

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

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

Copyright 2006.

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	3
1.1	Goals and Requirements	3
1.1.1	Requirements	3
1.1.2	Non-Requirements	4
2	Terminology and Notation	4
2.1	XML Namespaces	4
2.2	Notational Conventions	4
2.3	XML Schema Diagrams	5
2.4	XPath notation	6
3	The Process Documentation Model	6
3.1	The P-Structure	7
3.2	Interaction Views	8
3.3	Interaction P-Assertion Modelling	11
3.4	Identifying Interactions	12
3.5	Actor State P-Assertion Modelling	14
3.6	Relationship P-Assertion Modelling	15
3.7	Identifying P-Assertions	19
3.8	Identifying Data Items	20
3.9	Interaction Contexts and the P-Header	21
4	Conclusion	22

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⁺06a].
- 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	<code>http://www.pasoa.org/schemas/version023s1/PStruct.xsd</code>	[P-Structure]
ph	<code>http://www.pasoa.org/schemas/version023s1/PHeader.xsd</code>	[P-Header]
wsa	<code>http://schemas.xmlsoap.org/ws/2004/08/addressing</code>	[WS-Addressing]
xs	<code>http://www.w3.org/2001/XMLSchema</code>	[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.

`ts:testNumber`

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).

ts:test



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.

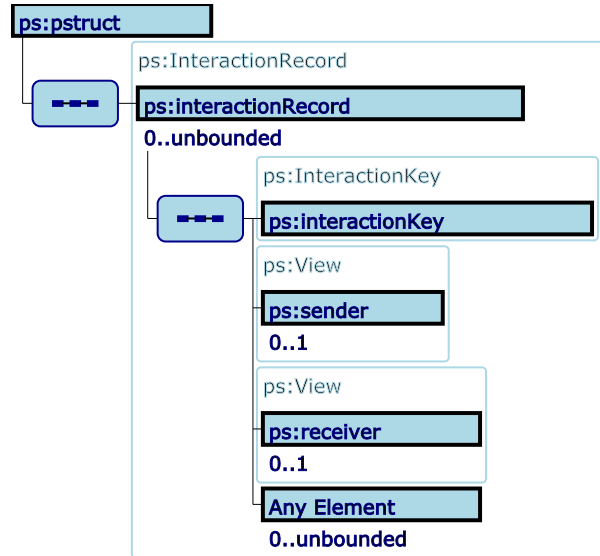


Figure 2: The P-Structure

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

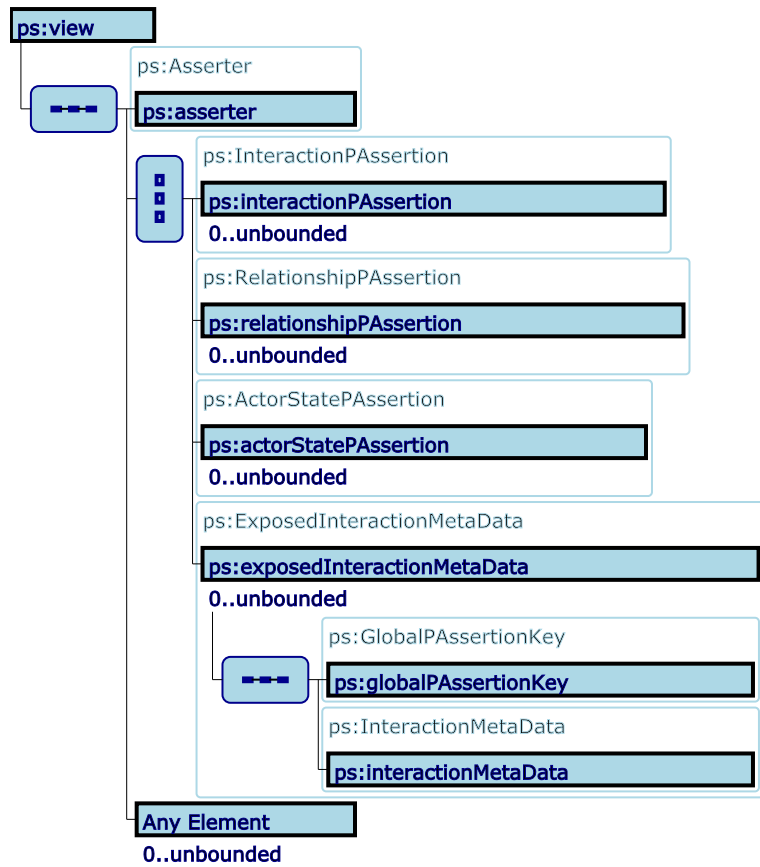


Figure 3: A View in the P-Structure

p-assertion is defined by an associated model described in Sections 3.3, 3.5 and 3.6.

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 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 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.

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**.

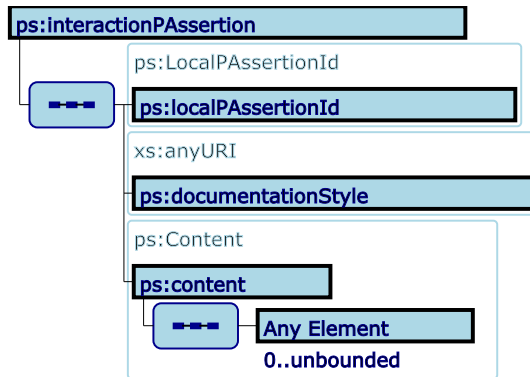


Figure 4: Model for an interaction p-assertion

`/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`.

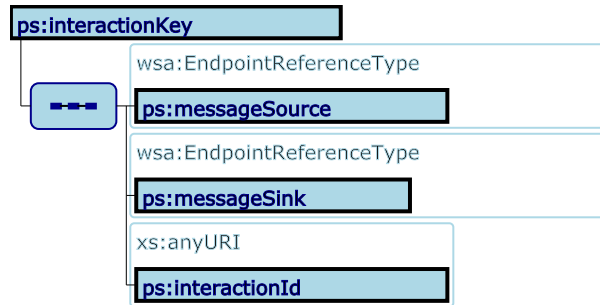


Figure 5: Model for identifying an interaction

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.

`/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.

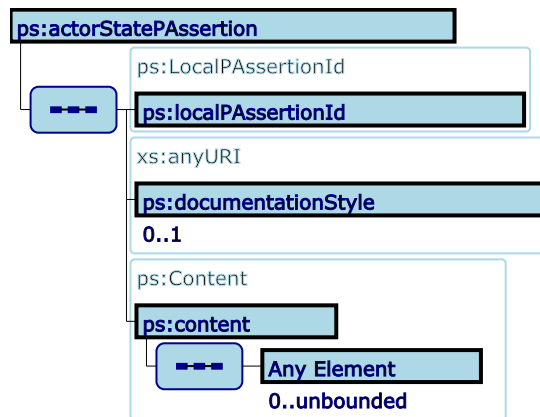


Figure 6: Model for an actor state p-assertion

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 (`objectIds`). 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.

`/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`.

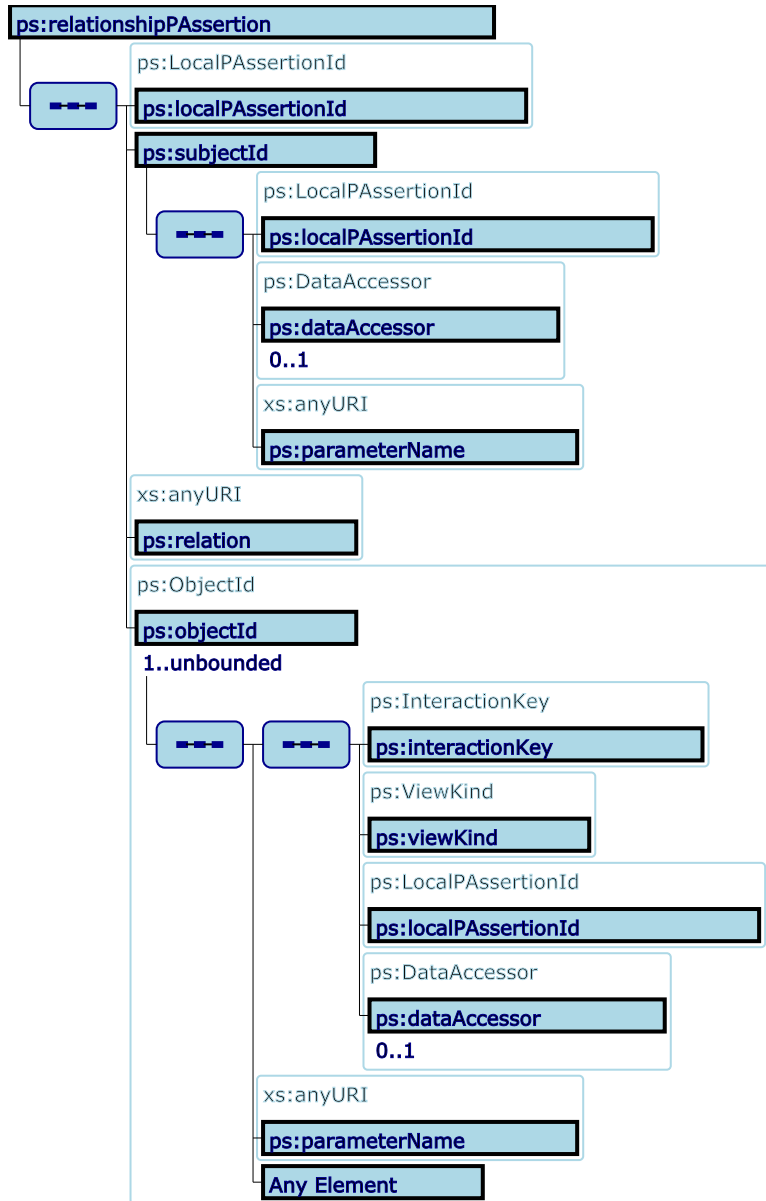


Figure 7: Relationship p-assertion model

`/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 `localPAssertionIdentifier`. 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.

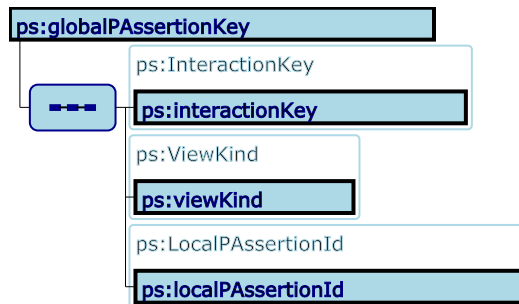


Figure 8: Global P-Assertion Key

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.

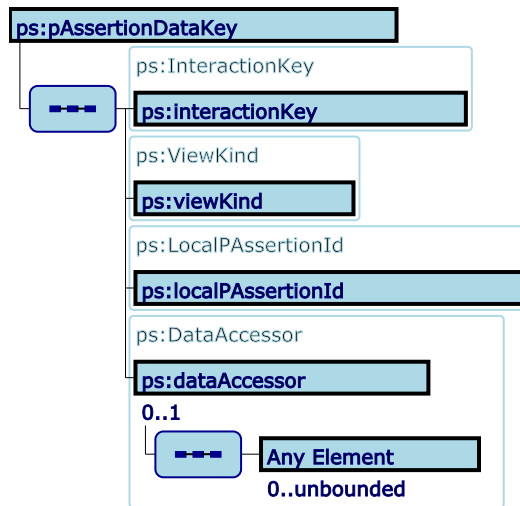


Figure 9: P-Assertion Data Key

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.

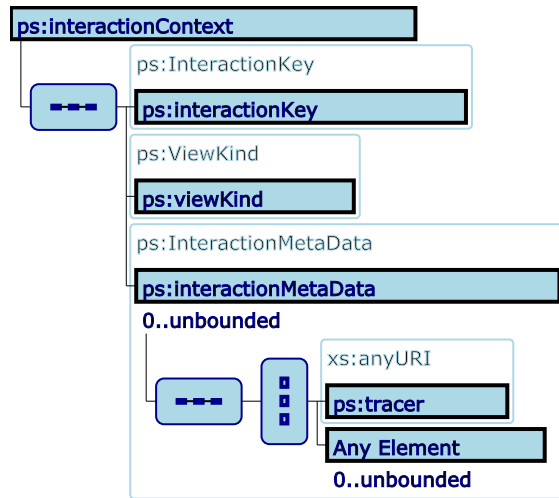


Figure 10: Model of an Interaction Context.

`/ps:interactionContext/ps:viewKind`

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⁺06a], security [TMG⁺06b], 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-structure 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: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:complexContent>
</xs:complexType>

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