Title: Scalability Specification

Author: Workpackage 5 (Scalability)

Editor: John Ibbotson

Authors: John Ibbotson (IBM)

Reviewers: All project partners

Identifier: D5.2.1

Type: Deliverable

Version: 2.0

Date: 27th February 2006

Status: Public

Summary

Scalability has been studied for provenance assertion recording and querying. Conceptually centralised, the provenance store will be distributed and its distributed instances will have to coordinate the storage of p-assertions. The provenance store will have to scale up with the quantity of p-assertions submitted, the number of actors being involved in workflow enactment, and the number of workflows being enacted simultaneously. When dealing with large amounts of data (or very congested networks), we assume that p-assertions are submitted asynchronously. As far as scalability of queries is concerned, the problem is to make sure that, even when p-assertions are distributed at different locations, they can easily be queried or navigated.

Copyright © 2006 by the PROVENANCE consortium

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

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

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

Foreword

This document has been edited by John Ibbotson (IBM) based on input from project partners.

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

Table of Contents

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

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

1 Introduction

After introducing a logical architecture for provenance systems in [CG05] chapter 3, this document now discusses the distribution and scalability of the provenance store component of the provenance architecture. We have identified two forms of scalability that can be applied to the provenance logical architecture. We refer to these as *Architectural* and *Implementation* scalability.

We consider *Architectural Scalability* to be scalability when considered from the point of view of the logical process architecture discussed in [CG05] chapter 1. This views the provenance store as a "black box" consisting of a set of interfaces and associated functionality. No consideration is given to how the internals of the "black box" are implemented. The *Architectural Scalability* will have an impact on any open standards the project proposes for the interfaces to a provenance store and is not influenced by any implementation technology.

Implementation Scalability breaks a provenance store into a set of components that are implemented using some computing technology. The technology is configured in such a way as to meet performance requirements such as numbers of connected actors, system response and p-assertion storage volumes. The *Implementation Scalability* issues are separate to the *Architectural Scalability* issues and do not impact any standards proposals developed by the project.

1.1 Overview of the document

The document has the following structure:

- Section 2 identifies a set of use cases originating from the Requirements workpackage WP2
- Section 3 discusses the Architectural Scalability issues and includes a set of recording patterns that identify communications between key architecture roles
- Section 4 discusses the Implementation Scalability issues
- Section 5 identifies a set of recommendations for addressing the architectural and implementation issues identified in the previous sections
- Section 6 concludes the document

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

2 Provenance scalability use cases

This section discusses a set of provenance store use cases derived from user and software requirements obtained in Workpackage 2.
.

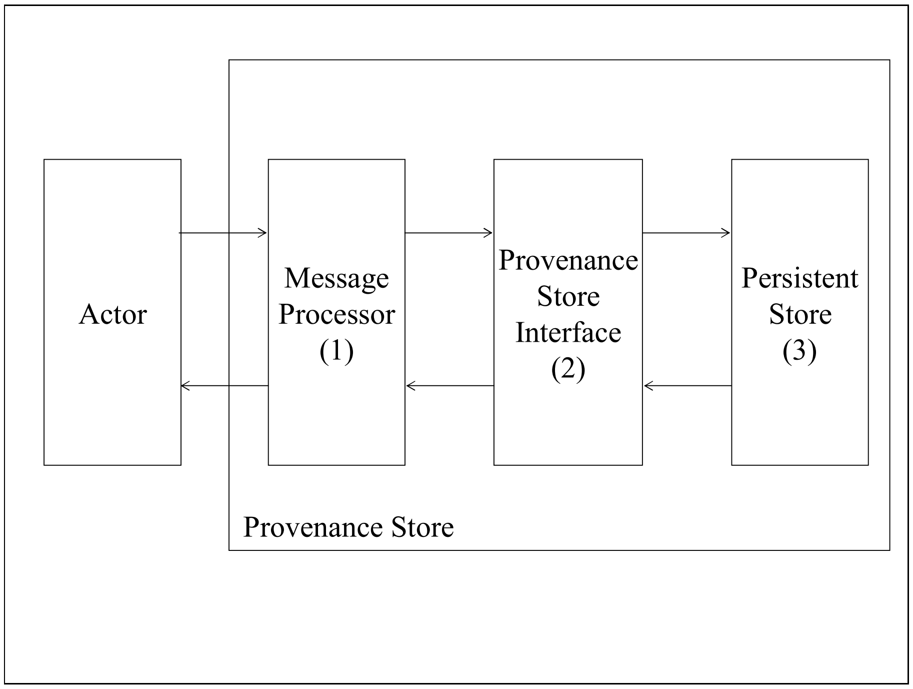

Figure 1: Components of a provenance store

For the purposes of this discussion, a provenance store consists of the following components:

1. An *Actor* is a client to the provenance store and interacts with it through the published provenance store record, query and management interfaces
2. A *Message Processor* receives request messages from actors and returns response messages to them
3. The *provenance store Interface* implements the operations specified by the published provenance store Interface
4. The *Persistent store* stores and maintains recorded p-assertions beyond the lifetime of an Actor or other Grid application

2.1 Large p-assertion messages

Use Case: An actor participates in a process that involves the movement of large amounts of data. Examples of this include the transfer of medical images as part of a Patient Healthcare Record. How does the provenance architecture deal with potentially large p-assertion messages being submitted to the provenance store? The p-assertions may be either interaction or actor state p-assertions. See requirement SC-1-1.

Description: This use case impacts the *Message Processor* and the messages that flow between the provenance store and the recording or querying actor. The scalability issues illustrated by this use case relate to the potential network delays caused by transporting large

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

messages and the potential creation of large objects within the *provenance store Interface* as the data is processed.

There are several alternative solutions to this use case:

1. The provenance store can be located locally to the actor with either a remote or locally available *Persistent store*.
    a. With a remote *Persistent store*, the delay between sending the large message and the *provenance store Interface* responding is minimised. There is still a possible network connection between the *provenance store Interface* and the *Persistent store* which can result in associated delays.
    b. With a locally available *persistent storage*, the network delay is eliminated. However, the locally available *Persistent store* may not be part of a fully managed set of storage resources and its contents will have to be replicated with *Persistent store* optimised for an extended lifetime.
2. Use of SOAP Message Transmission Optimisation Mechanism [GM05] to reduce the p-assertion message size. This recommendation from the W3C provides a way of packaging non-XML data with a SOAP message for transmission. Transforming non-XML data to XML by base 64 encoding is extremely inefficient and results in an increase in message size. The recommendation is compatible with the SOAP processing model and may be used to provide an efficient transfer of non-XML data from an actor to the provenance store.
3. The actor does not record the large amount of data within the p-assertion. Instead, it stores the data within an application datastore and includes a reference to the data within the submitted p-assertion.

2.2 References to data within a p-assertion

Use Case: An actor invokes a service to process a large amount of application data. Instead of sending the data, the actor provides a reference to the stored data and sends that in the invocation message. The processing actor uses that reference to access the data to be processed. At a later time, the provenance trace containing the reference is queried. What must the provenance architecture do to ensure that the reference is still resolvable?

See requirement SC-1-5a, SC-1-5b and SC-1-5c.

Description: This use case bridges the application data domain and the storage of p-assertions within a provenance store. The provenance logical architecture document discusses the identification of data items in order to query its provenance. Conversely, the data identifier allows the data to be identified following a query to a provenance store. For this to be successful, the identification name must be resolvable within the application data store. This implies that the application data store supports a naming convention that is persistent throughout the data lifetime; in other words application data identifiers can always be resolved.

The scope of the provenance project is the management of provenance p-assertions and not the application data to which they refer. Therefore it is assumed that the application database management system (DBMS) is resilient and allows application data identifiers to be resolved.

2.3 Multiple actors making submissions

Use Case: A provenance store is shared by a large number of actors that submit interaction and actor state p-assertions to it. How does the architecture deal with many actors submitting p-assertions simultaneously?

See requirement SC-1-3.

Description: This use case impacts the *Message Processor* of the provenance store. Each p-assertion message received typically results in a thread of execution within the provenance store. The number of concurrent threads that can be managed by the container hosting the implementation, and hence the number of actor requests that can be handled concurrently, is dependent on the design of the provenance store runtime software and the performance of the hardware it is running on.

Providing a solution for this use case relies on a combination of two scaling techniques:

1. **Scaling up** increases the performance of runtime software by increasing the performance of the hardware that is providing the runtime environment.
2. **Scaling out** provides multiple instances of the runtime software on separate processors to allow more connections to be made to the single network *endpoint*. This technique is known as *clustering* and is discussed as an implementation technique later in this document.

For this use case, the logical architecture is unchanged and scalability is introduced as part of the physical implementation of the architecture.

2.4 Multiple actors querying at the same time

Use Case: A provenance store is shared by a large number of actors that want to submit queries. How does the architecture deal with many actors querying a provenance store simultaneously?

See requirement SC-1-4.

Description: This use case has identical issues to the one described in section 2.3.

2.5 Management of large amount of p-assertions

Use Case: A *Persistent store* associated with a provenance store contains large numbers of p-assertions resulting in potentially Terabytes of data. How does the architecture support the management of large amounts of p-assertion data?

See requirement SC-1-6.

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

Description: The architecture should treat the *Persistent store* as a separate, closed component accessible by published interfaces. In most cases, the *Persistent store* will be implemented using a Database Management System (DBMS). Overall sizing of a provenance aware system will establish the expected volumes of p-assertions to be handled by the DBMS and a suitable configuration for the DBMS to support the large volumes must be deployed. The mechanism for managing large volumes of data in a DBMS is well known and outside the scope of the provenance project.

2.6 Degradation of performance whilst submitting p-assertions

Use case: Requirements state that adding provenance capability to an existing or newly designed system must not degrade its performance. The performance metrics listed in the requirements include:

1. Responsiveness of a user interacting via a GUI
2. End to end execution time of a workflow
3. Volume of provenance p-assertions as a percentage to application data

See requirements CR1-1-A, CR1-1-B, CR1-1-C, CR1-1-D, CR-1-2-A, CR-1-2-B and CR-1-2-C.

Description: Issues 1 and 2 identified in this use case may be addressed by using an asynchronous messaging model connect to the provenance store. The asynchronous messaging model does not block any application processing and this, together with good practice for overall system design, will minimise the overall impact of a provenance capability on the system.

Issue 3 may be addressed through system design. The contents of an interaction or actor state p-assertion message are determined by the system designer; constrained by their schemas. Therefore the p-assertion message sizes themselves are not dictated by the provenance architecture.

2.7 Large result sets

Use Case: An actor submits a query to a provenance store using the query interface. The result of the query contains a large number of results in the returned set. This has a potential negative impact on the performance of the provenance store. How does the architecture deal with returning large result sets from a query?

See requirement SC-1-2.

Description: Most information retrieval systems allow restrictions (such as the number of results to be returned) to be placed on submitted queries. The provenance query API should allow this feature and include operations for iterating through the returned result set. For large result sets, deferred presentation of results or redirection of result sets to a localised file systems are possible solutions. An implementation of the architecture should support these different solutions.

2.8 Access to p-assertions for query

Use Case: An actor is submitting p-assertions to a provenance store. At the same time, another actor attempts to query the same provenance store. The architecture should support the dynamic processing of p-assertions, i.e. recorded provenance data should be instantly accessible for queries even if a recording session is still in progress

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

See requirements TR-4-1 and TR-4-2.

Description: Use cases 2.3 and 2.4 address the issue of large numbers of actors recording and querying a provenance store. Each p-assertion record and query is independent of each other and each creates a separate thread of execution within the provenance store. Concurrent processing requests by the provenance store support the implementation of this use case.

2.9 Submission of repeated p-assertions

Use Case: How can an actor, involved in a long iteration in a workflow, avoid submitting the same p-assertions at every iteration step? Examples of this may include the actor functionality and data flow information. Information could be registered elsewhere (say, in a registry such as grimoires), possibly later consolidated into the provenance store, but the actor needs a mechanism to declare that information is to be found elsewhere.

This requirement does not form part of the set collected by workpackage WP2. It was identified during subsequent discussions between the partners

Description: Loops within a workflow will result in the same or partial p-assertion data being repeatedly submitted; albeit with different time stamps. This may result in extremely large amounts of p-assertion data being managed by a provenance store. To reduce this volume, the repeated data can be stored by reference. The reference would consist of an identifier that refers to the p-assertion data which is stored either in a registry or within the provenance store. To support this, the p-assertion schema would have to include suitable identifiers for the complete or partial tree of the p-assertion message. Changes to the schema to support references within the p-assertion would be required.

2.10 P-Assertions and databases

Use Case: Provenance has been studied intensively in database systems. In some specific cases (specialised query language, etc), the provenance of some data **within the database** can efficiently be reconstructed. In this case, if an actor is such a kind of database, the database will **not** submit actor state p-assertions describing data links etc, because these can be computed when and as required. Hence, p-assertions must indicate that useful information can be obtained from the database at provenance query time.

This requirement does not form part of the set collected by workpackage WP2. It was identified during subsequent discussions between the partners

Description: The description of this use case is outside the scope of this document.

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

3 Architectural Scalability

In this section, we discuss the issues relating to the *Architectural Scalability* of a provenance store. The Provenance Architecture describes a set of recording patterns that can be used to influence the design of a provenance store. Given these patterns, we can describe how actors within a workflow can each record their p-assertions in different provenance stores. These provenance stores may be in different domains for reasons of scalability and/or security. The set of p-assertions that documents the generation of some application data may be distributed throughout different provenance stores. The advantages of this architectural approach and a proposed linking mechanism that allows an actor using the provenance query interface to reconstruct a complete provenance trace are described in the Provenance Architecture document.

In this section, we discuss the staging of data within a distributed architecture as a way of scaling the architecture. In addition, we discuss methods to reduce the size of p-assertion messages to improve the performance of a provenance store.

3.1 Data staging

A provenance store that manages p-assertions over an extended lifecycle requires extensive management and data storage features of the kind provided by a commercial database management system. These are typically provided within an organisation on managed servers administered centrally. In most cases, actors will communicate directly to a provenance store whose interfaces provide connections to the data management infrastructure.

There are, however, cases where systems designers may choose to include data staging in their design. Data staging allows a provenance store to be placed closer to the recording actor for reasons of performance. An example of this might be where the documentation of execution is being recorded for a workflow running on a computing cluster with high speed networking support between the clustered processors. Slower speed networks connect the cluster to other servers which include a managed provenance store. Communicating directly with the managed provenance store would impact the performance of the clustered workflow.

For this use case, we identify the requirement for a provenance store to be able to export its p-assertions in a format that can be imported into another provenance store allowing locally staged p-assertions to be moved to a remotely managed provenance store independently of the recording interface. This may be achieved in either of two ways:

1. The provenance management interface includes import and export operations to allow p-assertions to be moved between provenance stores.
2. The provenance management interface allows an alternative protocol such as GridFTP to be used to transfer p-assertions from one provenance store to another.

Data staging may also be used to overcome performance bottlenecks when recording large amounts of p-assertion messages. However, two alternatives to recording large amounts of p-assertion data are now discussed.

3.2 By-Value versus By-Reference recording

Section 2.2 identifies a use case where references to data are included in a p-assertion rather than including the data by-value. Examples of this use case are where interaction p-assertions include large amounts of application data or where the data in a service request or response is private. Examples of private data may include patient records in a healthcare application. The patient records are usually held within a database that contains high levels of access security and the system designers may wish to have a lower level of security for the provenance store.

This use case can be supported by including application data references as optional elements within the p-assertion message schema. Chapter 6 of the provenance logical architecture [CG05] discusses the links between application data items and their associated provenance information. It identifies a set of naming conventions and patterns that can be used to link application data and provenance for querying. These conventions may be used within the p-assertion recording messages to support the by-reference use case.

For by-reference recording of data within a p-assertion, the lifetime of the application data has to be equivalent to that of the provenance store. To resolve a subsequent query to the provenance store, the querying actor also has to have access rights to the application data referred to by the p-assertion. An alternative approach would be for the referenced data to be managed by the provenance store. This approach however requires the provenance store to be at least as secure as the application data store. This scenario would not support the healthcare example cited earlier in this section.

The recommended approach for this scenario would be for the application data to be managed separately from the provenance store and references placed within the p-assertion message to link the p-assertion and application data records.

3.3 Record-Once versus Record-Many

Section 2.9 identifies a use case where the same (or similar) p-assertion message may be recorded multiple times leading to potentially large amounts of redundant information within the provenance store. This use case may be supported by implementing a strategy commonly used by compression algorithms. In compression algorithms, a catalogue of common patterns that appear in the data is created. Each entry in the catalogue has an identifier which is typically smaller than the pattern it identifies. To compress the data, when a particular pattern is identified it is replaced by its catalogue identifier which results in a compressed version of the data. Decompression of the data is performed by replacing the catalogue identifiers within the data with the matching patterns from the catalogue which is attached to the compressed data.

For the provenance recording protocol, a p-assertion is assigned a unique identifier. This is similar to the catalogue identifier described above. If an actor records a subsequent p-assertion that is the same as a previously recorded one, it may choose to replace the contents of the p-assertion message with the previous unique identifier thereby reducing the size of the p-assertion message. If there are multiple p-assertion messages to be recorded, it may repeatedly replace the p-assertion message with the unique identifier. Querying the p-assertion will include a lookup stage that replaces the recorded identifier with the contents of the message that matches the identifier. This is analogous to the decompression of data described above.

The previous description describes the replacement of a complete p-assertion message by its identifier. This mechanism can be extended by including identifiers at roots of sub-trees within the p-assertion message schema. This allows replacement at the sub-tree level within a p-assertion message in cases where only part of a p-assertion message is repeatedly recorded.

3.4 Architectural scalability requirements

The following requirements are identified for *Architectural Scalability*: The identifiers refer to requirements justified in section 10.3 of the provenance Logical Architecture.

1. **SL-1** The provenance query interface should support the retrieval of large result sets. This will provided by caching of the results within the provenance store. A set of interface operations will allow actors to iterate and retrieve the cached results in response to a query request.

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

2. **SL-2** The provenance recording interface should support the linking between application data and p-assertions using the by-reference model. P-assertion message schemas will allow references to application data to be included in instance messages. The provenance store will not be responsible for managing the application data lifecycle.
3. **SL-3** The provenance recording interface should support the recording of repeated p-assertions by identifying the message and its sub trees with an identifier. These identifiers may be used in subsequent p-assertions to replace the repeated information.
4. **SL-4, SL-5** The provenance management interface should support the import and export of p-assertions allowing provenance store to exchange their contents. Options for providing this support are:
    a. Explicit import and export operations on the management interface
    b. Operation(s) allowing provenance stores to communicate via another protocol (e.g. GridFTP)
5. **SL-4, SL-6** The provenance store recording interface should support the SOAP MTOM recommendations for non-XML data.

3.5 Use cases satisfied by architectural scalability

Architectural Scalability will satisfy the following Use Cases:

Large Result Sets
This is supported by *Architectural Scalability* recommendation 1.

References to Data within a P-Assertion
This is supported by *Architectural Scalability* recommendation 2.

Submission of Repeated P-Assertions
This is supported by *Architectural Scalability* recommendation 3.

Large P-Assertion Messages
This is supported by *Architectural Scalability* recommendations 4 and 5

3.6 Conclusions

This section has discussed the *Architectural Scalability* of the provenance architecture. This provides an external view of the logical architecture which will influence the draft open standards developed as part of the project. It has identified a set of recording patterns that can be used to distribute p-assertions between distributed provenance stores and proposed a linking mechanism for a complete set of p-assertions relating to application data to be assembled following a query to the provenance store. It has also discussed the use of data staging for use cases where large p-assertion messages are recorded.

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

4 Implementation Scalability

In this section, we discuss the issues relating to the *Implementation Scalability* of a provenance store. By *Implementation Scalability*, we mean aspects of scalability that are influenced by the way an architecture is implemented using a particular technology rather than the generic influences discussed in the previous section. We first identify a set of components and assess their impact on overall system scalability. The management of state within a provenance store is then discussed leading to the requirements for a clustered implementation. Finally we present a set of recommendations for implementing a scalable provenance store service.

4.1 Components of a provenance store

This section identifies components of the provenance logical architecture and assesses their impact on the scalability of the overall architecture. In particular we address components of the provenance store and their impact on overall system scalability.

The provenance logical architecture classifies the actors involved in the provenance lifecycle according to their role in a provenance system. Briefly, the responsibilities of each role are as follows.

- An *application actor* is responsible for carrying out the application's business logic.
- A *provenance store* is responsible for making persistent, managing and providing controlled access to recorded p-assertions.
- An *asserting actor* is an actor that creates p-assertions about an execution.
- A *recording actor* is an actor that submits p-assertions to a provenance store for recording.
- A *querying actor* is an actor that issues provenance queries to a provenance store.
- A *managing actor* is an actor that interacts with the provenance store for management purposes.

Central to the provenance architecture is the notion of a provenance store, which is a service designed to store and maintains a provenance representation beyond the lifetime of an application. Such a service may encapsulate at its core the functionality of a physical database, but also provides additional functionality pertinent to the requirements of the provenance architecture. In particular, the provenance store's responsibility is to offer long-term persistence of p-assertions.

We can further divide the provenance store into three components that jointly provide an implementation of the provenance store role. These are illustrated in the provenance store components figure:

1. A *Message Processor*. This component receives request messages from actors and returns response messages to them. It 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 *Message Processor* include translating the message into a format that is understandable by the *provenance store Interface*. For example, this may include converting an on-the-wire XML representation into a collection of Java objects and dispatching them to the provenance store Interface.
2. A *provenance store Interface*. This component implements the operations specified by the provenance store external interface. It receives translated request messages from the *Message Processor* and implements the functionality of the provenance recording, query and management interfaces. This component sits between the *Message Processor* and the *Persistent store*.

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

3. A *Persistent store*. This component stores and maintains p-assertions beyond the lifetime of a Grid application. Such a component encapsulates the functionality of a physical database; in particular it has the responsibility to provide long-term persistence of p-assertions.

4.2 Managing state

A typical interaction between an actor and a provenance store consists of a single session. An actor sends a request to the *Message Processor* which forwards it to the *provenance store Interface*. The *provenance store Interface* may then access the *Persistent store* to record or query p-assertions. The response to the actor's request then flows in the reverse direction resulting in the results of the requested operation being returned to the actor. This pipeline is fundamental to the operation of the provenance store; equally it is the cause of bottlenecks in system performance and its design and implementation must be carefully considered.

The role of a provenance store is to persistently manage a set of recorded p-assertions. The management of p-assertions is *stateful*. By this we mean that a recorded p-assertion can be subsequently retrieved from a provenance store for later analysis or processing. The term *state* in this context refers to the last-known or current status of an application or a process; in this case, the actor. The terms *maintaining state* and/or *managing state* refer to keeping track of the condition of the process. The provenance store, as far as the actor is concerned, has to be *stateless*. By this we mean that any request to the provenance store is processed without any knowledge of any previous requests. This is the model used by the World Wide Web which is intrinsically stateless because each request for a new Web page is processed without any knowledge of previous pages requested. Some commentators consider this is to be one of the drawbacks to the HTTP protocol. Because maintaining state is extremely useful, programmers have developed a number of techniques to add state to the World Wide Web. These include server APIs, such as NSAPI and ISAPI, and the use of cookies.

However, for a scalable architecture, a *stateless* provenance store is required. We can identify two components of state that must be considered to achieve an overall stateless design. These are *application state* and *connection state*.

Application state is represented by the set of p-assertions recorded by actors for subsequent retrieval via the provenance query interface. The *Persistent store* component of the provenance store has the responsibility for managing these p-assertions throughout the provenance lifecycle. Typically this component will be implemented using a database management system (DBMS) that can manage the p-assertions over an extended period of time.

Connection state only exists for the lifetime of a single thread of execution within the provenance store and consists of a session established between the requesting actor and the provenance store. The session exists to allow a response to the request to be returned to the actor. Once the response has been returned, the session may be torn down. This ensures that in terms of connection, a request by an actor is not dependent on any previous requests; its importance will be discussed later when clustering as an implementation technology is introduced.

Application state is passed between an actor and a provenance store through the schemas of request and response message instances. Examples of application state information may include:

- Identifiers for *Persistent stores*
- P-assertion headers and provenance context headers
- Security tokens used by the provenance store to authenticate actors

In summary, for a scalable provenance architecture, the provenance store has to appear stateless in terms of connections when viewed by a requesting actor. *Application state* is held within the *Persistent*

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

store component of the provenance store. References to the *application state* are contained within the request and response messages exchanged between an actor and the provenance store.

4.3 Scalability through clustering

The purpose of the provenance logical architecture is to describe the architecture in terms that are independent of any physical implementation. This section discusses how the components of a provenance store may be mapped to an implementation infrastructure that meets the scalability requirements Use Cases from section 2 that relate to actor connections and response.

The scalability strategy referred to as scaling-out provides for resources to be made available to support large numbers of connections to an application server used to implement a provenance store. The strategy is most commonly realized using clustering technology. When clustering is implemented, a *Load Balancing* component is added to a set of application servers that allows the servers to be viewed as a single logical network endpoint. Each application server will contain a replicated instance of a provenance store. This is illustrated in Figure 6.

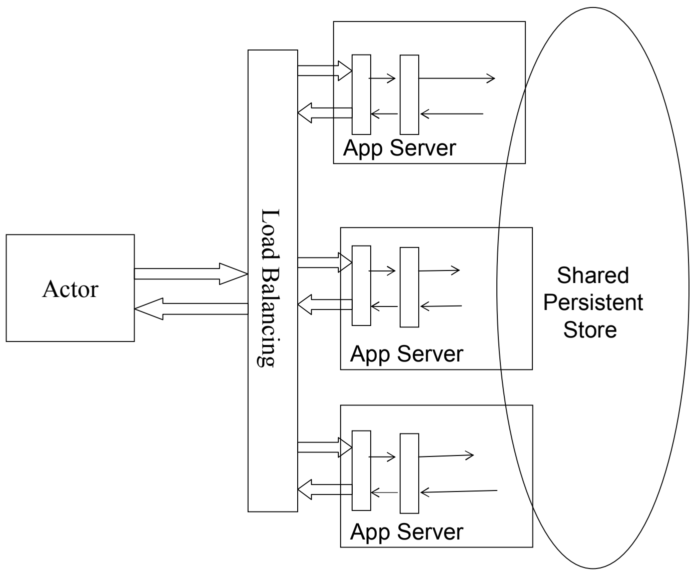

Figure 2 Clustered Containers

The Load Balancer allows connection requests from an Actor to be dynamically re-directed to one of any application servers within the cluster. These application servers are deployed on different physical machines each with its own IP address within the cluster. Two popular methods of load balancing in a cluster are *DNS round robin* and *hardware load balancing*. DNS round robin provides a single cluster logical name, returning any IP address of the nodes in the cluster. This option is inexpensive, simple, and easy to set up, but it does not provide any server affinity or high availability. In contrast, hardware load balancing solves the limitations of DNS round robin through virtual IP addressing. Here, the load balancer shows a single IP address for the cluster, which maps the addresses of each machine in the cluster. The load balancer receives each request and rewrites headers to point to other machines in the cluster. If any machine in the cluster is removed or fails, the changes take effect immediately. The advantages of hardware load balancing are server affinity and high availability; the disadvantages are that it's expensive and complex to set up.

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

There are many different algorithms to define the load distribution policy, ranging from a simple round robin algorithm to more sophisticated algorithms used to perform the load balancing. Some of the commonly used algorithms are:

- Round-robin
- Random
- Weight-based
- Minimum load
- Last access time
- Programmatic parameter-based (where the load balancer can choose a server based upon method input arguments)

Load-balancing algorithms affect statistical variance, speed, and simplicity. For example, the weight-based algorithm has a longer computational time than the other algorithms. Clustering is supported in many proprietary and open source servers.

A provenance store that is easily clusterable must have no affinity with any of the actors or applications that connect to it. To achieve this, the service must be stateless apart from handling any connection state as discussed in section 4.2. All state information that relates to a connecting actor or application must be stored in the associated stateful resources as described in previous sections. This is compatible with the provenance logical architecture where the stateful information from an actor is contained in the p-assertion schema. Connection state is sufficient to ensure that a response to a request is returned via the correct connection thread.

A further condition is that the stateful resources must be shared between all runtime containers within the cluster. Since there must be no affinity between an actor and the server it is connected to, all state must be accessible by all containers hosting instances of the provenance store within the cluster. For example, a workflow enactment engine that is recording many p-assertions to a provenance store will have all the submissions stored within the same Persistent store irrespective of which container instance the submission requests are processed by.

The provenance architecture will also make use of a number of system "middleware" services. Examples of such services will include naming and security authorisation and authentication. To ensure that there are no bottlenecks in overall system performance, these service implementations must also be scalable. These services will include commercial or open source database management systems to implement the *Persistent store* component of the provenance store.

4.4 Implementation scalability recommendations

The following requirements are identified for *Implementation Scalability*: The identifiers refer to requirements justified in section 10.3 of the provenance Logical Architecture.

1. **SL-9** provenance actors should have no affinity to a provenance store. Actors communicate with the single published network endpoint of the provenance store. This recommendation ensures that a provenance store can be scaled using clustering technology.
2. **SL-7** Implementations of the provenance recording, query and management interfaces should be clusterable.
3. **SL-8** The provenance store Persistent store component should be implemented using a proprietary or open source database management system.

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

4.5 Use cases satisfied by implementation scalability

Implementation Scalability will satisfy the following Use Cases:

Multiple actors making submissions
This is supported by *Implementation Scalability* recommendation 2.

Multiple actors querying at the same time
This is supported by *Implementation Scalability* recommendation 2.

Management of large amount of P-Assertions
This is supported by *Implementation Scalability* recommendation 3.

Degradation of Performance whilst Submitting P-Assertions
This is supported by *Implementation Scalability* recommendation 2.

4.6 Conclusions

This section has discussed how scalability use cases can be supported through the implementation of a provenance store. It considers how the three components of a provenance store may be implemented and what computing technologies would be appropriate to support a given set of scalability use cases.

The solutions discussed in this section do not impact the external or *Architectural Scalability* issues that would influence a set of open standards the project may propose.

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

5 Recommendations for a Reference Implementation

This section addresses in more detail the architectural and implementation recommendations made in previous sections.

5.1 Architectural scalability

This section addresses in more detail the recommendations from the Architectural Scalability section.

5.1.1 Large query result sets

Submitting a query to a provenance store may generate a result set that contains a large number of p-assertions. A mechanism is required to allow a querying actor to interact with the returned result set in a way that minimises network traffic and any client loading. To achieve this, the result set generated by the query should be cached within the provenance store and a set of operations provided to allow a client to iterate through the result set; retrieving only a subset of the total result set at a given time. This pattern is commonly used in information retrieval architectures and can be applied directly to the provenance Architecture.

An instantiation of this pattern is proposed by the GGF Database Access and Integration Services Working Group in their latest WS-DAIX specification document. This is shown in a simplified form in the following diagram.

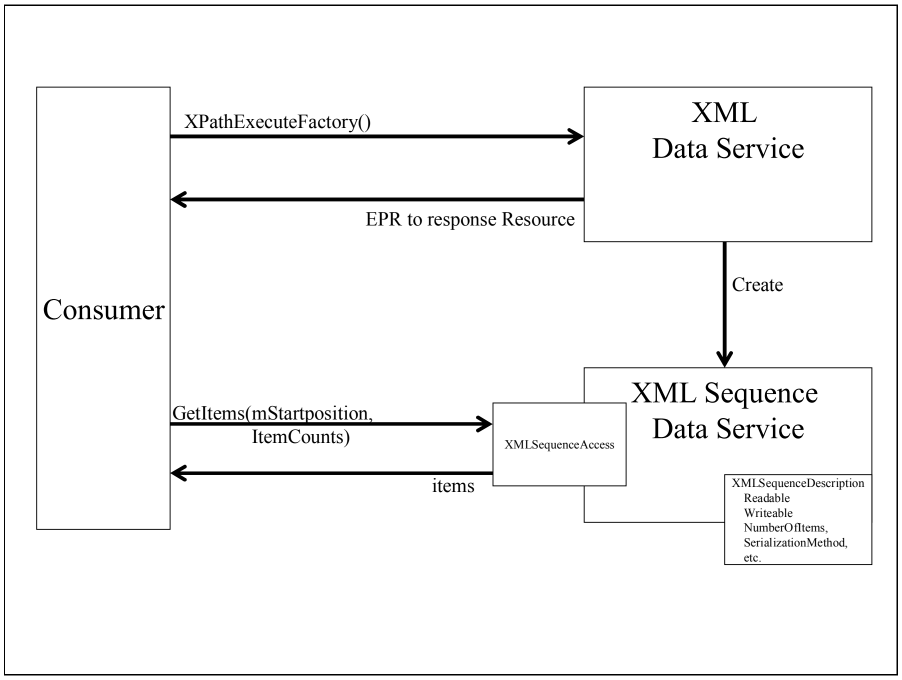

Figure 3: WS-DAIX XPathFactory pattern

In the XPathFactory pattern, the result set is not directly returned to a consumer in response to a query. The XPathExecuteFactory() request by a consumer contains the XPath expression for the query. Instead of responding with an XML document containing the result set, the data service creates a new resource; an XML Sequence Data Service. This resource contains the result set produced by the evaluation of the submitted XPath expression. Once created, the data service returns the WS-Addressing endpoint reference of the newly created resource to the requesting consumer.

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

The newly created resource represents the result set as an XML Sequence. It advertises via its properties the number of elements in the sequence and whether the resource is modifiable or not. The consumer is able to interrogate these properties and can retrieve items from the sequence using the GetItems(start, count) operation which responds with a set of items from the stored sequence. Full details of the XMLSequenceAccess interface are provided in the WS-DAIX specification.

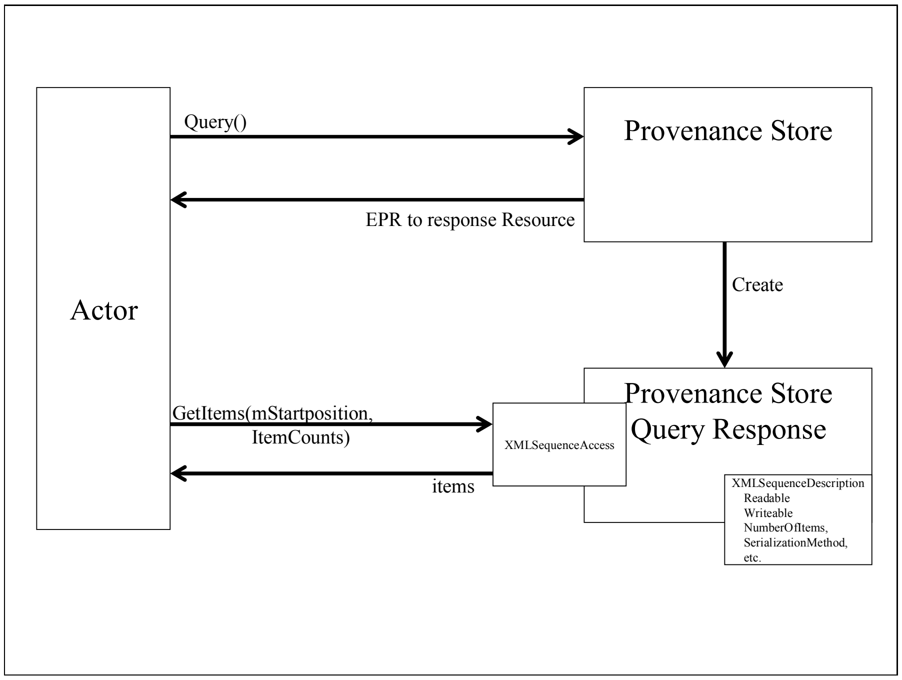

Figure 4: provenance use of a WS-DAIX compliant XML Sequence data service

The provenance store query interface will be a superset of the XML data service described in the WS-DAIX specification. It will support queries using an XPath expression, but it will also support query operations that are semantically meaningful to the provenance architecture. However, the provenance store should follow the WS-DAIX XPathFactory pattern and create a query response resource that is WS-DAIX compliant by supporting the XMLSequenceAccess interface. The WSDL for the XMLSequenceAccess interface is shown in the appendix.

By creating a new resource for the query result, we remove the overhead of lifecycle management and result interaction from the provenance store or querying actor and encapsulate it into the separate resource. The interaction operation GetItems() allows the querying actor to manage its access to the results instead of tying up resources within the provenance store. The element types contained within the XML sequence of the result set will differ depending upon the query expression. It is expected that the most frequently returned data type will be p-assertions. Support for this type will allow an export feature to be built onto the query interface. This is described in more detail in the import/export section.

Note that the query response resource may be created on a container remote from the provenance store resource allowing further distribution of the provenance logical architecture.

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

5.1.2 Application data linking by-reference

5.1.3 Repeated p-assertions

This use case does not appear in the Provenance project Aerospace or Organ Transplant Management applications. This recommendation will not be supported in the reference implementation of the architecture.

5.1.4 Import and export of p-assertions

Section 5.1.1 described the management of large result sets when querying a provenance store. It also states that a sequence of p-assertions may be returned as one example of the type of document resulting from a query. In this way, the contents of a provenance store may be extracted and exported to a querying actor. The XPath expression that created the result set then provides a filter that can be applied when exporting the contents of a provenance store.

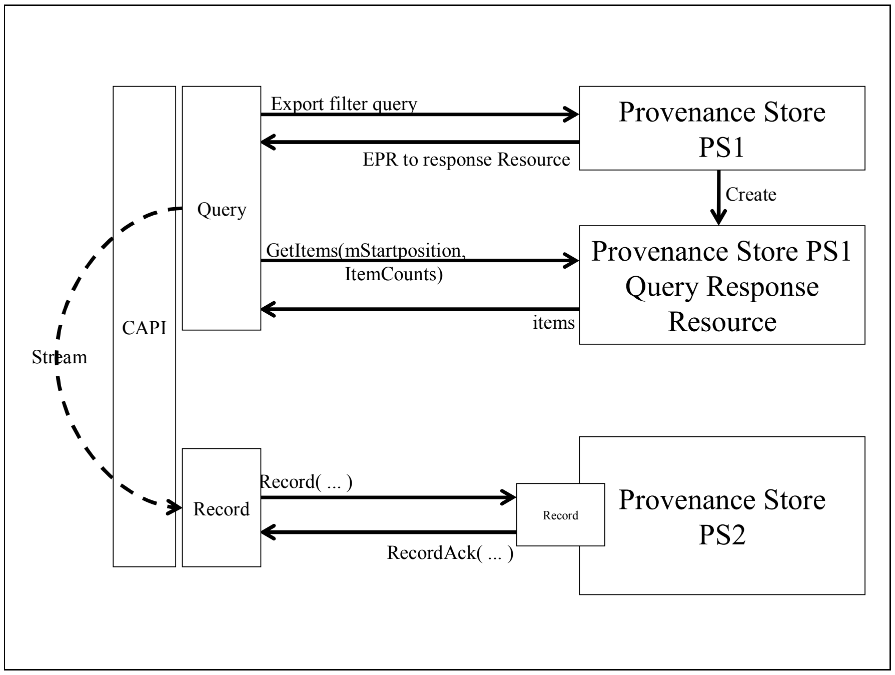

Figure 5: Import/Export between provenance stores

The import/export figure shows how two data service interfaces can be combined. It assumes that there is a Client API (CAPI) that includes a high level programming interface that hides the low-level interfaces to the provenance store operations. In this diagram, the provenance store query interface discussed previously provides the export functionality. A filter query is applied to the provenance store PS1 and the results extracted via the Query Response resource for PS1 using the XMLSequenceAccess interface. This provides a sequence of p-assertions that can be mapped directly to the Record interface of a provenance store that provides a bulk import capability.

The p-assertions recorded in a set of distributed provenance stores represent a directed graph of the historical processes that created some application data. To recover the complete graph, the set of p-assertions have to be assembled from the set of provenance stores. Within a p-assertion, the viewLink element includes the URL of a provenance store where the p-assertions linked to this interaction are stored. If a set of p-assertions are exported from one provenance store and imported into another, these links become invalid; this is analogous to dangling links from a web page.

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

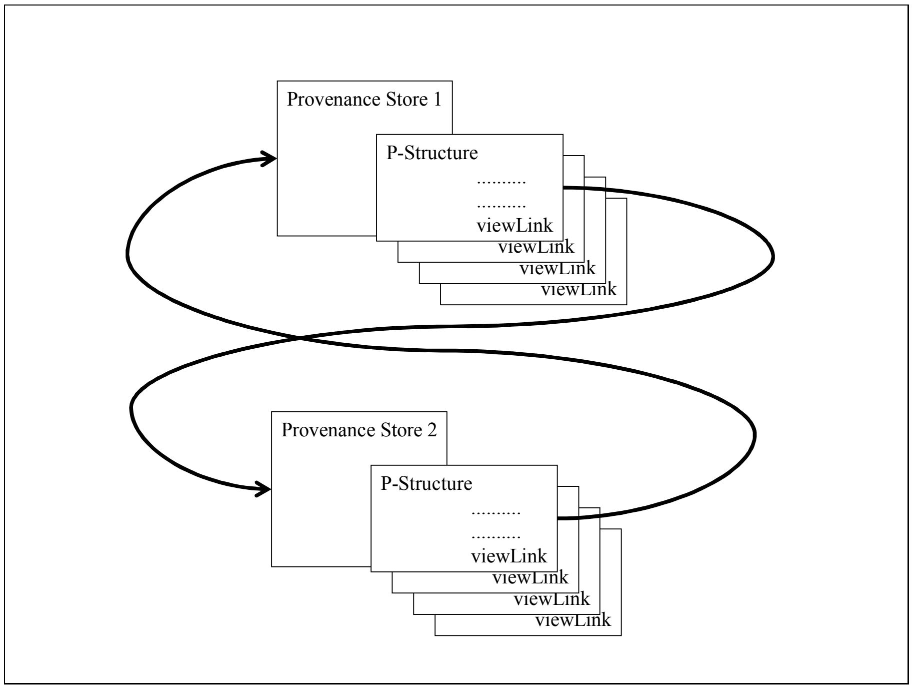

Figure 6: Cross linking between provenance stores

The cross linking is illustrated above. In this example, the documentation for a distributed process is recorded in two provenance stores. P-assertion elements within provenance store 1 will link to provenance store 2 with reverse links between provenance store 2 and provenance store 1. If the contents of provenance store 2 are exported and imported to provenance store 3, then the viewLinks in provenance store 1 will have to be modified to point to provenance store 3 instead of 2.

This viewLink update operation is implementation dependent and outside the scope of any standardised interfaces. An implementation approach has been identified:

1. viewLink updates are owned by the actor responsible for the export/import process. It traverses the exported set of p-assertions and identifies the set of provenance stores that need to be updated.
2. A provenance store management operation is implemented that updates the viewLinks associated with a set of interactions. This operation is invoked in the set of provenance stores identified in step 1.
3. Following successful completion of the update, the exported set of p-assertions are imported into the target provenance store

The query/record operations that implement the export/import functions will implement a copy operation. Since the provenance architecture does not permit deletion of information from a provenance store, a move semantic is not allowed.

5.1.5 Non-XML data in p-assertions

Many forms of data are not adequately represented in XML. Examples of these forms are multimedia types such as images, audio and video. These forms of binary (non-XML) data may be converted to XML using encoding techniques such as base-64, but this encoding results in a degradation of performance due to additional encoding/decoding processing and greatly increased message size for the encoded data. For scalability and performance, design principles advise that conversion of binary data into XML is should not be attempted unless absolutely essential.

For the provenance architecture, three issues need to be addressed:

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

1. The storage of non-XML data within a provenance store
2. The structure of record and query messages containing non-XML data
3. Analysis of p-assertions containing non-XML data

For issue 1, the storage of binary data within a database management system is common practice. In relational databases, binary information may be stored as a Binary Large Object (BLOB) either within a table or as a file on an associated file system. Similar support is starting to be provided for XML databases but so far it is not as sophisticated. An interim solution may be provided by storing the non-XML portion of a p-assertion as a file whose URL is stored within the provenance store PStructure. Retrieval of a p-assertion in response to a query would reconstruct the p-assertion by combining the XML and non-XML content into a single message.

Issue 2 may be addressed by the provenance store interfaces supporting the MTOM/XOP model proposed by the W3C XML Protocol Working Group [**GM05**]. This extension to the SOAP standard allows for non-XML data to be included in a MIME multipart message in addition to the XML data of SOAP.

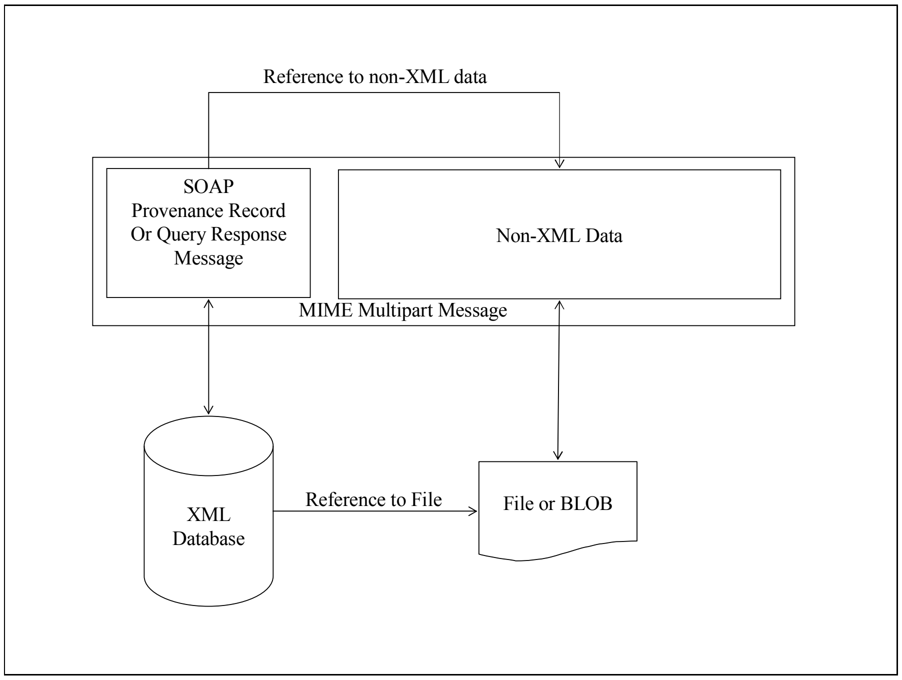

Figure 7: Managing non-XML Data

This is illustrated in the figure above. In the case of a recorded p-assertion, the client API constructs a MIME multipart message. The first part contains the SOAP message with a second part containing the non-XML data. When building the message, the client API includes a reference in the SOAP message that points to the non-XML part of the multipart message. This usually takes the form of the MIME content identifier for the part. Note that Axis2 provides support for this message model.

A provenance store would store the two message parts separately or together depending upon its capabilities. In all cases, the SOAP message would be mapped into a PStructure and stored in an XML database. If the database supported BLOBS, the non-XML data would be stored as a BLOB within the database. An alternative would be for the non-XML data to be stored as a file in an associated file system. In either case, the PStructure would contain a reference to the non-XML data. This reference would be directly in the case of BLOB support or indirectly via a URL when files are used. When generating a response to a query, the provenance store would generate a MIME multipart message containing the XML and non-XML data for returning to the querying actor.

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

Issue 3 is outside the scope of the provenance store and would be supported by tools that form part of the provenance architecture. Deliverable D6.1.1 describes a comparison mechanism that makes use of a plug-in algorithm that represents an application's understanding of similarity. This forms the basis of the analysis of p-assertions containing non-XML data.

5.2 Implementation scalability

This section addresses in more detail the Implementation Scalability Recommendations.

5.2.1 Clustering and non-affinity

As connections are made to an application server such as Apache Tomcat, the server identifies which installed service they are intended for and invokes the appropriate service. Typically, this is done on a thread per connection basis with each connection having its own thread of execution. For a single instance of an application server, there are a finite number of threads available. Once this number has been exceeded, the server refuses any more attempts to connect to it. The purpose of clustering is to increase the number of available connections to a service by increasing the amount of computing resources devoted to support connection requests.

Clustering allows a number of application servers to be run as a single logical entity. Each server runs on separate hardware allowing scalable deployment of the clustered application server. A load balancing application sits between the HTTP connection and the set of servers and allocates the connection request to a server instance. In this way, the number of connections served can be increased by adding more instances of the application server to the cluster. All instances of the application server have access to shared resources such as database resources allowing state to be shared.

A provenance store that is clusterable must have no affinity with any of the actors or applications that connect to it. This condition means that each connection request by an actor to a clustered application server must not be dependent on any previous connection state. All state information that relates to a connecting actor or application must be stored in an attached database. This is compatible with the provenance architecture where stateful information from an actor is expressed as a p-assertion. Connection state is sufficient to ensure that a response to a request is returned via the correct connection.

The WS-ResourceFramework (WS-RF) supported by the Globus Toolkit version 4 supports this recommendation by separating the stateful and stateless components of a service into a WS-Resource and associated web Service. There is no impact on the provenance architecture; clustering is supported through the architecture's implementation.

5.2.2 Persistent stores

Data services including support for XML document management will be provided by a persistent database component in the provenance implementation. The OGSA-DAI project already provides these services and will used in the provenance reference implementation. The properties of database management systems for storing large quantities of data are already well known and outside the scope of the provenance project.

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

6 Conclusion

In this document, we discussed scalability issues that are relevant in the context of provenance. The scalability architecture for the provenance store is then presented along with an explanation of the functionality of its constituent components including an illustration of how these various components interact when managing state within the provenance store. We present a set of use cases from the Requirements workpackage which are discussed with respect to the provenance store components.

Scalability may influence the provenance architecture generically or an implementation of the architecture. We identify which use cases may be discussed in the context of architectural or implementation scalability. Finally we provide a set of recommendations for an implementation of the architecture based on the Globus Toolkit GT4 which supports the Web Services Resource Framework set of standards.

Future work in this workpackage will focus on the scalable reference implementation. This work will be carried out in conjunction with the Security and Implementation workpackages WP4 and WP9.

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

References

[AIS77] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. A Pattern Language. Oxford University Press, 1977.

[Ale79] Christopher Alexander. The Timeless Way of Building. Oxford University Press, 1979.

[CG05] Chen L et al. An Architecture for provenance Systems. Internal document for provenance project.

[G05] P. Groth. provenance in Large Scale, Open, Distributed Systems, July 2005. Available from http://www.ecs.soton.ac.uk/~pg03r/mypapers/PaulMiniThesisFinal.pdf

[GM05] M. Gudgin et al. SOAP Message Transmission Optimization Mechanism, W3C Recommendation, January 2005. Available from http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns. Addison-Wesley Professional, 1995.

[DAIS-WG] Database Access and Integration Services Working Group. See https://forge.gridforum.org/projects/dais-wg

[XUPDATE] XML Update Language. See http://xmldb-org.sourceforge.net/xupdate/

Appendix A Interface Specifications

A.1 The XMLSequenceAccess Interface

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="wsdaix"
                  targetNamespace="http://www.ggf.org/namespaces/2005/09/WS-DAIX"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:wsdai="http://www.ggf.org/namespaces/2005/09/WS-DAI"
                  xmlns:wsdaix="http://www.ggf.org/namespaces/2005/09/WS-DAIX">

<!-- WSDL IMPORTS ################################################## -->
    <wsdl:import location="./wsdai_core_porttypes.wsdl"
                 namespace="http://www.ggf.org/namespaces/2005/09/WS-DAI"/>

<!-- WSDL TYPES #################################################### -->
    <wsdl:types>
        <xsd:schema targetNamespace="http://www.ggf.org/namespaces/2005/09/WS-DAIX"
                    elementFormDefault="qualified">
        <xsd:import namespace="http://www.ggf.org/namespaces/2005/09/WS-DAI"
                    schemaLocation="./wsdai_core_types.xsd" />
        <xsd:include schemaLocation="./wsdaix_xmlsequence_types.xsd" />

        <!-- ############################ -->
        <!-- ### Common Message Types ### -->
        <!-- ############################ -->

        <!-- ############################# -->
        <!-- ### XUpdate Message Types ### -->
        <!-- ############################# -->
        <xsd:element name="InvalidStartPositionFault">
            <xsd:complexType/>
        </xsd:element>

        <xsd:element name="InvalidCountFault">
            <xsd:complexType/>
        </xsd:element>

        <xsd:element name="GetItemsRequest">
            <xsd:complexType>
                <xsd:complexContent>
```

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

```
                    <xsd:extension base="wsdai:RequestType">
                        <xsd:sequence>
                            <xsd:element name="StartPosition" type="xsd:integer"/>
                            <xsd:element name="Count" type="xsd:integer"/>
                        </xsd:sequence>
                    </xsd:extension>
                </xsd:complexContent>
            </xsd:complexType>
        </xsd:element>

        <xsd:element name="GetItemsResponse">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="wsdai:Dataset"  minOccurs="1" maxOccurs="1"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:schema>
    </wsdl:types>

<!-- WSDL MESSAGES ################################################# -->

    <!-- ############################################### -->
    <!-- ### GetXMLSequencePropertyDocument Messages ### -->
    <!-- ############################################### -->
    <wsdl:message name="GetXMLSequencePropertyDocumentRequest">
        <wsdl:part name="GetXMLSequencePropertyDocumentRequest"
                   element="wsdai:GetDataResourcePropertyDocumentRequest" />
    </wsdl:message>

    <wsdl:message name="GetXMLSequencePropertyDocumentResponse">
        <wsdl:part name="GetXMLSequencePropertyDocumentResponse"
                   element="wsdaix:XMLSequencePropertyDocument" />
    </wsdl:message>

    <!-- ######################### -->
    <!-- ### GetItems Messages ### -->
    <!-- ######################### -->
    <wsdl:message name="GetItemsRequest">
        <wsdl:part name="GetItemsRequest"
                   element="wsdaix:GetItemsRequest"/>
    </wsdl:message>

    <wsdl:message name="GetItemsResponse">
        <wsdl:part name="GetItemsResponse"
                   element="wsdaix:GetItemsResponse"/>
    </wsdl:message>

    <wsdl:message name="InvalidStartPositionFault">
        <wsdl:part name="InvalidStartPositionFault"
                   element="wsdaix:InvalidStartPositionFault"/>
    </wsdl:message>

    <wsdl:message name="InvalidCountFault">
        <wsdl:part name="InvalidCountFault"
                   element="wsdaix:InvalidCountFault"/>
    </wsdl:message>

<!-- WSDL PORT TYPES ############################################### -->
    <wsdl:portType name="XMLSequenceAccessPT">

        <wsdl:operation name="GetXMLSequencePropertyDocument">
            <wsdl:input  name="GetXMLSequencePropertyDocumentRequest"
                         message="wsdaix:GetXMLSequencePropertyDocumentRequest" />
            <wsdl:output name="GetXMLSequencePropertyDocumentResponse"
                         message="wsdaix:GetXMLSequencePropertyDocumentResponse" />
            <wsdl:fault  name="InvalidResourceNameFault"
                         message="wsdai:InvalidResourceNameFault" />
        </wsdl:operation>

        <wsdl:operation name="GetItems">
            <wsdl:input message="wsdaix:GetItemsRequest"/>
            <wsdl:output message="wsdaix:GetItemsResponse"/>
```

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

```
            <wsdl:fault  name="InvalidResourceNameFault"
                         message="wsdai:InvalidResourceNameFault" />
            <wsdl:fault message="wsdai:InvalidDatasetFormatFault"
                        name="InvalidDatasetFormatFault"/>
            <wsdl:fault message="wsdai:NotAuthorizedFault"
                        name="NotAuthorizedFault"/>
            <wsdl:fault message="wsdai:ServiceBusyFault"
                        name="ServiceBusyFault" />
            <wsdl:fault name="InvalidStartPositionFault"
                        message="wsdaix:InvalidStartPositionFault" />
            <wsdl:fault name="InvalidCountFault"
                        message="wsdaix:InvalidCountFault" />
        </wsdl:operation>

    </wsdl:portType>

</wsdl:definitions>
```

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