Title:	Client Side Library Design and Implementation
Workpackage:	WP9
Author:	Sheng Jiang
Contributors:	Luc Moreau
	Paul Groth
	Simon Miles
	Victor Tan
	Steve Munroe
Reviewers:	All project partners
Identifier:	D9.3.3a
Type:	Deliverable
Version:	1.0
Date:	28 November 2006
Status:	Public

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

1 Introduction

The purpose of this document is to present the architecture of the *Client Side Library* (CSL) of a provenance system, its rationale, its implementation and a methodology guiding its use. According to Kruchten [1], "this is a development architecture, which focuses on the actual software module organisation."

The CSL is a collection of functions, which allows provenance-aware applications to communicate with provenance store services. It also provides functionality to help application developers enforce architecture rules or organise provenance relevant data items easier in their provenance-aware applications. In Section 2, we first enumerate the requirements, which are selected from the provenance architecture document [2] according to their relevancy to CSL design. In Section 3, we describe our implementation referring to implementation requirements in the provenance architecture document. In Section 4, we present a design of a layered model for the CSL and detail its implementation. In Section 5, we describe the development history of our implementation and its current status. The mechanism, which application developers can use for integrating the CSL, is introduced in Section 6.

Application developers should follow the procedures described in Section 6 primarily, and consult Section 4 for more details on the design when they need to.

2 Requirements

In this section, we enumerate the impact of various requirements on the CSL and explain how we support them in the CSL. The requirements are selected from the Chapter 9 in the provenance architecture document [2] because of their relevance to CSL.

ID	Description	Support Feature
[SR-1-1]	The CSL should include recording and query interfaces which are used respectively to store and retrieve p-assertions to/from the Provenance Store.	This requirement is supported by provided record, xquery and provenance query interfaces.
[SR-1-2]	The CSL should allow the retrieval of a provenance trace from the Provenance Store. Either a complete trace or a subset may be retrieved.	This requirement is support by provided xquery and provenance query interfaces.
[SR-1-3]	The CSL may store query results for future use.	This requirement may supported by the tools built upon the CSL. The CSL itself does not manage the storage of result.
[SR-1-4]	The CSL may allow comparisons to be made across Provenance Records.	This requirement may supported by the tools built upon the CSL.
[SR-1-9]	The CSL should be accessible as an API, which can be embedded into an existing application.	This requirement is supported by allowing our client side to be wrapped into a jar file, which can be easily embedded into applications.
[SR-1-10]	The CSL should allow the specification	This requirement is supported by

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

	of the identity of the Provenance Store to which data should be recorded, or which queries should be made to.	allowing application to assign the URL of the provenance store.
[SR-1-13]	The CSL may support the migration of provenance data among Provenance Stores.	This requirement is supported by the tools built upon the CSL.
[SR-1-16]	The CSL should provide support for maximum automation of the provenance recording mechanism.	This requirement is supported by providing a simpler Application API and the use of a creation utility to generate certain data items for users.
[SR-1-18]	The CSL should not block an executing workflow if any provenance services are unavailable.	The CSL is designed to be thread-safe so that its invocation by applications can be performed in separate threads.
[SR-2-1]	The additional execution overhead of the CSL should be kept to a minimum.	The CSL is designed and implemented while keeping this in mind.
[SR-3-1-2]	The CSL should allow application specific analysis and reasoning tools to be built upon.	The CSL provides the API, which tools can be built upon easily.
[SR-6-3]	The CSL may allow the configuration of different levels of security communication.	This requirement is supported by providing several security mechanisms in different levels within the CSL.
[SR-7-2]	The CSL should be easily integrated into applications. Integration costs for existing applications should be minimal; ideally existing system components should remain unaffected.	The CSL has been designed in a layer model, which exposes only a relevant simple Application API to applications.
[SL-2]	The CSL should support data translation with specific documentation styles, including references to data, anonymisation, security encryption, part of message, reduced form, etc. Also mentioned in [SL-4, OTM.1, OTM.2, OTM.8, OTM-18, EHCR.4, EHCR.5, AER-2, IR-APP-1, and IR-APP-2].	This requirement is supported by providing a documentation style function with certain implementation styles, including verbatim, reference, encryption, signing and replace. It also allows third-party implemented styles to be plugged in easily.
[SL-9]	The CSL should have no affinity to a Provenance Store.	The CSL is independent from provenance stores since it only relies on WSDL specification of their interfaces.
[AER-7]	The CSL should provide functionality to generate unique IDs.	This requirement is supported by providing create utilities for LocalPAssertionId and InteractionId.
[TSR-1-9]	The CSL should support a registry	This requirement was regarded as

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

	service to enable the software interfaces for all Tools to be visible.	beyond the scope of the CSL, which aims at interaction with provenance stores (and not registration).

3 Implementation Recommendation

Besides the general user requirement, we also have a series of implementation recommendations. In this section, we list these recommendations, suggest features and explain how we support them in the CSL implementation.

ID	Description	Design Feature
IR-ASL-1	The CSL should allow for identification of which provenance stores to be used. However, it should also ensure that all p-assertions pertaining to one interaction from a particular actor must be recorded in the same provenance store.	This recommendation is supported by implementing a ProvenanceService interface, which has an address parameter pointing to the in use provenance store. The CSL also helps applications in obeying architecture Rule 8.4 (Recording Consistency Rule) by adopting a boolean recorded parameter in ViewRecordImpl.
IR-ASL-2	The CSL should provide re-usable functionality to communicate and interact with the assigned provenance store.	This recommendation is supported by providing a Server API, which communicates to provenance store services; independently of the communication network (secure or not).
IR-ASL-3	The CSL may provide functionality to mutually authenticate with the assigned provenance store.	This recommendation is supported by combining the CSL with a security host environment, such as GT4; we adopt the GT4 security library in our implementation.
IR-ASL-4	An actor should provide functionality to record a view link to the provenance store that contains the corresponding actors view of the interaction if the view is in another provenance store.	This recommendation is supported by providing a P-Header helper. When a view is recorded in a provenance store, its view link is passed to its corresponding actors by using P-Header. P-Header with the view link is recorded at its corresponding actors.
IR-ASL-5	The CSL should provide functionality for generation of unique IDs.	This recommendation is supported by implementing a localPAssertionIdFactory, which enforces unique IDs. This also helps applications to comply with architecture Rule 8.1 (Unique Interaction Key Rule).

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

IR-ASL-6	The CSL may provide functionality to add assigned Interaction Keys into its asserting message.	This recommendation is supported by implementing a P-Header function. P-Header can hold the assigned Interaction Keys. We also provide utilities to help applications to embed PHeader into a message header. This also helps applications to comply with architecture Rule 8.2 (Interaction Key Transmission Rule).
IR-ASL-7	The CSL may provide functionality for the generation of a new session tracer and add it into the task message.	This recommendation is supported by implementing a tracer function within the PHeader function. This also helps applications to comply with architecture Rule 8.6 (Generation Rule).
IR-ASL-8	The CSL may provide functionality to add any session tracers received from a superior actor to all requests it makes to inferiors within the task started by the superior's request. This also complies with architecture Rule 8.7. (Propagation Rule: to inferior)	We provide the function to extract and inject session tracers from and into P-Headers. However, the burden is on the application to extract a tracer from a superior's interaction and inject it into the inferior interaction.
IR-ASL-9	The CSL may provide functionality to add session tracers supplied by its superior to its response to the superior. This also complies with architecture Rule 8.8 (Propagation Rule: to superior).	We provide the function to extract and inject session tracers from and into P-Headers. However, the burden is on the application to extract a tracer from a superior interaction and inject it into the response interaction.
IR-ASL-10	The CSL may support declarative policy specifications that specify what information needs to be recorded and when.	This recommendation is supported by implementing a policy function. This also helps applications to comply with architecture Rule 8.6 (Generation Rule).
IR-ASL-11	The CSL may be customised for a specific actor hosting environment to capture information automatically from existing logs or from the runtime environment. It may also use security functionality and credentials provided by the hosting environment.	This recommendation is supported by combining the actor side library with a security host environment, GT4. However, users have to configure the host environment themselves in order to get log information or use the security function and credentials provided by GT4.

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

IR-ASL-12	The CSL may provide a range of error handlers and other facilities to desynchronise application execution from execution documentation recording.	This recommendation is supported by implementing several exceptions. DataConstructorException handles all the data constructing errors on the client side. RecordException handles error during the recording process. This also helps applications to comply with architecture Rule 8.12 (Error Message Rule).
IR-ASL-13	The CSL may provide the necessary access to cryptographic functionality and material (such as key stores) in order to accomplish functionality such as signing or encrypting.	This recommendation is supported by implementing security-signing and security-encryption documentation style. (Note: the key store is expected to be provided by users.) This also helps applications to comply with architecture Rule 8.9 (Signature Rule).
IR-ASL-14	The CSL may provide functionality to transform messages according to particular documentation styles.	This recommendation is supported by implementing documentation style function. Verbatim, reference, encryption, signature, replace and complex documentation styles are supported in our implementation. This also helps applications to comply with architecture Rule 8.5 (Link Recording Rule).

4 Layered Model and Implementation

The CSL provides essential functionality to enable the interaction between provenance-aware applications and provenance store services. It also offers functions to help application developers develop provenance-aware applications easily. It contains the interfaces as shown in the following figure:

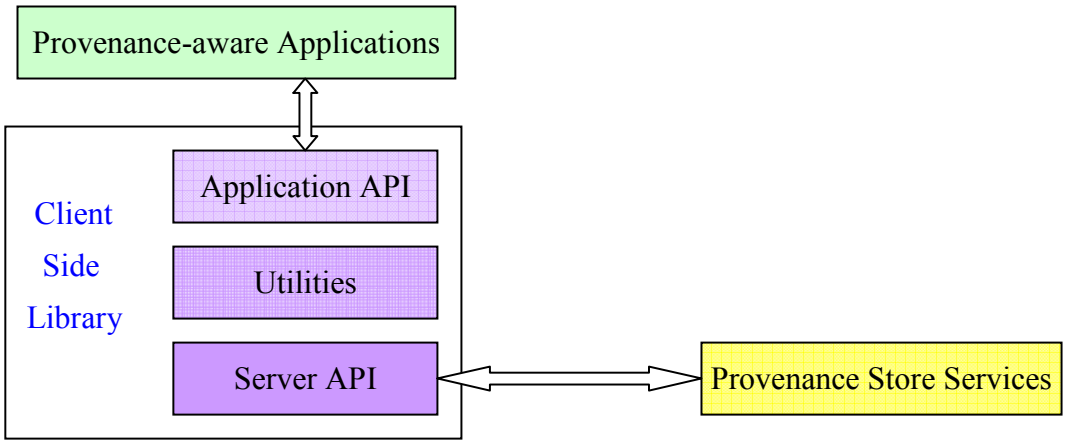

Figure 1: layered model of the CSL

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

In the above figure, we define three layers: the Server API (Application Programming Interface) interacts with provenance store services. However, the data structure of the Server API is too complicated to be used directly by application developers. Therefore, we have defined an alternate, relatively simpler Application API. Some utilities are also implemented in order to map the Application API to the Server API. Although application developers could invoke the Server API directly, it is recommended that they use only the Application API. Within the Application API layer, we also provide some helper functions that aid application developers to enforce architecture rules (*Chapter 8 Actor Behaviour, [2]*) in their provenance-aware applications. Helper functions are designed to be directly invoked by application developers. Therefore, they are introduced as part of the Application API.

We describe our implementation of the CSL in the following sections. We start from the bottom layer – the Server API, then the utilities layer, ending with the top layer – the Application API. Each layer has its corresponding implementation(s). For each implementation, we give the name of the Java package in which its primary interfaces are found.

4.1 The Server API

In practice, we defined our provenance store interfaces in a set of Web Service Definition Language (WSDL) files, according to our provenance specification [7, 8, 9, 10, 11, 12, 13]. Both server side and client side implementations are compatible with these WSDL definitions. Therefore, they are compatible with each other.

We then generated stubs from these WSDL files using a JAX-RPC (Java API for XML based Remote Procedure Call) [14] compliant tool, such as WSDL2Java [15] from the Globus Toolkit version 4. Its structure exactly mirrors the defined WSDL interfaces of our Web Service provenance store. We then use these stubs as our Server API (bottom layer). However, the Server API mainly aims at low-level programming and testing. It is not primarily intended to be used by application developers.

The Server API of the CSL is able to communicate with different types of provenance store servers, which are compatible with the Provenance Store WSDL (PSWSDL [5]). As an illustration, we have tested two implementations of provenance store services: a GT4-hosted WSRF compliant provenance store [4] and a pure Web Service provenance store (Provenance assertion Recording Services [3]).

Package: org.gridprovenance.client.generated

Implementation: The Server API is exactly the mapping of the WSDL to Java defined by the JAX-RPC.

4.2 Utilities

The programming model of the Server API is too complicated for applications or users to invoke directly. Instead, we designed utilities (middle layer) to hide the complexity of the Server API from the upper-layer applications. It maps the relatively simpler Application API, which we will describe in the next section, to the Server API.

4.2.1 Creation utilities

In order to simplify the user burden in creation provenance data, we provide several creation utilities. They can automatically create some data items, such as LocalPAssertionId and InteractionId, for provenance-aware applications or users. They also apply the relevant rules,

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

such as Rule 8.1 (Unique Interaction Key Rule), in order to make sure they obey the provenance architecture and are accepted by the Server API.

Package: org.gridprovenance.client.utilities

Implementation: a LocalPAssertionIdFactory and an InteractionIdFactory, which create unique LocalPAssertionIds and InteractionIds.

4.2.2 Data format converting utilities

We allow applications/users to input their data in simple format, mostly in String or Java generic URI. Our data format converting utilities will check whether they are acceptable and convert them into the formats that are accepted by Server API. These utilities also have functions to construct certain type of new data items. They also provide functions to convert certain type of data items into String or Element, which can be easily used by other functions.

Package: org.gridprovenance.client.utilities

Implementation: AsserterUtil, AddressUtil, DataAccessorUtil, DocumentUtil, ElementUtil, EndpointReferenceTypeUtil, GlobalPAssertionKeyUtil, InteractionKeyUtil, InteractionMetaDataUtil, LinkUtil, SecurityUtil, SignatureUtil, URIUtil and ViewKindUtil.

There are several new PAssertion() functions in the PAssertionFactory. They take necessary parameters to create the corresponding PAssertions. They have multiple methods in order to handle different combinations of parameters.

4.2.3 Error information utility

The client-side library is responsible for providing users with necessary error information. Potential errors include: client-side library errors, communication errors between client and server, and errors of the provenance store. In our implementation, every method has its own errorMessage parameter, which includes the name of method and its belonged class. If any error happens in a certain method, its errorMessage will be output alongside the Java error. Then, the user can know precisely where the error happens.

Package: org.gridprovenance.client.exception

Implementation: errors are wrapped into four types of Exceptions: DataConstructorException, PQueryException, RecordException and XQueryException. DataConstructorException handles all the data constructing errors on the client side. RecordException handles error during the recording process. XQueryException and PQueryException handle error during the xquerying or pquerying process.

4.3 The Application API

The Application API (top layer) of the CSL provides a relatively simple interface to applications. Ideally, the Application API is the only means by which application developers access the CSL.

4.3.1 ProvenanceService

The ProvenanceService interface is designed to facilitate interactions with a single provenance store at a given address. A ProvenanceServiceAddressingLocator is constructed using the given provenance store address. Each ProvenanceService instance contains a set of EndpointReference for record, xquery and pquery ports. When a ProvenanceService is initialized, it loads policy parameters from user's configuration. These policy parameters will

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

be used later in CSL. It also has methods to create new ViewRecord, XQuery or PQuery instances using its default properties.

Implementation:

Interface: org.gridprovenance.client.ProvenancService

Implement: org.gridprovenance.client.impl.ProvenanceServiceImpl

4.3.2 PortFactory

Each port has its own interface. We have defined them as interfaces for extensibility purposes. However, in order to reduce the programmer's burden, we have defined a PortFactory interface to create all relevant ports according to the different types of server. We have three implementations: PortFactoryPSImpl creates the ports when CSL works against a pure web-service provenance store; PortFactoryWSRFImpl creates the ports when CSL works against a WSRF compatible provenance store; PortFactoryUnkownImpl has a test function to decide the type of provenance store, and then creates the ports according to the type, whether Web Service or WSRF based.

Implementation:

Interface: org.gridprovenance.client.PortFactory, org.gridprovenance.client.PQueryPort, org.gridprovenance.client.RecordPort, org.gridprovenance.client.XQueryPort,

Implement: org.gridprovenance.client.impl.PortFactoryPSImpl,
org.gridprovenance.client.impl.PortFactoryWSRFImpl,
org.gridprovenance.client.impl.PortFactoryUnkownImp,
g.gridprovenance.client.impl.PQueryPortImpl,
org.gridprovenance.client.impl.RecordPortImpl,
org.gridprovenance.client.impl.XQueryPortImpl

4.3.3 ViewRecord & RecordResult

The ViewRecord interface is designed to hold the recording contents, i.e. PAssertions and exposedMetadata. It has added PAssertion() methods, which call corresponding new PAssetion() functions, also an addExposedMetadata() method. All the data within ViewRecord instances will be recorded by the record() method. The RecordResult interface is designed to hold the result of a recording.

Implementation:

Interfaces: org.gridprovenance.client.Record, org.gridprovenance.client.RecordResult;
Implements: org.gridprovenance.client.impl.RecordImpl,
org.gridprovenance.client.impl.RecordResultImpl

4.3.4 XQuery & XQueryResult

The XQuery interface is designed to hold the xquery parameters and carry out the xquery fucntion. It has a setXQueryString() method. The search string describes the specific search target. The XQuery is launched by the execute() method. The XQueryResult interface is designed to hold the result of xquerying.

Implementation:

Interfaces: org.gridprovenance.client.XQuery, org.gridprovenance.client.XQueryResult

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

Implements: org.gridprovenance.client.impl.XQueryImpl,
org.gridprovenance.client.impl.XQueryResultImpl

4.3.5 PQuery & PQueryResult

The PQuery interface is designed to hold the provenance query parameters and carry out the pquery function. It has a queryDatahandle and a relationshipTargetFilter alongside their namespaceMapping. They give the provenance store detailed search information. The PQuery is launched by the execute() method. The PQueryResult interface is designed to hold the result of pquerying.

Implementation:

Interfaces: org.gridprovenance.client.PQuery, org.gridprovenance.client.PQueryResult

Implements: org.gridprovenance.client.impl.PQueryImpl,
org.gridprovenance.client.impl.PQueryResultImpl

4.3.6 Security Mechanisms

Security, an important concern of the client-side library, targets three elements:

1 Establishing an encrypted communication connection to the provenance store service;

2 Providing credentials for authentication purposes to the provenance store on this communication path;

3 Creating a digital signature on p-assertions, which can subsequently be stored and verified.

In the Globus-Toolkit-hosted environments, the GT libraries can be employed to accomplish all 3 aims.

Implementation: security of communication is dependent on the hosting environment. We have integrated GT4 host security stubs into our client-side library. We also provide configuration parameter (useGT4SSL) for user to choose whether to use security or not.

4.3.7 Documentation style helper

The CSL provides support for documentation style. We implement the message transformation mechanisms that were initially defined in the architecture document [2] (*Chapter 6 Provenance Modelling, Section 6 Documentation Style Modelling*).

The atomic documentation styles that are supported by the CSL are the following: Verbatim, reference, encryption, signature and replace. Composite documentation styles that combine one or more of the above atomic documentation styles are also supported.

Package: org.gridprovenance.documentationstyles

Implementation: The users, at design time, define the transformations that they want to apply, and construct the related "*transformation*" files. Initially, the supplied atomic transformations are all that are provided to the user. Then, the user uses these files at runtime to apply the message transformation to the corresponding *input file*. To support this facility in the CSL, we provide a *"Transformer"* class, which take as an input the *transformation* and the *input file* (defined by the user), and produces the transformed *output file*. If more than one transformation is applied (composite documentation style), the *main function* of the *Transformer Class* is changed to call the appropriate *Transformer* (multiple times with different input files).

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

4.3.8 Policy helper

In order for users of a provenance store to be able to record p-assertions, several parameters of the store must be known and selected. To achieve this we adopt Apache's WS Policy implementation [16] to develop a policy framework, which allows users to discover a store's operational parameters their various options and to select those that are required (Shown if Figure 2). The framework provides a set of default parameters that are adopted if the user does not specify any policies in a policy document. If a user specifies a policy document, this is process using the WS Policy implementation to produce a policy object, which is then taken by a ConfigPolicies class that transfers these to the PolicyManager that sets the policies contained in the policy object into the PolicyParamHolder. The WS Policy implementation provides functions to merge different sets of policies and to discover common policy options from different sets of policies. This functionality provides much flexibility for future development of the policy framework and offers users significant opportunities for configuring provenance stores in different ways.

Package: org.gridprovenance.client.policy

Implementation: We have constructed a policyParameterHolder class and a defaultParameter class, which holds default policies. The PolicyManager handles PolicyObjects produced by the WSPolicy Implementation that creates these objects from XML PolicyDocuments, which are written by users to select the various different policy options provided by the provenance store's developers.

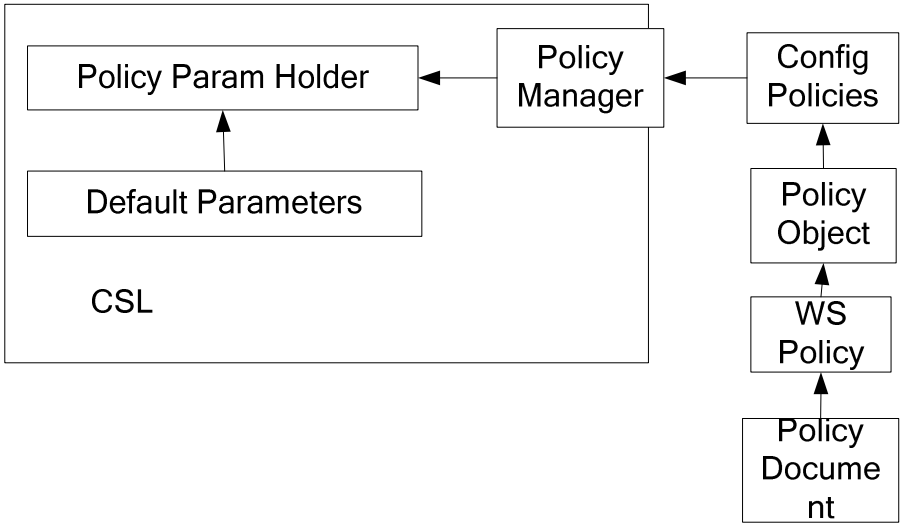

Figure 2: The Policy Framework

4.3.9 P-Header helper

The provenance architecture document [2], Section 3.4 introduces the concept of a P-Header (Definition 3.1). In the CSL, there is a P-Header function that helps applications to create and embed proper p-headers into the application messages. It also helps to extract the p-headers from messages on the receiver side.

Package: org.gridprovenance.client.pheader

Implementation: P-Header itself is a complex object. In order to create a P-Header easily, we have implemented InteractionContext, InteractionMetadata and tracer. We also implemented

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

a P-Header utility, which helps applications/users to embed a P-Header into a soap header and export a P-Header from a soap message.

4.3.10 Relationship helper

The provenance architecture document [2], Section 2.4 specifies relationships and relationship p-assertions (Definition 2.7). In the CSL, we implement a relation function that helps the applications/users to create relationshipPAssertion easily.

Package: org.gridprovenance.client.relation

Implementation: there are a RelationObject class, a RelationSubject class and a DataAccessor interface, which can implement different type of data accessors.

4.4 Tests and Examples

We provide examples to demonstrate how to use the CSL. We also implemented JUnit test cases to test the CSL itself.

Package: org.gridprovenance.client.test, org.gridprovnenace.client.test.junittest

Implementation: The main example is ApplicationAPIExample, in which we demonstrate how to use the CSL to record an interaction p-assertion, an actor state p-assertions and a relationship p-assertion, and query the recorded p-assertions back. It is well explained in the source code, and its execution is self explanatory.

There are also a few other examples listed below, with explanations provided in the source code itself.

- DocumentationStyleExample, demonstrates the use of the documentation style API;
- PHeaderExample, demonstrates the PHeader helper and utilities;
- XQueryExample, demonstrates the XQuery separately;
- SecureCommunicationExample, is a modification of ApplicationAPIExample that is capable of running in different security modes against a Provenance Service deployed in a secure manner;
- PQueryexample, demonstrates how to use PQuery. It works only after using RecordPQueryExampleData to record the example data.

A test suite (testCSL) has been implemented to test the CSL components independent of testing against a Provenance Store. It includes test cases for TestImplements, TestPHeader, TestRelationships and TestUtilities. They test each CSL components by creating testing instances and comparing them with the expecting values.

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

5 Roadmap

This section describes the CSL development roadmap.

5.1 Development History

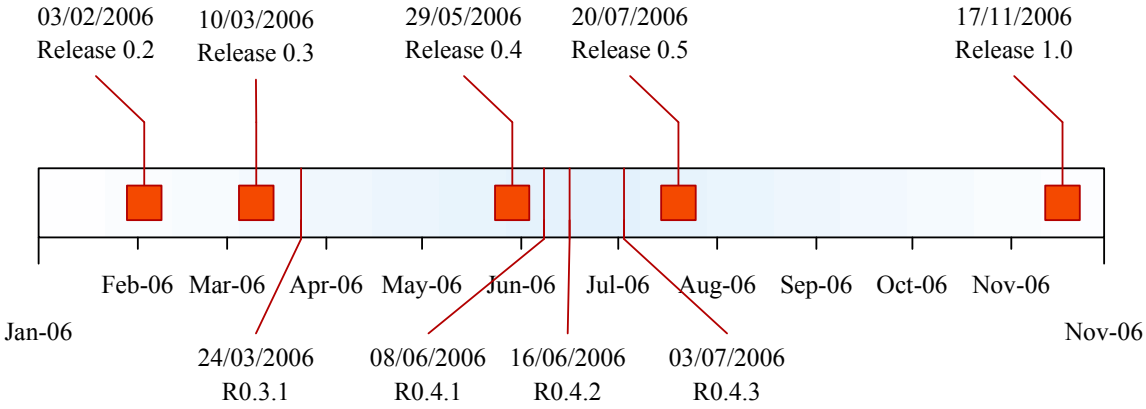

Figure 3: Release history of CSL

As shown in Figure 3, we started our design and implementation at the beginning of 2006. For the initial version (v0.1), we generated stubs from designed WSDL and constructed the implementation packages.

By 3rd February, we gave application developers our first release (v0.2). It includes only the generated Server API and an example to use it directly.

We then implemented the basic Application API, which includes ProvenanceService, ViewRecord and XQuery. Necessary utilities and the relation helper were implemented as well in order to map the Application API to the Server API. By 10th March 2006, when we had released v0.3, we also added an ApplicationAPIExample and an XQueryExample. In order to fix several reported bugs and new application requirements, we made a minor release v0.3.1 on 24th March 2006.

The next release (v0.4) we made on 29th May 2006. It was implemented against an improved schema. This version was fully tested working with IBM WSRF-compatible provenance store. In the mean time, we implemented documentation style and the P-Header helper. PortFactory was introduced in order to simplify the usage of the CSL. After interaction with application developers, we released a minor version (v0.4.1) for fixed bugs and new requirements. V0.4.2 was released later with refined exception and better support documents provided. V0.4.3 was provided with new ProvenanceLink functions and JUnit testcases for CSL. V0.5.0 was release 21st July 2006. In it, we provide the Provenance Query function; we introduce Policy Parameter, we also make secure communication available; alongside minor improvements and bugs fixes.

The current version is 1.0, which will be described in the next section.

During the development period, it was our intent that changes to the Application API was minimized to avoid affecting the development of provenance-aware applications. Whilst this

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

was desirable, changes could be made binary compatible and hence there would not be noticeable impact on application.

5.2 Current Release

We released the current version 1.0 17th November. In it, we support XPathIterator port-type and XPathQuery in Application API. We also provide a PortFactoryUnknownImpl, which can automatically detect the type of provenance store then create ports according to it. We adopted WS Policy in this release.

By now, the CSL is stable and completed. There are no activated bugs on the CSL implementation.

5.3 Support for Applications

We provide constant and solid support to the application developers in this project.

We have explained our design and implementation of the CSL to the application partners in detail during the two project face-to-face meeting, (first in Cardiff 22-24 February 2006; second in Cologne 31 May – 2 June 2006). We have provided well organized JavaDoc alongside with other support documents (Readme, ChangeLog, JDK1.4-guide, Lightweight-guide, SecureGuide, LinkingSupportGuide, DocumentationStyleReadme, and DocStyleProgrammerGuide).

We have taken part in the developer teleconferences, which were held every two weeks. We answered questions raised by application developers regarding to the usage of CSL. All the bugs or new requirements raised are handled as soon as possible. (See project bugzilla [6]). As described in earlier sections, minor releases were made for fixed bugs.

We have held two project Interop face-to-face meetings (Interop 1 19-21 July 2006 and Interop 2: 21-24 August 2006) at the University of Southampton in order to resolve technical problems for application developers, such as coding or library conflicts.

6 Integration Mechanism

The CSL provides application developers and users with generic support for using provenance functions. Application developers only need to make their applications invoke the Application API. They may embed the CSL into their applications. After integration with applications, the CSL becomes a part of provenance-aware applications and should be distributed with the applications.

In order to show how users use the CSL, we illustrate one recording example and one querying example with sequence diagrams. Since we have hidden the utilities and serverAPI from applications/users, we do not show detail of them except for the construction of these necessary data items.

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

6.1 Recording P-Assertions

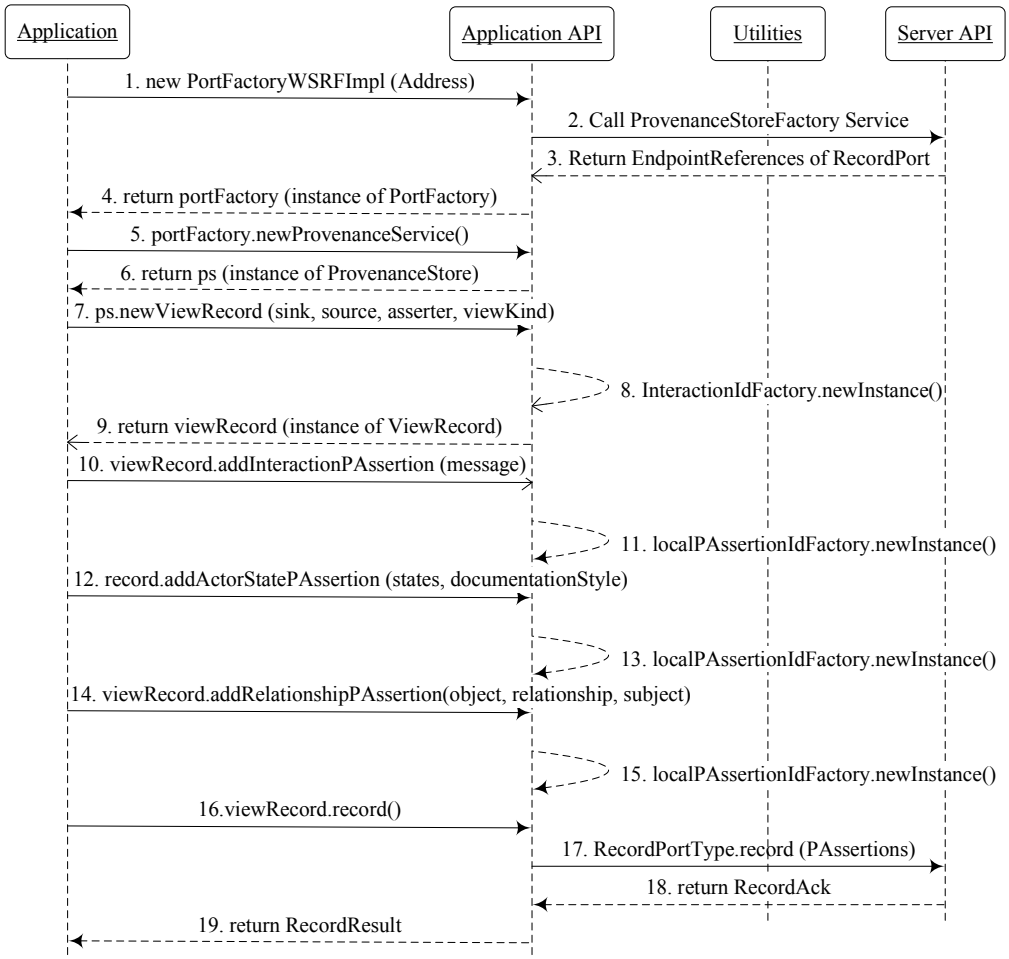

Figure 4: Recording sequence diagram

In Figure 4, we demonstrate how a provenance-aware application starts recording an interactionPAssertion. It calls the PortFactoryWSRFImpl with a given address (step 1). By calling the ProvenanceStoreFactory on the provenance store server (step 2), all relevant ports are created, including RecordPort (step 3). A portFactory instance, which holds all these ports, will be returned to the application (step 4). Then, the application needs to create a new ProvenanceService instance – ps (step 5, 6). After that, the application creates a new ViewRecord instance – viewRecord (step 7, 9). Its viewKind, sink, source and asserter parameters are used for all PAssertions that associate with the same view. IteractionId is auto-constructed by our InteractionIdFactory (step 8). In step 10, an interactionPAssertion is added in the viewRecord. In order to create a unique ID, the localPAssertionFactory is called (step 11) in background. A similar process is applied in order to add an ActorStatePAssertion to the record instance (step 12, 13). The difference is that the application provides actor states instead of messages. Before the application can add a relationshipPAssertion (step 14, 15),

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

the application should construct object and subject instance using the relation helper. In step 16, the record() method is called to record all these PAssertions. Behind it, RecordPortType of the Server API is called to communicate with provenance store service (step 17). RecordAck is returned from the Server API to the Application API. A RecordResult, which provides several methods to access the record result, is returned to the application. The recording process ends.

6.2 Querying for P-Assertions

Compared with the above recording process, the querying process is simpler, see Figure 5. The application constructs a new PortFactoryWSRFImpl instance (step 1, 2, 3, 4); then constructs ProvenanceService instance – ps (step 5, 6). It creates a new instance of XQuery with given xquery string (step 7, 8).

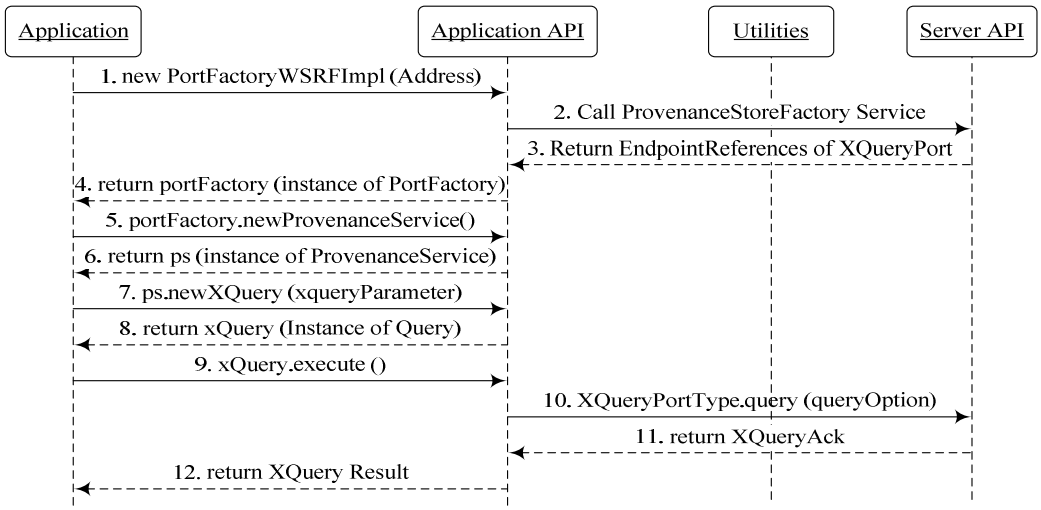

Figure 5: querying sequence diagram

In step 9, the execute() method launches the xquery to a provenance store. Behind it, XQueryPortType of the Server API is called to communicate with the provenance store service (step 10). XQueryAck is returned from the Server API to the Application API. XQueryResult, which provides several methods to access the xquery result, is returned to the application. The xquerying process ends.

7 Conclusion

In this document, we have presented a design for the CSL of provenance systems and our implementation details. The CSL allows provenance-aware applications to communicate with provenance store services. It also provides functionality to help application developers enforce architecture rules in their provenance-aware applications. We have enumerated the requirements in Section 2; and presented our supported functionalities referring to

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

implementation recommendations in Section 3. In Section 4, we have introduced our layered model and implementation details. We give the status of our current release and our future plan in Section 5. Finally, Section 6 gives indications how provenance-aware applications should use the CSL.

8 Bibliography

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

[2] Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, Victor Tan, Sofia Tsasakou, and Luc Moreau. **D3.1.1: An Architecture for Provenance Systems**. Technical report, University of Southampton, February 2006.

[3] PReServ (Provenance assertion Recording Services). http://twiki.pasoa.ecs.soton.ac.uk/bin/view/PASOA/SoftWare

[4] John Ibbotson. EU Provenance Project Deliverable 9.3.2. *Provenance Store (Server) Implementation Design*, February 2006.

[5] Provenance Service WSDL. http://www.pasoa.org/schemas

[6] EU Provenance project Bugzilla: http://gridprov.cs.cf.ac.uk

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

[8] Steve Munroe, Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. **A SOAP Binding For Process Documentation**. Technical report, University of Southampton, 2006. http://eprints.ecs.soton.ac.uk/13056

[9] Steve Munroe, Victor Tan, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. **A WS-Addressing Profile for Distributed Process Documentation**. Technical report, University of Southampton, 2006. http://eprints.ecs.soton.ac.uk/13057

[10] Paul Groth, Victor Tan, Steve Munroe, Sheng Jiang, Simon Miles, and Luc Moreau. **Process Documentation Recording Protocol**. Technical report, University of Southampton, 2006. http://eprints.ecs.soton.ac.uk/13053

[11] Simon Miles, Luc Moreau, Paul Groth, Victor Tan, Steve Munroe, and Sheng Jiang. **XPath Profile for the Provenance Query Protocol**. Technical report, University of Southampton, 2006. http://eprints.ecs.soton.ac.uk/13051

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

[13] Victor Tan, Steve Munroe, Paul Groth, Sheng Jiang, Simon Miles, and Luc Moreau. **Basic Transformation Profile for Documentation Style**. Technical report, University of Southampton, 2006. http://eprints.ecs.soton.ac.uk/13049

[14] JAX-RPC. http://java.sun.com/webservices/jaxrpc/

[15] WSDL2Java. http://www-unix.globus.org/api/javadoc-3.9.0-core/org/globus/wsrf/tools/wsdl/WSDL2Java.html

[16] WS Policy Implementation. ws.apache.org/commons/policy/index.html

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

Appendix: Example code

```
package org.gridprovenance.client.test;

import org.w3c.dom.Element;
import java.security.Principal;
import javax.security.auth.x500.X500Principal;
import java.net.URI;
import java.net.URISyntaxException;

import org.gridprovenance.client.PortFactory;
import org.gridprovenance.client.ProvenanceService;
import org.gridprovenance.client.ViewRecord;
import org.gridprovenance.client.RecordResult;
import org.gridprovenance.client.XQuery;
import org.gridprovenance.client.XQueryResult;
import org.gridprovenance.client.GlobalPAssertionKey;
import org.gridprovenance.client.ViewKind;

import org.gridprovenance.client.exception.DataConstructorException;
import org.gridprovenance.client.exception.RecordException;
import org.gridprovenance.client.exception.XQueryException;
import org.gridprovenance.client.impl.PortFactoryPSImpl;
import org.gridprovenance.client.impl.PortFactoryWSRFImpl;
import org.gridprovenance.client.impl.SenderViewKindImpl;

import org.gridprovenance.client.relationships.DataAccessor;
import org.gridprovenance.client.relationships.RelationSubject;
import org.gridprovenance.client.relationships.RelationObject;
import org.gridprovenance.client.relationships.XPathDataAccessor;

import org.gridprovenance.client.utilities.ElementUtil;

/**
 * Demonstrates how to use the Client Side Library (CSL) record<p>
 * interaction p-assertions, actor state p-assertions, relationship<p>
 * p-assertions and query the recorded p-assertions back.<p>
 * CSL of EU Provenance
 *
 * @author Sheng Jiang, Paul Groth
 * @version $Revision: 1.9 $
 */

public class ApplicationAPIExample {

   /**
    * Execution body.
    * @param args args[0] is the address of provenance store;
    */
   public static void main(final String[] args) {
      if ((args.length < 1) || args.length > 2) {
```


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

```
        System.out.println(" Usage:\tjava org.gridprovenance.test."
            + "ApplicationAPIExample <provenance store url>\n"
            + "\t\texample url: http://localhost:8080/wsrf/"
            + "services/ProvenanceStoreFactory (IBM WSRF ProvenanceStore)");
        System.exit(0);
    }
    try {
        new URI(args[0]);
    } catch (URISyntaxException e) {
        String errorMessage ="User input args 1 is not a valid URI";
        System.out.println(errorMessage);
        System.exit(0);
    }

    System.out.println("ApplicationAPIExample\n"
        + "The scenario():\tWe simulate a sender sending a "
        + "message to a receiver.\n\t\tThe sender uses part of its state in"
        + " the the message so\n\t\tit makes a relationship p-assertion "
        + "between its actor\n\t\tstate p-assertion and its interaction "
        + "p-assertion.\n\nPress any key to continue...");

    /**
     * The following parameters should be set up / provide by applications,
     * either from the configure files or generated during application
     * executing. Here, as a simple example, we use the pre-set values
     * instead.
     */
    PortFactory portFactory = null;
    try {
        portFactory = new PortFactoryWSRFImpl(args[0]);
    } catch (DataConstructorException e) {
        String errorMessage = "Error during create Ports!";
        System.out.println(errorMessage);
        System.exit(0);
    }
    //Create a ProvenanceService for recording and xquerying.
    ProvenanceService ps = portFactory.newProvenanceService();


    try {
        ViewKind vk = new SenderViewKindImpl();
        String sourceString = "http://www.sender.com/sender";
        String sinkString = "http://www.receiver.com/receiver";
        Principal asserter = new X500Principal(
            "CN=Sheng, OU=ecs, O=soton, C=UK");
        Element message = ElementUtil.newElement("<hello>Sheng</hello>");
        Element actorStates = ElementUtil.newElement("<cpu>1.2 GHz</cpu>");
        /**
         * In order to create a new record object, message source, message sink,
         * asserter and viewKind should be given by application.
         */
        ViewRecord viewRecordInstance = ps.newViewRecord(sinkString, sourceString,
            asserter, vk);

        /**
```


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

```
 * In the following two PAssertion, since no documentationStyle given,
 * Verbatim will be applied; for other, see DocumentationStyleExample.
 */
GlobalPAssertionKey gpakAPA = null, gpakIPA = null;
try {
   gpakIPA = viewRecordInstance.addInteractionPAssertion(message);
} catch (DataConstructorException e) {
   System.out.println("error when adding an InteractionPAssertion");
}

try {
   gpakAPA = viewRecordInstance.addActorStatePAssertion(actorStates);
} catch (DataConstructorException e) {
   System.out.println("error when adding an ActorStatePAssertion");
}

/*
 * We create a "usage" relationship between the above 2 PAssertions.
 * This relationship says message represented by InteractionPAssertion
 * used the actor state represented by ActorStatePAssertion. In order
 * to do so, there are a few more parameters that should be given by
 * applications. Applications may use relation helper to get or set
 * right relationships.
 *
 * In order to create the relationship PAssertion, the following
 * parameters should be set up / provide by applications. DataAccssor
 * are optional. It points to a certain element inside PAsserter.
 */
DataAccessor subjectDA = new XPathDataAccessor(
   "/soap:envelope/soap:body/message:hello");
URI subjectPAParameterName = null;
URI objectPAParameterName = null;
URI relationship = null;
try {
   subjectPAParameterName = new URI("http://www.pasoa.org/schemas/"
      + "ontologies/relationships/usage.user");
   objectPAParameterName = new URI("http://www.pasoa.org/schemas/"
      + "ontologies/relationships/usage.usedItem");
   relationship = new URI("http://www.pasoa.org/schemas/ontologies/"
      + "relationships/use");
} catch (URISyntaxException e) {}
DataAccessor objectDA = new XPathDataAccessor("/server:cpu");
String objectLinkString = "http://www.pasoa.org/xquery";
//where objectPA can be queried back

//Magic number 1, only because we know there are one object
RelationObject [] relationObject = new RelationObject [1];
relationObject [0] = new RelationObject(objectDA, objectPAParameterName,
   gpakAPA, objectLinkString);
RelationSubject relationSubject = new RelationSubject(subjectDA,
   subjectPAParameterName, gpakIPA);

try {
   viewRecordInstance.addRelationshipPAssertion(relationObject,
      relationship, relationSubject);
```


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

```
            } catch (DataConstructorException e) {
               System.out.println("error when adding an RelationshipPAssertion");
            }
            System.out.println("Recording the p-assertions in provenance service "
                     + ps.getRecordPortEPRType().toString());
            RecordResult rResult = viewRecordInstance.record();
            rResult.printOut();
            System.out.println("One Interaction PAassertion recorded");
         } catch (DataConstructorException e) {
            String errorMessage = "error when constructing data for ViewRecord";
            System.out.println(errorMessage);
         } catch (RecordException e) {
            String errorMessage = "Error during recording";
            System.out.print(errorMessage);
         }

         System.out.println("*****************************************\n"
            + "Querying the content of the provenance store\n"
            + "The entire content of the store will be displayed\n"
            + "Press any key to continue...");

         /**
          * Create a XQuery Instance
          */
         String xQueryString = "<result>{for $n in $ps:pstruct return $n}</result>";

         XQuery xQueryInstance = ps.newXQuery(xQueryString);
         try {
            XQueryResult xqResult = xQueryInstance.execute();
            xqResult.printOut();
            System.out.println("\nDone\nThanks for using ApplicationAPIExample"
               + " of this Client Side Library!\nHave fun creating your own"
               + " provenance aware application!\n");
         } catch (XQueryException e) {
            String errorMessage = "Error during xquerying";
            System.out.print(errorMessage);
         }
      }
   }
```


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