, and Luc Moreau. WS Provenance Glossary. Technical report, Electronics and Computer Science, University of Southampton, 2006.

[TMG+06a] Victor Tan, Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. A Profile for Non-Repudiable Process Documentation. Technical report, University of Southampton, June 2006.

[TMG+06b] Victor Tan, Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. Basic Transformation Profile for Documentation Style. Technical report, University of Southampton, June 2006.

[TMG+06c] Victor Tan, Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. The Provenance Standardisation Vision. Technical report, University of Southampton, June 2006.

[W3C99] W3C. XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999. http://www.w3.org/TR/xpath, 1999.

ws-prov-rec

Authors:
Paul Groth, U. Southampton
Simon Miles, U. Southampton
Victor Tan, U. Southampton
John Ibbotson, IBM
Luc Moreau, U. Southampton

August 24, 2006

The P-assertion Recording Protocol

Status of this Memo

This document provides information to the community regarding the specification of a protocol for recording process documentation into a provenance store and has the status of a working draft. The document does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

Related documents [MGJ+06, TMG+06] define schemas to be used for documentation about the execution of a process, *process documentation*, and introduce a *provenance store*, a type of Web Service with the capability for storing and giving access to process documentation. In particular, process documentation has a defined schema, the *p-structure*, which clients use when creating process documentation to be recorded. In this document, we specify an interface, the P-assertion Recording Protocol (PReP) [GLM04], by which a recording actor can communicate with a provenance store in order to record process documentation. This primarily takes the form of an abstract WSDL interface defining messages to be accepted and produced by a provenance store.

Contents

1 Introduction

Every provenance store supplies a Web Service interface for recording process documentation. It has a single operation, *record*, that takes Record document as input and returns an acknowledgement as result. This document defines the schema for the request and acknowledgement messages. The WSDL 1.1 description of the interface is given in Appendix B.

1.1 Goals and Requirements

The goal of this document is to define a Web Service interface, the P-assertion Recording Protocol (PReP), for recording process documentation into a provenance store.

1.1.1 Requirements

In meeting this goal, this document must address the following requirements:

- Define the schema of the record request message sent to the provenance store.
- Define the behaviour of a provenance store on receiving a record request.
- Define the schema of the acknowledgement message returned by the provenance store.

1.1.2 Non-Requirements

This document does not specify any properties or guarantees that the protocol may have. Such a specification is left to an abstract definition of the messages required for recording process documentation [GLM04]. This document also does not define the storage format, medium, or technology that a provenance store uses.

2 Terminology and Notation

All definitions for the concepts and structures found within this document can be found in [TGJ+06].

2.1 XML Namespaces

The XML Namespace URI that MUST be used by implementations of this specification is: `http://www.pasoa.org/schemas/version023s1/record/PRecord.xsd`

Table 1 lists XML namespaces that are used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

Prefix	XML Namespace	Specification(s)
pr	`http://www.pasoa.org/schemas/version023s1/record/PRecord.xsd`	[PRecord]
ps	`http://www.pasoa.org/schemas/version023s1/PStruct.xsd`	[P-Structure]
xs	`http://www.w3.org/2001/XMLSchema`	[XMLSchema]

Table 1: Prefixes and XML Namespaces used in this specification

2.2 Notational Conventions

The keywords "MUST ", "MUSTNOT ", "REQUIRED ", "SHALL ", "SHALLNOT ", "SHOULD ", "SHOULDNOT ", "RECOMMENDED ", "MAY ", and "OPTIONAL " in this document are to be interpreted as described in [Bra97].

2.3 XML Schema Diagrams

This document adopts a graphical notation to describe XML Schema. Figure 1 gives an example of a small XML Schema displayed as a diagram, which is now explained with reference to the figure.

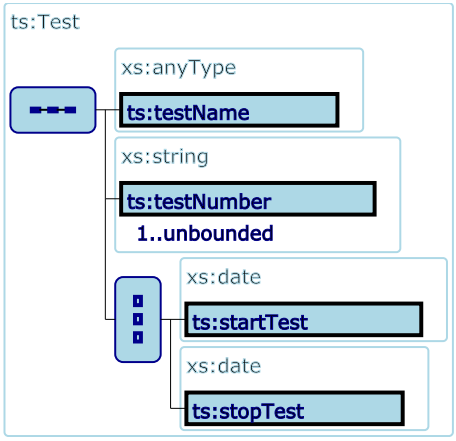


Figure 1: An example XML Schema diagram

Figure 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:stopTest`, 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.

2.4 XPath notation

In addition to the XML Schema diagrams, an XPath notation [W3C99] is used to identify each individual element in the specification along with its context, in order to describe precisely its meaning and use.

3 Recording Process Documentation

We specify below the request document schema, provenance store behaviour and acknowledgement document schema for recording process documentation. The full schema document, in which request and acknowledgement message structures are defined, is given in Appendix A. The WSDL 1.1 description of the interface taking and producing these messages is given in Appendix B. These schemas are based on those defined in the p-structure data model [MGJ+06], therefore, a familiarity with that document and the concepts it defines is assumed.

3.1 Request

A process documentation record request is a message sent by a *recording actor* to a *provenance store* that contains process documentation to be stored by the provenance store. It is instantiated as a document conforming to the schema depicted in Figure 2.

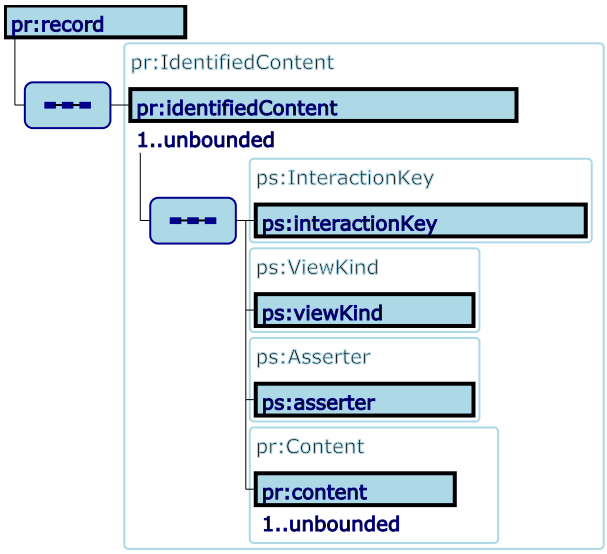


Figure 2: Process Documentation Record Request

One feature of the request message is to allow for bulk recording of multiple p-assertions and other associated information. For example, an actor may wish to record p-assertions at a time when network traffic is low or when the actor is not busy performing other tasks. Therefore, multiple pieces of information, which we call *content*, can be recorded about different interactions at the same time. Allowing content to be bundled together gives greater flexibility to the actor.

`/pr:record`

This element contains a record request.

`/pr:record/pr:identifiedContent`

This element is a container that associates a content element with the View that the content belongs to. There can be multiple `pr:identifiedContent` elements within a record request.

`/pr:record/pr:identifiedContent/ps:interactionKey`

This element identifies the key of the InteractionRecord that the content belongs to.

`/pr:record/pr:identifiedContent/ps:viewKind`

This element is the name of the View within the InteractionRecord that the content belongs to.

`/pr:record/pr:identifiedContent/ps:asserter`

This element is the identity of the asserter of the content.

`/pr:record/pr:identifiedContent/pr:content`

This element contains the information being recorded. This element is defined as a choice between the elements defined in Figure 3. This is to prevent any incorrectly typed information from being recorded. There can be multiple content elements within one `pr:identifiedContent` element.

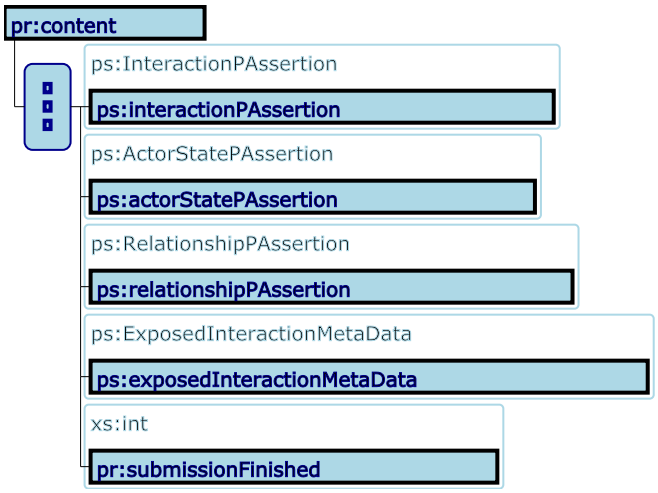


Figure 3: Content element definition

`/pr:record/pr:identifiedContent/pr:content/ps:interactionPAssertion`

An interaction p-assertion.

`/pr:record/pr:identifiedContent/pr:content/ps:actorStatePAssertion`

An actor state p-assertion.

`/pr:record/pr:identifiedContent/pr:content/ps:relationshipPAssertion`

A relationship p-assertion.

`/pr:record/pr:identifiedContent/pr:content/ps:exposedInteractionMetaData`

Some exposed interaction metadata.

`/pr:record/pr:identifiedContent/pr:content/pr:submissionFinished`

This element allows an actor to record how many p-assertions it expects to record in total. This allows querying actors to determine whether or not an actor is finished recording p-assertions.

3.2 Behaviour

On receiving a record message, a provenance store must store the contents of the message. The provenance store must return an acknowledgement, which may be sent synchronously or asynchronously depending on the underlying transport mechanism being used between the client and the provenance store. The provenance store must make the contents of the message available through the navigation of the p-structure via some query interface.

3.3 Acknowledgements

A record acknowledgement is sent by a provenance store to the actor that issued the corresponding record request. It is a document conforming to the schema depicted in Figure 4. Depending on the transport mechanism being used either a synchronous or asynchronous acknowledgement may be returned. The schema for the acknowledgement is defined here.

`/pr:recordAck`

This element contains a record acknowledgement. Each `pr:identifiedContent` element recorded has a corresponding acknowledgement element.

`/pr:recordAck/pr:synch_ack`

If the underlying transport is synchronous this element is used. The element has no internal data because the acknowledgement immediately follows the record request on the same communication channel, therefore, the sender already knows what record request `pr:recordAck` is acknowledging.

`/pr:recordAck/pr:ack`

This element is used when returning an acknowledgement asynchronously. In this case, clients may receive acknowledgements in any order, therefore, some means must be provided for clients to identify which record request each acknowledgement is associated with. Hence, several identifiers are provided that allow clients to make this association. These identifiers are defined below.

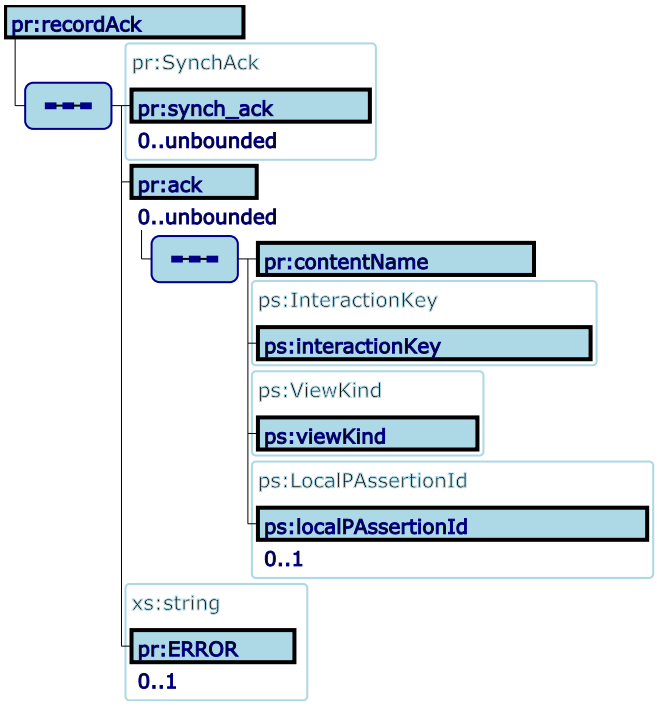


Figure 4: Process Documentation Record Acknowledgement

`/pr:recordAck/pr:ack/pr:contentName`

This element states the type of content that was recorded. The following standard strings are used, which map to element names defined in the `pr:content` element:

- interactionPAssertion
- actorStatePAssertion
- relationshipPAssertion
- exposedInteractionMetaData
- submissionFinished

Therefore, if a `pr:identifiedContent` element in the record request message contained an interaction p-assertion then the acknowledgement would contain a `pr:contentName` with the string interactionPAssertion as its contents.

`/pr:recordAck/pr:ack/ps:interactionKey`

The interaction key of the `pr:identifiedContent` element being acknowledged.

`/pr:recordAck/pr:ack/ps:viewKind`

The view kind of the `pr:identifiedContent` element being acknowledged.

`/pr:recordAck/pr:ack/ps:localPAssertionId`

If the recording of a p-assertion is being acknowledged, this is its local p-assertion id.

3.4 Faults

The acknowledgement message provides an element in which provenance stores can put implementation specific error messages.

`/pr:recordAck/pr:ERROR`

This element holds implementation specific error messages. This element may be present in both synchronous and asynchronous acknowledgement messages.

4 Security Considerations

This specification defines the process documentation record request and acknowledgement messages for any provenance store supporting the PReP protocol. In this context, there are two categories of security aspects that need to be considered: (a) securing the message exchanges and (b) securing the provenance store contents.

4.1 Securing Message Exchanges

When messages are exchanged between a recorder and provenance store when recording process documentation, it is recommended that the communication be secured using the mechanisms described in WS-Security [Var04]. In order to properly secure messages, the message body (record document or acknowledgement) and all relevant headers need to be included in the digital signature so as to prove the integrity of the message. In the event that a recorder frequently records process documentation, it is recommended that a security context be established using the mechanisms described in WS-Trust [Var05b] and WS-SecureConversation [Var05a], allowing for potentially more efficient means of authentication.

4.2 Securing Provenance Store Contents

Since this specification defines a mechanism to record process documentation in provenance stores, security policies should be established that ensure that only authorized actors can record p-assertions. A detailed architecture for securing provenance stores and their contents can be found in [GJ+06].

5 Conclusion

In this document, a protocol for recording process documentation into a provenance store is defined. The definition includes the format of request and acknowledgement messages and the expected behaviour of provenance stores with respect to those messages. Using the protocol an actor can record process documentation compatible with the p-structure.

A Process Documentation Record Schema

Below we give the full schema for process documentation record and record acknowledgement messages.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.pasoa.org/schemas/version023s1/record/PRecord.xsd"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified"
           xmlns:pr="http://www.pasoa.org/schemas/version023s1/record/PRecord.xsd"
           xmlns:ps="http://www.pasoa.org/schemas/version023s1/PStruct.xsd"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.pasoa.org/schemas/version023s1/PStruct.xsd ..\PStruct.xsd ">

    <xs:annotation>
     <xs:documentation>
     The Provenance Store Record schema
Author: Paul Groth Last
     Modified: 30 August 2005

        Copyright (c) 2006 University of Southampton
        See pasoalicense.txt for license information.
        http://www.opensource.org/licenses/mit-license.php

     </xs:documentation>
    </xs:annotation>


    <xs:import
     namespace="http://www.pasoa.org/schemas/version023s1/PStruct.xsd"
     schemaLocation="../PStruct.xsd" />

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

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

    <xs:element name="content" type="pr:Content"  />

    <xs:complexType name="RecordAck">
        <xs:sequence>
            <xs:element name="synch_ack" minOccurs="0"
             maxOccurs="unbounded" type="pr:SynchAck">
             <xs:annotation>
             <xs:documentation>
             If the provenance store is being accessed under
             a synchronous connection, (i.e. remote procedure
             call style) the provenance store may return a
             synch_ack instead of an ack element. Under such
             a situation, the provenance store does not need
             to parse the incoming message in order to send
             an acknowledgment.
             </xs:documentation>
             </xs:annotation>
            </xs:element>
            <xs:element name="ack" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="contentName">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="interactionPAssertion"/>
                                    <xs:enumeration value="actorStatePAssertion"/>
```

```
                                    <xs:enumeration value="relationshipPAssertion"/>
                                    <xs:enumeration value="exposedInteractionMetaData"/>
                                    <xs:enumeration value="submissionFinished"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>

                        <xs:element ref="ps:interactionKey"/>
                        <xs:element ref="ps:viewKind"/>
                        <xs:element ref="ps:localPAssertionId" minOccurs="0"/>

                    </xs:sequence>
                </xs:complexType>
            </xs:element>

            <xs:element name="ERROR" minOccurs="0" maxOccurs="1" type="xs:string">
             <xs:annotation>
             <xs:documentation>
             This field should be pressent if the messageName
             element's value is ERROR. As of yet we do not
             define the type of the error message that can be
             returned by the provenance store.
             </xs:documentation>
             </xs:annotation>
            </xs:element>
        </xs:sequence>
    </xs:complexType>

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

    <xs:complexType name="IdentifiedContent">
     <xs:sequence>
     <xs:element ref="ps:interactionKey"/>
     <xs:element ref="ps:viewKind"/>
     <xs:element ref="ps:asserter"/>
     <xs:element ref="pr:content" maxOccurs="unbounded"/>
     </xs:sequence>
    </xs:complexType>


    <xs:complexType name="Content">
     <xs:choice>
     <xs:element ref="ps:interactionPAssertion"/>
     <xs:element ref="ps:actorStatePAssertion" />
     <xs:element ref="ps:relationshipPAssertion"/>
     <xs:element ref="ps:exposedInteractionMetaData"/>
     <xs:element name="submissionFinished" type="xs:int" />
     </xs:choice>
    </xs:complexType>

    <xs:complexType name="SynchAck"></xs:complexType>

</xs:schema>
```

B Process Documentation Record WSDL

Below we give the WSDL document for process documentation record and acknowledgement messages.

```
<?xml version="1.0"?>

<definitions name="PRecord"
                  targetNamespace="http://www.pasoa.org/schemas/version023s1/record/PRecord.wsdl"
                  xmlns:tns="http://www.pasoa.org/schemas/version023s1/record/PRecord.wsdl"
                  xmlns="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:pr="http://www.pasoa.org/schemas/version023s1/record/PRecord.xsd">

    <documentation>
        The Provenance Store recording port type and messages
        Author: Paul Groth
        Last Modified: Feb 1 2005

        Copyright (c) 2006 University of Southampton
        See pasoalicense.txt for license information.
        http://www.opensource.org/licenses/mit-license.php
    </documentation>

    <import namespace="http://www.pasoa.org/schemas/version023s1/record/PRecord.xsd"
      location="./PRecord.xsd" />


    <message name="Record">
        <part name="body" element="pr:record"/>
    </message>

    <message name="RecordAck">
        <part name="body" element="pr:recordAck"/>
    </message>

    <portType name="RecordPortType">
        <operation name="Record">
           <input message="tns:Record"/>
           <output message="tns:RecordAck"/>
        </operation>
    </portType>

</definitions>
```

References

[Bra97] Scott Bradner. Key words for use in RFCs to indicate requirement levels. http://www.ietf.org/rfc/rfc2119.txt, 1997.

[GJ+06] Paul Groth, Sheng Jiang, , Simon Miles, Steve Munroe, Victor Tan, Sofia Tsasakou, and Luc Moreau. An Architecture for Provenance Systems. Technical report, Electronics and Computer Science, University of Southampton, 2006.

[GLM04] 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.

[MGJ+06] Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, Victor Tan, and Luc Moreau. Data model for Process Documentation. Technical report, University of Southampton, June 2006.

[TGJ+06] Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, and Luc Moreau. WS Provenance Glossary. Technical report, Electronics and Computer Science, University of Southampton, 2006.

[TMG+06] Victor Tan, Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. The Provenance Standardisation Vision. Technical report, University of Southampton, June 2006.

[Var04] Various authors. Web Services Security. http://www-128.ibm.com/developerworks/library/specification/ws-secure/, 2004.

[Var05a] Various authors. Web Services Secure Conversation Language. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-secon/, 2005.

[Var05b] Various authors. Web Services Trust Language. http://www-128.ibm.com/developerworks/library/specification/ws-trust/, 2005.

[W3C99] W3C. XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999. http://www.w3.org/TR/xpath, 1999.

ws-prov-pquery

Authors:
Simon Miles, U. Southampton
Luc Moreau, U. Southampton
Paul Groth, U. Southampton
Victor Tan, U. Southampton
Steve Munroe, U. Southampton
Sheng Jiang, U. Southampton

November 23, 2006

Provenance Query Protocol

Status of this Memo

This document provides information to the community regarding the specification of a protocol for querying the provenance of data items from process documentation and has the status of a working draft. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

A related document [MGJ+06] defines schemas to be used for documentation about the execution of a process, *process documentation*. It also defines the provenance of a data item as the process that led to that item. A *provenance query* is a query for the provenance of a data item and the results of such a query is documentation of the process that led to the item. In this document, we specify a protocol by which a querying actor and provenance store can communicate in performing a provenance query. This primarily takes the form of an abstract WSDL interface defining messages to be accepted and produced by a provenance store.

Contents

1 Introduction

The amount of information making up the provenance of an entity may be vast. The details of everything that ultimately caused an entity to be as it is would, generally, be an unmanageable amount. For example, to give the full provenance of an experiment's results, we have to describe the process that produced the materials used in the experiment, the provenance of materials used in producing those materials, the devices and software used in the experiment and their settings, the origin of the experiment design etc. Ultimately, if enough information was available, we would include details of processes back to the beginning of time. Similarly, given enough information, we could give finer and finer grained information on the processes that led to an entity, e.g. not just documenting that a sample was tested to see if a chemical was present, but the procedure by which this is done, the molecular interactions that took place in the testing procedure and so on. All the information about the provenance of an entity is potentially useful for someone with a particular question about that entity, but providing it all in all cases would be counter-productive.

Instead, anyone requiring the provenance of an entity should be able to get it by expressing the request as a *query* and *scoping* that query so that only the information relevant to them is returned. We define a *provenance query engine* as the actor that processes a provenance query and returns the results.

This document defines the schema for a provenance query request, the behaviour expected in processing that request and the resulting response. The WSDL 1.1 description of an interface for a service accepting such requests and producing such responses is given in Appendix B.

1.1 Goals and Requirements

The goal of this document is to define the protocol for querying the provenance of data items from process documentation.

1.1.1 Requirements

In meeting this goal, this document must address the following requirements:

- Define the schema of the provenance query request message sent to the provenance query engine. This definition includes how the data item of which to find the provenance is identified, and how the scope of the query results are expressed.
- Define the behaviour of a provenance query engine on receiving a provenance query request.
- Define how provenance queries over distributed process documentation can be resolved using the links defined in a supporting document [MTG+06].

- Define the schema of the provenance query results message returned by the provenance query engine.

1.1.2 Non-Requirements

This document specifies a synchronous version of the query protocol. Other documents may specify asynchronous provenance querying.

2 Terminology and Notation

All definitions for the concepts and structures found within this document can be found in [TGJ+06].

2.1 XML Namespaces

The XML Namespace URI that MUST be used by implementations of this specification is: `http://www.pasoa.org/schemas/version023s1/xquery/XQuery.xsd`

Table 1 lists XML namespaces that are used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

Prefix	XML Namespace	Specification(s)
pq	`http://www.pasoa.org/schemas/version023s1/pquery/ProvenanceQuery.xsd`	[PQuery]
ps	`http://www.pasoa.org/schemas/version023s1/PStruct.xsd`	[PStruct]
xs	`http://www.w3.org/2001/XMLSchema`	[XMLSchema]

Table 1: Prefixes and XML Namespaces used in this specification

2.2 Notational Conventions

The keywords "MUST ", "MUSTNOT ", "REQUIRED ", "SHALL ", "SHALLNOT ", "SHOULD ", "SHOULDNOT ", "RECOMMENDED ", "MAY ", and "OPTIONAL " in this document are to be interpreted as described in [Bra97].

2.3 XML Schema Diagrams

This document adopts a graphical notation to describe XML Schema. Figure 1 gives an example of a small XML Schema displayed as a diagram, which is now explained with reference to the figure.

Figure 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

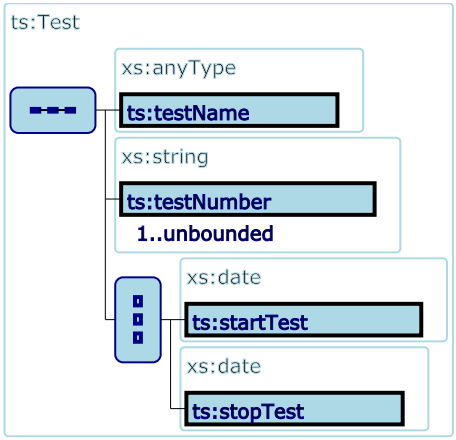


Figure 1: An example XML Schema diagram

`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:stopTest`, 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.

2.4 XPath notation

In addition to the XML Schema diagrams, an XPath notation [W3C99] is used to identify each individual element in the specification along with its context, in order to describe precisely its meaning and use.

3 Provenance Query

To execute a provenance query, a *provenance query request* is sent to a *provenance query engine* by a *querying actor* and a *provenance query result* is returned. We specify below the request document schema, provenance query engine behaviour and response document schema for a provenance query. The full schema document, in which request and response message structures are defined, is given in Appendix A. The WSDL 1.1 description of the interface taking and producing these messages is given in Appendix B.

4 Provenance Query Request

A provenance query request is a message sent by a querying actor to a provenance query engine to perform a query to find the provenance of a data item over the contents of that, and linked, stores. A provenance query request includes a *query data handle*, identifying the entity of which to find the provenance, and a *relationship target filter*, specifying the query's scope.

A provenance query request is instantiated as a document conforming to the schema depicted in Figure 2.

`/pq:provenanceQuery`

This element defines a provenance query request.

`/pq:provenanceQuery/pq:queryDataHandle`

This element defines a search over process documentation to find the record of an entity at a given event for which the querying actor wishes to find the provenance.

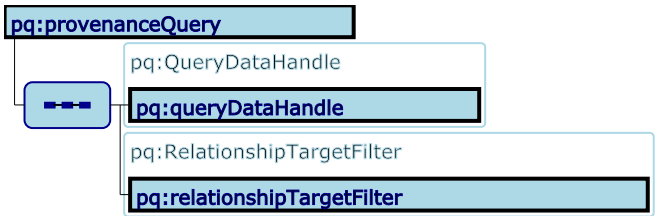


Figure 2: Provenance Query Request

`/pq:provenanceQuery/pq:relationshipTargetFilter`

This element contains the criteria by which the querying actor specifies whether any given entity in the documentation, and its relations, should be included in the query results.

4.1 Query Data Handle

When a querying actor asks for an entity's provenance, it identifies the entity such that a provenance query engine can find documentation of the entity. The identification is called a *query data handle*. For the actor, a query data handle identifies an entity at a given event. For the engine, a query data handle identifies a search for p-assertion data items in process documentation.

Not all p-assertion contents may be in the format required by the search expression. In this case, the querying actor can provide a set of *document language mappings*, which specify how to convert formats so that the search can take place. 0 mappings mean that all the p-assertion contents are expected to be in the language required by the search, e.g. XML. More than one mapping means that different p-assertion contents are mapped, e.g. one for CORBA messages, one for BLAST results etc.

A query data handle is instantiated as a document conforming to the schema depicted in Figure 3.

`/pq:queryDataHandle`

This element contains all details used to discover documentation of a data item.

`/pq:queryDataHandle/pq:search`

This element includes a definition of a search over process documentation to find the record of an entity at a given event for which the querying actor wishes to find the provenance. Different provenance query engines may support different types of search specification.

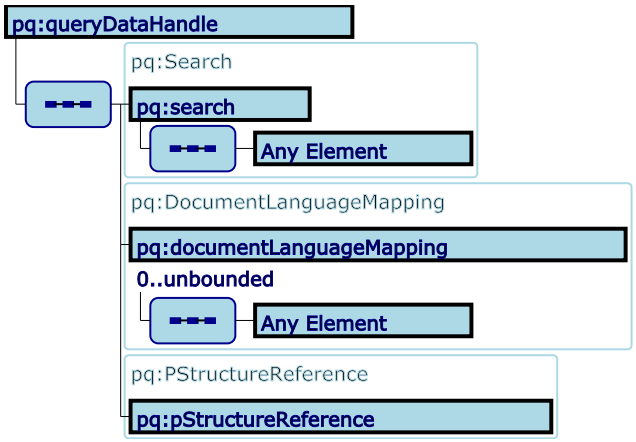


Figure 3: Query Data Handle

`/pq:queryDataHandle/pq:documentLanguageMapping`

This element includes a specification of how p-assertion contents are mapped to the document language required by the search, e.g. if p-assertion contents are comma-separated values, it could be mapped to an XML format for the search. The mappings required depend on the search language, and so different provenance query engines may support different types of search specification.

`/pq:queryDataHandle/pq:pStructureReference`

This element includes a definition of the exact set of process documentation over which the query data handle searches.

A `pq:pStructureReference`, which identifies the set of process documentation over which the query data handle search will be performed, is instantiated as a document conforming to the schema depicted in Figure 4.

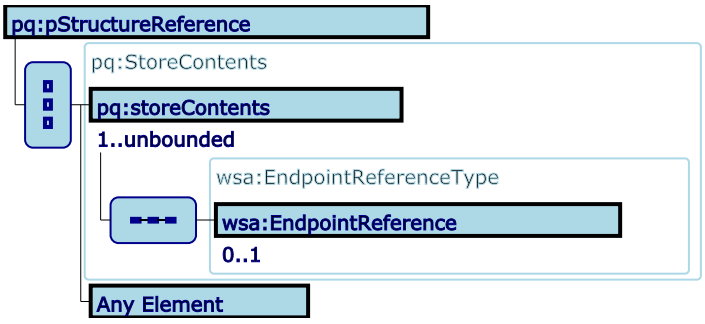


Figure 4: P-Structure Reference

`/pq:pStructureReference`

This element includes a definition of the exact set of process documentation over which the query data handle searches.

`/pq:pStructureReference/pq:storeContents`

The presence of this element is a statement that the query data handle search should be performed over the full contents of a provenance store. This element may occur multiple times, to specify that the search is over multiple stores. Where the element has no contents, this states that the search should be performed over the default store for the provenance query engine, usually the store that the engine is deployed as a component of.

`/pq:pStructureReference/pq:storeContents/wsa:EndpointReference`

If present, this element gives the address of a provenance store over which the search is conducted. This is a provenance store service port as defined in [MTG+06].

`/pq:pStructureReference/xs:any`

Where the storeContents element is not used, the set of process documentation over which the query data handle searches must be specified in another way. This element, if present, defines this search space in a way that the provenance query engine can interpret.

4.2 P-Assertion Data Key Query Data Handles

All provenance query engines can interpret query data handles of a particular form: a p-assertion data key as specified in the process documentation data model. This search will find at most one p-assertion data item, i.e. the one referred to by the key.

An example is shown below.

```
<pq:queryDataHandle>
  <pq:search>
    <ps:pAssertionDataKey>
      <ps:interactionKey>
        <ps:messageSource>...</ps:messageSource>
        <ps:messageSink>...</ps:messageSink>
        <ps:interactionId>...</ps:interactionId>
      </ps:interactionKey>
      <ps:viewKind xsi:type="ps:SenderViewKind"/>
      <ps:localPAssertionId>...</ps:localPAssertionId>
      <ps:dataAccessor>...</ps:dataAccessor>
    </ps:pAssertionDataKey>
  </pq:search>
  <pq:pStructureReference>
    <pq:storeContents/>
  </pq:pStructureReference>
<pq:queryDataHandle>
```

4.3 Relationship Target

A p-assertion data item is a part, or whole, of a p-assertion's content. 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 is used to determine whether a p-assertion data item is within scope for the provenance query results or not.

A relationship target is instantiated as a document conforming to the schema depicted in Figure 5.

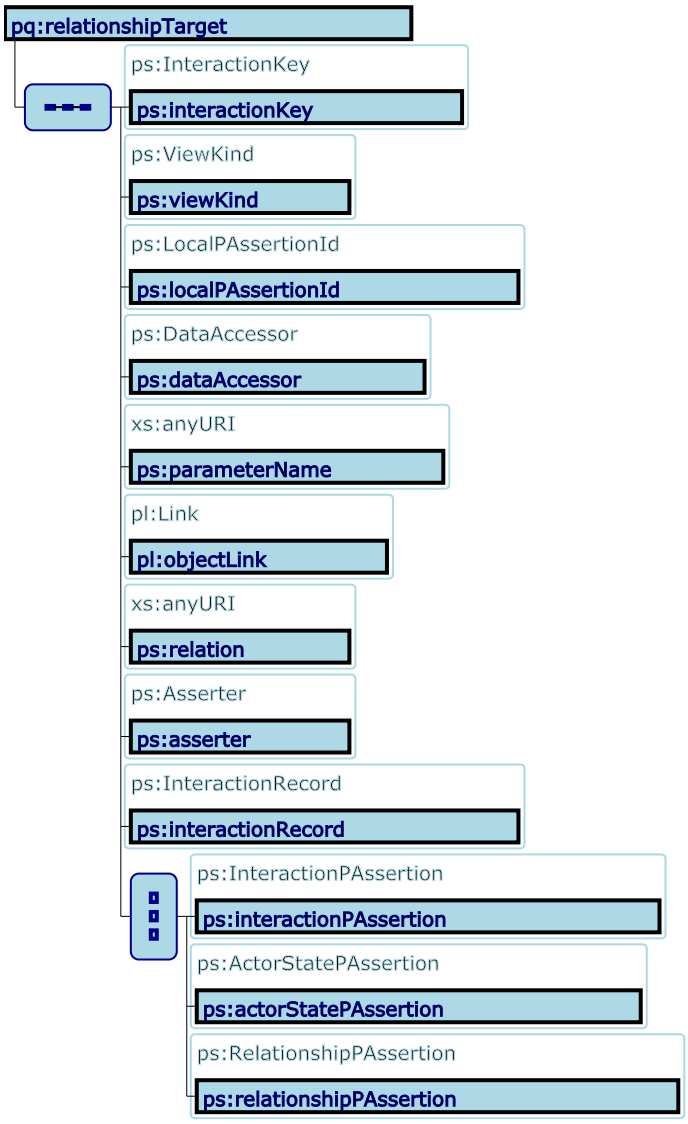


Figure 5: Relationship Target

`/pq:relationshipTarget`

This element contains all the information on one p-assertion data item.

`/pq:relationshipTarget/ps:interactonKey`

This is the interaction key of the interaction that the p-assertion containing the data item documents.

`/pq:relationshipTarget/ps:viewKind`

This is the view kind of the asserter of the p-assertion containing the data item.

`/pq:relationshipTarget/ps:localPAssertionId`

This is the local p-assertion ID of the p-assertion containing the data item.

`/pq:relationshipTarget/ps:dataAccessor`

This is the location of the data item within the content of its containing p-assertion.

`/pq:relationshipTarget/ps:parameterName`

This is the role played by the data item in the relationship of which this item is a target.

`/pq:relationshipTarget/pl:objectLink`

This is a link to the provenance store in which the p-assertion data item is documented.

`/pq:relationshipTarget/ps:relation`

This is the type of the relationship of which the data item is a target.

`/pq:relationshipTarget/ps:asserter`

This is the identity of the asserter of the p-assertion containing the data item.

`/pq:relationshipTarget/ps:interactionRecord`

This is the full set of p-assertions recorded about the interaction in which the data item is asserted.

`/pq:relationshipTarget/ps:interactionPAssertion`

If present, this is the interaction p-assertion in which the data item is contained.

`/pq:relationshipTarget/ps:actorStatePAssertion`

If present, this is the actor state p-assertion in which the data item is contained.

`/pq:relationshipTarget/ps:relationshipPAssertion`

If present, this is the relationship p-assertion which is itself the object of a relationship (and so the data item in this context).

4.4 Relationship Target Filter

The set of process documentation that may be returned in response to a provenance query request 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 query results. This is the purpose of the *relationship target filter*. A relationship target filter is used to filter each object of a relationship p-assertion found while evaluating the provenance query, and filters out those that are not in scope. To do this, the provenance query engine constructs a relationship target for each relationship object, which encapsulates all relevant data about the data item the object identifies. The relationship target filter then defines a function that operates over a relationship target and returns true or false depending on whether the relationship target is in scope or not.

A relationship target filter is instantiated as a document conforming to the schema depicted in Figure 6.

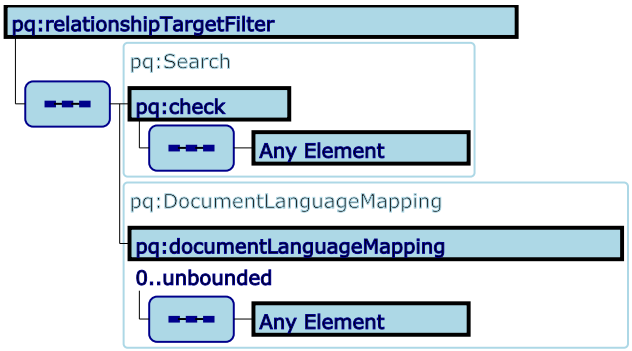


Figure 6: Relationship Target Filter

`/pq:relationshipTargetFilter`

This element includes a specification of the scope of a provenance query.

`/pq:relationshipTargetFilter/pq:check`

This element contains a definition for a function that operates on a relationship target that determines whether the target is in scope or not.

`/pq:relationshipTargetFilter/pq:check/xs:any`

This is the function definition. The form depends on what is supported by the provenance query engine, and may be suited to a particular type of process documentation.

`/pq:relationshipTargetFilter/pq:documentLanguageMapping`

This element includes a specification of how p-assertion contents are mapped to the document language required by the filter, e.g. if p-assertion contents are comma-separated values, it could be mapped to an XML format for the search. The mappings required depend on the search language, and so different provenance query engines may support different types of search specification.

5 Behaviour

A provenance query request is processed as follows.

1. The search for a data item expressed by the query data handle is performed on the process documentation contained in the given search space (`pStructureReference`). The result of this search is a set of p-assertion data items, addressed by p-assertion data keys.

2. For each relationship, asserted in the process documentation, of which one of those p-assertion data items is a subject, take each object of the relationship, express it as a relationship target, and execute the relationship target filter on it. As the provenance store containing the data item may not include both views of the interaction in which the data item was exchanged, the other view, referred to by a view link, may be retrieved by submitting process documentation and provenance queries to the linked store.

3. Where a relationship object is accepted by the relationship target filter, find the provenance of the p-assertion data item referred to by that object using the steps above and the same relationship target filter. As the object may be in another provenance store, specified by the object link, this may involve submitting process documentation and provenance queries to that store.

4. The final results of the query are comprised of two parts: the p-assertion data keys for every item discovered by the query data handle search; and, for every relationship object accepted by the relationship target filter, the relationship between that object and the subject of the relationship.

5.1 Data Accessors

The process documentation data model allows data accessors of any form to be used in the subjects and objects of relationship p-assertions. It is left to querying actors to correctly interpret these. However, a provenance query engine, following the above algorithm, must also be able to interpret them in order to

determine whether the object of one relationship is the same as the object of another relationship. The exact data accessors that a provenance query engine can understand is not restricted, but in order to use them the engine must be able to perform the following operations. For each operation, we state exactly why it is needed, i.e. what necessary operation would be impossible if the operation was not defined.

Get Accessor For Item Get the data accessor for a result of a query data handle search. Without this operation, an engine is unable to determine whether the subject of a relationship p-assertion refers to a data item found by the query data handle.

Test Accessor Equality Given the data accessor in the subject of a relationship and the data accessor in an object of a relationship, determine whether they refer to the same item. This may or may not involve navigating the p-assertion content. Without this operation, an engine cannot iteratively follow the relationships of each data item in the process.

6 Provenance Query Result

The final results of the query are comprised of two parts: the p-assertion data items from the query data handle search (the start data items); and, for every relationship object accepted by the filter, the relationship between that object and the subject in that relationship.

A provenance query result is instantiated as a document conforming to the schema depicted in Figure 7.

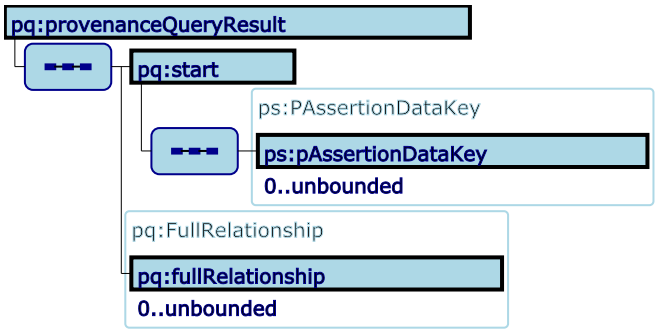


Figure 7: Provenance Query Result

`/pq:provenanceQueryResult/pq:start`

This element contains the p-assertion data keys for the items found by the query data handle search.

`/pq:provenanceQueryResult/pq:start/ps:pAssertionDataKey`

This is a p-assertion data key for an item found by the query data handle search.

`/pq:provenanceQueryResult/pq:fullRelationship`

This element contains the details of a relationship between two data items in the results of the provenance query.

6.1 Full Relationship

A *full relationship* is the full details of a relationship between two p-assertion data items. The information it contains is derived from a relationship p-assertion, but there are two differences. First, there is only a single object per full relationship, because each relationship object has been evaluated independently as to whether it is in scope for the provenance. Second, it is not contained within an interaction record, so the interaction key and view kind of the subject is included in a full relationship.

By matching a start p-assertion data key with those in the subjects and objects of full relationships, a graph of relationships describing the provenance of that start item can be determined.

A full relationship is instantiated as a document conforming to the schema depicted in Figure 8.

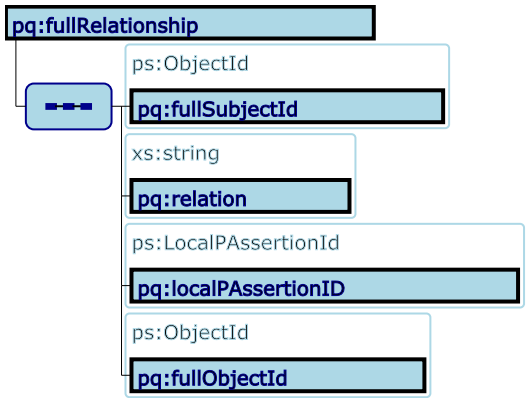


Figure 8: Full Relationship

`/pq:fullRelationship`

This element contains the details of a relationship between two p-assertion data items.

`/pq:fullRelationship/pq:fullSubjectId`

This is the subject of the relationship. It has exactly the same contents as the object ID of a relationship p-assertion as defined for the process documentation data model [MGJ+06].

`/pq:fullRelationship/ps:relation`

This is the type of relationship between subject and object.

`/pq:fullRelationship/ps:localPAssertionId`

This is the local p-assertion ID of the relationship p-assertion from which the full relationship is derived. The other identifiers of the p-assertion, its interaction key and view kind, are the same as in the fullSubjectId.

`/pq:fullRelationship/pq:fullObjectId`

This is the object of the relationship. It has exactly the same contents as the object ID of a relationship p-assertion as defined for the process documentation data model [MGJ+06].

6.2 Faults

Provenance query faults are exchanged by using extensions of the ProvenanceQueryFault type. A default element is specified for this type, and is shown in Figure 9.

Figure 9: Provenance Query Fault

`/pq:provenanceQueryFault`

This element, or extensions of it, represents a fault caused by the evaluation of a provenance query request.

7 Default Port Name

In order to aid addressing of provenance stores, we require that each interface that can be assumed to be present in every provenance store be given a default port context name [MTG+06], i.e. the last part of the URL addressing the port supporting that interface. For the provenance query interface, the default port name is `pquery`.

8 Security Considerations

This specification defines the process documentation query request and response messages for any provenance store supporting the process documentation query interface. In this context, there are two categories of security aspects that need to be considered: (a) securing the message exchanges and (b) securing the provenance store contents.

8.1 Securing Message Exchanges

When messages are exchanged between a querier and provenance store in a process documentation query, it is recommended that the communication be secured using the mechanisms described in WS-Security [Var04]. In order to properly secure messages, the message body (query expression or results) and all relevant headers need to be included in the digital signature so as to prove the integrity of the message. In the event that a querier frequently performs process documentation queries on a store it is recommended that a security context be established using the mechanisms described in WS-Trust [Var05b] and WS-SecureConversation [Var05a], allowing for potentially more efficient means of authentication.

8.2 Securing Provenance Store Contents

Since this specification defines a mechanism to retrieve the contents of provenance stores, security policies should be established that ensure that only authorized queriers can access the p-assertions.

9 Conclusions

The aim of a process documentation data model is, first among many requirements, to enable the provenance of items to be determined. In this document, we have presented the data models for expressing provenance queries and their results, and stated the behaviour that can be expected of an engine evaluating such a query.

A Provenance Query Schema

Below we give the full schema for provenance queries and their results.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.pasoa.org/schemas/version023s1/pquery/ProvenanceQuery.xsd"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:ps="http://www.pasoa.org/schemas/version023s1/PStruct.xsd"
           xmlns:pq="http://www.pasoa.org/schemas/version023s1/pquery/ProvenanceQuery.xsd"
           xmlns:pl="http://www.pasoa.org/schemas/version023s1/distribution/PLinks.xsd"
           xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">

     <xs:annotation>
   <xs:documentation>
         Author: Simon Miles
     Last Modified: 28 Feb 2006

                Copyright (c) 2006 University of Southampton
                See pasoalicense.txt for license information.
                http://www.opensource.org/licenses/mit-license.php

            </xs:documentation>
  </xs:annotation>


     <xs:import namespace="http://www.pasoa.org/schemas/version023s1/PStruct.xsd"
                schemaLocation="../PStruct.xsd"/>
     <xs:import namespace="http://www.pasoa.org/schemas/version023s1/distribution/PLinks.xsd"
                schemaLocation="../distribution/PLinks.xsd"/>
     <xs:import namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"
                schemaLocation="../wsaddressing.xsd"/>

  <xs:element name="provenanceQuery" type="pq:ProvenanceQuery"/>

  <xs:element name="provenanceQueryResult" type="pq:ProvenanceQueryResult"/>

  <xs:element name="provenanceQueryFault" type="pq:ProvenanceQueryFault"/>

  <xs:complexType name="ProvenanceQueryFault"/>

  <xs:complexType name="ProvenanceQuery">
    <xs:sequence>
      <xs:element ref="pq:queryDataHandle"/>
      <xs:element ref="pq:relationshipTargetFilter"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="ProvenanceQueryResult">
    <xs:sequence>
      <xs:element name="start">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="ps:pAssertionDataKey" minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element ref="pq:fullRelationship" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="queryDataHandle" type="pq:QueryDataHandle"/>
```

```
<xs:element name="documentLanguageMapping" type="pq:DocumentLanguageMapping"/>

<xs:complexType name="DocumentLanguageMapping">
  <xs:sequence>
    <xs:any/>
  </xs:sequence>
</xs:complexType>

<xs:element name="pStructureReference" type="pq:PStructureReference"/>

<xs:element name="storeContents" type="pq:StoreContents"/>

<xs:element name="relationshipTargetFilter" type="pq:RelationshipTargetFilter"/>

<xs:complexType name="PStructureReference">
  <xs:choice>
    <xs:sequence>
      <xs:element ref="pq:storeContents" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:any/>
  </xs:choice>
</xs:complexType>

<xs:complexType name="StoreContents">
  <xs:sequence>
    <xs:element ref="wsa:EndpointReference" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="QueryDataHandle">
  <xs:sequence>
    <xs:element name="search" type="pq:Search"/>
    <xs:element ref="pq:documentLanguageMapping" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="pq:pStructureReference"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="RelationshipTargetFilter">
  <xs:sequence>
    <xs:element name="check" type="pq:Search"/>
    <xs:element ref="pq:documentLanguageMapping" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="relationshipTarget" type="pq:RelationshipTarget"/>

<xs:complexType name="RelationshipTarget">
  <xs:sequence>
    <xs:element ref="ps:interactionKey"/>
    <xs:element ref="ps:viewKind"/>
    <xs:element ref="ps:localPAssertionId"/>
    <xs:element ref="ps:dataAccessor"/>
    <xs:element ref="ps:parameterName"/>
    <xs:element ref="pl:objectLink"/>
    <xs:element ref="ps:relation"/>
    <xs:element ref="ps:asserter"/>
    <xs:element ref="ps:interactionRecord"/>
    <xs:choice>
      <xs:element ref="ps:interactionPAssertion"/>
      <xs:element ref="ps:actorStatePAssertion"/>
      <xs:element ref="ps:relationshipPAssertion"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

```
  <xs:element name="fullSubjectId" type="ps:ObjectId"/>

  <xs:element name="fullObjectId" type="ps:ObjectId"/>

  <xs:complexType name="FullRelationship">
    <xs:sequence>
      <xs:element ref="pq:fullSubjectId"/>
      <xs:element ref="ps:relation"/>
      <xs:element ref="ps:localPAssertionID"/>
      <xs:element ref="pq:fullObjectId"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="fullRelationship" type="pq:FullRelationship"/>

  <xs:complexType name="Search">
    <xs:sequence>
      <xs:any/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

B Provenance Query WSDL

Below we give the WSDL document for provenance query and response messages.

```
<?xml version="1.0"?>
<definitions name="PQuery"
             targetNamespace="http://www.pasoa.org/schemas/version023s1/pquery/PQuery.wsdl"
             xmlns:tns="http://www.pasoa.org/schemas/version023s1/pquery/PQuery.wsdl"
             xmlns:pq="http://www.pasoa.org/schemas/version023s1/pquery/ProvenanceQuery.xsd"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

    <documentation>
        The Provenance Store provenance query port type and messages
        Author: Simon Miles
        Last Modified: 28 Feb 2006

        Copyright (c) 2006 University of Southampton
        See pasoalicense.txt for license information.
        http://www.opensource.org/licenses/mit-license.php
    </documentation>

    <import namespace = "http://www.pasoa.org/schemas/version023s1/pquery/ProvenanceQuery.xsd"
            location  = "./ProvenanceQuery.xsd"/>

<!-- Defines the Provenance Query Port type which offers one operation:
           the ProvenanceQuery operation. This operation takes in a ProvenanceQuery message defined
           by the pq:provenanceQuery element in ProvenanceQuery.xsd. The operation returns an output
           message ProvenanceQueryResult defined by the pq:provenanceQueryResult element in ProvenanceQuery.xsd  -->

    <portType name = "PQueryPortType">
        <operation name = "ProvenanceQuery">
           <input message = "tns:ProvenanceQuery"/>
           <output message = "tns:ProvenanceQueryResult"/>
           <fault message = "tns:ProvenanceQueryFault"/>
        </operation>
    </portType>

    <message name = "ProvenanceQuery">
        <part name = "body" element = "pq:provenanceQuery"/>
    </message>

    <message name = "ProvenanceQueryResult">
        <part name = "body" element = "pq:provenanceQueryResult"/>
    </message>

    <message name = "ProvenanceQueryFault">
        <part name = "body" element = "pq:provenanceQueryFault"/>
    </message>
</definitions>
```

References

[Bra97] Scott Bradner. Key words for use in RFCs to indicate requirement levels. http://www.ietf.org/rfc/rfc2119.txt, 1997.

[MGJ+06] Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, Victor Tan, and Luc Moreau. Data model for Process Documentation. Technical report, University of Southampton, June 2006.

[MTG+06] Steve Munroe, Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. WSRF Data Model Profile for Distributed Provenance. Technical report, University of Southampton, June 2006.

[TGJ+06] Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, and Luc Moreau. WS Provenance Glossary. Technical report, Electronics and Computer Science, University of Southampton, 2006.

[Var04] Various authors. Web Services Security. http://www-128.ibm.com/developerworks/library/specification/ws-secure/, 2004.

[Var05a] Various authors. Web Services Secure Conversation Language. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-secon/, 2005.

[Var05b] Various authors. Web Services Trust Language. http://www-128.ibm.com/developerworks/library/specification/ws-trust/, 2005.

[W3C99] W3C. XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999. http://www.w3.org/TR/xpath, 1999.

ws-prov-xquery

Authors:
Simon Miles, U. Southampton
Steve Munroe, U. Southampton
Paul Groth, U. Southampton
Sheng Jiang, U. Southampton
Victor Tan, U. Southampton
John Ibbotson, IBM
Luc Moreau, U. Southampton

August 29, 2006

Process Documentation Query Protocol

Status of this Memo

This document provides information to the community regarding the specification of a protocol for querying the process documentation contained in a provenance store and has the status of a working draft. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

A related document [MGJ+06] defines schemas to be used for documentation about the execution of a process, *process documentation*, and introduce a *provenance store*, a type of Web Service with the capability for storing and giving access to process documentation. In particular, process documentation has a defined schema, the *p-structure*, which clients of a provenance store can navigate in queries to extract particular pieces of process documentation. In this document, we specify a protocol by which a querying actor and provenance store can communicate in performing a process documentation query. This primarily takes the form of an abstract WSDL interface defining messages to be accepted and produced by a provenance store.

Contents

1 Introduction

Every provenance store supplies a Web Service interface for querying process documentation by navigating the p-structure. It has a single operation, *query*, that takes an XQuery expression as input and returns a set of XML documents, whose schemas are dependent on the XQuery, as result. This document defines the schema for the request and response messages. The WSDL 1.1 description of the interface is given in Appendix B.

1.1 Goals and Requirements

The goal of this document is to define the protocol for querying process documentation contained in a provenance store using XQuery.

1.1.1 Requirements

In meeting this goal, this document must address the following requirements:

- Define the schema of the query request message sent to the provenance store.
- Give the additional requirements on the XQuery expression for navigating process documentation.
- Define the behaviour of a provenance store on receiving a query request.
- Define the schema of the query response message returned by the provenance store.

1.1.2 Non-Requirements

This document specifies a synchronous version of the query protocol. Other documents may specify asynchronous querying.

2 Terminology and Notation

All definitions for the concepts and structures found within this document can be found in [TGJ+06].

2.1 XML Namespaces

The XML Namespace URI that MUST be used by implementations of this specification is: `http://www.pasoa.org/schemas/version023s1/xquery/XQuery.xsd`

Table 1 lists XML namespaces that are used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

Prefix	XML Namespace	Specification(s)
xq	`http://www.pasoa.org/schemas/version023s1/xquery/XQuery.xsd`	[PDXQuery]
xs	`http://www.w3.org/2001/XMLSchema`	[XMLSchema]

Table 1: Prefixes and XML Namespaces used in this specification

2.2 Notational Conventions

The keywords "MUST ", "MUSTNOT ", "REQUIRED ", "SHALL ", "SHALLNOT ", "SHOULD ", "SHOULDNOT ", "RECOMMENDED ", "MAY ", and "OPTIONAL " in this document are to be interpreted as described in [Bra97].

2.3 XML Schema Diagrams

This document adopts a graphical notation to describe XML Schema. Figure 1 gives an example of a small XML Schema displayed as a diagram, which is now explained with reference to the figure.

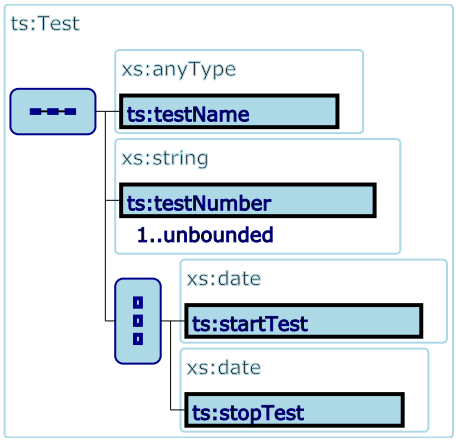


Figure 1: An example XML Schema diagram

Figure 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:stopTest`, 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.

2.4 XPath notation

In addition to the XML Schema diagrams, an XPath notation [W3C99] is used to identify each individual element in the specification along with its context, in order to describe precisely its meaning and use.

3 Process Documentation Query

We specify below the request document schema, query expression form, provenance store behaviour and response document schema for a process documentation query. The full schema document, in which request and response message structures are defined, is given in Appendix A. The WSDL 1.1 description of the interface taking and producing these messages is given in Appendix B.

3.1 Request

A process documentation query request is a message sent by a *querying actor* to a *provenance store* to perform an XQuery over the contents of the store. It is instantiated as a document conforming to the schema depicted in Figure 2.

Figure 2: Process Documentation Query Request

The document contents are described in detail as follows.

`/xq:query`

This element represents a request for a provenance store to perform an XQuery operation over its contents, and return the results.

`/xq:query/xq:xquery`

This element contains the XQuery expression itself, conforming to the details given in the next section.

3.2 Query Expressions

The query operation treats the contents of a provenance store as a single XML document conforming to the p-structure XML Schema. The expression should follow the XQuery 1.0 specification [BCF+06], but with the following caveats.

First, the result of evaluating the expression should be one or more XML documents, i.e. not just string, integer or other literal values. This is to allow the results to be returned unambiguously in a response document.

Second, as an XQuery expression can be performed over multiple documents or document nodes, queriers need to specify which document they wish to query. By default all provenance stores implementing the process documentation query interface must recognise the pre-bound variable $ns:pstruct in an XQuery expression, where ns is any prefix bound to the namespace `http://www.pasoa.org/schemas/version023s1/PStruct.xsd`. This variable should be bound to a document node that follows the schema of the *pstruct* element defined in the p-structure data model [MGJ+06], with the expectation that the contents of the document is a p-structure containing the full contents of the provenance store. Particular implementations may provide bindings to other documents where applicable.

Specific provenance-store implementation may provide built-in XQuery functions for common or advanced traversal of the store contents.

3.2.1 Examples

The following example returns the whole provenance store contents as a p-structure.

```
declare namespace ps = "http://www.pasoa.org/schemas/version023s1/PStruct.xsd";

$ps:pstruct
```

The following example returns an HTML unordered list summarising the relationship p-assertions in a store. The format of each item is *subject relation object1 object2...* The subjects and objects are identified by the interaction ID of the interaction record in which the data item is contained.

```
declare namespace ps = "http://www.pasoa.org/schemas/version023s1/PStruct.xsd";

<UL> {
  for $r in $ps:pstruct//ps:relationshipPAssertion
  return <LI> {
    (data($r/../../ps:interactionKey/ps:interactionId),
     ' ', data($r/ps:relation), ' ')}
    {for $o in $r/ps:objectId return (data($o/ps:interactionKey/ps:interactionId), ' ')
  }</LI>
}</UL>
```

3.3 Behaviour

On receiving a process documentation query request, a provenance store is expected to evaluate the query and synchronously return the result in a response message. The result of the query is exactly that document returned by the XQuery submitted.

3.4 Response

A process documentation query response is sent by a provenance store to the querying actor that issued the corresponding request. It is instantiated as a document conforming to the schema depicted in Figure 3.

Figure 3: Process Documentation Query Response

The document contents are described in detail as follows.

`/xq:queryResult`

This element represents the results of an XQuery.

`/xq:queryResult/xs:any`

Each child element contains a document fragment produced by evaluating the request's XQuery expression.

3.5 Faults

Process documentation query faults are exchanged by using extensions of the XQueryFault type, the default element for which is shown in Figure 4.

Figure 4: XQuery Fault

`/xq:queryFault`

This element, or extensions of it, represents a fault caused by the evaluation of a process documentation query request.

4 Default Port Name

In order to aid addressing of provenance stores, we require that each interface that can be assumed to be present in every provenance store be given a default port context name [MTG+06], i.e. the last part of the URL addressing the port supporting that interface. For the process documentation query interface, the default port name is `xquery`.

5 Security Considerations

This specification defines the process documentation query request and response messages for any provenance store supporting the process documentation query interface. In this context, there are two categories of security aspects that need to be considered: (a) securing the message exchanges and (b) securing the provenance store contents.

5.1 Securing Message Exchanges

When messages are exchanged between a querier and provenance store in a process documentation query, it is recommended that the communication be secured using the mechanisms described in WS-Security [Var04]. In order to properly secure messages, the message body (query expression or results) and all relevant headers need to be included in the digital signature so as to prove the integrity of the message. In the event that a querier frequently performs process documentation queries on a store it is recommended that a security context be established using the mechanisms described in WS-Trust [Var05b] and WS-SecureConversation [Var05a], allowing for potentially more efficient means of authentication.

5.2 Securing Provenance Store Contents

Since this specification defines a mechanism to retrieve the contents of provenance stores, security policies should be established that ensure that only authorized queriers can access the p-assertions.

6 Conclusions

Querying the details of a past process, as documented in process documentation, requires a query language and a means by which queries expressed in that language can be exchanged with provenance stores containing the documentation. This document describes a protocol, defined in an interface description, for querying process documentation.

A Process Documentation Query Schema

Below we give the full schema for process documentation queries.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.pasoa.org/schemas/version023s1/xquery/XQuery.xsd"
  xmlns:xq="http://www.pasoa.org/schemas/version023s1/xquery/XQuery.xsd"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.pasoa.org/schemas/version023s1/xquery/XQuery.xsd"
  elementFormDefault="qualified" attributeFormDefault="unqualified">

  <xs:annotation>
    <xs:documentation>
      An XQuery interface for Web Services
      Author: Simon Miles
      Last Modified: 28 Feb 2006
      Copyright (c) 2006 University of Southampton
      See pasoalicense.txt for license information.
      http://www.opensource.org/licenses/mit-license.php
    </xs:documentation>
  </xs:annotation>

  <xs:element name="query" type="Query">
    <xs:annotation>
      <xs:documentation>Query operation</xs:documentation>
    </xs:annotation>
  </xs:element>

  <xs:complexType name="Query">
    <xs:annotation>
      <xs:documentation>Query operation type</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="xquery" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="queryResult" type="QueryResult">
    <xs:annotation>
      <xs:documentation>Query results</xs:documentation>
    </xs:annotation>
  </xs:element>

  <xs:complexType name="QueryResult">
    <xs:annotation>
      <xs:documentation>Query results type</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="queryFault" type="QueryFault">
    <xs:annotation>
      <xs:documentation>Query fault</xs:documentation>
    </xs:annotation>
  </xs:element>

  <xs:complexType name="QueryFault">
    <xs:annotation>
      <xs:documentation>Fault in XQuery evaluation</xs:documentation>
    </xs:annotation>
  </xs:complexType>
</xs:schema>
```

B Process Documentation Query WSDL

Below we give the WSDL document for process documentation query and response messages.

```
<?xml version="1.0"?>
<definitions name="XQuery"
  targetNamespace="http://www.pasoa.org/schemas/version023s1/xquery/XQuery.wsdl"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://www.pasoa.org/schemas/version023s1/xquery/XQuery.wsdl"
  xmlns:xq="http://www.pasoa.org/schemas/version023s1/xquery/XQuery.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns="http://schemas.xmlsoap.org/wsdl/">
  <documentation>
    An XQuery interface for Web Services
    Author: Simon Miles
    Last Modified: 28 Feb 2006
    Copyright (c) 2006 University of Southampton
    See pasoalicense.txt for license information.
    http://www.opensource.org/licenses/mit-license.php
  </documentation>
  <import namespace="http://www.pasoa.org/schemas/version023s1/xquery/XQuery.xsd" location="./XQuery.xsd"/>
  <message name="Query">
    <part name="body" element="xq:query"/>
  </message>
  <message name="QueryAck">
    <part name="body" element="xq:queryAck"/>
  </message>
  <portType name="XQueryPortType">
    <operation name="Query">
      <input message="tns:Query"/>
      <output message="tns:QueryAck"/>
    </operation>
  </portType>
</definitions>
```

References

[BCF+06] Scott Boag, Don Chamberlin, Mary F. Fernandez, Daniela Florescu, Jonathan Robie, and Jerome Simeon. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery, 2006.

[Bra97] S. Bradner. Key words for use in RFCs to indicate requirement levels. http://www.ietf.org/rfc/rfc2119.txt, 1997.

[MGJ+06] Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, Victor Tan, and Luc Moreau. Data model for Process Documentation. Technical report, University of Southampton, June 2006.

[MTG+06] Steve Munroe, Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. WSRF Data Model Profile for Distributed Provenance. Technical report, University of Southampton, June 2006.

[TGJ+06] Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, and Luc Moreau. WS Provenance Glossary. Technical report, Electronics and Computer Science, University of Southampton, 2006.

[Var04] Various authors. Web Services Security. http://www-128.ibm.com/developerworks/library/specification/ws-secure/, 2004.

[Var05a] Various authors. Web Services Secure Conversation Language. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-secon/, 2005.

[Var05b] Various authors. Web Services Trust Language. http://www-128.ibm.com/developerworks/library/specification/ws-trust/, 2005.

[W3C99] W3C. XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999. http://www.w3.org/TR/xpath, 1999.

ws-prov-sign

Authors:
Victor Tan, U. Southampton
Simon Miles, U. Southampton
Steve Munroe, U. Southampton
Paul Groth, U. Southampton
Sheng Jiang, U. Southampton
John Ibbotson, IBM
Luc Moreau, U. Southampton

August 30, 2006

A Profile for Non-Repudiable Process Documentation

Status of this Memo

This document provides information to the community regarding a profile for creating process documentation that is non-repudiable through the use of signatures, and has the status of a working draft. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

The data model for process documentation [MGJ+06] describes p-assertions as individual units for documenting process. These p-assertions can be cryptographically signed by asserting actors in order to establish accountability for their creation. This document extends on the data model for the basic p-assertions (relationship, actor-state and interaction) to include support for signatures.

Contents

1 Introduction

The data model for process documentation [MGJ+06] provides descriptions of different ways in which processes may be documented. Individual items of process documentation, p-assertions, are created by asserting actors with unique identities. Signing these p-assertions links their identities to these p-assertions and establishes accountability, and subsequent liability, for their creation. This document shows how the data model and schema for p-assertions is modified to include a Signature element, and how this element is used in establishing non-repudiation.

1.1 Goals and Requirements

The goal of this document is to provide an extended data model for p-assertions which permits support for the use of signatures.

1.1.1 Requirements

In meeting this goal, this document must address the following requirements:

- Explain how signatures in the extended data model can be used for non-repudiation

1.1.2 Non-Requirements

This document does not intend to meet the following requirements:

- Explain alternative mechanisms (other than signatures) to achieve non-repudiation.
- Prescribe measures to be undertaken if p-assertions are not signed when they are expected to be, or if the signature on a p-assertion is determined to be invalid.
- Propose a technology specific approach to implementing signatures within the data model.

2 Terminology and Notation

All definitions for the concepts and structures found within this document can be found in [TGJ+06].

2.1 XML Namespaces

The XML Namespace URI that MUST be used by implementations of this specification is: `http://www.pasoa.org/schemas/version023s1/PStruct.xsd`

Table 1 lists XML namespaces that are used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

Prefix	XML Namespace	Specification(s)
ps	`http://www.pasoa.org/schemas/version023s1/PStruct.xsd`	[P-Structure]
wsa	`http://schemas.xmlsoap.org/ws/2004/08/addressing`	[WS-Addressing]
xs	`http://www.w3.org/2001/XMLSchema`	[XMLSchema]

Table 1: Prefixes and XML Namespaces used in this specification

2.2 Notational Conventions

The keywords "MUST ", "MUSTNOT ", "REQUIRED ", "SHALL ", "SHALLNOT ", "SHOULD ", "SHOULDNOT ", "RECOMMENDED ", "MAY ", and "OPTIONAL " in this document are to be interpreted as described in [Bra97].

2.3 XML Schema Diagrams

This documents adopts a graphical notation to describe XML Schema. Figure 1 gives an example of a small XML Schema displayed as a diagram, which is now explained with reference to the figure.

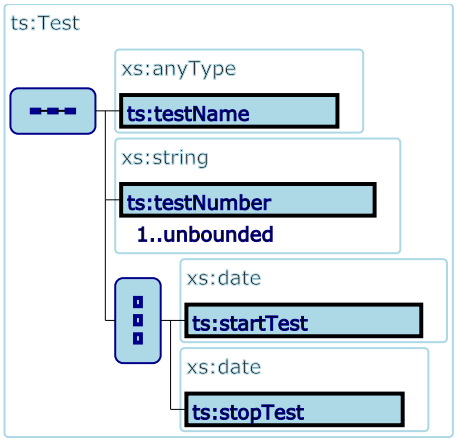


Figure 1: An example XML Schema diagram

Figure 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:stopTest, 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.

2.4 XPath notation

In addition to the XML Schema diagrams, an XPath notation [W3C99] is used to identify each individual element in the specification along with its context, in order to describe precisely its meaning and use.

3 Signatures in P-Assertions

The provenance of a data item can only be determined through the analysis of its related process documentation. However, process documentation in the form of p-assertions created by an actor only represents that actor's subjective view of events in the process it is documenting. It is therefore important that actors be held accountable for p-assertions they create, since p-assertions with erroneous information will affect other information processing functions. If such accountability is desirable in a specific provenance aware application, it can be made compulsory for actors to sign p-assertions they create. This establishes a direct link between the signed p-assertions and an actor's unique identity, and makes it impossible for it to deny having created a specific p-assertion (the property of non-repudiation).

The use of signatures assumes that actors operate within an environment that supports the use of a public key infrastructure through which certificates are distributed. The verification of a signature on a p-assertion can be achieved by the repository to which p-assertions are submitted for storage, or by querying actors that retrieve p-assertions from this repository. The use of signatures does not guarantee the truth (or otherwise) of the contents of a p-assertion, it only establishes the identity of the actor involved in its creation. In an application environment where accountability is vital, a provenance system that stores or processes p-assertions may be configured to reject p-assertions that are unsigned or whose signatures are invalid.

The determination of how accurately a p-assertion describes an actual event in a past process is unrelated to the use of signatures and has to be achieved through alternative means. This may, for example, involve comparing p-assertions relating to the same message or actor state made by asserting actors with different levels of trustworthiness.

3.1 Signed Interaction P-Assertion

The model for a signed interaction p-assertion is shown in Figure 2.

The basic addition to the data model for a normal interaction p-assertion described in Section 3.3 in [MGJ+06] is the `ps:Signature` element. This contains the signature of the asserting actor in a manner appropriate to the representational format of the p-assertion. If the interaction p-assertion is structured as XML, signing is performed using the XML Signature Recommendation [BBF+02], where the `ds:Signature` element from the XML Signature schema (`http://www.w3.org/TR/xmldsig-core/ xmldsig-core-schema.xsd`) is used directly in place of the `ps:Signature` element.

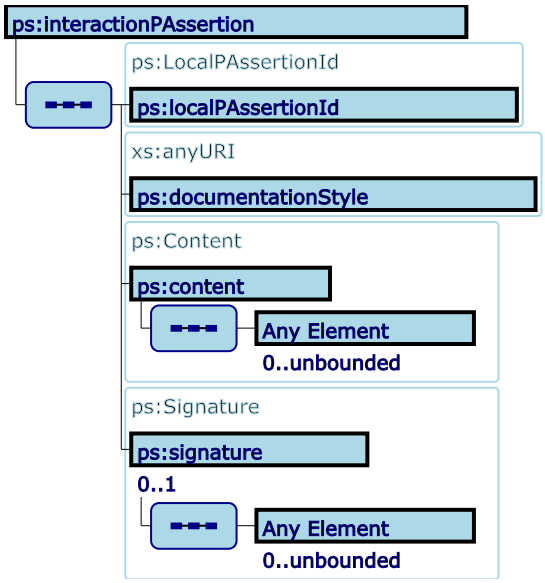


Figure 2: Signed Interaction P-Assertion

3.2 Signed Actor State P-Assertion

The model for a signed actor state p-assertion is shown in Figure 3.

The basic addition to the data model for a normal actor state p-assertion described in Section 3.5 in [MGJ+06] is the `ps:Signature` element. This contains the signature of the asserting actor in a manner appropriate to the representational format of the p-assertion. If the actor state p-assertion is structured as XML, signing is performed using the XML Signature Recommendation [BBF+02], where the `ds:Signature` element from the XML Signature schema (`http://www.w3.org/TR/xmldsig-core/ xmldsig-core-schema.xsd`) is used directly in place of the `ps:Signature` element.

3.3 Signed Relationship P-Assertion

The model for a signed relationship p-assertion is shown in Figure 4.

The basic addition to the data model for a normal relationship p-assertion described in Section 3.6 in [MGJ+06] is the `ps:Signature` element. This contains the signature of the asserting actor in a manner appropriate to the representational format of the p-assertion. If the relationship p-assertion is structured as XML, signing is performed using the XML Signature Recommendation [BBF+02], where the `ds:Signature` element from the XML Signature schema (`http://www.w3.org/TR/xmldsig-core/ xmldsig-core-schema.xsd`) is used directly in place of the `ps:Signature` element.

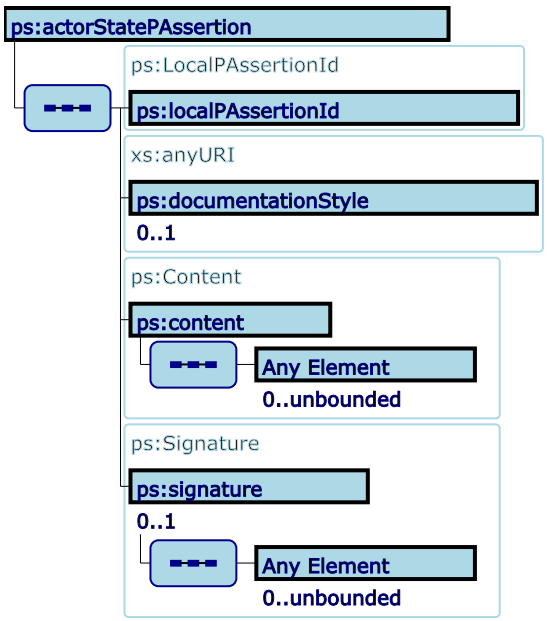


Figure 3: Signed Actor State P-Assertion

4 Conclusion

In this document, we have described how the data model and schema for process documentation [MGJ+06] can be extended in order to accomodate the inclusion of a Signature element. The inclusion of this element allows p-assertions to be signed, which links the identities of the asserting actors to these p-assertions and ensures that these actors cannot subsequently deny accountability for their contents (the property of non-repudiation). This is important as process documentation in the form of p-assertions created by an actor merely represents that actor's subjective view of events in the process it is documenting.

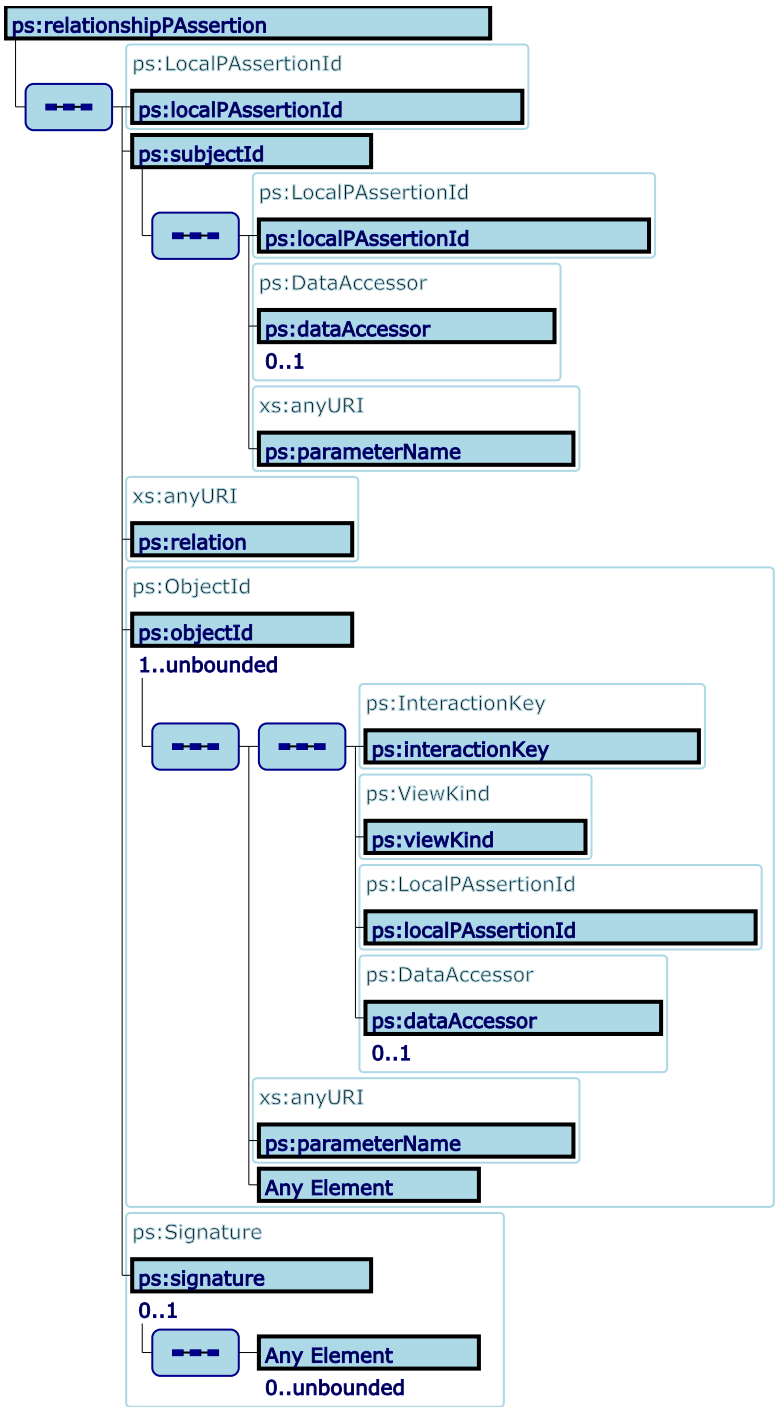


Figure 4: Signed Relationship P-Assertion

A Schema for Signed Process Documentation

Below we give the full schema for the data model of process documentation that includes signatures for interaction, relationship and actor state p-assertions.

```
 <?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:ps="http://www.pasoa.org/schemas/version023s1/PStruct.xsd"
                     xmlns:xs="http://www.w3.org/2001/XMLSchema"
                     xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
                     attributeFormDefault="unqualified" elementFormDefault="qualified"
                     targetNamespace="http://www.pasoa.org/schemas/version023s1/PStruct.xsd">

       <xs:import
        namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"
        schemaLocation="./wsaddressing.xsd"/>

       <xs:annotation>
        <xs:documentation>
            The P-Structure. This is a logical view of the contents of a provenance store.
            The P-Structure contains a set of interaction records that document interactions
            between actors.

            Author: Paul Groth


                  Copyright (c) 2006 University of Southampton
                  See pasoalicense.txt for license information.
                  http://www.opensource.org/licenses/mit-license.php

              </xs:documentation>
    </xs:annotation>

      <!-- We define the global elements of the p-struture here so that
         They can be referenced by external schemas. Below we define
         the types of the p-structure. The prefix ps: refers to this
         document. -->
    <xs:element name="pstruct" type="ps:PStructure">
        <xs:annotation>
            <xs:documentation>
                (Root Element Start Here) An instance of the
                p-structure. Each instance of the p-structure contains a
                set of interaction records.
            </xs:documentation>
        </xs:annotation>
    </xs:element>

    <xs:element name="interactionRecord" type="ps:InteractionRecord">
        <xs:annotation>
            <xs:documentation>
                An interaction record describes the client and service
                view of a particular interaction. An interaction record
                is identified by an interaction key.
            </xs:documentation>
        </xs:annotation>
    </xs:element>

    <xs:element name="view" type="ps:View">
        <xs:annotation>
            <xs:documentation>
                     A view of an interaction by an actor.
            </xs:documentation>
        </xs:annotation>
```

```
</xs:element>

<xs:element name="interactionKey" type="ps:InteractionKey">
    <xs:annotation>
        <xs:documentation>
            An interaction key contains a message source,
            a message sink, and an interaction id.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="messageSource" type="wsa:EndpointReferenceType">
    <xs:annotation>
        <xs:documentation>
            The source of the message within the sender.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="messageSink" type="wsa:EndpointReferenceType">
    <xs:annotation>
        <xs:documentation>
            The sink of the message within the receiver.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="interactionId" type="xs:anyURI">
    <xs:annotation>
        <xs:documentation>
            A URI that uniquely identifies this interaction .
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="asserter" type="ps:Asserter">
    <xs:annotation>
        <xs:documentation>
            Each view (either client or service) comes from a
            particular actor.  The actor that asserts p-assertion
            in a particular view is termed the asserter. The identity
            of the asserter is documented in the corresponding view inside
            the interaction record.
        </xs:documentation>
    </xs:annotation>
</xs:element>

 <xs:element name="numberOfExpectedAssertions" type="ps:NumberOfExpectedAssertions">
    <xs:annotation>
        <xs:documentation>
            The number of expected p-assertions to be contained
            within a view as documented by the asserting actor.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<!-- The following elements define the three types of p-assertions. -->
<xs:element name="interactionPAssertion" type="ps:InteractionPAssertion">
    <xs:annotation>
        <xs:documentation>
            Assertion as to the content of an interaction.
        </xs:documentation>
    </xs:annotation>
</xs:element>
```

```
<xs:element name="actorStatePAssertion" type="ps:ActorStatePAssertion">
    <xs:annotation>
        <xs:documentation>
            Information supplied by an actor about its state in the
            context of this interaction . Examples include the
            script that was used in running a service or the time
            when an invocation was sent/received.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="relationshipPAssertion" type="ps:RelationshipPAssertion">
    <xs:annotation>
        <xs:documentation>
            Describes a relationship between a p-assertion recorded
            in this view and another p-assertion made by the
            asserting actor. This can be seen as a triple: subject
            identifier, relation, object identifier.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<!-- End P-assertion defintions -->

<xs:element name="viewKind" type="ps:ViewKind">
    <xs:annotation>
        <xs:documentation>
            Whether a view is from the sender or receiver.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="localPAssertionId" type="ps:LocalPAssertionId">
    <xs:annotation>
        <xs:documentation>
            Uniquely identifies a p-assertion within a view.
        </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="dataAccessor" type="ps:DataAccessor">
    <xs:annotation>
        <xs:documentation>
            An application dependent mechanism for referencing a
            piece of data within a p-assertion.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="parameterName" type="xs:anyURI">
    <xs:annotation>
        <xs:documentation>
            The parameter name of a data item referenced
            in a relationship p-assertion.
        </xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="documentationStyle" type="xs:anyURI">
    <xs:annotation>
        <xs:documentation>
            The style of documentation used when recording
            an interaction p-assertion.
        </xs:documentation>
    </xs:annotation>
</xs:element>
```

```
<xs:element name="pAssertionDataKey" type="ps:PAssertionDataKey"/>

<xs:element name="objectId" type="ps:ObjectId"/>

<xs:element name="relation" type="xs:anyURI"/>

<xs:element name="globalPAssertionKey" type="ps:GlobalPAssertionKey"/>

<xs:element name="interactionMetaData" type="ps:InteractionMetaData"/>

<xs:element name="interactionContext" type="ps:InteractionContext"/>

<xs:element name="senderViewKind" type="ps:SenderViewKind"/>
<xs:element name="exposedInteractionMetaData" type="ps:ExposedInteractionMetaData"/>

<!-- Type Definitions -->
<xs:complexType name="InteractionKey">
    <xs:sequence>
        <xs:element ref="ps:messageSource"/>
        <xs:element ref="ps:messageSink"/>
        <xs:element ref="ps:interactionId"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="PStructure">
    <xs:sequence>
        <xs:element maxOccurs="unbounded" minOccurs="0" ref="ps:interactionRecord"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="InteractionRecord">
    <xs:sequence>
        <xs:element ref="ps:interactionKey" />
        <xs:element minOccurs="0" name="sender" type="ps:View">
            <xs:annotation>
                <xs:documentation>
                    The senders's view of the interaction .
                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element minOccurs="0" name="receiver" type="ps:View">
            <xs:annotation>
                <xs:documentation>
                    The receiver's view of the interaction.
                    WARNING!!! In future revisions the receiver view
                    may not be allowed to include relationship
                    p-assertions. If you have an example of the
                    usage of relationship p-assertions in this view,
                    please contact the authors of the schema.
                    Thanks!

                </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:any namespace="##other" processContents="lax"
            minOccurs="0" maxOccurs="unbounded" />

    </xs:sequence>
</xs:complexType>

<xs:complexType name="Asserter">
    <xs:sequence>
        <xs:any namespace="##other" maxOccurs="unbounded"
            minOccurs="0" />
    </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="Asserter">
    <xs:sequence>
        <xs:any namespace="##other" maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>

<xs:simpleType name="NumberOfExpectedAssertions">
    <xs:restriction base="xs:positiveInteger"/>
</xs:simpleType>



<!-- Following the WS-Security spec, we allow any type of identiification -->


    <xs:complexType name="View">
    <xs:sequence>
        <xs:element ref="ps:asserter" />
        <xs:choice maxOccurs="unbounded" minOccurs="0">
            <xs:element maxOccurs="unbounded" minOccurs="0"
                ref="ps:interactionPAssertion" />
            <xs:element maxOccurs="unbounded" minOccurs="0"
                ref="ps:relationshipPAssertion" />
            <xs:element maxOccurs="unbounded" minOccurs="0"
                ref="ps:actorStatePAssertion" />
            <xs:element maxOccurs="unbounded" minOccurs="0"
               ref="ps:exposedInteractionMetaData"/>
        </xs:choice>

        <xs:any namespace="##other" processContents="lax"
            minOccurs="0" maxOccurs="unbounded" />

    </xs:sequence>
</xs:complexType>

<xs:complexType name="RelationshipPAssertion">
    <xs:sequence>
        <xs:element ref="ps:localPAssertionId"/>
        <xs:element name="subjectId">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="ps:localPAssertionId"/>
                    <xs:element ref="ps:dataAccessor" minOccurs="0" />
                    <xs:element ref="ps:parameterName"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element ref="ps:relation"/>
        <xs:element maxOccurs="unbounded" ref="ps:objectId"/>
        <xs:element maxOccurs="1" minOccurs="0" name="signature" type="ps:Signature" />
    </xs:sequence>
</xs:complexType>

<xs:complexType name="GlobalPAssertionKey">
    <xs:sequence>
        <xs:element ref="ps:interactionKey"/>
        <xs:element ref="ps:viewKind"/>
        <xs:element ref="ps:localPAssertionId"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="PAssertionDataKey">
    <xs:complexContent>
        <xs:extension base="ps:GlobalPAssertionKey">
            <xs:sequence>
```

```
                <xs:element minOccurs="0" ref="ps:dataAccessor"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="InteractionPAssertion">
    <xs:sequence>
        <xs:element maxOccurs="1" minOccurs="1" ref="ps:localPAssertionId"/>
        <xs:element maxOccurs="1" minOccurs="1" ref="ps:documentationStyle"/>
        <xs:element maxOccurs="1" minOccurs="1" name="content" type="ps:Content"/>
        <xs:element maxOccurs="1" minOccurs="0" name="signature" type="ps:Signature"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="ActorStatePAssertion">
    <xs:sequence>
        <xs:element maxOccurs="1" minOccurs="1" ref="ps:localPAssertionId"/>
        <xs:element maxOccurs="1" minOccurs="0" ref="ps:documentationStyle"/>
        <xs:element maxOccurs="1" minOccurs="1" name="content" type="ps:Content"/>
        <xs:element maxOccurs="1" minOccurs="0" name="signature" type="ps:Signature"/>
    </xs:sequence>
</xs:complexType>

<xs:simpleType name="LocalPAssertionId">
    <xs:union memberTypes="xs:long xs:string xs:anyURI"/>
</xs:simpleType>


<xs:complexType name="ViewKind" abstract="true">
    <xs:annotation>
        <xs:documentation>
               Instance documents must select something that is derived
        </xs:documentation>
    </xs:annotation>
</xs:complexType>

<xs:complexType name="SenderViewKind">
    <xs:complexContent>
        <xs:restriction base="ps:ViewKind"></xs:restriction>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="ReceiverViewKind">
    <xs:complexContent>
        <xs:restriction base="ps:ViewKind"></xs:restriction>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="ObjectId">
    <xs:complexContent>
        <xs:extension base="ps:PAssertionDataKey">
            <xs:sequence>
                <xs:element ref="ps:parameterName"/>
                <xs:any namespace="##other" processContents="lax"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="DataAccessor">
    <xs:sequence>
        <xs:any maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
```

```
    <xs:complexType name="Content">
        <xs:sequence>
            <xs:any namespace="##any" maxOccurs="unbounded" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>


    <xs:complexType name="InteractionMetaData">
        <xs:sequence>
           <xs:choice maxOccurs="unbounded">
                <xs:element name="tracer" type="xs:anyURI"/>
                <xs:any namespace="##other" maxOccurs="unbounded" minOccurs="0"/>
           </xs:choice>
       </xs:sequence>
    </xs:complexType>

    <xs:complexType name="InteractionContext">
        <xs:sequence>
            <xs:element ref="ps:interactionKey" />
            <xs:element ref="ps:viewKind"/> <!-- View Kind of the actor who created the metadata -->
            <xs:element ref="ps:interactionMetaData"
                maxOccurs="unbounded" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>


     <xs:complexType name="ExposedInteractionMetaData">
         <xs:sequence>
              <xs:element ref="ps:globalPAssertionKey"/>
               <xs:element ref="ps:interactionMetaData"/>
              <xs:element maxOccurs="1" minOccurs="0" name="signature" type="ps:Signature" />
         </xs:sequence>
    </xs:complexType>

    <xs:complexType name="Signature">
         <xs:sequence>
              <xs:any namespace="##any" maxOccurs="unbounded" minOccurs="0"/>
         </xs:sequence>
    </xs:complexType>

</xs:schema>
```

References

[BBF+02] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, and Ed Simon. XML Signature Syntax and Processing. http://www.w3.org/TR/xmldsig-core/, 2002.

[Bra97] Scott Bradner. Key words for use in RFCs to indicate requirement levels. http://www.ietf.org/rfc/rfc2119.txt, 1997.

[MGJ+06] Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, Victor Tan, and Luc Moreau. Data model for Process Documentation. Technical report, University of Southampton, June 2006.

[TGJ+06] Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, and Luc Moreau. WS Provenance Glossary. Technical report, Electronics and Computer Science, University of Southampton, 2006.

[W3C99] W3C. XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999. http://www.w3.org/TR/xpath, 1999.

ws-prov-link

Authors:
Steve Munroe, U. Southampton
Paul Groth, U. Southampton
Sheng Jiang, U. Southampton
Simon Miles, U. Southampton
Victor Tan, U. Southampton
Luc Moreau, U. Southampton
John Ibbotson, IBM
Javier Vazquez, UPC

August 24, 2006

A WS-Addressing Profile for Distributed Process Documentation

Status of this Memo

This document provides the specification of a data model for distributed process documentation and has the status of a working draft. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

Process documentation can often be distributed across different provenance stores. To enable the discovery of related process documentation, a mechanism is required to link disparate but related process documentation to enable the effective collection of such documentation in order to answer provenance queries. This document represents a WS-addressing profile on distributed process documentation that provides mechanisms to solve this problem.

Contents

1 Introduction

Using the provenance recording protocol [GTM+06] actors may record p-assertions to any number of different provenance stores. This means that the documentation of a process [MGJ+06] that led to a result can be distributed at different locations. Given this, there must be some mechanism to retrieve and reconstruct these p-assertions in order to validate, visualise or replay the represented process. To facilitate such retrieval, the notion of a *link* is introduced which, intuitively, is a pointer to a provenance store. This document defines the schema necessary to represent linking for distributed process documentation. The XML document that describes this model is presented in Appendix A.

1.1 Goals and Requirements

The goal of this document is to define a linking mechanism that allows distributed process documentation to be discovered by queriers [MMG+06]. This document exists as part of a family of documents specifying provenance. It is most closely associated to [MMG+06], which describes how queries over process documentation can be achieved.

1.1.1 Requirements

In meeting this goal, this document must address the following requirements:

- Define the schema for linking different but related actor views for a given interaction.
- Define the schema to define how to locate p-assertions [MGJ+06] that appear as objects in a relation.

1.1.2 Non-Requirements

This document does not intend to meet the following requirements:

- Supply definitions and scope of data provenance. This is covered in [TMG+06c].
- Supply a model for the transformation of process documentation — called documentation styles. This aspect of data provenance is covered in [TMG+06a].
- Supply a model querying process documentation. This aspect of data provenance is described in [MMG+06].
- Supply a model for Provenance security. This aspect of data provenance is described in [TMG+06b].

2 Terminology and Notation

All definitions for the concepts and structures found within this document can be found in [TGJ+06].

2.1 XML Namespaces

The XML Namespace URI that MUST be used by implementations of this specification is: `http://www.pasoa.org/schemas/version023s1/PLinks.xsd`

Table 1 lists XML namespaces that are used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

Prefix	XML Namespace	Specification(s)
pl	`http://www.pasoa.org/schemas/version023s1/PLinks.xsd`	[PLinks]
wsa	`http://schemas.xmlsoap.org/ws/2004/03/addressing`	[WS-addressing]
xs	`http://www.w3.org/2001/XMLSchema`	[XMLSchema]

Table 1: Prefixes and XML Namespaces used in this specification

2.2 Notational Conventions

The keywords "MUST ", "MUSTNOT ", "REQUIRED ", "SHALL ", "SHALLNOT ", "SHOULD ", "SHOULDNOT ", "RECOMMENDED ", "MAY ", and "OPTIONAL " in this document are to be interpreted as described in [Bra97].

2.3 XML Schema Diagrams

This document adopts a graphical notation to describe XML Schema. Figure 1 gives an example of a small XML Schema displayed as a diagram, which is now explained with reference to the figure.

Figure 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:stopTest`, either of which must contain a date.

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

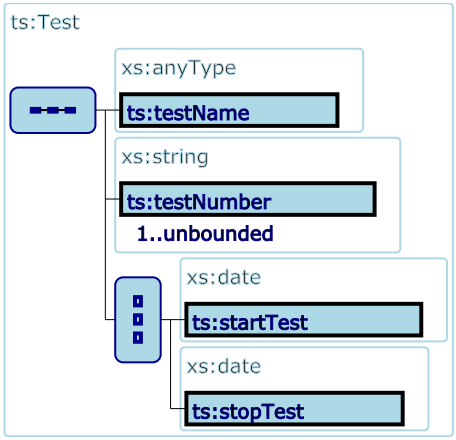


Figure 1: An example XML Schema diagram

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.

2.4 XPath notation

In addition to the XML Schema diagrams, an XPath notation [W3C99] is used to identify each individual element in the specification along with its context, in order to describe precisely its meaning and use.

3 Linking

Process documentation may be distributed across any number of provenance stores, and must be retrieved in order to answer provenance queries. A querier, finding a fragment of process documentation, must be able to locate related fragments. This is achieved by providing explicit *links* to locations where other, related process documentation can be found.

This document provides a linking mechanism to achieve this using the Web-Service addressing's endpoint reference mechanism [GHR06]. Thus, provenance stores are modelled as Web Services that are locatable at an endpoint reference.

Two forms of linking are required: Linking together process documentation that represent different *views* of an interaction, and providing links to data items that play the role of *objects* in a relation. Each mechanism is described below.

3.1 Linking Together Different Views of an Interaction

The different views of actors in an interaction can be linked together by providing links from one set of interaction p-assertions to the location of the associated set (i.e. the set belonging to the other actor in the interaction). This is achieved by providing textttviewLinks, which are links that can be found within p-assertions pointing to the provenance store holding the other view of an interaction.

Linking information is passed between actors as part of an interaction's meta-data, and may be contained within a *p-header* (a full description of which is given in [MTG+06]). The p-header allows for the provision of extra information about an interaction in its `interactionMetaData` element, which contains an extensibility element that can be used to provide application specific information within interactions, and is shown in the xpath expression below.

```
/ps:pheader/ps:interactionMetaData/xs:Any
```

One such use of this extensibility element is to provide the ability to connect process documentation that is spatially distributed via the use of the link mechanism. This can be achieved by redefining the `Any` element as a `viewLink`, resulting in the following xpath expression:

```
/ps:pheader/ps:interactionMetaData/pl:viewLink
```

This model is described by a schema document represented by Figure 2.

Figure 2: A WS-addressing endpoint reference pointing to the location of another view on an interaction

The document content is further described below.

`/pl:viewLink`

This element represents a link to the location of an actor's view of an interaction.

`/pl:viewLink/pl:provenanceStoreRef`

A `viewLink` is instantiated as a WS-addressing `EndpointReference` pointing to the provenance store where the associated view for an interaction can be found.

The instantiation of an EndpointReference can be achieved by utilising WS-Addressing's `ReferenceParameters` element. This is described in Section 4.

3.2 Linking to Objects of a Relationship P-Assertion

Relationship p-assertions allow uni-directional relationships between both messages and data to be expressed. Relationship p-assertions are modelled 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 complete model is presented in the Data Model for Process Documentation document [MGJ+06]). The triple consists of a subject identifier (`subjectId`), a relation, and several object identifiers (`objectIds`).

Relationship p-assertions express 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 caused the output to be as it is. An `objectId` element contains several child elements and is fully defined in [MGJ+06]. In distributed process documentation scenarios, given that an actor can use multiple provenance stores to document its involvement in a process, an `objectId` element will contain an `objectLink` element giving the address of the provenance store in which the data item acting as an object in a relationship is kept. In the complete data model for process documentation presented in [MGJ+06], this element instantiates the element in

the `objectId` structure contained within a relationship p-assertion as shown by the xpath below:

```
/ps:relationshipPAssertion/ps:objectId/xs:Any
```

The Any element is then instantiated in the WS-addressing profile by an object link as follows:

```
/ps:relationshipPAssertion/ps:objectId/pl:objectLink
```

Object links are modelled as shown in Figure 3. The model is further described as follows.

Figure 3: An WS-addressing endpoint reference pointing to the location of an object in a relation

`/pl:objectLink`

This element represents a link to the location of a data item that acts as an object in a relationship p-assertion.

`/pl:objectlink/pl:provenanceStoreRef`

The link to the location of a data item acting as an object in a relationship p-assertion is instantiated as a WS-addressing `EndpointReference` pointing to the provenance store where the data item can be found.

4 Identifying Provenance Store Service Ports

Provenance stores offer different functionality via their public interfaces exposed as ports. When following a link to a provenance store it is necessary to be able to identify the correct port in order to perform the requested operation. The WS-addressing's EndpointReference provides a number of characteristics to enable such identification via `ReferenceParameters`. The `ReferenceParameter` element is defined as an xs:Any, thus it allows for extensibility and enables the specification of a `PortContext` type to provide a way to identify which port is to be used in a communication with a provenance store. Below is shown an instance of a `PortContext` type that defines a key-value pair. The key is given as the

portName and the value is given as the context of the port, i.e. the identifier used to identify the port.

```
<pl:portContext>
    <pl:portName>XQuery</pl:portName>
    <pl:context>xquery</pl:context>
</pl:portContext>
```

A ProvenanceStoreRef is now defined in which the ReferenceParameters element is used to identify two ports, one for recording and one for querying, using the portContext type.

```
<wsa:EndpointReference>
    <wsa:Address>http://www.pasoa.org/provenancestore1/</wsa:Address>

    <wsa:ReferenceParameters>
        <pl:portContext>
            <pl:portName>Record</pl:portName>
            <pl:context>myrecord</pl:context>
       </pl:portContext>
       <pl:portContext>
            <pl:portName>XQuery</pl:portName>
            <pl:context>superxquery</pl:context>
      </pl:portContext>
      ...
    </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

In this way connecting to a provenance store entails specifying the EndpointReference of the provenance store plus the context for the required port. The following example shows an EndpointReference that identifies a record port.

```
http://www.pasoa.org/provenancestore1/myrecord
```

The formal model of this is further described below.

/pl:provenanceStoreRef/wsa:ReferenceParameters/pl:portContext

This element holds the information about one of this provenance store's ports.

/pl:provenanceStoreRef/wsa:ReferenceParameters/pl:portContext/pl:portName

The name of the port for this provenance store.

/pl:provenanceStoreRef/wsa:ReferenceParameters/pl:portContext/pl:context

The context of the specified port for this provenance store.

It should be noted that in cases where process documentation must be kept for long periods of time, it is recommended that the references used for machines and ports should use virtual URI's that can the n be mapped onto real IP addresses, since physical machines may be upgraded or changed over time.

5 Conclusion

This document has provided a WS-addressing profile for the distribution of process documentation. It provides a detailed data model of a linking mechanism instantiated using WS-addressing `EndpointReferences`. This enables process documentation necessary to connect different views of an interaction to be found, and to enable data items acting as objects in a relationship p-assertion to be discovered.

Other forms of linking could be considered where process documentation is distributed but not stored in provenance stores. Specific profiles should be defined for each of these.

Appendix A

The following XML document describes the linking types and elements used in this document.

```
<?xml version="1.0" encoding="UTF-8"?> <xs:schema
xmlns:pl="http://www.pasoa.org/schemas/version023s1/distribution/PLinks.xsd"
                      xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
                    xmlns:xs="http://www.w3.org/2001/XMLSchema"
                    targetNamespace="http://www.pasoa.org/schemas/version023s1/PLinks.xsd"
                    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:annotation>
        <xs:documentation>
            Defines extensions to the P-Structure that allow for linking between
            provenance stores from within the p-structure.
            Author: Paul Groth
            Last Modified:15 May 2006

                  Copyright (c) 2006 University of Southampton
                  See pasoalicense.txt for license information.
                  http://www.opensource.org/licenses/mit-license.php

             </xs:documentation>
    </xs:annotation>

      <xs:import namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"
        schemaLocation="../wsaddressing.xsd"/>

    <xs:element name="viewLink" type="pl:Link">
        <xs:annotation>
            <xs:documentation>
                 A link between this provenance store and another provenance
                 store. Where the provenance store linked to will contain
                 the other view of the interaction record.
             </xs:documentation>
        </xs:annotation>
    </xs:element>

    <xs:element name="objectLink" type="pl:Link">
        <xs:annotation>
            <xs:documentation>
                 A link between this provenance store and another provenance
                 store. Where the provenance store linked to will contain the p-assertion identified by the
                 p-assertion data key in the object id where this link is located.
             </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="Link">
        <xs:sequence>
            <xs:element name="provenanceStoreRef" type="wsa:EndpointReferenceType">
                <xs:annotation>
                    <xs:documentation>
                        The actual link to the Provenance Store.
                    </xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

References

[Bra97] Scott Bradner. Key words for use in RFCs to indicate requirement levels. http://www.ietf.org/rfc/rfc2119.txt, 1997.

[GHR06] Martin Gudgin, Marc Hadley, and Tony Rogers. Web services addressing 1.0 - core w3c recommendation 9 may 2006. http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/, 2006.

[GTM+06] Paul Groth, Victor Tan, Steve Munroe, Sheng Jiang, Simon Miles, and Luc Moreau. Process Documentation Recording Protocol. Technical report, University of Southampton, June 2006.

[MGJ+06] Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, Victor Tan, and Luc Moreau. Data model for Process Documentation. Technical report, University of Southampton, June 2006.

[MMG+06] Simon Miles, Steve Munroe, Paul Groth, Sheng Jiang, Victor Tan, John Ibbotson, and Luc Moreau. Process Documentation Query Protocol. Technical report, University of Southampton, June 2006.

[MTG+06] Steve Munroe, Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. A SOAP Binding For Process Documentation. Technical report, University of Southampton, June 2006.

[TGJ+06] Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, and Luc Moreau. WS Provenance Glossary. Technical report, Electronics and Computer Science, University of Southampton, 2006.

[TMG+06a] Victor Tan, Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. Basic Transformation Profile for Documentation Style. Technical report, University of Southampton, June 2006.

[TMG+06b] Victor Tan, Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. Data Model for Provenance Security. Technical report, University of Southampton, June 2006.

[TMG+06c] Victor Tan, Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. The Provenance Standardisation Vision. Technical report, University of Southampton, June 2006.

[W3C99] W3C. XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999. http://www.w3.org/TR/xpath, 1999.

ws-prov-ds

Authors:
Victor Tan, U. Southampton
Paul Groth, U. Southampton,
Sheng Jiang, U. Southampton
Simon Miles, U. Southampton
Steve Munroe, U. Southampton
Sofia Tsasakou, U. Southampton
Luc Moreau, U. Southampton

November 23, 2006

Basic Transformation Profile for Documentation Style

Status of this Memo

This document provides information to the community regarding the specification of a data model for documentation style. It is intended to supplement the data model for process documentation [MGJ+06], and to be read in conjunction with that document. This document does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

The data model for process documentation [MGJ+06] provides descriptions of different ways in which processes may be documented. The primary method involves recording messages exchanged between interacting services, as well as the state of those services at the time of message exchange. These messages and states may have their contents transformed when recorded due to application dependent security and scalability requirements. The documentation style describes the types of transformations that can be performed. An actor that processes the recorded documentation must understand the transformations performed on it in order to interpret or utilise it appropriately. This document is a profile of several basic documentation style transformations that are likely to be useful in application domains that use process documentation. It is not intended to be exhaustive; other profiles may be provided of alternative documentation style transformations which may be generic or more specific in nature.

Contents

1 Introduction

The documentation style data model is presented here in the context of the process documentation model [MGJ+06]. In the process documentation model, p-assertions are atomic units of process documentation of which there exists three forms: relationship p-assertions, actor state p-assertions and interaction p-assertions. When an actor in a provenance-aware system documents an interaction, it constructs an interaction p-assertion, which states the content of a message received or sent by that actor. An actor can also make assertions about its internal state in the context of a specific interaction through an actor state p-assertion. This can be the state of the actor prior to or after a message exchange. Relationship p-assertions allow uni-directional relationships between both messages and data to be expressed.

The activity of constructing an interaction p-assertion from a message can be considered as a single atomic transformation. This transformation needs to be adequately defined by the actor creating that p-assertion in order that actors who retrieve that p-assertion from the provenance store are able to understand the nature of the transformation applied. This is equally true for an actor state p-assertion. Documentation styles are essentially descriptions of the types of transformations that can be applied to a message or to the internal state of an actor. Relationship p-assertions do not utilize documentation styles; this is further clarified in Section 3.

1.1 Goals and Requirements

The goal of this document is to develop an open, interoperable approach to process documentation.

1.1.1 Requirements

This specification intends to meet the following requirements:

- Provide a profile of a data model that describes some basic documentation style transformations.
- Provide extensibility in the data model for describing new types of documentation style transformations.

1.1.2 Non-Requirements

This document does not intend to meet the following requirements:

- Provide an implementation specific approach to realizing these documentation style transformations.

- Provide an exhaustive list of documentation style transformations relevant to process documentation.

2 Terminology and Notation

All definitions for the concepts and structures found within this document can be found in [TGJ+06].

2.1 XML Namespaces

The XML Namespace URI that MUST be used by implementations of this specification is: `http://www.pasoa.org/schemas/version023s1/docstyle`

Table 1 lists XML namespaces that are used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

Prefix	XML Namespace	Specification(s)
pds	`http://www.pasoa.org/schemas/version023s1/docstyle`	[DocumentationStyle]
xenc	`http://www.w3.org/2001/04/xmlenc#`	[XML Encryption]
ds	`http://www.w3.org/2000/09/xmldsig#`	[XML Signature]
xs	`http://www.w3.org/2001/XMLSchema`	[XML Schema]

Table 1: Prefixes and XML Namespaces used in this specification

2.2 Notational Conventions

The keywords "MUST ", "MUSTNOT ", "REQUIRED ", "SHALL ", "SHALLNOT ", "SHOULD ", "SHOULDNOT ", "RECOMMENDED ", "MAY ", and "OPTIONAL " in this document are to be interpreted as described in [Bra97].

2.3 XML Schema Diagrams

This document adopts a graphical notation to describe XML Schema. Figure 1 gives an example of a small XML Schema displayed as a diagram, which is now explained with reference to the figure.

Figure 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:stopTest, either of which must contain a date.

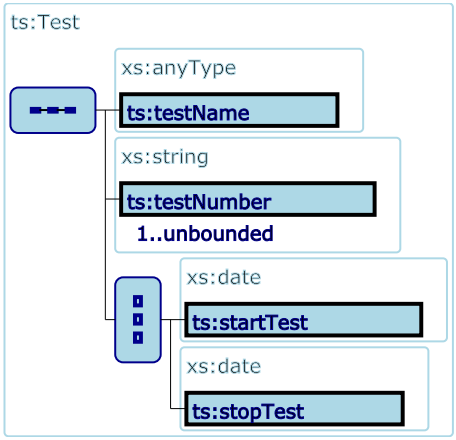


Figure 1: An example XML Schema diagram

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.

2.4 XPath notation

In addition to the XML Schema diagrams, an XPath notation [W3C99] is used to identify each individual element in the specification along with its context, in order to describe precisely its meaning and use.

3 Data Model for Transformation Description Document

In the process documentation model [MGJ+06], the content of an interaction p-assertion is intended to provide information about a single message exchanged between two actors. In some cases, the content may simply be the actual message itself verbatim. However, there may often be an application-specific need to transform or modify the actual message in some way before storing it as the content of an interaction p-assertion. For example, parts of the message may contain highly sensitive information and there may be a corresponding security requirement within the application environment to obscure these parts in some manner or remove them prior to placing the message itself into a p-assertion. A message may contain a large amount of raw data that is irrelevant to process documentation; the raw data could effectively be replaced with an external reference to minimize the size of the interaction p-assertion created.

Actor state p-assertions contain state information about an actor which may be structured in an arbitrary fashion by the actor concerned. This structured information can also be transformed in the same ways as interaction p-assertions, based on similar motivations as well.

Documentation style transformations then refer in general to all the possible types of transformations that can be applied to a message exchanged between actors or state information provided by an actor in order to obtain a transformed output that will become the content of either an interaction p-assertion or actor state p-assertion. It is possible, but not mandatory, that documentation style transformations are reversible i.e. the content of a p-assertion can be reverse-transformed to produce the input that it was originally derived from.

The information required to perform a specific transformation must be provided as a document (the *transformation definition document*). The transformation is performed using this document to produce an output that should be properly typed. For the case when the output of a transformation is an XML document, this requires that it should be well formed and be capable of being validated against a schema. The transformed output is then stored within a p-assertion while the transformation definition document is then made available at a repository. This should be accessible to all actors that process p-assertions in order to allow them to understand how the contents of a specific p-assertion was created.

Documentation style transformations shall not be applicable to relationship p-assertions. The information in the fields of a relationship p-assertion have an explicit meaning within the context of the process documentation data model, and must not be altered in any way in order to ensure that provenance queries that use them produce correct results.

3.1 Transform Definition and Transform Operation

The structure of a transformation definition document is shown in Figure 2. It describes a documentation style transformation and the nature of the input and output (an XML document, Java object, CORBA object) that it functions on. Both the input and output are application dependent and may not be based on the same technology. For example, a transformation could operate on a Java object as an input and produce an XML document as output. The location of the transformation definition document must be specified as a URI in the `/ps:interactionPAssertion /ps:documentationStyle` component of an interaction p-assertion (Figure 4 [MGJ+06]). The constituent components of this structure as described below must be provided, unless otherwise stated.

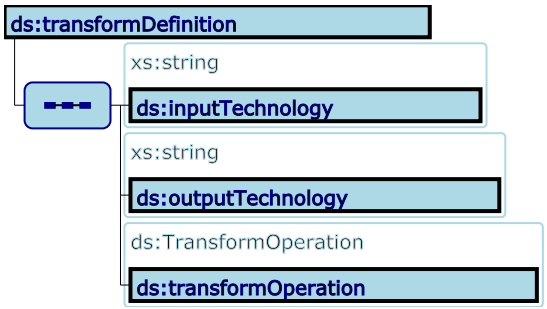


Figure 2: Model for a transform definition

`/ds:transformDefinition`

This is the root element and provides a logical structure for the document that describes the documentation style transform operation

`/ds:transformDefinition/ds:inputTechnology`

This element specifies the underlying technology for the object to be transformed, whether it is an XML document, CORBA object, etc.

`/ds:transformDefinition/ds:outputTechnology`

This element specifies the technology for the transformed object, whether it is an XML document, CORBA object, etc.

`/ds:transformDefinition/ds:transformOperation`

This component provides the relevant information required in the execution of the specific transform operation.

All the documentation style transformations that are described in the remaining sections in this document shall be extensions of the `/ds:transformDefinition /ds:transformOperation` element, each containing information specific to themselves. Any new user-defined documentation style should be defined as an instance in a similar manner as well.

3.2 Verbatim Documentation Style

The verbatim documentation style (Figure 3) denotes a null transformation applied to a given input; the output of the transformation is identical to its input. The main intention of this style is to indicate the creation of an interaction p-assertion where the contents of the p-assertion is the exchanged message as it is.

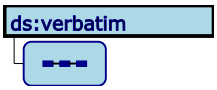


Figure 3: Model for a verbatim documentation style

`/ds:verbatim`

This is the root element for the description of the verbatim documentation style.

3.3 Reference Documentation Style

The reference documentation style (Figure 4) denotes a transformation of an input 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. This transformation operation may include the computation of a digest on the contents to be replaced. The digest, if included, shall be appended to the transformed message in a manner that associates it with the reference URI. The primary functionality of the digest is to verify that in a reverse transformation, data retrieved from a reference URI is identical to the data on which the digest was originally computed.

For the case of an XML document, the transformed result shall have a new namespace corresponding to the additional elements appended into it. In addition, namespaces of existing untransformed elements in the original document

may be changed to new namespaces in the transformed document in order to reflect that a transformation has occurred. If such namespace change is desired, a list mapping namespaces in the original document to new namespaces must be provided to allow the transformation functionality to make these namespace substitutions when constructing the transformed document. Subsequent validation of the output document against a specified schema, if so desired, shall then be based on these new namespaces.

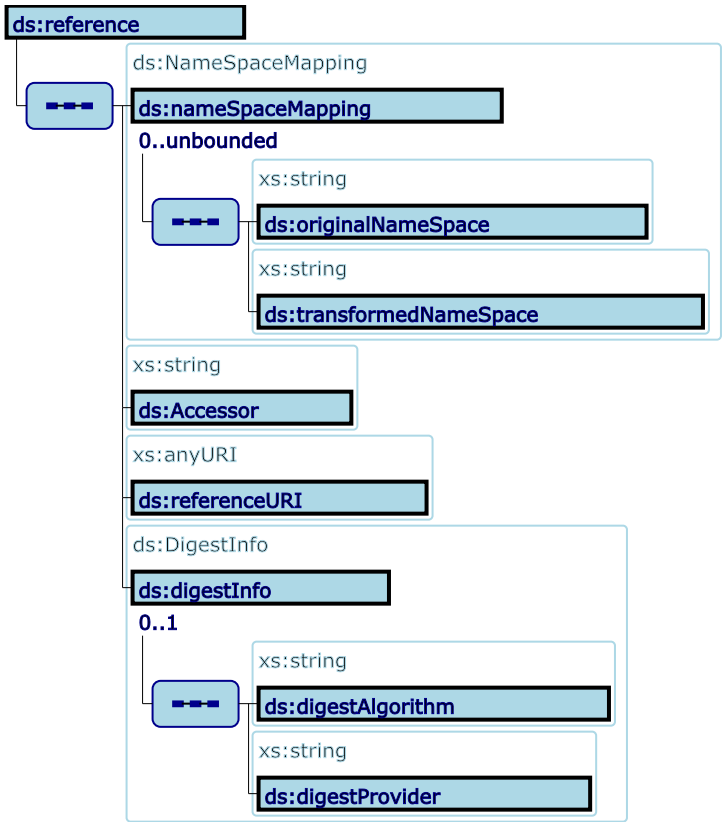


Figure 4: Model for a reference documentation style

`/ds:reference`

This is the root element for the description of the reference documentation style.

`/ds:reference/ds:nameSpaceMapping`

This element is a list mapping namespace URIs in the original XML document to namespace URIs in the transformed XML document and must be provided if the original document contains any namespace URIs. Such mapping must encompass all namespaces in the original document in order

for the transformed document to be meaningful. The namespace mapping must be applied in an equivalent manner for a reverse transformation to produce the original document.

`/ds:reference/ds:Accessor`

This element is used to specify the appropriate part of the input that the reference transformation shall operate on.

`/ds:reference/ds:referenceURI`

This element shall provide the URI for the location where the referenced contents can be found.

`/ds:reference/ds:digestInfo`

This optional element specifies that a digest operation is computed on the referenced contents using the digest algorithm value specified in the `digestAlgo` and the digest provider value specified in the `digestProvider` elements. If not provided, no digest is computed.

The schema for the new elements introduced into the transformed XML document is shown below, with the prefix rd.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
targetNamespace="http://www.gridprovenance.org/documentationstyle/referenceOutput"
elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:rd="http://www.gridprovenance.org/documentationstyle/referenceOutput"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="referenceURI" type="xs:anyURI"/>
  <xs:element name="referenceDigest" type="xs:string"/>
</xs:schema>
```

3.4 Signature Documentation Style

The security-signing documentation style (Figure 5) denotes a transformation of an input by which a part of (or the whole of) its contents has been signed. The signature itself shall appear as a new element in the transformed output.

`/ds:sign`

This is the root element for the description of the signature documentation style.

`/ds:sign/ds:nameSpaceMapping`

This element is a list mapping namespace URIs in the original XML document to namespace URIs in the transformed XML document and must be provided if the original document contains any namespace URIs. Such

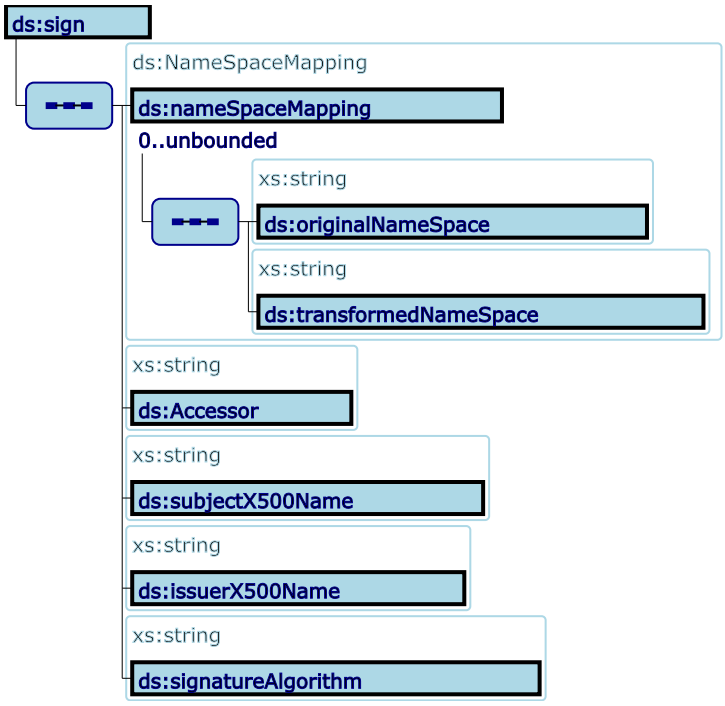


Figure 5: Model for a signature documentation style

mapping must encompass all namespaces in the original document in order for the transformed document to be meaningful. The namespace mapping must be applied in an equivalent manner for a reverse transformation to produce the original document.

`/ds:sign/ds:Accessor`

This element is used to specify the appropriate part of the input that the signature transformation shall operate on.

`/ds:sign/ds:subjectX500Name`

This element is used to specify the subject X500 distinguished name of the X509 certificate [HFPS99] used in the signature operation.

`/ds:sign/ds:issuerX500Name`

This element is used to specify the issuer X500 distinguished name of the X509 certificate [HFPS99] used in the signature operation.

`/ds:sign/ds:signatureAlgorithm`

This element is used to specify the signature algorithm used.

For an XML document, signing should be performed using the XML Signature Recommendation [BBF+02], and the corresponding schema for the new signature element in the transformed output is given at `http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd`.

3.5 Encryption Documentation Style

The encryption documentation style (Figure 6) denotes a transformation of an input by which a part of (or the whole of) its contents has been encrypted. The encrypted content shall appear in replacement of the original content in the transformed output.

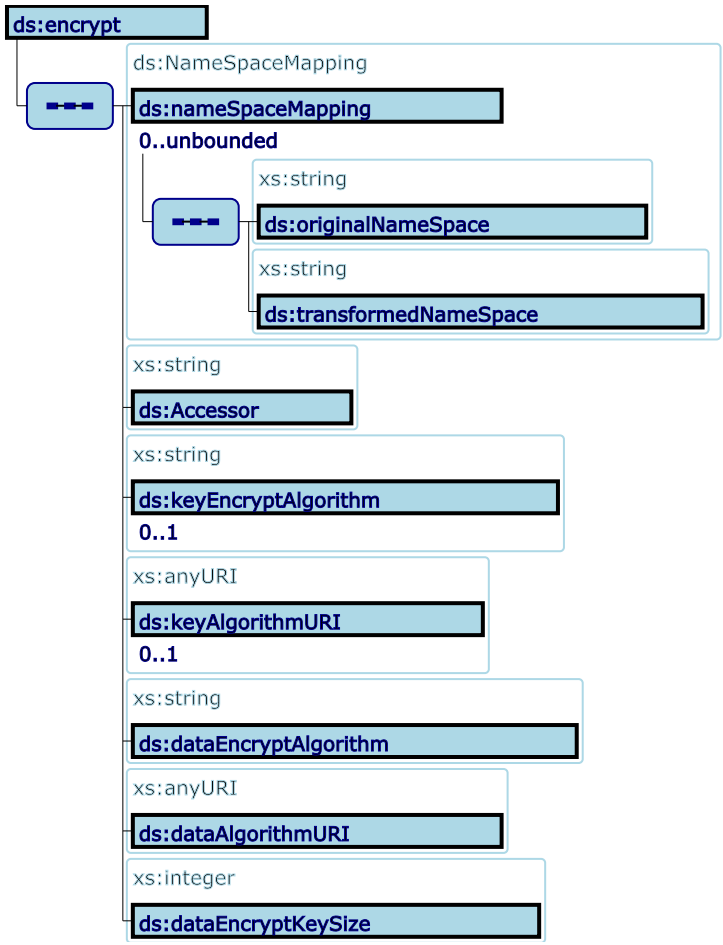


Figure 6: Model for an encryption documentation style

`/ds:encrypt`

This is the root element for the description of the encryption documentation style.

`/ds:encrypt/ds:nameSpaceMapping`

This element is a list mapping namespace URIs in the original XML document to namespace URIs in the transformed XML document and must be provided if the original document contains any namespace URIs. Such mapping must encompass all namespaces in the original document in order for the transformed document to be meaningful. The namespace mapping must be applied in an equivalent manner for a reverse transformation to produce the original document.

`/ds:encrypt/ds:Accessor`

This element is used to specify the appropriate part of the input that the encryption transformation shall operate on.

`/ds:encrypt/ds:keyEncryptAlgorithm`

This element is used to specify the encryption algorithm to be applied to the secret key used in the encryption operation.

`/ds:encrypt/ds:keyAlgorithmURI`

This element is used to specify the URI describing the previous algorithm.

`/ds:encrypt/ds:dataEncryptAlgorithm`

This element is used to specify the encryption algorithm to be applied to the designated contents to be encrypted in the encryption operation.

`/ds:encrypt/ds:dataAlgorithmURI`

This element is used to specify the URI describing the encryption algorithm above.

`/ds:encrypt/ds:dataEncryptKeySize`

This element is used to specify the size of the secret key used in encrypting the designated contents.

For an XML document, the encryption should be performed using the XML Encryption Recommendation [IDS02], and the corresponding schema for the new signature element in the transformed output is given at `http://www.w3.org/TR/xmlenc-core/xenc-schema.xsd`.

3.6 Composite Sequence Documentation Style

A composite sequence documentation style denotes a sequence of documentation styles or transformations that is applied in succession to a given input (message or actor state information). These transformations include the previously discussed documentation styles (encryption, signature, reference and verbatim), and may also include the composite style itself. The output of a transformation in a sequence becomes input to the next transformation in that same sequence, so that the final output of the documentation style is an initial input that has undergone all transformations specified within the sequence. Conversely, a reverse transformation involves the application of all the transformations in a sequence in a reversed order on a transformed input.

Figure 7: Model for composite sequence documentation style

`/ds:compositeSequence`

This is the root element for the description of the composite documentation style.

`/ds:compositeSequence/ds:transformOperation`

This element is a sequence of documentation style transformations unbounded in length, which may additionally include other composite sequence transformations within it.

3.7 Example

We demonstrate how documentation style transformations affect a given input that is an XML document using an example involving the a composite sequence transformation that encapsulates an encryption and reference documentation style. Consider two actors that exchange a single XML structured message between each other as shown in Figure 8.

This original message can optionally be validated against a schema prior to performing transformation to ensure that errors do not arise from attempting to transform an incorrectly typed input. An example of such a schema for the above message is shown in Figure 9.

Consider now a composite transformation involving an encryption documentation style followed by a reference documentation style to be enacted upon this

```
<?xml version="1.0" encoding="UTF-8"?>
<pd:author xmlns:pd="http://www.myexample.com/personaldetails"
xmlns:ad="http://www.myexample.com/addressdetails">

  <pd:firstname>John</pd:firstname>
  <pd:lastname>Doe</pd:lastname>
  <ad:address>
    <ad:street>123 High Street</ad:street>
    <ad:city>Gotham City</ad:city>
  </ad:address>
  <pd:biography>He led an undistinguished life explained in 200 pages</pd:biography>

</pd:author>
```

Figure 8: Original XML message

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.myexample.com/personaldetails"
elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ad="http://www.myexample.com/addressdetails">

  <xs:element name="author">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstname" type="xs:string"/>
        <xs:element name="lastname" type="xs:string"/>
        <xs:element ref="ad:address" />
        <xs:element name="biography" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Figure 9: Original XML message schema

sample message in order to produce a transformed message, that will become the content of an interaction p-assertion. The first transformation specifies that the contents of the **pd:lastname** element be encrypted. The next transformation specifies that the contents of the **pd:biography** element be removed and stored at a public repository, and the URI pointing to this stored content is placed in the transformed message along with a digest initially computed on the removed contents. The transformation definition document that specifies this composite transformation itself is shown in Figure 10.

The **Accessor** element within both transformations specifies XPath expressions that identify the specific element within the original message that would be operated upon. Also, the two namespaces present in the original message are now mapped to new namespaces through the namespace mapping specified in both transformations. The transformed message is shown in Figure 11.

This transformed message can optionally be validated against an output schema if one is provided. This output schema for the transformed message can be

```
<transformDefinition xmlns="http://www.gridprovenance.org/documentationstyle"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:pds="http://www.gridprovenance.org/documentationstyle">

  <inputTechnology>XML</inputTechnology>
  <outputTechnology>XML</outputTechnology>

  <transformOperation xsi:type="CompositeSequence">

    <transformOperation xsi:type="Encrypt">

      <nameSpaceMapping>
        <originalNameSpace>http://www.myexample.com/personaldetails</originalNameSpace>
        <transformedNameSpace>http://www.myexample.com/personaldetails/enc</transformedNameSpace>
      </nameSpaceMapping>
      <nameSpaceMapping>
        <originalNameSpace>http://www.myexample.com/addressdetails</originalNameSpace>
        <transformedNameSpace>http://www.myexample.com/addressdetails/enc</transformedNameSpace>
      </nameSpaceMapping>
      <Accessor>author/lastname</Accessor>
      <keyEncryptAlgorithm>DESede</keyEncryptAlgorithm>
      <keyAlgorithmURI>http://www.w3.org/2001/04/xmlenc#kw-tripledes</keyAlgorithmURI>
      <dataEncryptAlgorithm>AES</dataEncryptAlgorithm>
      <dataAlgorithmURI>http://www.w3.org/2001/04/xmlenc#aes128-cbc</dataAlgorithmURI>
      <dataEncryptKeySize>128</dataEncryptKeySize>

    </transformOperation>


    <transformOperation xsi:type="Reference">

      <nameSpaceMapping>
        <originalNameSpace>http://www.myexample.com/personaldetails/enc</originalNameSpace>
        <transformedNameSpace>http://www.myexample.com/personaldetails/enc/ref</transformedNameSpace>
      </nameSpaceMapping>
      <nameSpaceMapping>
        <originalNameSpace>http://www.myexample.com/addressdetails/enc</originalNameSpace>
        <transformedNameSpace>http://www.myexample.com/addressdetails/enc/ref</transformedNameSpace>
      </nameSpaceMapping>
      <Accessor>author/biography</Accessor>
      <referenceURI>http://www.mystorage.com/biography.txt</referenceURI>
      <digestInfo>
        <digestAlgorithm>MD5</digestAlgorithm>
        <digestProvider>Java</digestProvider>
      </digestInfo>

    </transformOperation>

  </transformOperation>

</transformDefinition>
```

Figure 10: Transformation definition document

```
<?xml version="1.0" encoding="UTF-8"?>

<pd:author xmlns:pd="http://www.myexample.com/personaldetails/enc/ref"
xmlns:ad="http://www.myexample.com/addressdetails/enc/ref"
xmlns:rd="http://www.gridprovenance.org/documentationstyle/referenceOutput"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">

  <pd:firstname>John</pd:firstname>
  <pd:lastname>

    <xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Content">
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc" />

      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <xenc:EncryptedKey>
          <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-tripledes"/>
            <xenc:CipherData>
              <xenc:CipherValue>
                dbZHVtHrAXoWDX3awB0G7RAYWd/lMgQz4o4B+wEh4yg=
              </xenc:CipherValue>
            </xenc:CipherData>
        </xenc:EncryptedKey>
      </KeyInfo>

      <xenc:CipherData>
        <xenc:CipherValue>
          g9Fe6ppIg4aN0wpxPA5KFUz+==
        </xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>

  </pd:lastname>
  <ad:address>
    <ad:street>123 High Street</ad:street>
    <ad:city>Gotham City</ad:city>
  </ad:address>
  <pd:biography>
    <rd:referenceURI>http://www.mystorage.com/biography.txt</rd:referenceURI>
    <rd:referenceDigest>1nIldFXC2BVSbqE9njc6gQ==</rd:referenceDigest>
  </pd:biography>

</pd:author>
```

Figure 11: Transformed XML message

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.myexample.com/personaldetails/enc/ref"
elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ad="http://www.myexample.com/addressdetails/enc/ref"
xmlns:rd="http://www.gridprovenance.org/documentationstyle/referenceOutput"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">


  <xs:element name="author">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstname" type="xs:string"/>
        <xs:element name="lastname">
          <xs:complexType>
            <xs:sequence>
             <xs:element ref="xenc:EncryptedData"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>

        <xs:element ref="ad:address" />
        <xs:element name="biography">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="rd:referenceURI"/>
              <xs:element ref="rd:referenceDigest"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>

      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Figure 12: Transformed XML message schema

predefined prior to runtime by the application user, or generated at runtime using the schema for the original message and the transformation definition document. Such an output schema should contain the new namespaces for the untransformed elements as well reference schemas for the new elements introduced as a result of the transformation. An example of such an output schema, which incorporates the predefined schema for reference elements (Section 3.3), as well as the schema for XML Encryption (`http://www.w3.org/TR/xmlenc-core/xenc-schema.xsd`) is shown in Figure 12.

The schema for the original and transformed message can be stored in a public repository in order for an actor to retrieve them and use them in a reverse transformation when processing interaction p-assertions. Optionally, an actor may also construct its own output schema for validation purposes using the namespace mapping provided in the transformation definition document.

4 Conclusion

In this document, we have presented a data model for some standard documentation style transformations that can be applied to either a message or actor state in order to produce a transformed output that becomes the content of an interaction p-assertion or actor state p-assertion. An example is provided to illustrate how the model can be used on XML type documents. This model is intended as a complement to the process documentation data model [MGJ+06], which describes the logical organisation of process documentation as well as models of different forms of p-assertions

A Schema for transformation definition document

Below we give the full schema for a transformation definition document that is structured in XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.gridprovenance.org/documentationstyle"
elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:ds="http://www.gridprovenance.org/documentationstyle"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="transformDefinition" type="ds:TransformDefinition"/>
  <xs:element name="transformOperation" type="ds:TransformOperation"/>
  <xs:element name="nameSpaceMapping" type="ds:NameSpaceMapping"/>
  <xs:element name="verbatim" type="ds:Verbatim"/>
  <xs:element name="sign" type="ds:Sign"/>
  <xs:element name="encrypt" type="ds:Encrypt"/>
  <xs:element name="reference" type="ds:Reference"/>
  <xs:element name="compositeSequence" type="ds:CompositeSequence"/>
  <xs:complexType name="TransformDefinition">
    <xs:sequence>
      <xs:element name="inputTechnology" type="xs:string"/>
      <xs:element name="outputTechnology" type="xs:string"/>
      <xs:element ref="ds:transformOperation"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="TransformOperation" abstract="true"/>
  <xs:complexType name="Verbatim">
    <xs:complexContent>
      <xs:extension base="ds:TransformOperation"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Sign">
    <xs:complexContent>
      <xs:extension base="ds:TransformOperation">
        <xs:sequence>
          <xs:element ref="ds:nameSpaceMapping" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element name="Accessor" type="xs:string"/>
          <xs:element name="subjectX500Name" type="xs:string"/>
          <xs:element name="issuerX500Name" type="xs:string"/>
          <xs:element name="signatureAlgorithm" type="xs:string"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Encrypt">
    <xs:complexContent>
      <xs:extension base="ds:TransformOperation">
        <xs:sequence>
          <xs:element ref="ds:nameSpaceMapping" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element name="Accessor" type="xs:string"/>
          <xs:element name="keyEncryptAlgorithm" type="xs:string" minOccurs="0"/>
          <xs:element name="keyAlgorithmURI" type="xs:anyURI" minOccurs="0"/>
          <xs:element name="dataEncryptAlgorithm" type="xs:string"/>
          <xs:element name="dataAlgorithmURI" type="xs:anyURI"/>
          <xs:element name="dataEncryptKeySize" type="xs:integer"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Reference">
```

```
    <xs:complexContent>
      <xs:extension base="ds:TransformOperation">
        <xs:sequence>
          <xs:element ref="ds:nameSpaceMapping" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element name="Accessor" type="xs:string"/>
          <xs:element name="referenceURI" type="xs:anyURI"/>
          <xs:element name="digestInfo" type="ds:DigestInfo" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="CompositeSequence">
    <xs:complexContent>
      <xs:extension base="ds:TransformOperation">
        <xs:sequence>
          <xs:element ref="ds:transformOperation" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="NameSpaceMapping">
    <xs:sequence>
      <xs:element name="originalNameSpace" type="xs:string"/>
      <xs:element name="transformedNameSpace" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="DigestInfo">
    <xs:sequence>
      <xs:element name="digestAlgorithm" type="xs:string"/>
      <xs:element name="digestProvider" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

References

[BBF+02] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, and Ed Simon. XML Signature Syntax and Processing. http://www.w3.org/TR/xmldsig-core/, 2002.

[Bra97] Scott Bradner. Key words for use in RFCs to indicate requirement levels. http://www.ietf.org/rfc/rfc2119.txt, 1997.

[HFPS99] R. Housley, W. Ford, W. Polk, and D. Solo. Request For Comment: Internet X.509 Public Key Infrastructure Certificate and CRL Profile. http://www.ietf.org/rfc/rfc2459.txt, 1999.

[IDS02] Takeshi Imamura, Blair Dillaway, and Ed Simon. XML Encryption Syntax and Processing. http://www.w3.org/TR/xmlenc-core/, 2002.

[MGJ+06] Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, Victor Tan, and Luc Moreau. Data model for Process Documentation. Technical report, University of Southampton, June 2006.

[TGJ+06] Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, and Luc Moreau. WS Provenance Glossary. Technical report, Electronics and Computer Science, University of Southampton, 2006.

[W3C99] W3C. XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999. http://www.w3.org/TR/xpath, 1999.

ws-prov-pquery-xpath

Authors:
Simon Miles, U. Southampton
Luc Moreau, U. Southampton
Paul Groth, U. Southampton
Victor Tan, U. Southampton
Steve Munroe, U. Southampton
Sheng Jiang, U. Southampton

November 23, 2006

XPath Profile for the Provenance Query Protocol

Status of this Memo

This document provides information to the community regarding the specification of a profile for using XPath in querying the provenance of data items from process documentation and has the status of a working draft. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

The provenance query protocol has been defined in a separate document [MMG+06], and includes the data models for provenance query requests which can be executed by a provenance query engine. Many parts of the request document are unspecified, being dependent on the provenance query engine implementation. This document defines an XPath-based profile by which provenance queries can be fully specified against process documentation that is in, or can be mapped to, XML format.

Contents

1 Introduction

The provenance query request data model [MMG+06] has many unspecified parts, being dependent on the provenance query engine implementation. This document defines an XPath-based profile by which provenance queries can be fully specified against process documentation that is in, or can be mapped to, XML format.

This document defines a method by which query data handles can be specified as XPath expressions evaluated against the p-structure, data accessors can be specified as XPaths that are usable by the provenance query engine, and relationship target filters can be expressed using XPaths over relationship targets.

1.1 Goals and Requirements

The goal of this document is to define extensions to the provenance query protocol to give a complete XPath-based approach to expressing and evaluating such queries.

1.1.1 Requirements

In meeting this goal, this document must address the following requirements:

- Define the schema of an XPath query data handle.
- Define the schema of an unambiguous (so usable by the provenance query) XPath-based data accessor.
- Define the schema of an XPath relationship target filter.

1.1.2 Non-Requirements

No relevant non-requirements have been determined for this specification.

2 Terminology and Notation

All definitions for the concepts and structures found within this document can be found in [TGJ+06].

2.1 XML Namespaces

The XML Namespace URI that MUST be used by implementations of this specification is: `http://www.pasoa.org/schemas/version023s1/xquery/XQuery.xsd`

Table 1 lists XML namespaces that are used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

Prefix	XML Namespace	Specification(s)
xp	`http://www.pasoa.org/schemas/version023s1/pquery/XPathPQuery.xsd`	[XPath]
pq	`http://www.pasoa.org/schemas/version023s1/pquery/ProvenanceQuery.xsd`	[PQuery]
ps	`http://www.pasoa.org/schemas/version023s1/PStruct.xsd`	[PStruct]
xs	`http://www.w3.org/2001/XMLSchema`	[XMLSchema]

Table 1: Prefixes and XML Namespaces used in this specification

2.2 Notational Conventions

The keywords "MUST ", "MUSTNOT ", "REQUIRED ", "SHALL ", "SHALLNOT ", "SHOULD ", "SHOULDNOT ", "RECOMMENDED ", "MAY ", and "OPTIONAL " in this document are to be interpreted as described in [Bra97].

2.3 XML Schema Diagrams

This document adopts a graphical notation to describe XML Schema. Figure 1 gives an example of a small XML Schema displayed as a diagram, which is now explained with reference to the figure.

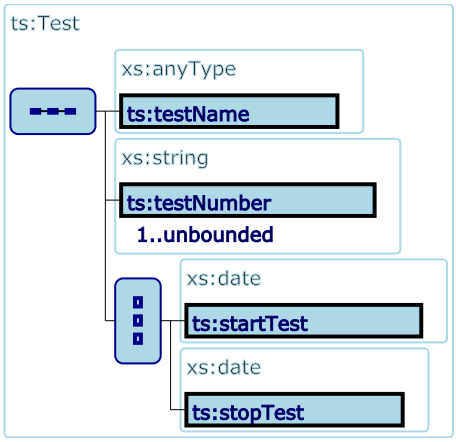


Figure 1: An example XML Schema diagram

Figure 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:stopTest`, 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.

2.4 XPath notation

In addition to the XML Schema diagrams, an XPath notation [W3C99] is used to identify each individual element in the specification along with its context, in order to describe precisely its meaning and use.

3 XPath Query Data Handle

A query data handle can be expressed as an XPath over the p-structure. On evaluating this XPath on a set of process documentation following the p-structure, it

will return a set of nodes, which should be interaction or actor state p-assertions or nodes within interaction or actor state p-assertion contents. If the path evaluates to any other node in the p-structure, e.g. a whole view or interaction record, then this is an error and a fault should be returned. These are the start data items for the provenance query algorithm. Note that while a provenance query is primarily intended to find the provenance of a single item, there is nothing preventing a query data handle from referring to multiple items, and the provenance of all of these will be determined and returned by the provenance query engine.

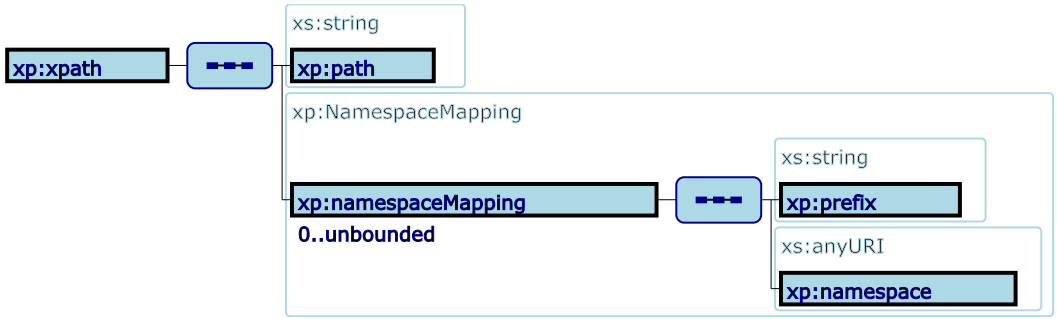


Figure 2: Provenance Query Request

An XPath search is a document instantiating the schema shown in Figure 2.

`/xp:xpath`

Contains an XPath definition.

`/xp:xpath/xp:path`

Contains the actual XPath string itself.

`/xp:xpath/xp:namespaceMapping`

Contains a mapping from a prefix used in the XPath string to a namespace.

`/xp:xpath/xp:namespaceMapping/xp:prefix`

Contains the prefix in a prefix-to-namespace mapping.

`/xp:xpath/xp:namespaceMapping/xp:namespace`

Contains the namespace in a prefix-to-namespace mapping.

The following is an example of an XPath query data handle. It finds an element named “ex:data” in the sender view of interactions with a given message sink.

```
<pq:queryDataHandle>
  <pq:search>
    <xp:xpath>
      <xp:path>/ps:pstruct/ps:interactionRecord[ps:interactionKey/
         ps:messageSink[wsa:Address="http://www.example.com/store"]/ps:sender/
         ps:interactionPAssertion/ps:content/ex:envelope/ex:store/ex:data</xp:path>
      <xp:namespaceMapping>
        <xp:prefix>ps</xp:prefix>
        <xp:namespace>http://www.pasoa.org/schemas/023s1/PStruct.xsd</xp:namespace>
      </xp:namespaceMapping>
      <xp:namespaceMapping>
        <xp:prefix>wsa</xp:prefix>
        <xp:namespace>http://www.ws.addressing</xp:namespace>
      </xp:namespaceMapping>
      <xp:namespaceMapping>
        <xp:prefix>ex</xp:prefix>
        <xp:namespace>http://www.example.com</xp:namespace>
      </xp:namespaceMapping>
    </xp:xpath>
  </pq:search>
</pq:queryDataHandle>
```

4 Single Node XPath Data Accessor

A data accessor for which the operations required by a provenance query engine can be supported is a *single node XPath*. This is an XPath, using a subset of the XPath notation, which explicitly refers to one node in a p-assertion's content and has a normalised form which can be directly compared with other single node XPaths.

A *single node XPath part* is made up of one of the following concatenated sequences of text strings:

- A slash ("/"), a namespace prefix followed by a colon, e.g. "ns:", an element name, e.g. "element", an integer index in brackets, e.g. "[1]".
- A slash ("/"), an at symbol ("@"), a namespace prefix followed by a colon, e.g. "ns:", an attribute name, e.g. "attribute".
- A slash ("/"), the string "text()", an integer index in brackets, e.g. "[1]".

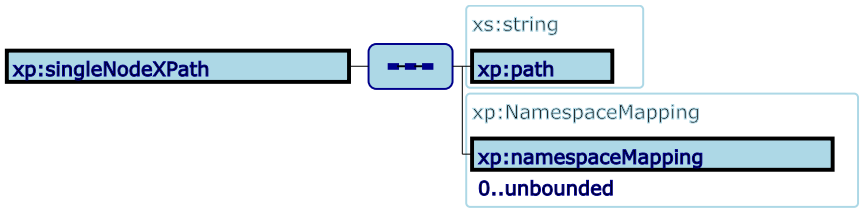


Figure 3: Provenance Query Request

A single node XPath is made up of a concatenated sequence of single node XPath parts (where only the first form may be followed by other forms). The single node XPath is a document instantiating the schema shown in Figure 3.

`/xp:singleNodeXPath`

Contains an XPath definition where the path is a single node XPath.

In order to enable the required data accessor operations to be performed, we define a *normalised form* of a single node XPath. This form is obtained by replacing each namespace prefix, e.g. "ns:" with the namespace it denotes in braces, e.g. "{http://www.example.org}".

The operations required for a provenance query engine to use this data accessor, defined in [MMG+06], are performed as follows.

Get Accessor For Item The data accessor for a node in an p-assertion's XML content is obtained by constructing a single node XPath part for each element from the root of the content down to the node, and concatenating them.

Test Accessor Equality Two single node XPath data accessors are equal if their normalised forms are exact matches.

The following is an example of a single node XPath data accessor. It refers to a particular element named "ex:data" inside a p-assertion's content.

```
<ps:dataAccessor>
  <xp:singleNodeXPath>
    <xp:path>/ex:envelope[0]/ex:store[0]/ex:data[0]</xp:path>
    <xp:namespaceMapping>
      <xp:prefix>ex</xp:prefix>
      <xp:namespace>http://www.example.com</xp:namespace>
    </xp:namespaceMapping>
    </xp:xpath>
  </xp:singleNodeXPath>
</ps:dataAccessor>
```

5 XPath Relationship Target Filter

A relationship target filter can be expressed as an XPath over documents instantiating the relationship target schema. If the XPath evaluates to 1 or more nodes, then the result of the filter is true, i.e. the relationship target is in scope. If the XPath evaluates to 0 nodes, then the result of the filter is false, i.e. the relationship target is out of scope.

When expressed as an XPath, the relationship target filter follows exactly the same format as given for the XPath Query Data Handle in Section 3, as shown in Figure 2.

The following is an example of an XPath relationship target filter. It returns true for a given relationship target only if the parameter name of the relationship target is *not* "http://www.example.com#divisor", i.e. it excludes all divisor data items from the scope of the provenance query.

```
<pq:relationshipTargetFilter>
  <pq:search>
    <xp:xpath>
      <xp:path>/pq:relationshipTarget[ps:parameterName!="http://www.example.com#divisor"]</xp:path>
      <xp:namespaceMapping>
        <xp:prefix>ps</xp:prefix>
        <xp:namespace>http://www.pasoa.org/schemas/023s1/PStruct.xsd</xp:namespace>
      </xp:namespaceMapping>
      <xp:namespaceMapping>
        <xp:prefix>pq</xp:prefix>
        <xp:namespace>http://www.pasoa.org/schemas/023s1/pquery/ProvenanceQuery.xsd</xp:namespace>
      </xp:namespaceMapping>
    </xp:xpath>
  </pq:search>
</pq:relationshipTargetFilter>
```

6 Conclusions

This document describes a concrete profile for expressing provenance queries using the XPath query language. It also defines a supporting data accessor format, also using XPath.

References

[Bra97] Scott Bradner. Key words for use in RFCs to indicate requirement levels. http://www.ietf.org/rfc/rfc2119.txt, 1997.

[MMG+06] Simon Miles, Luc Moreau, Paul Groth, Victor Tan, Steve Munroe, and Sheng Jiang. Provenance Query Protocol. Technical report, University of Southampton, June 2006.

[TGJ+06] Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, and Luc Moreau. WS Provenance Glossary. Technical report, Electronics and Computer Science, University of Southampton, 2006.

[W3C99] W3C. XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999. http://www.w3.org/TR/xpath, 1999.

ws-prov-soap

Authors:
Steve Munroe, U. Southampton
Paul Groth, U. Southampton
Sheng Jiang, U. Southampton
Simon Miles, U. Southampton
Victor Tan, U. Southampton
John Ibbotson, IBM
Luc Moreau, U. Southampton

August 24, 2006

A SOAP Binding for Provenance P-headers

Status of this Memo

This document provides information to the community regarding the specification of a data model for process documentation used to describe a SOAP binding of the process documentation model and has the status of a working draft. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

This document describes a SOAP binding for the process documentation p-header. It presents a specification of the p-header and can be considered an extension of the process documentation data model presented in [MGJ+06].

Contents

1 Introduction

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, a further obligation is placed 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, such as a SOAP message [Mit03], this information is termed the *p-header*, defined as follows.

Definition 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 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. This document presents a specification of the data model for the p-header.

A full overview document is available that describes the vision for the standardisation effort [TMG+06].

1.1 Goals and Requirements

The goal of this document is to define an open, interoperable model for the p-header and its location in a SOAP header.

1.1.1 Requirements

In meeting this goal, this document must address the following requirements:

- Define the data items necessary for the p-header and their logical organisation.
- Locate the p-header within a SOAP header.

- Provide the basis for an open, interoperable set of standards.
- Provide extensibility for more sophisticated and/or currently unanticipated scenarios.

2 Terminology and Notation

All definitions for the concepts and structures found within this document can be found in [TGJ+06].

2.1 XML Namespaces

The XML Namespace URI that MUST be used by implementations of this specification is: `http://www.pasoa.org/schemas/version023s1/PStruct.xsd`

Table 1 lists XML namespaces that are used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

Prefix	XML Namespace	Specification(s)
ph	`http://www.pasoa.org/schemas/version023s1/PHeader.xsd`	[P-Header]
xs	`http://www.w3.org/2001/XMLSchema`	[XMLSchema]

Table 1: Prefixes and XML Namespaces used in this specification

2.2 Notational Conventions

The keywords "MUST ", "MUSTNOT ", "REQUIRED ", "SHALL ", "SHALLNOT ", "SHOULD ", "SHOULDNOT ", "RECOMMENDED ", "MAY ", and "OPTIONAL " in this document are to be interpreted as described in [Bra97].

2.3 XML Schema Diagrams

This documents adopts a graphical notation to describe XML Schema. Figure 1 gives an example of a small XML Schema displayed as a diagram, which is now explained with reference to the figure.

Figure 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:stopTest`, either of which must contain a date.

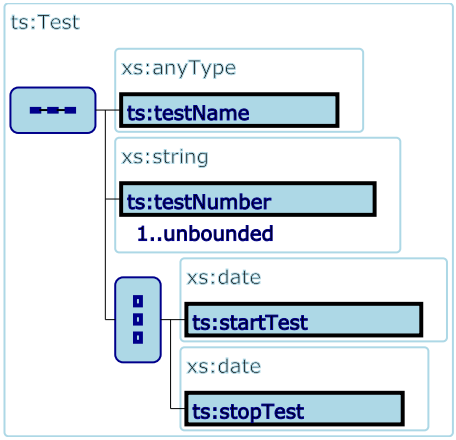


Figure 1: An example XML Schema diagram

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.

2.4 XPath notation

In addition to the XML Schema diagrams, an XPath notation [W3C99] is used to identify each individual element in the specification along with its context, in order to describe precisely its meaning and use.

3 The P-Header

The p-header's intended functionality is to provide a way for actors to pass meta-information about interactions. Though the model is technology neutral, we focus on the use of SOAP messages as the vehicle to pass around application messages and thus forms the basis of the interaction model in this document.

The p-header is constructed and placed within the header of a SOAP message. The information contained within a p-header is of three types. First, the p-header contains an interaction key. This key is used to link p-assertions made at different times and by different actors to a specific interaction. Any p-assertion made about a specific interaction should use the same interaction key. This enables queriers to later come along and find all those p-assertions about a given interaction by examining their interaction keys and selecting all those that have the same one. Thus, when an actor sends a message to another actor it must include an interaction key so that the receiver can use this within its own p-assertions about the interaction to point to that interaction. Along with the interaction key, a p-header contains an optional set of interaction metadata (see [MGJ+06] for the detailed specification of the elements defined within the p-header, i.e. `InteractionKey`, `InteractionMetada` and `InteractionContext`). This data contains provenance related information about the interaction such as pointers to the location where p-assertions made by the actor are stored, any tracers that are being used to denote a process and any other application specific information deemed necessary. The information described so far all refer to the current interaction, i.e. the contents of the SOAP message to which the p-header is attached.

The final form of information contained within a p-header is an optional set of interaction contexts relating to other interactions, i.e. interactions other than the one that the above discussed interaction key and interaction metadata are about. This provides the means to propagate view and object links around. This information includes the interaction key of the interaction being referred to, and other information relating to where p-assertions relating to this information are stored, i.e. interaction metadata about the interaction. The model of the p-header is shown in Figure 2.

The contents of a p-header are further described as follows:

`/ph:pheader`

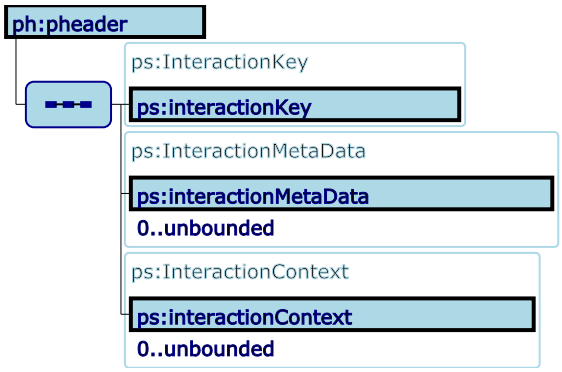


Figure 2: Model of the PHeader.

The root element of the p-header. It contains a sequence of three components that enable actors to add interaction context elements to messages: an interaction key, an OPTIONAL set of interaction metadata about the identified interaction, and an OPTIONAL set of interaction contexts about other interactions.

`/ph:pheader/ps:interactionKey`

The intent of this component is to uniquely identify the interaction whose message this p-header is attached. The full definition of `interactionKey` is given in [MGJ+06].

`/ph:pheader/ps:interactionMetaData`

The intent of this component is to hold meta data about the above identified interaction. The full description nd formal definition of `interactionMetaData` is given in [MGJ+06].

`/ph:pheader/ps:interactionContext`

The intent of this component is to provide information about the context of the above identified interaction by identifying those other interactions that are relevant to this one. The full description and formal definition of `interactionContext` is given in [MGJ+06].

3.1 The P-header's location in a SOAP Message

Given the full specification of the p-header as above, its location within a SOAP message can be described. Figure 3 shows a SOAP message and its component parts. A SOAP message contains an envelope that, as its name would suggest, serves as a container for the other elements of the SOAP message. Those other

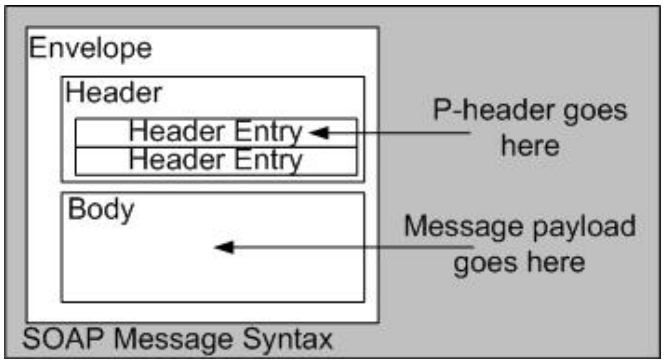


Figure 3: The p-header's location in a SOAP message

elements comprise the message body that contains the application specific information that is to be passed, as well as the SOAP header.

The SOAP header is used to supply extra information about the message that does not properly belong in the message body. For example, the information in the message body may need to be linked to other messages — it is the purpose of the SOAP header to carry such information. Since the p-header is designed to carry context specific information about p-assertions, that is, extra information about the enclosed p-assertions, then its natural place within a SOAP message is in the SOAP header. This is shown the figure, where the location of the p-header is shown as being within one of the SOAP header entries.

The XML snippet below shows a soap message with a p-header and its elements enclosed within the SOAP message's Header element. The namespaces shown under the Envelope element provide the prefixes for each of the parts of the p-header and the application data contained within the Body element.

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
              soap:encodingStyle="http://schemas.xmlsoap.org/soap/
              encoding/">
              xmlns:ph="http://www.pasoa.org/schemas/version023s1/PHeader.xsd">
              xmlns:ps="http://www.pasoa.org/schemas/version023s1/PStructure.xsd">
              xmlns:ap="http://www.application/data/Data.xsd">
  <soap:Header>
    <ph:pheader>
        <ps:interactionKey>...</ps:interactionKey>
        <ps:interactionMetaData>...</ps:interactionMetaData>
        <ps:interactonContext>...</ps:interactionContext>
    </ph:pheader>
  </soap:Header>
  <soap:Body>
    <ap:applicationData>010011100111</ap:applicationData>
  </soap:Body>
</soap:Envelope>
```

4 Conclusion

In this document the specification of the process documentation p-header was presented. The p-header is used to transfer provenance-based context information within the headers of application specific messaging protocols such as SOAP. This enables actors to relate messages to other messages and thus helps to bring together different views on a given interaction.

Appendix A

The following illustrates the p-header types and elements used in this document.

```
<?xml version="1.0" encoding="UTF-8"?> <xs:schema
targetNamespace="http://www.pasoa.org/schemas/version023s1/PHeader.xsd"
          elementFormDefault="qualified"
          attributeFormDefault="unqualified"
          xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns:ph="http://www.pasoa.org/schemas/version023s1/PHeader.xsd"
          xmlns:ps="http://www.pasoa.org/schemas/version023s1/PStruct.xsd">

   <xs:annotation>
        <xs:documentation>
           The PHeader schema
           Author: Paul Groth


           Copyright (c) 2006 University of Southampton
           See pasoalicense.txt for license information.
           http://www.opensource.org/licenses/mit-license.php

        </xs:documentation>
    </xs:annotation>


    <xs:import namespace="http://www.pasoa.org/schemas/version023s1/PStruct.xsd"
                    schemaLocation="./PStruct.xsd"/>

    <xs:element name="pheader" type="ph:PHeader"/>

    <xs:complexType name="PHeader">
        <xs:annotation>
            <xs:documentation>Provenance Specific Header Information</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="ps:interactionKey" maxOccurs="1" minOccurs="1" />
            <xs:element ref="ps:interactionMetaData" maxOccurs="unbounded" minOccurs="0"/>
            <xs:element ref="ps:interactionContext" maxOccurs="unbounded" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

References

[Bra97] Scott Bradner. Key words for use in RFCs to indicate requirement levels. http://www.ietf.org/rfc/rfc2119.txt, 1997.

[MGJ+06] Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, Victor Tan, and Luc Moreau. Data model for Process Documentation. Technical report, University of Southampton, June 2006.

[Mit03] N. Mitra. Soap version 1.2 part 0: Primer. http://www.w3.org/TR/soap12-part0/, 2003.

[TGJ+06] Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, and Luc Moreau. WS Provenance Glossary. Technical report, Electronics and Computer Science, University of Southampton, 2006.

[TMG+06] Victor Tan, Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. The Provenance Standardisation Vision. Technical report, University of Southampton, June 2006.

[W3C99] W3C. XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999. http://www.w3.org/TR/xpath, 1999.

ws-prov-gloss

Authors:
Victor Tan, U. Southampton
Paul Groth, U. Southampton,
Sheng Jiang, U. Southampton
Simon Miles, U. Southampton
Steve Munroe, U. Southampton
Luc Moreau, U. Southampton

October 23, 2006

WS Provenance Glossary

Status of this Memo

This document provides a glossary of terms intended to be used as a reference to other specification documents pertaining to the data model for process documentation [MGJ+06]. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

This glossary defines a set of terms used in the draft Provenance standard specification documents ([MGJ+06], [TMG+06b], [MTG+06], [TMG+06a], [GTM+06], [MMG+06b], [MMG+06a]) for the process documentation data model. The terms described here are intended to be implementation and technology independent, with the intent that they can be analyzed and applied to as many contexts as possible.

Contents

1 Introduction

The provenance of data and information generated within a computer system is the set of documentation that describes the processes that resulted in their creation. These processes can be viewed as sequences of causally related events that commence at some specific point in the past and eventually lead up to the event of creating the data or information of interest. The documentation of this sequence of events is based on a representational model that describes the storage, retrieval and processing of the documentation in a generic and technology independent manner. The data model for process documentation [MGJ+06] provides a detailed description of this model. This glossary assembles and defines a set of terms used in the model description. It also acts as a repository of all defined terms for the provenance standardisation documents related to the data model document.

1.1 Goals and Requirements

The goal of this document is to provide a set of defined terms used in all provenance standardization documents related to the data model for process documentation [MGJ+06].

1.1.1 Requirements

In meeting this goal, this document must address the following requirement:

- It must provide definitions for all terms used in all provenance standardization documents.

1.1.2 Non-Requirements

This document does not intend to meet the following requirement:

- Provide further clarification and context for the definitions provided. Such material will be provided in the documents from which these terms are collated from.

2 Provenance Glossary

Glossary

Actor

An actor is an entity with a distinct identity capable of undertaking some autonomous action within a provenance system. See also Asserting Actor and Querying Actor.

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.

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.

Asserter Identity

An asserter identity is an identifier for an actor that asserts p-assertions.

Asserting Actor

An asserting actor is an actor that creates or asserts a p-assertion, which may subsequently be recorded to a provenance store.

Assertion Category Policy

An assertion category policy specifies what categories of p-assertions a service can record.

Composite Documentation Style

A composite documentation style indicates that more than one atomic documentation style has been applied to a message.

Data Accessor

A data accessor is a reference to the location of a p-assertion data item within a p-assertion's content.

Data Staging Policy

A data staging policy specifies whether or not a provenance store is capable of data staging. If so, it identifies 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 data staging), or both.

Data Upload Policy

A data upload policy specifies if and how frequently a provenance store has to resolve a reference contained in a p-assertion.

Document Language Mapping

A document language mapping is a definition of how to transform documents formatted in one document language into another document language.

Documentation Style

Documentation style is a representation of the transformation according to which the content of a message is asserted in an interaction p-assertion or the state of an actor is asserted in an actor-state p-assertion.

Documentation Style Policy

A documentation style policy describes the various different kinds of documentation style a recording service offers.

Global P-Assertion Key

A global p-assertion key is used to uniquely identify a p-assertion throughout the provenance system, and consists of an interaction key, a view kind and the local p-assertion identifier for that p-assertion.

Index Management Policy

The index management policy states whether a provenance store is capable of performing indexing and, if it can, the different kinds of indexing offered by the store must be enumerated.

Interaction Context

A set of interaction metadata about an identified interaction.

Interaction Identifier

A value that is unique for a given interaction message sent from one message source to one message sink.

Interaction Key

A globally unique identifier for an interaction.

Interaction Metadata

Provenance-related data about an interaction.

Interaction P-Assertion

An interaction p-assertion is an assertion of the contents of an interaction message by an actor that has sent or received that message; the message must include information that allows it to be identified uniquely.

Interaction Record

An interaction record encapsulates all the p-assertions and identifiers related to one interaction, and is uniquely identified by an interaction key.

Internal Reference Documentation Style

The internal reference documentation style specifies 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.

Link

A link is a reference to another provenance store.

Local P-Assertion Identifier

A value that is unique for each p-assertion made by one asserting actor about one interaction.

Message Sink

The address to which an interaction message was sent.

Message Source

The address from which an interaction message was sent.

Object Identifier

An object identifier uniquely identifies the objects of an asserted relationship p-assertion.

P-Assertion

A p-assertion is an assertion that is made by an actor and pertains to a process. See also Actor State P-Assertions, Relationship P-Assertion and Interaction P-Assertion.

P-Assertion Category

A p-assertion category is classification of a p-assertion on the basis of the type of information it can record.

P-Assertion Data Item

Part, or all, of a p-assertion.

P-Assertion Data Key

A globally unique identifier for a p-assertion data item.

P-Header

The p-header of an interaction is provenance-related contextual information, sent along with the interaction message.

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.

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.

Parameter Name

A parameter name identifies an entity's role in a relationship, where that entity is documented by a p-assertion data item.

Process Documentation

The documentation of a process consists of a set of p-assertions made by the actors involved in the process.

Provenace Store Template Policy

A provenance store template policy indicates the ability of the store to accept templates from asserting actors, the languages for producing p-assertions the store supports, and the recording actors it can accept templates from.

Provenance of a piece of data

The provenance of a piece of data is the process that led to that piece of data.

Provenance Query Result Full Relationships

A provenance query result full relationship is the relationship between two p-assertion data items in the provenance of an entity found by a query.

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.

Provenance Store

A provenance store is a repository dedicated for purpose of storing p-assertions created by asserting actors, and subsequently retrievable by querying actors.

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.

Query Store Policy

A query store policy states which provenance stores a querying service has access to.

Querying Actor

A querying actor is an actor that retrieves p-assertions from a provenance store for purposes of answering process documentation related queries.

Recipient Store Policy

A recipient store policy specifies which provenance stores a service can record to.

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.

Reference-Digest Documentation Style

The reference-digest documentation style specifies 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.

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.

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 passertion.

Relationship Target Filter

A relationship target filter is a mechanism by which a querying actor can scope the provenance query results.

Search Language Policy

A search language policy identifies the search languages the provenance store supports.

Security Signature Checking Policy

A security signature checking policy specifies 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.

Security-encryption Documentation Style

The security-encryption documentation style specifies a transformation of a message by which a part of (or the whole of) its contents has been encrypted.

Security-signing Documentation Style

The security-signing documentation style specifies the transformation of a message by which a part of (or the whole of) its contents has been signed.

Subject Identifier

A subject identifier uniquely identifies a data item or message acting as the subject of an asserted relationship p-assertion.

Tracer

A tracer is a piece of information used to associate an interaction with other, related interactions on the basis of some shared information.

Verbatim Documentation Style

The verbatim documentation style denotes a null transformation applied to the contents of a message.

View

A view is the set of p-assertions asserted by an actor about a specific interaction.

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.

References

[GTM+06] Paul Groth, Victor Tan, Steve Munroe, Sheng Jiang, Simon Miles, and Luc Moreau. Process Documentation Recording Protocol. Technical report, University of Southampton, June 2006.

[MGJ+06] Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, Victor Tan, and Luc Moreau. Data model for Process Documentation. Technical report, University of Southampton, June 2006.

[MMG+06a] Simon Miles, Luc Moreau, Paul Groth, Victor Tan, Steve Munroe, and Sheng Jiang. XPath Profile for the Provenance Query Protocol. Technical report, University of Southampton, June 2006.

[MMG+06b] Simon Miles, Steve Munroe, Paul Groth, Sheng Jiang, Victor Tan, John Ibbotson, and Luc Moreau. Process Documentation Query Protocol. Technical report, University of Southampton, June 2006.

[MTG+06] Steve Munroe, Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. WSRF Data Model Profile for Distributed Provenance. Technical report, University of Southampton, June 2006.

[TMG+06a] Victor Tan, Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. A Profile for Non-Repudiable Process Documentation. Technical report, University of Southampton, June 2006.

[TMG+06b] Victor Tan, Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. Basic Transformation Profile for Documentation Style. Technical report, University of Southampton, June 2006.