Title:	*Provenance Store (Server) Implementation Design*
Workpackage:	*WP9*
Authors:	*John Ibbotson, Neil Hardman, Alexis Biller (IBM)*
Editor:	*Alexis Biller*
Reviewers:	*All project partners*
Identifier:	*D9.3.2*
Type:	*Deliverable*
Version:	*1.1*
Date:	*28 February 2006*
Status:	*Public*

Summary

This document provides an implementation design for the Provenance deliverable D9.3.2., the first public release of Provenance Store Server Implementation. It includes a description of the major interfaces, components and subcomponents together with a mapping to the architectural requirements it supports. This is a working document and is subject to change before the deliverable is complete.

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

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 e.V. Germany
- Universitat Politecnica de Catalunya Spain
- Magyar Tudományos Akadémia Számítástechnikai és Automatizálási Kutató Intézet Hungary

Forward

This document describes:

- Component overview for the first public release of Provenance implementation, Provenance Store Server version 1.0.
- How the Web Services Resources Framework (WSRF) has been applied to provide stateful offering.
- The storing of Provenance records (p-assertion) to an XML database, followed by subsequent retrieval and management.
- The considerations for set-up of a Provenance Store and runtime implications.

The primary audience of this document include: administrators and developers of the Provenance Store; and developers of Provenance Client Library.

The document has been edited by Alexis Biller (IBM) based on input from project partners.

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

Change History

0.1 Initial draft document for review by project partners

0.2 Further documentation and initial partner feedback:

- **Title**: to specify Server Implementation and thus differentiate from Client code.
- **Forward**: included to summarise key topic matter
- **Supported Requirements**: moved to appendix section and includes pointer to component in which requirement is met.
- **Non-Supported Requirements**: moved to appendix section and includes justification for non-support in Grid Provenance Server Implementation version 1.0.
- **Component Terminology**: moved into List of Acronyms (after Table of Contents)
- **The Provenance Store as a WS-Resource**: restructured and expanded.
- **Use of term "Provenance Store":** modified to Provenance Store Service
- **GT4 package naming:** altered to reflect server component
- **Manage Operation:** now includes justification following PASOA release and comments from University of Southampton.
- **Use of term "Provenance code":** re-phrased to be more explicit
- **Use of term "Provenance information":** changed to p-assertions
- **The eXist XML database:** additional commentary regarding justification of use
- **Data Storage:** corrected invalid use of InteractionKey
- **XML Insert and Append operations:** additional remarks to emphasise difference between the two operations
- **Influences on the Component Design:** removed High Availability, inserted additional remarks to National Language Support, Test Considerations and Test Scenarios.
- **Exception and Notification Handling:** inserted details of Provenance Exception and use throughout implementation, together with use in log4j logging.

0.3 Following further partner feedback

- **Title**: as Provenance Store (Server).
- **Forward**: primary audience
- **Introduction**: purpose and links to other Provenance documents
- **Reference to GT4 in package naming**: changed to WSRF to clearly indicate compliance
- **Choice of Data Repository:** created specific section for further clarification
- **Provenance Exception**: upon first use refers reader to section regarding such topic
- **WSRF Factory pattern**: additional figure and textual description.
- **Shared Stateful Resource**: removed. Was surplus information.
- **Uniqueness of InteractionKey**: relationship between architecture and implementation docs.
- **WS-Security**: added to both overview and design
- **Conclusion and further work**: included in an extra chapter before appendices
- **WSDL in appendix**: updated to PASOA version 0.22
- **UML in appendix**: added and referenced in body of document
- **Sequence diagrams in appendix**: added and referenced in body of document

1.0 Public release

- **Test scenarios**: reference specific section in Architecture
- **InteractionKey**: error conditions
- **Security**: further elaboration and details regarding properties in external files

1.1 Following Partner discussion

- **New Terminology:** added prior to Acronyms
- **Consistency of terminology**: reference to Provenance Store Server Implementation
- **Links to other docs**: refer to final architecture and include D4.2.1 security reference
- **Component Requirements**: added text to separate service and data store
- **Provenance Store Factory**: state what factory invocation returns and sets
- **Fig 3**: arrow from Destroy service to Resource
- **Packaging and Delivery Considerations**: binary and source download reference
- **Conclusion and further work**: enhanced application of WSRF as extension to 3 services
- **Bibliography**: Security Requirements added

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

Contents

New Terminology

The following terminology is introduced within this document as an extension to those of the Provenance Architecture [GJM+06]:

- **ProvenanceStore**: a WS-Resource, consisting of ProvenanceService and ProvenanceStore Resources
- **ProvenanceService**: the stateless Web Service
- **ProvenanceStore Resources**: persistent contents of database: p-assertions, interaction records and policies.

List of Acronyms

The following terms are used in this design document:

- **ACL:** Access Control List
- **AXIS**: The Apache SOAP engine provided as part of the GT4 release. See http://ws.apache.org/axis/
- **Collection**: A subdivision of an XML database.
- **CVS**: Concurrent Versions System. Open source tool for source code versioning.
- **DBMS**: Database Management System.
- **DOM**: The Document Object Model used for containing and searching XML documents or document fragments. See http://www.w3.org/DOM/
- **EPR**: Endpoint Reference. This document uses EPR to refer to a pointer to a WS-Resource.
- **eXist:** An open source implementation of an XML database. See http://exist.sourceforge.net/ and http://exist-db.org/webdb.pdf
- **GT4**: The Globus Toolkit 4 is a software toolkit which can be used to build Grid systems. It includes resources monitoring and discovery services, job submission infrastructure, security infrastructure and data management infrastructure. See http://www.globus.org
- **GT4 Container**: refers to combination of SOAP engine and an application server. Optionally also HTTP server. Provides runtime environment for a Web Service. GT4 includes a simple Java Web Services container based on Apache Axis (version 1.2 RC3).
- **GT4 WS Core**: a Java Runtime on which all the Globus-supplied Web Services resources have bee implemented. Examples include GRAM and RTF.
- **GAR**: Grid Archive. A single file containing all the data required by the Web Services container to deploy the service. The structured file includes the WSDL bindings, compiled stub classes from WSL and compiled service implementation.
- **OGSA**: Open Grid Services Architecture, defines a common, standard, and open architecture for grid based applications.
- **OGSA-DAI**: OGSA Data Access and Integration. Open source software that provides an abstraction layer from an underlying database technology. This includes a client API to provide a simpler interface to access the OGSA-DAI services. See http://www.ogsadai.org

- **PASOA**: Provenance Aware Service Orientated Architecture. See http://www.pasoa.org
- **Pivot Point:** The position in a Web Services interaction where AXIS handlers are invoked.
- **Tomcat:** The Apache Java based application server. The GT4 WS-RF implementation may be run as a service within Tomcat. See http://tomcat.apache.org/
- **XMLDB:** A programmatic interface to XML databases.
- **XPath:** An XML based language for searching XML documents and databases. See http://www.w3.org/TR/xpath
- **XUpdate**: An XML based language for updating XML documents/databases. See http://xmldb-org.sourceforge.net/xupdate/
- **WS:** Web Services, a service provided over the internet using protocols such as SOAP and HTML.
- **WS-Addressing**: Specifies a construct called an endpoint reference, containing at least a URL pointer to a Web Service.
- **WS-Resource**: The pairing of a Web Service with a resource. From the different mechanisms of addressing a WS-Resource the preferred way is by use of WS-Addressing specification.
- **WS-RF**: Specifies stateful Web Services through the management of WS-Resources. Formed as collection of specifications: WS-ResourceProperties, WS-ResourceLifetime, WS-ServiceGroup and WS-BaseFaults. WSRF can be expressed as the infrastructure for the OGSA architecture. See http://www.oasis-open.org/

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

1 Introduction

The components of the Provenance Store server implementation, the server setup and runtime, are presented within the framework of Web Services specifications and the architecture of Provenance. This document is aimed at a target audience of:

- **Administrators of a Provenance Store:** with responsibility for defining policy and executing set-up and review of runtime.
- **Developers of the Provenance Store:** those who continue to extend the features available from the server and apply corrections.
- **Developers of Provenance Client Library:** who will make server implementation features easier to use for developers of client applications.
- **Client Application Developers**: implementing actors which will interact with a Provenance Store, through the defined Provenance interfaces. Developers using Client Library API would not be direct consumers of this document.
- **Developers of PASOA**: to observe use of technology and contribute to further development and deployment progress.

1.1 Purpose of this document

This document outlines components that form the server implementation of a Provenance Store. These components are built on Web Service standards and apply models which are advised when applying WS specifications. The implementation must be seen in the context of Provenance Architecture, with extension for compliance with Web Services Resource Framework (WSRF).

The document is structured to convey an understanding of the implementation, with functional description, chapter 2, acting as preface to component design, chapter 3. The Web Services initially described in chapter 2 are cast into implementation for each component and extended from initial invocation to back-end updates to a database, including security considerations. Chapter 3 also details initiation, runtime and termination of key components such as Services container and XML database. Influences on implementation design are discusses in chapter 4, with external dependencies listed in chapter 5. Requirements provided by WP2 [And05] are tabulated in chapter 6. Two tables are included, the first for requirements which have been incorporated into the implementation; the second contains those which have not been included. Chapter 7 details packaging and delivery for the implementation, how to obtain resources for both running a Provenance system and further developing the implementation. Conclusion and future work are communicated in chapter 9. Outstanding issues are listed in chapter 10, detailing impact and mitigation. Building on the work of others has saved us from losing too much hair, to recognise this help a bibliography is provided in chapter 11. An appendix contains diagrams such as sequence diagrams, WSDL, UML and exception handling.

1.2 Links to other Provenance Documents

This document makes use of content in the following existing Provenance project documents:

- WP2: Requirements identified in D2.2.1. Those requirements which have been incorporated into the implementation are listed, together with components in which they can be found. Requirements which have not been incorporated are also listed, together with a justification.
- WP3: The final architecture document D3.1.1. This architecture is implemented, with extension for compliance with WS-RF. Test scenarios are built from discussion points in the architecture document.
- WP4: Security D4.2.1. Harnessing standard security features of Globus Toolkit permits authentication and authorisation to be achieved. Privacy concerns are also addressed.
- WP5: Scalability D5.2.2. By applying WS-RF model the deployment of Provenance into more demanding systems becomes feasible more easily.

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

2 Component Functional Description

2.1 Component Requirements

The purpose of the Provenance Services provide three Web-Service interfaces:

1. Record: provides an interface through which provenance assertions can be recorded.
2. Query: provides an interface through which the data in the provenance store can be queried.
3. Manage: provides an interface through which the administration of the Provenance Store can be performed.

The component provides access to the Provenance Store and through which the underlying data store for recording and retrieval of data. This component uses the services of the security component to establish the authenticity and authentication of users of the component.

The supported requirements and non-supported requirements are tabled in Section 5. This includes justification for non-support.

2.2 The Provenance Store as a Stateful Web Service

In their 2004 white paper Foster et al describe how stateful resources may be modelled with Web Services. In the paper, they describe a set of conventions intended to formalise how a service requestor interacts with state as represented by a set of stateful resources visible through an associated Web Service. This modelling approach leads to a set of conventions on using Web Service technologies.

In this section, we review this modelling approach and show how it can be used to describe the resources provided by a Provenance Store. The term “Provenance Store Resource” is introduced to refer to the extension to provide WSRF support. These resources then provide the basis for developing a scalable implementation of the Provenance architecture. For additional information see Scalability document [Ibb06].

2.2.1 The Web Service Environment

There are several components associated with a Web Service. These are shown in the following figure and then described in more detail.

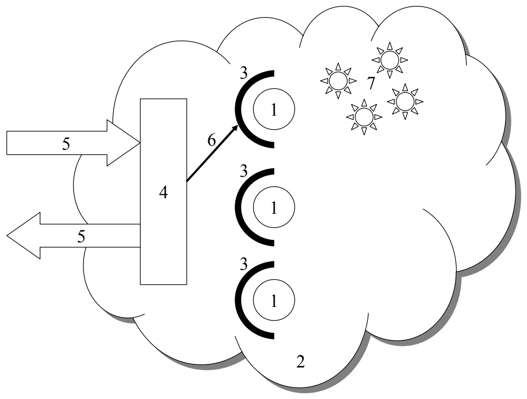


Figure 1 Components related to a Web Service

The *PROVENANCE* Project receives research funding from the European Community’s Sixth Framework Programme

In Figure 1, a Web Service (1) is a software component that performs some function. It provides a set of operations which access or manipulate the state held in other resources managed within the system. The Web Service is a component that is deployed within some runtime infrastructure (2). An example of this runtime infrastructure is an Application Server such as Tomcat or a proprietary product such as IBM's Websphere Application Server. The runtime environment is responsible for hosting the Web Service code and dispatching messages to the Web Service. The environment may also provide other qualities of service such as security and transactions.

The Web Service's interface (3) is described by an interface description language such as WSDL. It defines the Web Service's capabilities in terms of a collection of operations that can be invoked by service requestors in a distributed system. Each operation is described in terms of message exchanges that define the format of the message used to invoke an operation and the format(s) of messages returned by the Web Service as responses to the requestor; these also include the format of any fault messages.

The runtime environment also provides a message processing facility (4) that can receive messages (5) from service requestors. This component may support one or more transport protocols such as HTTP, SMTP or IIOP. The term *endpoint* is often applied to this component since it is made available to the distributed computing fabric at an advertised network address. Other responsibilities of the endpoint component include translating the message into a format that is understandable by the Web Service. For example, this may include converting the on-the-wire XML representation into a collection of Java objects and dispatching them (6) to the Web Service. The target Web Service is identified by the endpoint address (URI) and other components of the message. Note that the Web Services are deployed within a particular runtime environment at a particular network address, each Web Service in turn is itself uniquely identified by an address that combines the endpoint address together with some additional identification specific to each deployed Web Service.

The Web Service implementation is responsible for receiving the message and processing it. The processing may include interactions with other Web Services and stateful resources (7). Following the processing, the Web Service may format and send a response message to the client requesting the service. Note that these stateful resources can be shared between different Web Services; the decision to share a resource is made as part of an overall system design.

For Provenance, we need to consider a Web Service that provides access to or manipulates a set of logical stateful resources based on messages it sends and receives. Under these circumstances, the Web Service implementation itself is stateless since it delegates all management of state to another resource such as a database or file system. A stateless implementation of the Web Service enhances its security and scalability and is generally viewed as good engineering practice for Web Service implementations. A consequence of this statelessness requirement is that any dynamic state needed for a given message exchange must be:

- Provided explicitly within a request message whether directly by-value or indirectly by-reference, and/or
- Maintained implicitly within other system components with which the Web Service can interact.

Since the stateless Web Service implementation is interacting with a set of stateful resources, the identity of the stateful element(s) may either be passed as part of the Web Service request message or maintained as static data by the Web Service. Finally it is worth noting that although the implementation of the Web Service is stateless, the interface offered is clearly stateful since its behaviour is defined with respect to the underlying state.

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

2.2.2 Requirements for Modelling State in Web Services

Web services are an inherently stateless medium. All data passed during invocation is contained in the WS request. Persistence of data between interactions is therefore restricted. A stateful resource exists between interactions even when you're not interacting with it. As an analogy, a database exists even when you're not querying it. That state is defined by a group of properties. Change a property and the state changes, and vice-versa. A WS-Resource is the combination of a stateful resource and a Web service that interacts with it.

The following section extracts a set of requirements for modelling state within a Web Service and introduces the term WS-Resource.

- A WS-Resource is composed of a Web Service and a stateful resource such as a database or file system. The stateful resource has a specific set of state data that can be expressed as an XML document. The stateful resource has a well defined lifecycle and can be acted upon by one or more Web Services.
- WS-Resources can be created or destroyed.
- The definition of a stateful resource can be associated with the interface definition of a Web Service to enable well-formed queries to be submitted to the WS-Resource. The state of the WS-Resource can be queried and modified via Web Services message exchanges.

2.2.3 The Provenance Store as a WS-Resource

The implication of using WS-RF is that a WS-Resource needs to be created and destroyed. Creation is achieved by use of a Factory pattern and is further described in section 1.2.5. Destruction is requested by a calling application when submissions are complete, thus freeing server resources. The destroy operation is not implemented to delete any p-assertions held in an XML database; it simply removes the instance of a WS-Resource created on the server.

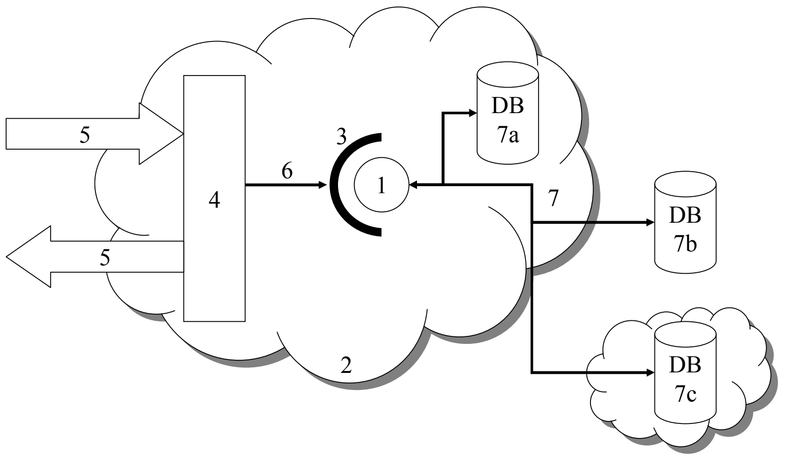


Figure 2 Components related to a Provenance Store

Figure 2 illustrates the components of a Provenance Store implemented using the WS-RF; it combines the Web Service (1) with a set of stateful resources (7). The interface (3) reflects the Submission, Query and Management operations identified in the Provenance Architecture.

The operations advertise the stateful interface to the underlying resources which typically consist of a set of databases though an implementation may include file systems as an alternative form of persistent store.

The databases may be positioned anywhere within a distributed computing infrastructure based on system design requirements and connected to the Web Service implementation by a remote connection mechanism. An alternative implementation strategy would be to treat the stateful resources as data services such as OGSA-DAI. In this case, the connection would be via a Web Services based protocol. Three examples of database positioning are illustrated:

- Within the same container as the Provenance Store (7a)
- As a completely separate managed component (7b)
- Within a separate, remote container (7c)

Note that these represent physical instances of a database management system (DBMS). WS-Resources may be mapped to these physical instances for finer granularity. For example, the following WS-Resources may be implemented by the same physical DBMS:

- Personal WS-Resources for different Actors and Users
- WS-Resources with different security policies
- WS-Resources that have different views of the Provenance Store (XML or Relational)

In general a physical instance of a DBMS can implement multiple stateful resources within a Provenance Store. For example, this can be observed by defining multiple collections in a single XML database, where each collection holds p-assertion related to a private instance of a Provenance Store.

Note that the general model of the WS-ResourceFramework (WSRF) is not restricted to data storage. Any architectural object such as a Provenance interaction or query result set can be considered a WS-Resource. The WS-RF supports the concept of notifications. This allows notifications to be published whenever a WS-Resource is changed. This pattern will be investigated in later versions of the Provenance Store implementation.

2.2.4 Provenance Store Factory

The factory pattern is the recommended approach for WS-RF in GT4; it has therefore been applied to the implementation for Provenance Server version 1.0. The factory pattern is commonly used in software development to create objects and is appropriate for creating instances of a Resource. A Resource factory is a Web Service that is capable of creating a Resource. To pair a Web Service with a resource the factory links the WS address with a resource ID, producing an Endpoint Reference (EPR). The factory is responsible for creating a ProvenanceStore Resource, achieved by:

1. Create the new stateful resource
2. Assigning an identity to the new resource
3. Create the association between the new resource and its Web Services implementation

The response message of the Resource factory contains a qualified EPR containing an identifier that refers to the newly created stateful resource. The factory returns the identifier to an Actor that requested the creation of the resource and places the identifier into a registry for later retrieval.

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

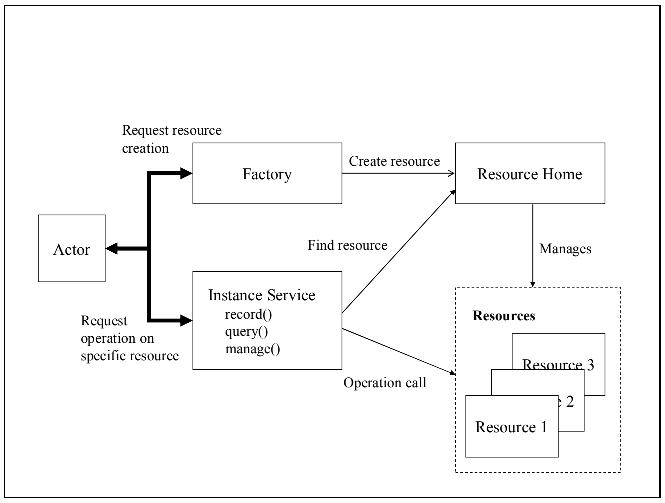


Figure 3: Resource Factory Pattern

The factory interface is provided to facilitate the creation of a new ProvenanceStore Resource. The factory is only used for the creation of new resources. It results in the creation of a container service. These new resources can be accessed through separate services, namely the standard Provenance defined service.

There may be many different types of Web Services that implement the Resource factory pattern. However, unless the request results in the creation of a new Resource with an identifier provided in the response message, the message exchange is not considered to be an instance of the WS-Resource factory pattern. This pattern is described simply as a usage pattern, not an explicit operation in its own right. The factory pattern may be implemented in a number of different Web Service operations that may for example create one or many WS-Resources.

For completeness, WS-Resources must be fully managed over their lifecycle. This implies an operation to destroy a Resource and free any resources it uses. The destroy operation has been implemented to only remove the Resource instance and does not delete data held in the XML database.

2.2.5 Web Services Security

This version of the reference implementation includes access control using the Globus Policy Decision Point mechanism described in the EU Provenance security specification [IT06].

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

2.3 Component Overview

Server implementation may be invoked by Provenance Client API, the interfaces and inter-operates with other components is described below.

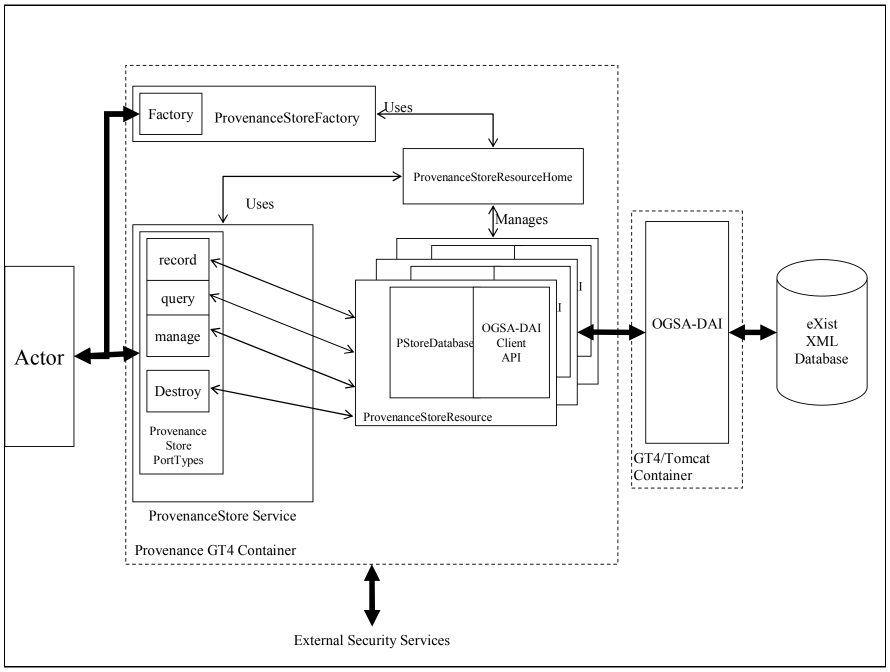


Figure 3: Provenance Store Components

The stateful Web Service pattern is described in the Stateful Web Service section of this document. We now identify the components of a Provenance Store that implement this pattern. These are shown in the Provenance Store Components figure.

1. The ProvenanceStoreFactory creates an instance of a ProvenanceStoreResource. This is an instance of a Resource accessed by an actor through the Provenance Store public record, query, manage and Destroy interfaces.
2. The ProvenanceStoreResourceHome manages a set of ProvenanceStoreResource objects. It is used by the ProvenanceStoreFactory to create new instances of ProvenanceStoreResource objects and by the ProvenanceStore implementation of the Web Service to access previously created Resources.
3. The ProvenanceStore is an implementation of the Provenance Store Web Service whose interfaces are defined by the record, query, manage and Destroy operations. These operations are defined as part of a ProvenanceStorePortType within the interface WSDL.
4. The ProvenanceStoreResource is an implementation of a WS-Resource. An instance of this Resource is created by the ProvenanceStoreFactory for use by an actor. The ProvenanceStoreResource may be destroyed using the Destroy operation. Many ProvenanceStoreResource objects can be created and managed in this implementation.
5. Persistent XML data storage is provided by an OGSA-DAI data service. This service may be hosted in the same container as the Provenance Store or alternatively on a separate, distributed container.
6. An eXist XML database provides the persistent database for the OGSA-DAI service.

These components are described in more detail in the Component Design section of this document. The reader is also directed towards the Provenance Client API documentation.

3 Component Design

3.1 Pre-requisite components

This Provenance Store implementation has the following pre-requisite components:

- Java 1.4.2 - http://java.sun.com (or IBM Java 1.4.2).
- Apache Ant 1.5 - http://ant.apache.org
- Globus Toolkit WSRF-compliant WS java container version 4.0.1 http://www.globus.org/toolkit/downloads/4.0.1/#wscore_bin
    - Contains Apache AXIS 1.2 RC2
- OGSA-DAI version OGSA-DAI WSRF 2.1 http://www.ogsadai.org.uk
- eXist 1.0 latest snapshot - http://exist.sourceforge.net

3.2 Component Details

This section provides more details of the components identified in the Component Overview. The GT4 WS Java Core in a Java implementation of WSRF. These Java classes are provided with a container to host the Web Services, together with scripts used to interact with Resources.

The appendix contains sequence diagrams which present the role of each component, both for resource creation and resource invocation.

3.2.1 The ProvenanceStoreFactory component

The WSDL describing the interface to the ProvenanceStoreFactory is shown in The ProvenanceStoreFactory Interface appendix. It consists of a single ProvenanceStoreFactoryPortType with a single createResource operation. The input message to the operation is of type CreateResourceRequest and is empty. The response is of type CreateResourceResponse. The response contains a WS-Addressing EndpointReference (EPR) to the newly created ProvenanceStoreResource. The EPR is used by subsequent invocations of the ProvenanceStore Web service to correctly identify which ProvenanceStoreResource should be used.

An example EPR serialized as a string is shown:

```
<ns1:ProvenanceStoreResourceReference xsi:type="ns2:EndpointReferenceType"
    xmlns:ns1="http://www.gridprovenance.org/schemas/ProvenanceStore"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <ns2:Address xsi:type="ns2:AttributedURI">
        http://9.20.147.198:8080/wsrf/services/ProvenanceStore
    </ns2:Address>
    <ns2:ReferenceProperties xsi:type="ns2:ReferencePropertiesType">
        <ns1:ProvenanceStoreResourceKey xmlns:ns1="http://org.gridprovenance.gt4">
            1832807742
        </ns1:ProvenanceStoreResourceKey>
    </ns2:ReferenceProperties>
    <ns2:ReferenceParameters xsi:type="ns2:ReferenceParametersType"/>
</ns1:ProvenanceStoreResourceReference>
```

The returned EPR contains the Address of the Web Service to be used to access the ProvenanceStoreResource together with a key to identify the newly created resource.

The GT4 client API allows a portType to be returned given an EPR as an input argument. The following client code fragment illustrates the 2 stage process of invoking the ProvenanceStoreFactory followed by the Provenance Store Web Service:

```
String factoryURI = "http://127.0.0.1:8080/wsrf/services/ProvenanceStoreFactory";
EndpointReferenceType factoryEPR, instanceEPR;
ProvenanceStoreFactoryPortType psFactory;
ProvenanceStorePortType pStore;

// Get the ProvenanceStoreFactory portType
factoryEPR = new EndpointReferenceType();
factoryEPR.setAddress(new Address(factoryURI));
psFactory = factoryLocator.getProvenanceStoreFactoryPortTypePort(factoryEPR);

// Create resource and get endpoint reference of WS-Resource.
// This resource is our Provenance Store "instance".
CreateResourceResponse createResponse = psFactory.createResource(new CreateResource());
instanceEPR = createResponse.getEndpointReference();

// Get instance PortType
pStore = instanceLocator.getProvenanceStorePortTypePort(instanceEPR);

// Create a Record structure .....
Record rec = new Record();

// ..... and invoke the Provenance Store Web service
RecordAck recAck = pStore.record(rec);
.....
```

Note: The ProvenanceStoreFactory class is contained in the org.gridprovenance.wsrf.server package.

3.2.2 The ProvenanceStoreResourceHome component

The ProvenanceStoreResourceHome class extends the GT4 ResourceHomeImpl class which implements the ResourceHome interface. The Provenance Store implementation has to provide a create() method which:

- Creates an instance of a ProvenanceStoreResource
- Initialises the resource
- Gets a key for the resource and adds it to the home set of keys
- Returns the key

The ProvenanceStoreResourceHome class is contained in the org.gridprovenance.wsrf.impl package.

3.2.3 The ProvenanceStoreResource component

The ProvenanceStoreResource class implements the GT4 Resource, ResourceIdentifier and ResourceProperties interfaces. Its purpose is to:

- Encapsulate the interactions between the Provenance Store Web Service and the OGSA-DAI data service
- Be a managed object that conforms to the lifecycle rules of WS-RF

The ProvenanceStoreResource.initialize() method is invoked by ProvenanceStoreResourceHome.create() when creating a new resource. This constructs a new instance of a PStoreDatabase that exists for the lifetime of the ProvenanceStoreResource instance.

The ProvenanceStoreResource also has methods for each of the three Provenance operations; record, query and manage. Each method receives as input argument the message sent by an actor to the Provenance Store Web service. It then invokes the appropriate method (record, query or manage) on the PStoreDatabase for processing. The returned acknowledgements from the PStoreDatabase are then returned by the ProvenanceStoreResource object.

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

WS-RF lifecycle management allows a resource to be destroyed using a Destroy operation to the ProvenanceStorePortType. This is shown in the ProvenanceStore Interface WSDL. The Destroy operation is implemented by a DestroyProvider operation which is part of the GT4 toolkit. This provider is loaded when the ProvenanceStore service is initialized by the GT4 container and is defined as part of the deploy-server.wsdd configuration file. Note that the Destroy operation does not destroy the persistent data stored in any database. It simply destroys the ProvenanceStoreResource object that encapsulates the interfaces to the persistent XML database.

The ProvenanceStoreResource class is contained in the org.gridprovenance.wsrf.impl package.

3.2.4 The ProvenanceStore component

The ProvenanceStore component implements an instance Web Service through which all interactions with the Provenance Store take place. Its interface is described by the WSDL in the ProvenanceStore Interface appendix.

The interface consists of a single WSDL port type called ProvenanceStorePortType containing the operations to support the three Provenance interfaces (record, query and manage) together with the Destroy lifecycle operation. See the previous section for details of the Destroy operation.

3.2.4.1 The Record Operation

The record operation messages are defined in the PRecord WSDL and its associated XML Schema file PRecord.xsd. The schema describes the record interface messages developed in the Provenance Architecture document.

The request object for the record interface is a Record object. This contains an array of IdentifiedContent objects, each one containing an array of Content objects that in turn contain either an individual p-assertion or a SubmissionFinished indicator. The record method of the ProvenanceStore class passes the Record object to the ProvenanceStoreResource object for processing. The ProvenanceStoreResource iterates through each IdentifiedContent object and calls the submit method of the PStoreDatabase object to record the entire contents to the database.

The response object for the record interface is a RecordAck object. This contains an array of RecordAckAck objects that correspond to each p-assertion and SubmissionFinished indicator that have been successfully inserted into the Provenance Store database.

A RecordAckAck contains a string describing the type of object that was inserted, and also the InteractionKey for the interaction under which they were inserted.

3.2.4.2 The Query Operation

The query operation messages are defined in the PQuery WSDL and its associated XML Schema file PQuery.wsdl.

The request object for the query interface is a QueryRequest object. This contains two elements:

- An XPath query expression
- An optional namespace mapping element

The response object for the query interface is a QueryResponse object. This contains a single element of any type.

3.2.4.3 The Manage Operation

The GridProvenance Architecture document (D3.1.1) did not specify management functionality as this is not intended to be standardised. The justification being that management operation is not deemed Provenance specific.

For the reference implementation the following operations have been defined with regards to the manage interface:

- CreateRepository
- DeleteRepository
- ListRepository
- UpdateRepository

Each operation is defined with input message and corresponding acknowledgement message.

3.2.5 The choice for data repository

IBM partner members made a design decision to adopt XML database technology as the best choice for storing Provenance data. This decision was based on the best knowledge available at the time regarding the feasibility of storage within an XML database, coupled with the ability to search using XPath. This implementation decision was presented to partners during early design discussions.

The choice of XML database as repository for p-assertions was made based on the format of the data exchanged between Web Services, i.e. XML; and the availability of Open Source XML Database implementation. The use of OGSA-DAI (see following section) does not preclude the use of another data repository at a later stage. For example, implementation could be written to shred the XML data across Relational database Table instead of into XML database Collection. Additionally, the use of OGSA-DAI displays the positive re-use of another European Community Framework resource.

3.2.6 The OGSA-DAI Data Service

The database access has been abstracted away from the core server side Provenance (Provenance Store) implementation by employing OGSA-DAI to own the connection to the database. This allows the Provenance Store implementation to focus on the key business functionality rather than database specific issues.

OGSA-DAI, is itself a web service that is hosted from a container such as Apache Tomcat or GT4. Just like the Provenance code, it uses the same client/server model with SOAP messages being sent between client applications and the OGSA-DAI service. The OGSA-DAI service is positioned at the pivot point of an AXIS handler chain.

OGSA-DAI provides two ways of interacting with a database:

- Perform document: This is an XML document that is constructed and passed to OGSA-DAI. It specifies the interaction to perform against the database, such as a query or an insert.
- Client API: This is a Java API for interacting with OGSA-DAI. It does not require the programmer to develop perform documents as it does this on our behalf. The API provides a neater, faster and more reliable way of using OGSA-DAI.

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

The Provenance database access implementation uses the OGSA-DAI client API to interact with the database. This OGSA-DAI client API is referenced in only a single Provenance Store class (PStoreDatabase.java).

The fact that the database is abstracted away does not hide all database problems from us. OGSA-DAI makes no attempt to "normalize" database access, so we are still subject to the problems inherent with databases from multiple vendors, such as varying data languages and varying support at the JDBC/XMLDB driver level.

As the database access code is being written to access an XML database, we must use XPath and XUpdate protocols to search and update the database. These protocols are exposed in the OGSA-DAI client API through the XPathQuery and XUpdate classes in the uk.org.ogsadai.client.toolkit.activity.xmldb package.

See section Known outstanding issues for details regarding XQuery support in OGSA-DAI.

3.2.7 The eXist XML database

OGSA-DAI provides support for two XML databases; eXist and Xindice from the Apache consortium. The eXist XML database has been widely accepted and received recognition through market awards. Additionally, previous experience and testing further assisted the selection of eXist database for this implementation.

The eXist database manages XML documents as a set of collections. A database can contain multiple collections with each collection containing multiple XML documents. Collections can be created and deleted within eXist. For storing p-assertions (Provenance records), a single collection is used called ProvenanceStore. This is referred to in the set-up documentation by a constant named "ProvenanceCollection". All documents within this collection conform to the p-structure type defined in the Provenance architecture. All p-structures contain a unique InteractionKey as specified in the Architecture document (section 8.2, rule 8.1). The names for the OGSA-DAI data service URI, database resource, and collection name are specified using the GT4 deploy-server.wsdd file as detailed in Administration section.

The following sections detail some of the issues encountered in developing with XML databases which so far are not as well established or mature as relational database technology. For more details of the eXist database, see the paper "eXist: An Open Source Native XML Database" available at http://exist-db.org/webdb.pdf.

3.2.7.1 XML Namespaces

XML uses namespaces to provide a means of avoiding naming conflicts when document structures are derived from multiple schemata. An XML document within eXist will contain declaration for each namespace used within the document. This takes the form of a prefix and a namespace value to which the prefix is associated. For example, the PASOA schema used by the Provenance project would have:

- Prefix: ps
- Value: http://www.pasoa.org/schemas/version022/PStruct.xsd

When the OGSA-DAI client API is used to record, query or update data in an XML database, the namespace prefix and values must be pre-registered with OGSA-DAI. The OGSA-DAI client API supports this for XPath and XUpdate. For an XPath query, the prefix and values are set by calling the setNamespaceBinding() method on the XPathQuery object.

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

For updating an existing document using XUpdate, the prefix and value must be included in the XUpdate modifications element as an attribute, a sample is provided:

```
<xu:modifications version="1.0"
     xmlns:xu=http://www.xmldb.org/xupdate
     xmlns:ps="http://www.pasoa.org/schemas/version022/PStruct.xsd">
     <xu:update select="/ps:pstruct/ps:interactionRecord">
      <xu:element name="ps:client">
          <ps:data> some data </ps:data>
      </xu:element>
    </xu:update>
</xu:modifications>
```

Note: The process that led to the creation of this sample is described in Insert and Append sections.

3.2.7.2 Data Storage

The p-assertions (Provenance records) are stored in the database as a p-structure document within a collection. Each p-structure document contains an IdentifiedContent that is keyed on the InteractionKey element. Within an XML database there is no implicit enforcement of unique keys, should a request to add a new entry with identical key be made then a duplication of keyed item will occur. Should an update request be expressed against the key, then both items will be updated. This feature of XML databases is countered within the implementation for Provenance Store (Server) by first issuing a query to check for any match in the database:

- no match found, then a fresh record is inserted into the database
- match found, the submitted data is appended to the existing record;

Further details regarding XML Update are detailed in Insert and Append sections.

The recording actor may request an InteractionKey from the Provenance Client API, or may generate its' own. The InteractionKey comprises three elements:

- interactionId: a URI
- messageSource: an address that can contain an array of any elements.
- MessageSink: an address that can contain an array of any elements.

In theory, any element could be placed in the messageSink and messageSource. However, for this implementation; the messageSource and messageSink are based on WS-Addressing EPRs. When testing with the eXist database, it was shown that eXist is intolerant to searching for the p-structure element if the Address elements contained in the messageSource and messageSink had a namespace that was different from that of the main document. To resolve this, a runtime transformation is applied: the namespace for the messageSource and messageSink contents has been changed from

- xmlns:wsa=http://www.w3.org/2005/03/addressing to
- xmlns:ps=http://www.pasoa.org/schemas/version022/PStruct.xsd.

Note: This work around should be re-visited in future implementations. A bug report to the eXist project may be submitted.

3.2.7.3 Recording Provenance

When the record operation on the ProvenanceStore Web service is invoked, a Record object is generated by AXIS and is passed to the service. The object contains the data from the recording actor to be recorded and the interface must determine whether to insert the data as a new p-structure document or whether to update an existing document. This difference is required as the XML XUpdate instruction is structured differently and must therefore be handled as two distinct cases. To identify which type of update is required the database must be searched to retrieve any p-structure

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

documents that have a key that corresponds to the one we are recording. The search is performed with an XPath statement of the form:

```
/ps:pstruct[ps:interactionRecord/ps:interactionKey/ps:interactionId ="http://interactionId/URI" and
ps:interactionRecord/ps:interactionKey/ps:messageSource//ps:Addr = "http://messageSource/URI"
and ps:interactionRecord/ps:interactionKey/ps:messageSink//ps:Addr = http://messageSink/URI"]
```

This returns an entire p-structure document if the key is matched, otherwise it will return an empty set. Note that this will return all matching documents, so if more than 1 document has the same key then they will all be returned.

Note that the messageSink/messageSource contains an element ps:Addr. This is not defined within the http://www.pasoa.org/schemas/version022/PStruct.xsd schema even though it has been given the same namespace prefix as elements that are. This is to work around a problem with the eXist database where elements that contain children with different namespaces could not be found (this is our interpretation of the problem; however the behaviour was somewhat unpredictable).

If the result set of the search is empty, then the Provenance Store inserts the p-structure into the database. If the result set is not empty, then it performs an append to the existing p-structure entry.

3.2.7.4 Insert operation

Inserts of p-assertions into the database are accomplished as follows:

A Record object is received by the PStoreDatabase.submit() method. This contains the data from the recording actor to be stored in the database. The IdentifiedContent element (which is a keyed record) is converted into a p-structure object and then written into the database by calling the OGSA-DAI client API. This is shown in the following code fragment:

```
// Create a pstruct from the identifiedContent.
PStructure pstruct = make_pstruct(collection, key, iDContent);

// Now serialize the PStructure into an XML string.
String result = Utils.PStructureToXMLString(pstruct);

XMLCreateResource xmlCreateResource = new XMLCreateResource(result);
xmlCreateResource.setParentCollectionName(collection);
service.perform(xmlCreateResource);
ActivityOutput output = xmlCreateResource.getOutput();
String response = output.getData();
```

3.2.7.5 Append operation

Appends of p-assertions into the database are accomplished in a manner similar to that described in the following pseudo-code:

```
Determine the type of View we're updating
Extract the appropriate View Element from the identified content
If we got an element
    We know the view already exists
    Extract all the p-assertions from the identified Content and add them to an array of appends
    If schema non-compliance detected
        Throw ProvenanceException
Else
    We need to create a new View of the appropriate type
    Extract all the p-assertions from the identified Content and add them to the View
    Convert the entire View into a single append entry
EndIf

For Each entry in the append array
   Append the entry to the document in the database
```

```
Done
```

Should non compliance with p-structure schema be detected following a request to append p-assertion then a *ProvenanceException* will be thrown. An example where this might occur is if two or more entries were submitted for *NumberOfExpectedPAssertions*. This is not permitted in the p-structure schema definition, only a single entry can be held per record per view. This allows a sender (client) or receiver (service) to submit only a single *NumberOfExpectedPAssertions*. Additionally, an exception would be reported if submission from an actor does match existing triple: InteractionKey, senderId and receiverId. This extends from Architecture rule 8.1 (from section 8.2).

Note: The Java code to support the Update function is much larger than the pseudo-code illustrated above.

3.3 Web Services Security

Globus certificates are used to identify both client and services in the WSRF compliant Provenance implementation. To validate client access, the signature of the client must be held in the access control list (ACL) for the Web Service. This ACL defines the operations which the client is permitted to invoke. Should the client attempt to access without correct authorisation then an Axis exception is thrown back to calling application. The security service is provided by the GT4 container and has no impact on the provenance implementation. A benefit from this model is that the security can be further developed in successive implementations; or rather deployment can apply various permutations of security.

For simplicity in the first release of Provenance Store (Server), the security is set on the service operations for ProvenanceStore and ProvenanceStoreFactory. This requires that security is defined on both the initial request to Factory, to create a WS-Resource; and the subsequent operations request to a specific resource. The generation of certificates can be achieved by using Globus security resources. These resources are packaged into the full GT4 implementation, but are not available in the WS Java Core. Note that the creation and management of user certificates forms part of the management of a GT4 environment and is out of scope for this implementation.

Access control to the Provenance Store is managed using entries in the provenance-authfile.xml file. In this file, users are identified with their certificate Distinguished Name (DN) and are permitted to access operations of the ProvenanceStoreFactory and ProvenanceStore service. The syntax for this file matches that of the Grimoires ACL file described at http://www.ecs.soton.ac.uk/~wf/grimoires/docs/security.html#Grimoires_ACL. To access a ProvenanceStore and its associated ProvenanceStoreResource, users should be permitted to invoke the createResource operation on the ProvenanceStoreFactory which allows them to create a ProvenanceStoreResource. Thereafter they may be permitted to access the ProvenanceStore operations at the discretion of the systems administrator.

This access control support is disabled in the release of the implementation when downloaded. To enable it, administrators should follow the instructions provided in the release documentation. The package org.gridprovenance.gt4.security includes the custom Policy Decision Point code to support the Provenance Store access control in the class ProvenanceAuthorisationPDP.java.

3.4 Initialisation and Termination

There will always be other permutations of component configuration other than that described in the application documentation. If the user opts for a different set-up then they should be aware of the implications. It may not be feasible to support each set-up. The following items describe the execution process together with the system administration.

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

3.4.1 Initialisation GT4

When a service request is made, the GT4 container assigns a thread to execute the request. This thread will call the AXIS engine to process the inbound request, de-serialize the SOAP message into a request object and then call the registered service implementation for that service passing the request object.

On entry into a service implementation, the AXIS thread of execution is able to access a context object that is managed by AXIS. This object provided to allow the service and all the handlers to pass information between each other. The context is accessible through a static AXIS API MessageContext.getCurrentContext().

3.4.2 Initialisation eXist

The eXist service can be exposed in several configurations, see eXist Administration. The documented approach is to launch as a stand-alone server. Invocation is achieved either by:

- OGSA-DAI Data Service requests, using the XML:DB API.
- eXist interactive client application, http://www.exist-db.org/client.html
- web interface, http://localhost:8080/exist (default port is 8080, but can be altered)

The default number of threads available for connection is 5, although this can be modified when launching the server. Should a connection request be received when there are no available threads in the pool, then it will block until one becomes available.

3.4.3 Termination GT4

On exit from a service implementation, the interface specifies that a response object must be passed back to AXIS (and ultimately for delivery to the client application).

Depending on the service type the response object will vary in structure.

3.4.4 Termination eXist

Upon completion of invoked task, the eXist thread of execution will be returned to the connection pool and the response passed to the calling thread. For Provenance, this will be returning to the OGSA-DAI Data Service request.

3.5 Administration

3.5.1 Administration GT4 and OGSA-DAI

There are values that must be set as parameter entries in the deploy-server.wsdd file used by GT4:

- provenanceServiceURI: the URI of the provenance data service exposed by OGSA-DAI
- provenanceDatabaseName: the name of the OGSA-DAI data resource
- provenanceCollectionName: the name of the eXist collection

If these parameters cannot be found, then a Provenance Exception will be thrown; see section Exception & Notification Handling. Changes to these values will require alterations to other property files; see Other Externals and Dependencies.

3.5.2 Administration eXist

Separate to the GT4 Container, which hosts the Provenance and OGSA-DAI Web Services, there is the eXist XML database service. This is run in a separate JVM as a stand-alone eXist server. The server can be started as a service,

This is by no means the only feasible server deployment, another configuration includes embedding the eXist database within a client application.

Some of the features of the different configurations are listed below:

- Stand-alone eXist
  - Separate JVM (should provide increases reliability)
  - Control the maximum number of threads available to client requests (will block until a thread is available)
  - No support for SOAP nor for Cocoon
- Embedded eXist
  - Same JVM as client application
  - No network listeners
  - Database not accessible outside of the client application

For more information refer to the eXist Developers Guide and Deployment Guide: http://exist.sourceforge.net/devguide.html and http://exist.sourceforge.net/deployment.html

3.6 Application Development Tooling

The design has no effect on Application Development tooling. Some descriptions may refer to files located in a shared Concurrent Versions System (CVS) repository for project partners.

4 Influences on the Component Design

4.1 Security

The design assumes that

- User authentication is managed by the GT4 container
- Access to the database is managed by the OGSA-DAI data service

4.2 Reliability Availability and Serviceability (RAS)

Log4J has been used throughout to provide the logging functionality. Log4J requires a log4j.properties file to be available and correctly configured in order for the logging trace-points to output their data.

4.3 Performance

The performance of this component is dependant on the ability to access and update the XML database. The initial intention was to perform all updates to the data as single operations to ensure efficient processing. However, the reality of using the XUpdate interface has meant that each p-assertion must be added individually, which is inefficient.

4.4 Monitoring and Statistics

None.

4.5 Platform Considerations

Use of Apache AXIS allows the use of AXIS parameters and access to the AXIS MessageContext.

4.6 National Language Support (NLS) and Internationalisation Considerations

Messages exposed to user are translated based on default locale or locale of calling method. A resource bundle is loaded which contains the translated messages for the application. A set of convenience methods are provided to retrieve and format the message from the source bundle.

4.7 Test Considerations

The test scenarios shall be based on the patterns detailed in the public document "Architecture for Provenance Systems" section 5.1 [GJM+06].

4.7.1 Test Scenarios

Whilst more will be added once test plan is completed, the initial test strategies are:

- Recording of a new interaction.
- Recording of additional data in an interaction.
- Recording of the returned ViewLink.
- Query the data stored in the database with an XPath query.

4.7.2 JUnit Strategy

JUnit tests can be used to test the submit method by passing a known Record object. However, as the development progresses, the need for testing the insertion of the ViewLinks will mean that testing will have to migrate to a more sophisticated framework where the link can be extracted from the

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

PHeader and appended into the view. This will require a real client/server setup rather than a simple JUnit setup.

4.7.3 Test Hooks

The test hooks are the principal method entry points for submit, query and manage.

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

5 Other Externals and Dependencies

This code relies on the OGSA-DAI Client API and the Apache AXIS API. The configuration of OGSA-DAI and Apache AXIS is done through configuration files that can be manually edited.

Component	File name	Location / Purpose
AXIS	deploy-server.wsdd	located in CVS root
		defines the exposed server services
		used at deploy time to generate server-config.wsdd in GT4 root
		Defined services include ProvenanceStore and ProvenanceStoreFactory.
	server-config.wsdd	located in GT4 server /etc/globus_wsrf_gridprovenance/
		generated during GT4 Provenance deploy based on data within deploy-server.wsdd in CVS
		defines the Provenance services.
	client-config.wsdd	located in CVS root
		defines client-side AXIS handlers for Globus WSRF security
		also found in GT4 root and OGSA-DAI root
		could be extended, for client application, to include provenance-handler to automate Interaction PAssertions submissions.

Component	File name	Location / Purpose
OGSA-DAI	data.service.resource.properties	located in CVS /docs/ (to be copied to server and used during server setup)
		permits OGSA-DAI to connect to eXist XML database
		defines resource identify, user credentials and eXist server parameters.
	NStoPkg.properties	located in CVS root
		defines mappings between namespace and implementation packages
		Includes ProvenanceStore, ProvenanceStore bindings and ProvenanceStore service.
	DatabaseRoles.xml	located in OGSA-DAI /skeletons/
		defines database credentials

Component	File name	Location / Purpose
GT4	deploy-jndi-config.xml	located in CVS root
		defines the resource parameters to the service definitions for both ProvenanceStore and ProvenanceStoreFactory
		ProvenanceStore service contains parameter for ACL file which needs to be uncommented and correctly set for when security is enabled.
		ProvenanceStoreFactory service contains parameters for security descriptor, ACL and loadOnStartup. These need to be uncommented and ACL value set for when security is enabled.
	jndi-config.xml	located in GT4 gt4/etc/globus_wsrf_gridprovenance/
		generated from deploy-jndi-config.xml to define the Provenance service parameters
	security-config-provenance.xml	located in CVS /etc/
		defines Globus security parameters
	provenance-authfile.xml	located in CVS /etc/
		defines authorisation list using signatures
	Windows-Certificates.txt	located in CVS /docs/
		describes how to setup Globus certificates in Windows.
	container-log4j.properties	located in GT4 root
		defines log4j properties to be used following GT4 container start-up
		example usage: to debug security settings

6 Supported and Non-Supported Requirements

This implementation will support the following requirements from the Provenance Store architecture:

ID	Description	Design Feature	Component
IR-PS-4	Provenance stores for an application can be deployed in security domains to which all querying actors have credentials to access. Tools can provide single-sign on by storing credentials and using as appropriate for each provenance store.	This recommendation may be supported by a federated security infrastructure with actors having access to multiple security domains via a single sign-on infrastructure.	GT4 via WS-Security
IR-PS-5	Process documentation cannot be removed from the data storage using the provenance store APIs; instead, it may be removed, following data retention policies, by making direct access to the storage layer, using curation methodology typical of long term storage.	The data storage system is a separate sub-component of the Provenance Store. The Provenance Store interfaces do not support the removal of process documentation, but this does not preclude its removal by direct access to the data storage by a user with an administration role.	
IR-PS-6	The access control functionality of the provenance store may provide a way to model the expression of access control policies and rules so that they are functionally and / or semantically equivalent to the access control policies of other data stores.	For a Provenance Store to implement the same access control policies as other data stores, it must share the definition of those policies. The implementation will allow access control policies to be shared with other data stores.	
IR-PS-7	Scalability of query results can be implemented by caching of the results within the provenance store. A set of interface operations allows actors to iterate and retrieve the cached results in response to a query request.	Query results will be implemented as a resource allowing querying actors to iterate over large result sets. This feature will be supported in addition to the basic operation of returning a result set in response to a query.	OGSA-DAI and DIAS
IR-PS-8	Management of large number of p-assertions can be addressed by implementing the provenance store persistent store component using a proprietary or open source database management system.	The Provenance Store persistent store component is accessed via an OGSA-DAI Data Service. This allows proprietary database components to be used.	OGSA-DAI
IR-PS-9	Implementations of the provenance recording, query and management interfaces should permit clustering.	The Provenance Store is identified by a single logical network endpoint address which may represent a clustered implementation. Identification of stateful resources managed by the Provenance Store will be by standardised Web Services specifications. The specifications permit clustering.	GT4 and WS-RF model facilitate clustering. A later test scenario may be created to verify this requirement.

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

This implementation will not support the following requirements from the Provenance Store architecture:

ID	Description	Design Feature	Justification
IR-PS-1	A provenance store implementation may use a metadata infrastructure such as RDF to annotate p-assertions with metadata information about time at which p-assertions are received through the recording interface or stored in persistent storage.	P-Assertion schemas include elements of any type that are intended for application specific information and metadata. These elements may be used to annotate p-assertions with RDF metadata for application specific processing.	This is deemed application specific. Context of p-assertion is open and thus Actor may submit with RDF triple using a suitably defined DocumentStyle.
IR-PS-2	A provenance store implementation may document its interactions with asserters by storing p-assertions in some provenance store (itself or another one). This allows the provenance to document back ups, replication, time, etc.	A Provenance Store can itself be an actor that records p-assertions. Recording may be to itself or a separate Provenance Store by including the actor side libraries in an implementation of the Provenance Store. The Provenance Store implementation developed as part of the project will not include this feature.	Not included in version 1.0 release of Provenance Server Implementation. If priority deemed sufficient, then this may be included in future release of Provenance Server Implementation.
IR-PS-3	A provenance store implementation may use a storage layer that offers replication of p-assertions in order to be fault-tolerant.	The Provenance Store implementation developed as part of this project separates the storage layer which is represented as a data service using OGSA-DAI. This separation allows a fault-tolerant database configuration to be used if required.	To date, XML databases do not support fault tolerance so this feature will not be supported by the project.
IR-PS-10	Implementations of provenance store should consider long term storage of p-assertions and the necessity to keep a long term copy of public keys (which may expire) to verify signed assertions.	Security lifecycle management considers the changes over an extended period of time of data security as threats to the data change.	This is an ongoing research and development subject and out of scope for the Provenance project.

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

7 Packaging and Delivery Considerations

Project partners may retrieve source code for building by use of project CVS repository. The build process results in a *gar* file for deployment to Globus. Additional support documentation and set-up files are provided within CVS project.

Binary and source code is available for download from external project server, subject to acceptance of registration terms. Software download details are available at the following URL: www.gridprovenance.org

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

8 Known Outstanding Issues

8.1 XQuery Support

OGSA-DAI does not currently support XQuery. This support is anticipated in 2Q2006 following which support will also be included in Provenance implementation.

8.2 Federated Query

The implementation to support federated query will be provided after the initial public release in February 2006. This will be achieved by inclusion of DIAS patterns.

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

9 Conclusion and further work

The documentation has presented the components which together form the server-side Provenance Store. The deployment of the system is presented in terms of each internal and external component. The application of standards, principally WS-ResourceFramework, is both presented in abstract and in the instance for this implementation.

The exposure of WSRF is an extension to Provenance services to record, query and manage. Future implementations shall apply further WSRF specifications to yield increased benefit from WS-Resource model to access provenance data.

This should now be viewed with consideration for client side library development. The linkage of both Client Library and Provenance Store is crucial for success of Provenance implementation. These need to be displayed to function correctly with Test Scenarios. It should also take into account deployment to more strenuous environments, where the number of actors or the size of submitted records may have implications on deployment configuration. Such considerations will also take look at implementation to cater for more complex query: both to permit XQuery and to return large result sets in a manner that the calling client can use. This is the direction that the implementation of the Provenance Store will take over the coming months, until September 2006.

10 Bibliography

[And05] A. Andics: EU Provenance Deliverable 2.2.1 Software Requirements Document. Public document published on www.gridprovenance.org project website.

[GJM+06] P. Groth, S. Jiang, S. Miles, S. Munroe, V. Tan, S. Tsasakou, L. Moreau: Architecture for Provenance Systems. Public document published on www.gridprovenance.org project website.

[Ibb06] J. Ibbotson: EU Provenance Deliverable 5.3.2 Scalability for Provenance. Public project document published on www.gridprovenance.org project website.

[IT06] J. Ibbotson and V. Tan : EU Provenance Deliverable 4.2.1 Security Requirements. Public document published on www.gridprovenance.org project website.

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

11 Appendices

11.1 Appendix A – Sequence Diagrams

The following diagrams are provided to assist comprehension of the role of each component.

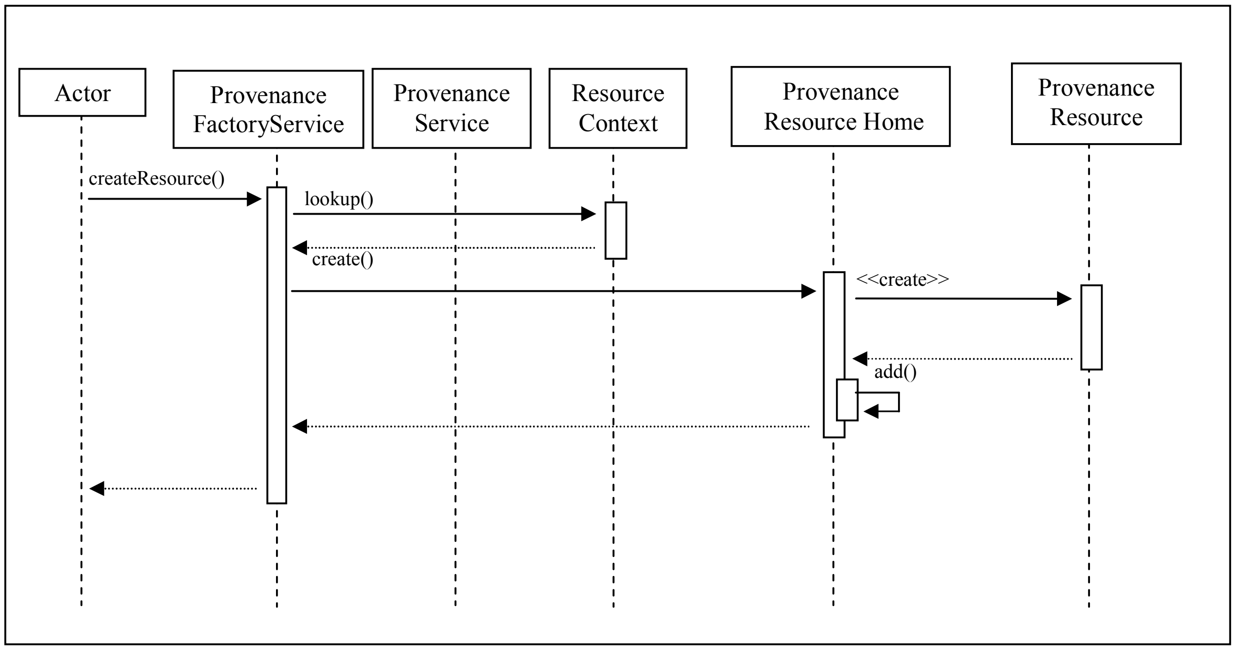


Figure 5: Sequence diagram for creating a Provenance Store Resource

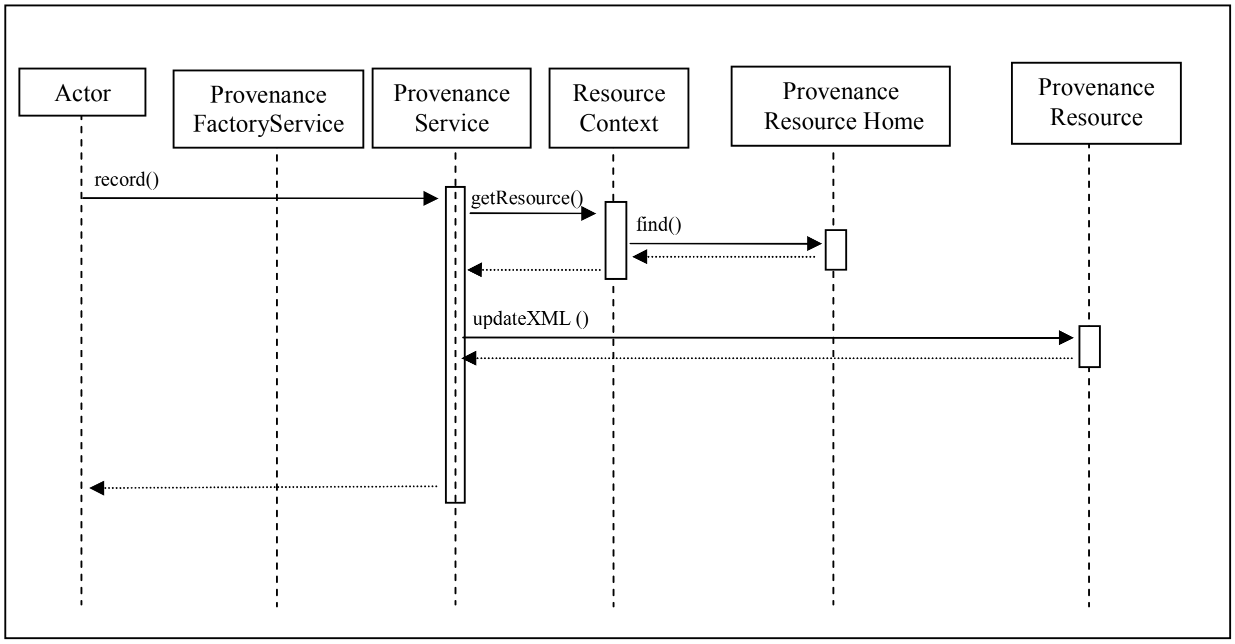


Figure 6: Sequence diagram for recording to a Provenance Store Resource

11.2 Appendix B – External APIs

The following external APIs are described using WSDL and XML Schemas. The PASOA (version 0.22) definitions have been imported into the WSRF compliant Provenance Store schema. Note that these are preliminary WSDL definitions of the interfaces and are subject to change as the project moves towards a standardised set of interfaces.

11.2.1 ProvenanceStoreFactory Interface

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- (c) Copyright International Business Machines Corporation 2005. -->
<!--              See CPL.txt for licensing information.             -->
<definitions name="ProvenanceStoreFactory"
    targetNamespace="http://www.gridprovenance.org/namespaces/ProvenanceStoreFactory"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://www.gridprovenance.org/namespaces/ProvenanceStoreFactory"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <documentation>
        The ProvenanceStoreFactory interface.
        Author: John Ibbotson
        Last Modified: 03/01/2006
    </documentation>

    <!-- T Y P E S -->
    <types>
        <xsd:schema
            targetNamespace="http://www.gridprovenance.org/namespaces/ProvenanceStoreFactory"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:tns="http://www.gridprovenance.org/namespaces/ProvenanceStoreFactory">

           <xsd:import namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
                       schemaLocation="../ws/addressing/WS-Addressing.xsd" />

           <!-- REQUESTS AND RESPONSES -->
           <xsd:element name="createResource">
               <xsd:complexType/>
           </xsd:element>
           <xsd:element name="createResourceResponse">
               <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element ref="wsa:EndpointReference"/>
                    </xsd:sequence>
               </xsd:complexType>
           </xsd:element>
        </xsd:schema>
    </types>

    <!-- M E S S A G E S -->
    <message name="CreateResourceRequest">
           <part name="request" element="tns:createResource"/>
    </message>
    <message name="CreateResourceResponse">
           <part name="response" element="tns:createResourceResponse"/>
    </message>

    <!-- P O R T T Y P E -->
    <portType name="ProvenanceStoreFactoryPortType">
       <operation name="createResource">
              <input message="tns:CreateResourceRequest"/>
              <output message="tns:CreateResourceResponse"/>
           </operation>
    </portType>
</definitions>
```

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

11.2.2 ProvenanceStore Interface

```
<?xml version="1.0"?>
<!-- (c) Copyright International Business Machines Corporation 2005. -->
<!--              See CPL.txt for licencing information.             -->
<definitions name="ProvenanceStore"
    targetNamespace="http://www.gridprovenance.org/namespaces/ProvenanceStore"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://www.gridprovenance.org/namespaces/ProvenanceStore"
    xmlns:prw="http://www.pasoa.org/schemas/version022/PRecord.wsdl"
    xmlns:pqw="http://www.pasoa.org/schemas/version022/WSXQuery.wsdl"
    xmlns:pmw="http://www.pasoa.org/schemas/version022/PManage.wsdl"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsrlw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.wsdl"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
    xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-
01.wsdl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <documentation>
        The ProvenanceStore interface.
        Author: John Ibbotson
        Last Modified: 03/01/2006
    </documentation>

    <import namespace="http://www.pasoa.org/schemas/version022/PRecord.wsdl"
              location="./PRecord.wsdl"/>
    <import namespace="http://www.pasoa.org/schemas/version022/WSXQuery.wsdl"
              location="./WSXQuery.wsdl"/>
    <import namespace="http://www.pasoa.org/schemas/version022/PManage.wsdl"
              location="./PManage.wsdl"/>
    <import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-
01.wsdl" location="../wsrf/lifetime/WS-ResourceLifetime.wsdl" />

    <!-- T Y P E S -->
    <types>
           <xsd:schema
                targetNamespace="http://www.gridprovenance.org/namespaces/ProvenanceStore"
                xmlns:tns="http://www.gridprovenance.org/namespaces/ProvenanceStore"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema">
               <!-- RESOURCE PROPERTIES -->
               <xsd:element name="Name" type="xsd:string"/>
               <xsd:element name="ProvenanceStoreResourceProperties">
                   <xsd:complexType>
                       <xsd:sequence>
                            <xsd:element ref="tns:Name" minOccurs="1" maxOccurs="1"/>
                        </xsd:sequence>
                   </xsd:complexType>
               </xsd:element>
           </xsd:schema>
    </types>

    <!-- P O R T T Y P E -->
    <portType  name="ProvenanceStorePortType"
              wsrp:ResourceProperties="ProvenanceStoreResourceProperties">
           <operation name="record">
               <input message="prw:Record"/>
               <output message="prw:RecordAck"/>
           </operation>

           <operation name="query">
               <input message="pqw:Query"/>
               <output message="pqw:QueryAck"/>
            </operation>

            <operation name="CreateRepository">
               <input message="pmw:CreateRepository"/>
               <output message="pmw:CreateRepositoryAck"/>
            </operation>
            <operation name="DeleteRepository">
               <input message="pmw:DeleteRepository"/>
               <output message="pmw:DeleteRepositoryAck"/>
            </operation>
            <operation name="ListRepositories">
```

```
                <input message="pmw:ListRepositories"/>
                <output message="pmw:ListRepositoriesAck"/>
            </operation>
            <operation name="updateRepository">
                 <input message="pmw:UpdateRepository"/>
                 <output message="pmw:UpdateRepositoryAck"/>
            </operation>

            <wsdl:operation name="Destroy">
                <input message="wsrlw:DestroyRequest"
                       wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime/Destroy"/>
                <output message="wsrlw:DestroyResponse"
                        wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime/DestroyResponse"/>
                <fault
                    message="wsrlw:ResourceNotDestroyedFault"
                    name="ResourceNotDestroyedFault"/>
                <fault
                    message="wsrlw:ResourceUnknownFault"
                    name="ResourceUnknownFault"/>
            </wsdl:operation>
    </portType>
</definitions>
```

The WSRF compliant ProvenanceStore includes the following WSDL files:

11.2.2.1 PRecord WSDL

```
<?xml version="1.0"?>
<definitions name="PRecord"
             targetNamespace="http://www.pasoa.org/schemas/version022/PRecord.wsdl"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:tns="http://www.pasoa.org/schemas/version022/PRecord.wsdl"
             xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
             xmlns:pr="http://www.pasoa.org/schemas/version022/PRecord.xsd"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema">

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

    <import namespace="http://www.pasoa.org/schemas/version022/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>
```

11.2.2.2 WSXQuery WSDL

The following WSDL shall be modified to include the full WSXQuery version 0.22 following release of Provenance version 1.0. The updated version shall provide extended query capabilities.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="WSXQuery"
             targetNamespace="http://www.pasoa.org/schemas/version022/WSXQuery.wsdl"
             xmlns:xs="http://www.w3.org/2001/XMLSchema"
             xmlns:tns="http://www.pasoa.org/schemas/version022/WSXQuery.wsdl"
             xmlns:xq="http://www.pasoa.org/schemas/version022/WSXQuery.xsd"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

    <documentation>
        An XQuery interface for Web Services
        Author: Simon Miles
        Last Modified: 25 January 2005
    </documentation>

    <import namespace="http://www.pasoa.org/schemas/version022/WSXQuery.xsd"
location="./WSXQuery.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>
```

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

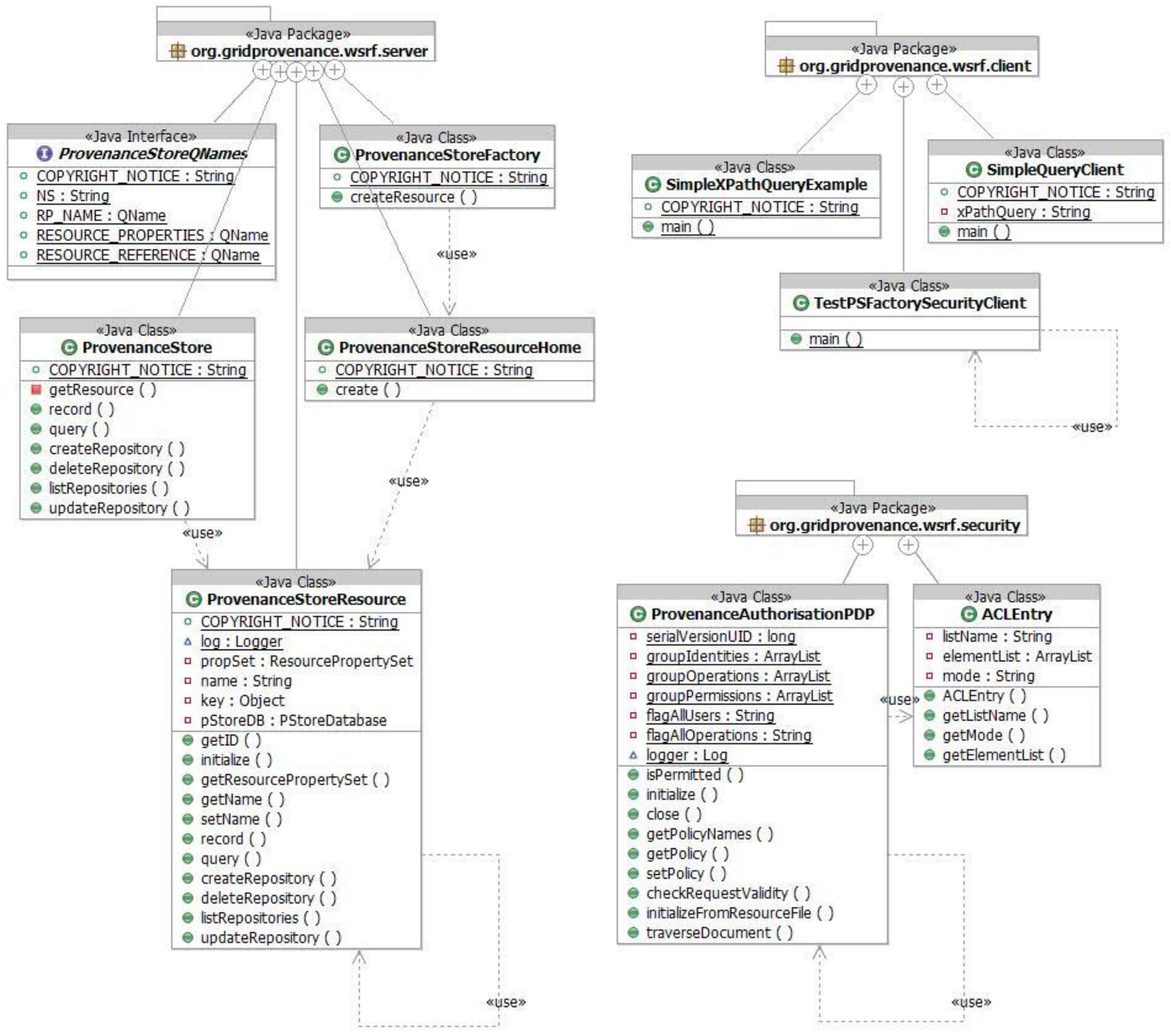
11.2.2.3 PManage WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="PManage"
    targetNamespace="http://www.pasoa.org/schemas/version022/PManage.wsdl"
    xmlns:tns="http://www.pasoa.org/schemas/version022/PManage.wsdl"
    xmlns:pm="http://www.pasoa.org/schemas/version022/PManage.xsd"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:p="http://www.w3.org/2001/XMLSchema">

    <documentation>
        The Provenance Store management port type and messages.
        Author: Paul Groth
        Last Modified: 17 November 2003
    </documentation>

    <import namespace="http://www.pasoa.org/schemas/version022/PManage.xsd"
               location="./PManage.xsd"/>

    <!-- A Provenance Store may have one or more repositories that act as independently from one
another. Fundamentally, this can be seen as a Provenance Service having one or more provenance stores
-->
    <portType name="ManagePortType">
        <operation name="CreateRepository">
           <input message="tns:CreateRepository"/>
           <output message="tns:CreateRepositoryAck"/>
        </operation>
        <operation name="DeleteRepository">
           <input message="tns:DeleteRepository"/>
           <output message="tns:DeleteRepositoryAck"/>
        </operation>
        <operation name="ListRepositories">
           <input message="tns:ListRepositories"/>
           <output message="tns:ListRepositoriesAck"/>
        </operation>
        <operation name="updateRepository">
           <input message="tns:UpdateRepository"/>
           <output message="tns:UpdateRepositoryAck"/>
        </operation>
    </portType>

    <message name="CreateRepository">
        <part name="body" element="pm:createRepository"/>
    </message>

    <message name="CreateRepositoryAck">
        <part name="body" element="pm:createRepository"/>
    </message>

    <message name="DeleteRepository">
        <part name="body" element="pm:deleteRepository"/>
    </message>

    <message name="DeleteRepositoryAck">
        <part name="body" element="pm:deleteRepository"/>
    </message>

    <message name="ListRepositories">
        <part name="body" element="pm:listRepositories"/>
    </message>

    <message name="ListRepositoriesAck">
        <part name="body" element="pm:listRepositoriesAck"/>
    </message>

    <message name="UpdateRepository">
       <part name="UpdateRepository" element="pm:updateRepository"/>
    </message>

    <message name="UpdateRepositoryAck">
       <part name="UpdateRepositoryAck" element="pm:updateRepositoryAck"/>
    </message>
</definitions>
```

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

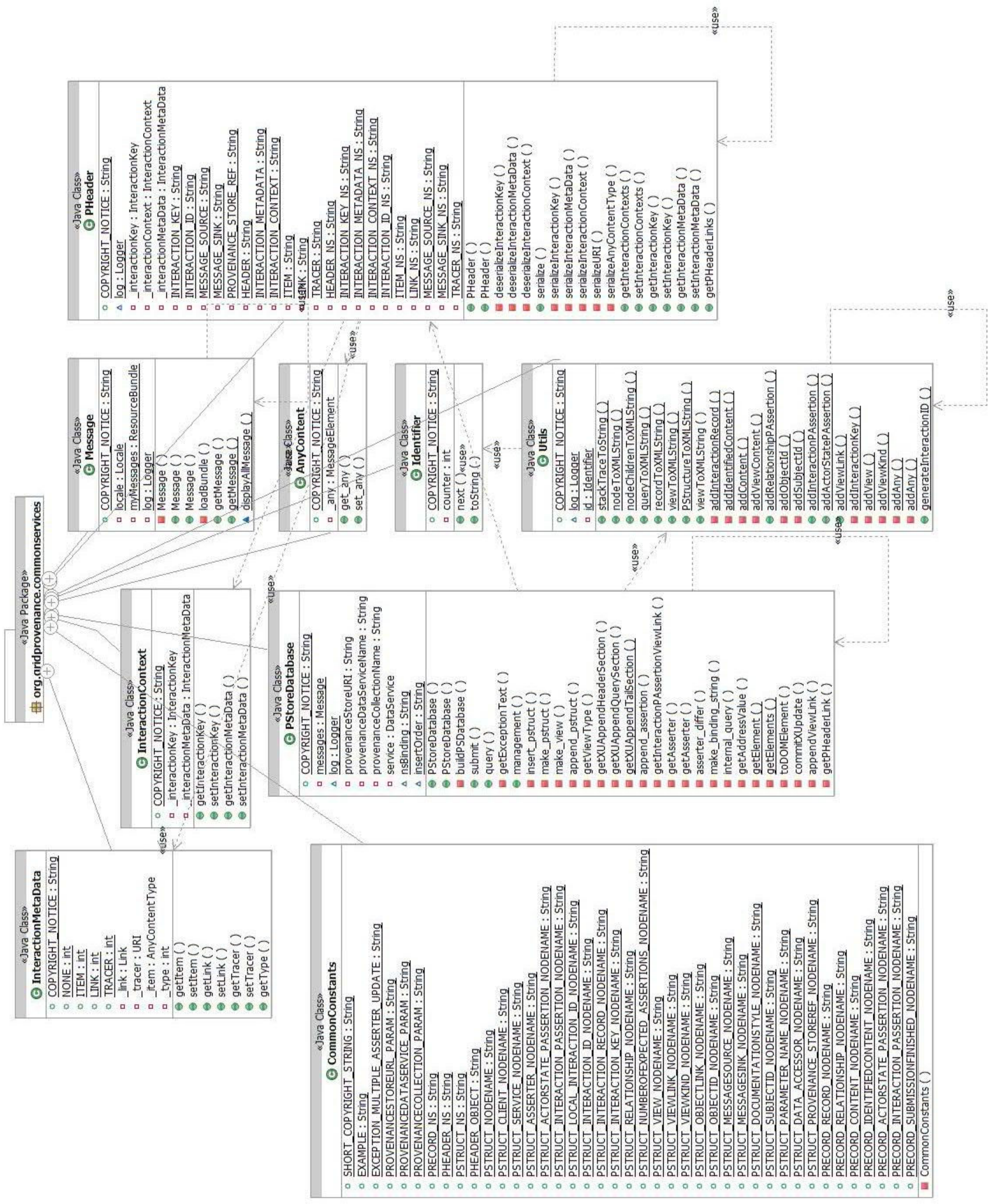
11.3 Appendix C – UML Class Diagrams

11.3.1 Class diagram for org.gridprovenance.wsrf package, including security



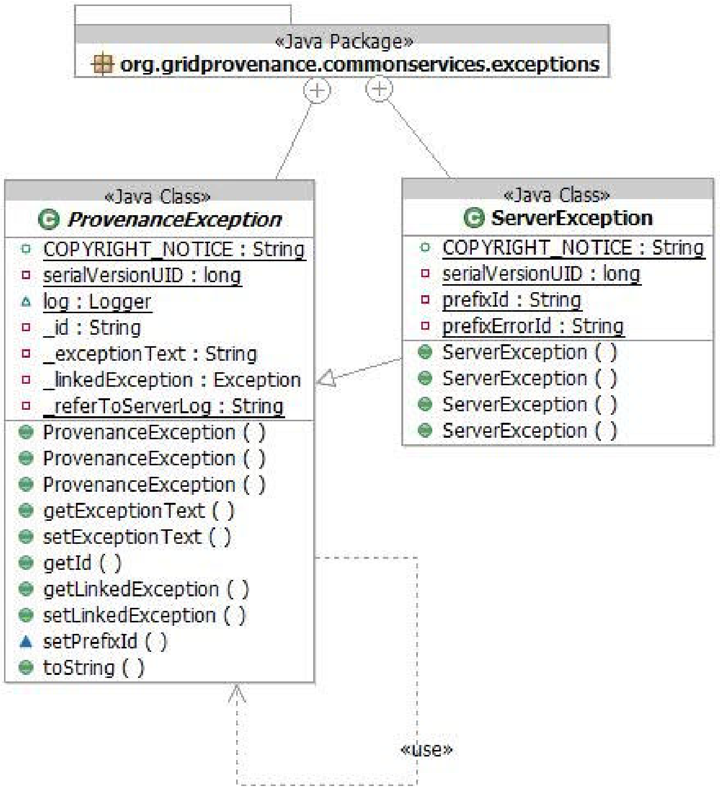
The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

11.3.2 Class diagram for org.gridprovenance.commonservices package

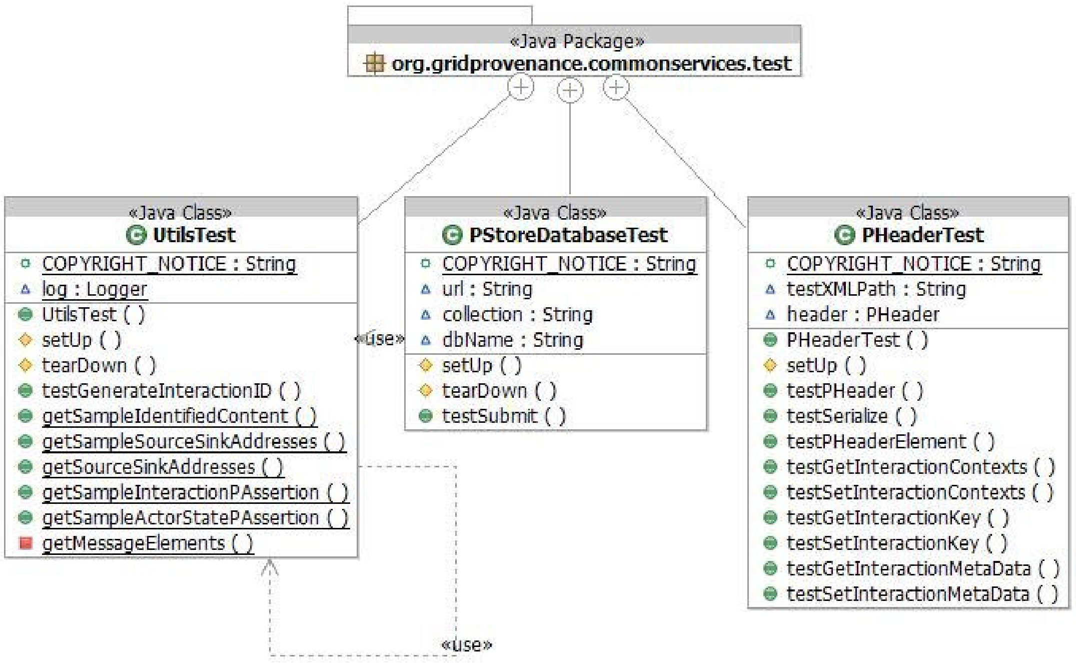

«Java Package» org.gridprovenance.commonservices
«Java Class» PHeader
COPYRIGHT_NOTICE : String
log : Logger
_interactionKey : InteractionKey
_interactionContext : InteractionContext
_interactionMetaData : InteractionMetaData
INTERACTION_KEY : String
INTERACTION_ID : String
MESSAGE_SOURCE : String
MESSAGE_SINK : String
PROVENANCE_STORE_REF : String
HEADER : String
INTERACTION_METADATA : String
INTERACTION_CONTEXT : String
ITEM : String
LINK : String
TRACER : String
HEADER_NS : String
INTERACTION_KEY_NS : String
INTERACTION_METADATA_NS : String
INTERACTION_CONTEXT_NS : String
INTERACTION_ID_NS : String
ITEM_NS : String
LINK_NS : String
MESSAGE_SOURCE_NS : String
MESSAGE_SINK_NS : String
TRACER_NS : String
PHeader ( )
PHeader ( )
deserializeInteractionKey ( )
deserializeInteractionMetaData ( )
deserializeInteractionContext ( )
serialize ( )
serializeInteractionKey ( )
serializeInteractionMetaData ( )
serializeInteractionContext ( )
serializeURI ( )
serializeAnyContentType ( )
getInteractionContexts ( )
setInteractionContexts ( )
getInteractionKey ( )
setInteractionKey ( )
getInteractionMetaData ( )
setInteractionMetaData ( )
getPHeaderLinks ( )
«Java Class» Message
COPYRIGHT_NOTICE : String
locale : Locale
myMessages : ResourceBundle
log : Logger
Message ( )
Message ( )
Message ( )
loadBundle ( )
getMessage ( )
getMessage ( )
displayAllMessage ( )
«Java Class» AnyContent
COPYRIGHT_NOTICE : String
_any : MessageElement
get_any ( )
set_any ( )
«Java Class» Identifier
COPYRIGHT_NOTICE : String
counter : int
next ( )
toString ( )
«Java Class» Utils
COPYRIGHT_NOTICE : String
log : Logger
id : Identifier
stackTraceToString ( )
nodeToXMLString ( )
nodeChildrenToXMLString ( )
queryToXMLString ( )
recordToXMLString ( )
viewToXMLString ( )
PStructureToXMLString ( )
viewToXMLString ( )
addInteractionRecord ( )
addIdentifiedContent ( )
addContent ( )
addViewContent ( )
addRelationshipPAssertion ( )
addObjectId ( )
addSubjectId ( )
addInteractionPAssertion ( )
addActorStatePAssertion ( )
addViewLink ( )
addInteractionKey ( )
addView ( )
addViewKind ( )
addAny ( )
addAny ( )
generateInteractionID ( )
«Java Class» InteractionContext
COPYRIGHT_NOTICE : String
_interactionKey : InteractionKey
_interactionMetaData : InteractionMetaData
getInteractionKey ( )
setInteractionKey ( )
getInteractionMetaData ( )
setInteractionMetaData ( )
«Java Class» PStoreDatabase
COPYRIGHT_NOTICE : String
messages : Message
log : Logger
provenanceStoreURI : String
provenanceDataServiceName : String
provenanceCollectionName : String
service : DataService
nsBinding : String
insertOrder : String
PStoreDatabase ( )
PStoreDatabase ( )
buildPSDatabase ( )
submit ( )
query ( )
getExceptionText ( )
management ( )
insert_pstruct ( )
make_pstruct ( )
make_view ( )
append_pstruct ( )
getViewType ( )
getXUAppendHeaderSection ( )
getXUAppendQuerySection ( )
getXUAppendTailSection ( )
append_assertion ( )
getInteractionPAssertionViewLink ( )
getAsserter ( )
getAsserter ( )
asserter_differ ( )
make_binding_string ( )
internal_query ( )
getAddressValue ( )
getElement ( )
getElements ( )
toDOMElement ( )
commitXUpdate ( )
appendViewLink ( )
getPHeaderLink ( )
«Java Class» InteractionMetaData
COPYRIGHT_NOTICE : String
NONE : int
ITEM : int
LINK : int
TRACER : int
_link : Link
_tracer : URI
_item : AnyContentType
_type : int
getItem ( )
setItem ( )
getLink ( )
setLink ( )
getTracer ( )
setTracer ( )
getType ( )
«Java Class» CommonConstants
SHORT_COPYRIGHT_STRING : String
EXAMPLE : String
EXCEPTION_MULTIPLE_ASSERTER_UPDATE : String
PROVENANCESTOREURL_PARAM : String
PROVENANCEDATASERVICE_PARAM : String
PROVENANCECOLLECTION_PARAM : String
PRECORD_NS : String
PHEADER_NS : String
PSTRUCT_NS : String
PHEADER_OBJECT : String
PSTRUCT_NODENAME : String
PSTRUCT_CLIENT_NODENAME : String
PSTRUCT_SERVICE_NODENAME : String
PSTRUCT_ASSERTER_NODENAME : String
PSTRUCT_ACTORSTATE_PASSERTION_NODENAME : String
PSTRUCT_INTERACTION_PASSERTION_NODENAME : String
PSTRUCT_LOCAL_INTERACTION_ID_NODENAME : String
PSTRUCT_INTERACTION_ID_NODENAME : String
PSTRUCT_INTERACTION_RECORD_NODENAME : String
PSTRUCT_INTERACTION_KEY_NODENAME : String
PSTRUCT_RELATIONSHIP_NODENAME : String
PSTRUCT_NUMBEROFEXPECTED_ASSERTIONS_NODENAME : String
PSTRUCT_VIEW_NODENAME : String
PSTRUCT_VIEWLINK_NODENAME : String
PSTRUCT_VIEWKIND_NODENAME : String
PSTRUCT_OBJECTLINK_NODENAME : String
PSTRUCT_OBJECTID_NODENAME : String
PSTRUCT_MESSAGESOURCE_NODENAME : String
PSTRUCT_MESSAGESINK_NODENAME : String
PSTRUCT_DOCUMENTATIONSTYLE_NODENAME : String
PSTRUCT_SUBJECTID_NODENAME : String
PSTRUCT_PARAMETER_NAME_NODENAME : String
PSTRUCT_DATA_ACCESSOR_NODENAME : String
PSTRUCT_PROVENANCE_STOREREF_NODENAME : String
PRECORD_RECORD_NODENAME : String
PRECORD_RELATIONSHIP_NODENAME : String
PRECORD_CONTENT_NODENAME : String
PRECORD_IDENTIFIEDCONTENT_NODENAME : String
PRECORD_ACTORSTATE_PASSERTION_NODENAME : String
PRECORD_INTERACTION_PASSERTION_NODENAME : String
PRECORD_SUBMISSIONFINISHED_NODENAME : String
CommonConstants ( )
«use»

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme

11.3.3 Class diagram for org.gridprovenance.exceptions package



11.3.4 Class diagram for org.gridprovenance.commonservices.test package


«Java Package»
org.gridprovenance.commonservices.test
«Java Class»
UtilsTest
COPYRIGHT_NOTICE : String
log : Logger
UtilsTest ( )
setUp ( )
tearDown ( )
testGenerateInteractionID ( )
getSampleIdentifiedContent ( )
getSampleSourceSinkAddresses ( )
getSourceSinkAddresses ( )
getSampleInteractionPAssertion ( )
getSampleActorStatePAssertion ( )
getMessageElements ( )
«Java Class»
PStoreDatabaseTest
COPYRIGHT_NOTICE : String
url : String
collection : String
dbName : String
setUp ( )
tearDown ( )
testSubmit ( )
«Java Class»
PHeaderTest
COPYRIGHT_NOTICE : String
testXMLPath : String
header : PHeader
PHeaderTest ( )
setUp ( )
testPHeader ( )
testSerialize ( )
testPHeaderElement ( )
testGetInteractionContexts ( )
testSetInteractionContexts ( )
testGetInteractionKey ( )
testSetInteractionKey ( )
testGetInteractionMetaData ( )
testSetInteractionMetaData ( )
«use»
«use»

11.4 Exception and Notification Handling

The *ProvenanceException* class is used for error reporting together with log4j logging. This is sub-classed for more specific error handling and defined in package *org.gridprovenance.commonservices.exceptions.* Each instance of a *ProvenanceException* is allocated an identifier, this is to assist debugging by prefixing log4j error messages.

Exceptions thrown within the Provenance server implementation are caught and wrapped into *ServerExceptions*. It is not intended that server-side exceptions be reported to user directly, but should be handled by the implementation.

The *PROVENANCE* Project receives research funding from the European Community's Sixth Framework Programme