Title:	*Tools Description Document*
Author:	*Work Package 6 ("Tools and Setup")*
Editor:	*Omer F. Rana (UWC)*
Reviewers:	*Luc Moreau (SOTON), Simon Miles (SOTON), Steven Willmott (UPC), Guy Kloss (DLR), Javier Vazquez (UPC)*
Identifier:	*D6.1.1*
Type:	*Deliverable*
Version:	*6.0*
Date:	*December 14, 2005*
Status:	*Public*

Summary

This document contains a description of different software libraries ("tools") that may be used to access a Provenance Store. Such tools make up an integral part of a "Provenance System", and enable an application to make queries, submit data and manage one or more Provenance Stores. Relationships to the Software Requirements Document (D2.1.2) and the Logical Architecture are also provided.

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

Foreword

This document has been compiled by Omer Rana (UWC). Other contributors to the document include:

- John Ibbotson and Alexis Biller
- Luc Moreau, Paul Groth and Simon Miles
- Arnaud Contes, Vikas Deora, Ian Wootten

Table of Contents

Foreword **3**

Table of Contents **4**

Table of illustrations **5**

List of definitions and acronyms **6**

1 Introduction **7**

1.1 Purpose of the document *7*
1.2 Overview of the document *7*

2 Tool Requirements **8**

2.1 Relationship to Software Requirements Document *8*
2.2 Relationship to Logical Architecture *10*
Table 1 ..... *19*
2.3 Types of Tools and Scope *19*

3 Tool Suite Description and Implementation **22**

3.1 Navigation *22*
3.2 Relationship Definitions *22*
3.3 Analysis *23*
3.3.1 PAAM : Provenance Assertion Abstraction Mechanism *24*
3.3.2 Assertion Engine *25*
3.4 Comparison *26*
3.5 Conflict *26*
3.6 Temporal *27*
3.7 Graphical Interface and Portlets *27*

4 Tools and PS Interaction **30**

5 Tools and Application Interaction **31**

6 Types of Users **35**

6.1 Examples of Usage **37**

7Interaction with other Work Packages **39**

8Summary and Future Plan **40**

References **42**

Table of illustrations

Figures:

1. High level interaction between Tool Suite, Provenance Store and Application.
2. Comparison Plug-in Architecture.
3. Assertion combining User and Provenance defined data.
4. UML Class diagram of the Tool Suite.
5. UML Sequence diagram of the Tool Suite (on two pages).
6. Tool Interaction.
7. eXo Portal Framework.

List of definitions and acronyms

- *Actor:* An individual or an organisation that is involved in a data manipulation process.
- *API*: Application Programming Interface.
- The *provenance of a piece of data* is the documentation of the process that produced that data.
- *GUI:* Graphical User Interface.
- *JSR:* Java Specification Request.
- *WSRF:* Web Services Resource Framework.
- *WSRP:* Web Services for Remote Portlets.
- *PA:* A p-assertion is an assertion that is made by an actor and pertains to a process.
- *PS:* Provenance Store.
- *Process Documentation:* The documentation of a process consists of a set of PAs made by the actors involved in the process.
- *Workflow:* The process by which a series of tasks are executed in a specific sequence; including the specification of how outputs of tasks are routed to the inputs of other tasks or stored, whichever action is required.
- *Workflow enactment engine:* A software program that conducts the execution of a workflow in accordance with the specification of the workflow. In distributed computational environments the workflow enactment engine is usually a service that makes use of and coordinates other services in order to execute a given workflow submitted to the engine by a client.
- *Validation Assertion (VA)*: An atomic query that must evaluate to either "True" or "False" based on Provenance Assertions within one or more Provenance Stores. Validation Assertions can be chained together using *AND* or *OR* operators.
- *Provenance Trace (PT):* A collection of PAs that have been retrieved as a result of a query.
- *SRD*: Software Requirements Document – produced by WP2.
- *URD*: User Requirements Document – produced by WP2.

1 Introduction

WP6 is aiming to produce tools for navigating, accessing and reasoning over process documentation placed in one or more PSs. Such tools are intended to be "generic", i.e. application independent, and would interact with the PS using a Management, Query and Submission interface provided by the PS. Functionality provided by the Tool Suite is therefore expected to support extended capabilities to an application user, allowing ease of query and analysis over process documentation which is currently not possible with the use of PS Application Programming Interface (API). The PS API (in the Logical Architecture document from WP3) may be too low level for applications to make use of conveniently. It is therefore intended that the functionality provided by the Tool Suite provides a more useful, higher level set of capabilities.

A key deliverable from this WP will be the "analysis" tool and the assertion checking mechanism that it supports. The remaining tools will make use of this mechanism. An "interface" in this instance is assumed to be an API, that may be made use of by an external program. As a simple definition, it is assumed that a PS contains a collection of PAs, that follow some pre-defined schema. The approach used to store a PA in the PS is not made public (hence, a number of possible database technologies could be used), however the schema of each PA may be made public via the Query or Management API. A schema contains a set of elements that represent the structure of the PA, and each of the element contains one or more data items associated with it.

1.1 Purpose of the document

The purpose of this document is to describe various components and their interactions, as part of the software deliverable from WP6. A UML class and sequence diagram is also provided to illustrate this interaction. Further sections provide details about each of the tools forming part of the WP6 Tool Suite.

1.2 Overview of the document

This document provides a description of the tools that access data in the PS. Tools in this case constitute a set of "helper classes" that can be accessed via an application program. A minimal graphical interface is also discussed which makes use of these helper classes to enable a user to view the contents of the PS. Section 2 describes relationships between the tools, the types of tools that are being supported, and assumptions being made about the architecture within which the tools are being deployed. Details about the content of the Tool Suite are provided in Section 3, with specific information about each tool. Interaction between the tools and the PS are described in Section 4, and between the tools and the application in Section 5. Some simple examples of use are provided in Section 6, along with a description of the types of users who could most benefit from such tools. Section 7 relates the work being undertaken in WP6 with other WPs in the EU Provenance project. It may be useful to note that Sections 1 and 2 describe the design of the Tool Suite, whereas Section 3 is focused on implementation aspects of it.

2 Tool Requirements

Three types of tools are envisioned as an outcome of WP6:

1. Tools for accessing and analysing the contents of the PS. Such tools are expected to provide functionality beyond that available within the Management and Query interface made available at the PS. In the simplest case, such functionality may aggregate capability already available within the Management and Query interfaces.
2. Configuration tools that allow a user to specify how they wish to view the PAs that are returned as a response to a query. Configuration information in this context refers to particular user preferences for properties associated with each tool.
3. Tools for setting up a Provenance recording session. Such tools may be used by a client for specifying: (1) the collection of components/services involved in the recording of process documentation, (2) the type of security protocol to be used (in collaboration with WP4), (3) the location of the PS, and (4) the type of data that must be recorded for each component (in collaboration with WP7 and WP8). No recording or submission support for PAs is being provided in the Tool Suite.

The capability of tools available within each of these categories are described in more detail below. Reference is also made to the Software Requirements Document (SRD) and the Logical Architecture. Access to the Tool Suite is made available only via the Application Programming Interface (API). A Graphical User Interface (GUI) and other components which are part of the Tool Suite make use of these APIs to perform the required operation.

2.1 Relationship to Software Requirements Document

As described in [D2.2.1, section 2.5] the SRD, a provenance system should provide various services to store, manage and query process documentation. Based on this description, th Tool Suite from WP6 must provide query and analysis capabilities on the stored process documentation (recorded as a set of PAs).

Some of the examples of the capabilities offered by the tools that would be used by the application services would include:

1. Capability to interact with the provenance system to analyze the stored process documentation. This corresponds to retrieving a set of PAs that can be related to a given process. Depending on the terms used to specify the query, a user may restrict the number of PAs that are returned for a given process.
2. Query capabilities that allow navigation on process documentation. It is worth mentioning that this capability should be independent of the query language used by the application and should not be limited to the capabilities provided by an individual PS on which the query would be performed. The general notion of "navigating" process documentation can range from simply viewing one or more PAs from the PS. A user may view these in a graph format – whereby relationships between PAs are explicitly captured, or may view a PA as an XML document, whereby elements contained in the document could be expanded to provided a greater degree of detail. It can also imply comparing one PA with another (or group of other PAs).

The WP6 Tool Suite provides an Application Programming Interface (API) to all the services offered as referenced in SRD [D2.2.1, section 2.5] to insulate from the details of

how the provenance system is implemented and also to provide a standardized set of operations that may be embedded within another system.

Some of the operational capabilities that are described in this document relate to the following requirements captured in SRD:

1. Allow the retrieval of a provenance trace from the PS. Either a complete trace or a subset may be retrieved [D2.2.1, section 3.4.1]. This capability is provided by the navigation tool in conjunction with the graph tool of the WP6 Tool Suite. The details of these tools are explained further in Section 3.1 and 3.7 of this document.
2. Allow the back-up of a Store to be taken. This will generally include an archiving facility that reads all the data from a PS, and records this as a disk file [D2.2.1, section 3.4.2]. This capability is provided by the navigation tool, which includes a trigger that is either initialized automatically over a particular time interval or manually triggered by the application. The trigger will read the entire contents of the PS and would write it to a remote location. Currently no support for back-up has been provided in the Tool Suite. This constitutes work that will be considered in the future.
3. Allow comparison to be undertaken with reference to a particular element (or groups of elements) within each PA [D2.2.1, section 3.4.3]. This capability is provided by the comparison tool that performs the "similarity" check and the conflict tool that helps in detection of any conflict between PAs. A detail description of both these tools is provided in Sections 3.4 and 3.5 respectively of this document.
4. Allow the results of a query to the PS to be recorded into a file. A query in this context must be specified with reference to the XML schema used to specify the contents of a PS. This schema may be different from the particular recording format used internally within the PS. Such a schema may be a combination of a general purpose schema (application independent) and an application specific schema. A query must use terms specified in the schema, and may include conjunction or disjunction of terms (where this is appropriate) [D2.2.1, section 3.4.4]. This capability is provided by the navigation tool, which includes a logging function.
5. Allow a user to access a PA based on the time and date at which the assertion was stored. This assumes that each PA will have an associated time stamp, and an XML element that provides a suitable annotation for this. The exact source of this time stamp is not of concern within the Tool Suite [D2.2.1, section 3.4.5]. The use of an available time stamp is made by the temporal tool of the WP6 Tool Suite. The details of this tool are explained further in Section 3.6 of this document.
6. Allow a user to specify a set of rules which can be verified with respect to the contents of a PS. Verification in this instance implies that the rule may be used to confirm whether the contents associated with a particular schema element meets the constraints defined in the rule. A script may therefore be used to encode a "policy". This capability is provided by the analysis tool in conjunction with the assertion tool of the WP6 Tool Suite. A detail description on both of these tools is provided in Section 3.2 and 3.3 of this document.
7. Allow a user to specify a time period in the future at which a provenance query may be submitted to a PS. A scheduler may be provided that allows queries to be stored to disk, and dispatched to the store in the future [D2.2.1, section 3.4.7]. This capability is provided by the navigation tool which includes a simple scheduler that accepts queries with time values, to allow the query to be run at a future time. This capability is based on the assumption that the results would be retrieved instantly on the execution of the query.
8. Allow a service or user to specify the identity (this may be an address of the hosting server or may be an identifier that is dynamically resolved) of the PS to which data must

be recorded. Allow a service or user to choose the level of security they wish to associate with the recording process. The level of security can range from no security, encrypted data, to more complex security mechanisms. Allow a user to specify the location of the archive to which data must be sent from a PS [D2.2.1, Sections 3.4.10-3.4.12]. Such capability to determine where a PA may be submitted by an application, and for an application to initialize its recording process, is provided by the set-up protocol (provided in the next deliverable from WP6).

9. Access to the capabilities provided by the tools should be made available as an API. This is to allow such capabilities to be embedded within an existing application. This capability is fulfilled by all the tools that form part of the WP6 Tool Suite.
10. Allow a user to visualize the contents of a PS. The visualization support may allow an easier way to access the XML-based content accessible from the PS. The graphical user interface is part of the navigation tool that allows a software application or a human user to visually navigate and analyze the process documentation. A detailed description of the navigation tool is covered in Section 3.1 of this document.

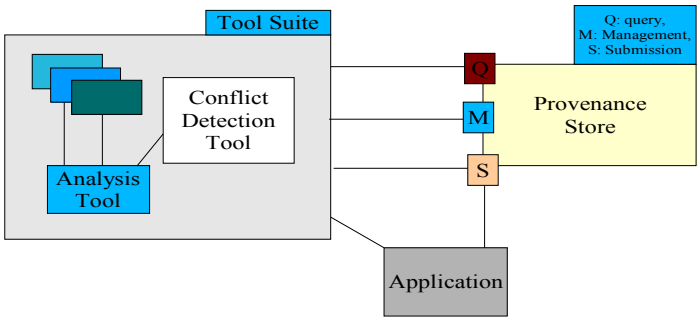

Figure 1: High level interaction between Tool Suite, PS and Application.

Figure 1 illustrates the interaction between the tools, the PS and an application. This identifies that an application user can make a direct submission to the PS. A tools relationship diagram is illustrated in Figure 6 (in Section 5).

Schema Description

We assume in this document that each PA follows a P-Structure [WP3] that has been defined according to an XML schema. It is also assumed that some of the elements that comprise a P-Structure relate to data that is specific for an actor within a particular application – we refer to this as the application schema.

2.2 Relationship to Logical Architecture

The following requirement have been identified for the logical architecture:

1. A workflow description is provided by an application user. The representation should be in some XML format – which describes the set of activities and dependencies between the activities. It is assumed that each of these activities has corresponding PAs in the PS. The workflow description can then be verified by retrieving records from the PS. Work is underway with the applications (WP7) to specify this XML workflow representation.

2. A workflow can be derived by analyzing PAs stored in the PS. This assumes that there is enough information within each PA to enable a workflow graph to be reconstructed.
3. It is required that PS should be able to at least process XPath queries. Other query capability may also be made available at the PS beyond the use of XPath.

   It is assumed that workflows are pre-defined, and specified by the application user/developer.

Some of the operational capabilities that are described in this document are satisfied by the following functionality provided by the logical architecture document.

1. The architecture provides a user interface which allows visualization of query results and processing services' outputs. [WP3, Logical Architecture document v0.2, section 3.3]. The navigation tool provided as part of the Tool Suite offers capability related to the logical architecture to perform browsing over PSs, visualize differences in different execution, and illustrate execution from a semantic viewpoint.
2. The architecture provides processing services for analyzing and reasoning over recorded PAs [WP3, Logical Architecture document v0.2, section 3.3]. The analysis, comparison, and conflict detection tool provided as part of the Tool Suite offers this capability. The tools may be used to compare the processes that generate several data items, and verify that a given execution was semantically valid.

   Based on overall logical architecture diagram in Figure 3.1 [WP3] the graphical tools provide support that can be added to an application GUI, the comparison tools provide aspects of matchmaking to allow comparison between PAs. Similarly, functionality within a trace comparator is supported by both the comparison tool and the conflict detection tool. In general, the tools outlined in this document try to provide instantiations of some of the capability that has been outlined as necessary within the logical architecture.

Based on the application requirements identified in the SRD, it is possible to identify the following requirements that tools would place on the architecture. In Table 1 below, each functional requirement from the SRD (column 1) along with the basis for this requirement obtained from the URD (column 3) is listed, followed by an interpretation of these requirements with reference to the tools being developed in WP6.

SRD ID, flags	***Text of Software Requirement*** ***Text of Architecture Requirement***	***Source (URD ID)***
Functional requirements		
TSR-1-1 *essential, critical (OTM, TENT)*	The provenance architecture should provide for the recording and querying of interaction and actor provenance. Source (URD ID): AR-1-1, AR-1-2, AR-1-3, AR-1-5, AR-1-6, AR-1-7, AR-2-1, AR-2-2, AR-2-3, AR-2-4, AR-3-1, AR-3-2, AR-3-3, AR-5-1, AR-5-4, AR-5-5, AR-5-6, AR-5-7, AR-5-8, AR-5-9, AR-5-10, AR-5-12, AR-5-13, AR-6-2, AR-7-1, TR-1-1-A-1, TR-1-1-A-2, TR-1-1-A-3, TR-1-1-A-4, TR-1-1-B-1, TR-1-1-B-2, TR-1-1-B-3, TR-1-1-B-4, TR-1-1-C-1, TR-1-1-C-2, TR-1-1-C-3, TR-1-1-C-3, TR-1-1-C-5, TR-1-1-C-6, TR-1-1-D-1, TR-1-1-D-2, TR-1-1-D-3, TR-1-1-E-1, TR-1-1-E-2, TR-1-1-E-3, TR-1-1-F-1, TR-1-1-F-2, TR-1-1-F-3, TR-1-1-F-4, TR-1-1-F-5, TR-1-1-G-1, TR-1-1-G-2, TR-1-1-G-3, TR-1-1-H-1, TR-1-1-H-2, TR-1-1-i-1, CR-3-1, CR-3-2, CR-5-1, CR-5-4 Tools Requirement: Provenance architecture should support a Query Interface and a Submission/Recording Interface to the PS. Tools may make use of this interface to retrieve one or more PAs from the PS.	
TSR-1-2 *essential*	The provenance architecture should allow the retrieval of a provenance trace from the PS. Either a complete trace or a subset may be retrieved. Source (URD ID): AR-1-1, AR-1-2, AR-1-3, AR-1-5, AR-1-6, AR-1-7, AR-2-3, AR-2-4, AR-3-2, AR-3-3, AR-5-1, AR-5-2, AR-5-3, AR-5-4, AR-5-5, AR-5-6, AR-5-8, AR-5-10, AR-5-12, AR-5-13, AR-6-1, TR-1-1-A-1, TR-1-1-A-2, TR-1-1-A-3, TR-1-1-G-2, TR-1-1-G-3, TR-1-1-H-1, TR-1-1-H-2, TR-4-1, CR-3-1 Tools Requirement: Provenance architecture should provide support for storing results generated from a query locally at the PS. These results can then be read from a temporary storage area by the tools.	
TSR-1-3 *essential*	The provenance architecture should allow the back-up of a PS to be taken. This will generally include an archiving facility that allows data within a PS to be saved for future use. Tools Requirement: Provenance Architecture should not impose constraints on access to the store via the Query or Management interfaces. Tools may be able to read all or part of the PS directly.	TR-3-6

SRD ID, flags	***Text of Software Requirement*** ***Text of Architecture Requirement***	***Source (URD ID)***
TSR-1-4 *essential*	The provenance architecture should allow comparisons to be made across Provenance Records within a PS with reference to particular data attributes within a Provenance Record. Tools Requirement: The Query interface at the PS should allow search to be carried out on a data attribute or value. For instance, the tools may generate an XPath query that needs to be processed by the PS, and this query would be defined with reference to one or more namespaces.	AR-1-1, AR-1-6, AR-5-6, TR-1-1-C-1, TR-1-1-C-2, TR-1-1-C-3, TR-1-1-C-4, TR-1-1-C-5
TSR-1-5 *essential*	The provenance architecture should allow the results of a query to the PS to be captured for future use. A query in this context must be specified with reference to the structure of the PS. Tools Requirement: Requirement covered by TSR-1-2.	TR-2-1-C, TR-2-1-D, TR-3-1, TR-3-3, TR-3-7
TSR-1-6 *desirable*	The provenance architecture should allow a user to access a Provenance Record based on the time and date (calendrical information) at which the Record was stored. Tools Requirement: If a time stamp already exists in a PA, the tools can make use of this. Generally, no timestamp is produced by the tools.	AR-3-3, AR-5-1, TR-1-1-D-2, T-R-1-1-F3, T-R-1-1-G1, CR-3-1
TSR-1-7 *desirable*	The provenance architecture should allow a user to verify the contents of a PS against a specified set of rules. Verification in this context means that the contents of the PS meets the set of constraints expressed by the set of rules. Tools Requirement: Requirement covered by TSR-1-3.	AR-1-1, AR-1-7, AR-3-1, AR-4-1, AR-5-3, AR-5-4, AR-5-5, AR-6-1, AR-6-2, TR-1-1-C-6, TR-5-2
TSR-1-8 *essential*	The provenance architecture should allow a user to specify a time period in the future at which a provenance query may be submitted to a PS. A scheduler will be made available that allows queries to be stored to disk, and dispatched to the store in the future. Tools Requirement: The Query interface at the PS should provide some indication to the tools about the list of queries that are waiting to be answered by the PS. Essentially, if a message queue exists at the PS, the contents and size of this query should be accessible via the Management interface at the PS.	TR-4-2

SRD ID, flags	***Text of Software Requirement*** ***Text of Architecture Requirement***	***Source (URD ID)***
TSR-1-9 *essential*	SR-1-9: The provenance architecture should allow capabilities provided by the tools to be accessible as an API. This is to allow such capabilities to be embedded within an existing application. Tools Requirement: The Provenance architecture should support a registry service to enable the software interfaces for all tools to be visible. This is particularly true for any visualisation tools that may be application specific. Information about new tools, or updated versions of existing tools should be accessible through such a registry. This would be associated with "Processing Services" in the logical architecture [WP3].	TR-4-3, TR-5-3, TR-6-1, TR-6-4-A, TR-6-4-B
TSR-1-10 *essential*	As part of the initialisation of the provenance recording process, the provenance architecture should allow a service or user to specify the identity of the PS to which data should be recorded. Tools Requirement: The Management interface at the PS should enable the identity of the PS to be queried.	Omer Rana
TSR-1-11 *desirable*	The system should support the multiple storage of a provenance record, i.e. the system should provide a way to store copies of a provenance record in more than one repository. Tools Requirement: Not applicable.	TR-3-1
TSR-1-12 *essential*	The system should support the recording of different provenance information views related to an event or an entity. Tools Requirement: Not applicable.	TR-3-2
TSR-1-13 *essential*	The provenance architecture should support the migration of provenance data among PSs. Tools Requirement: Covered by TSR-1-3.	TR-3-3
TSR-1-14 *essential*	The system should support the storage of recorded provenance data for an indefinite period of time. Tools Requirement: Covered by TST-1-3.	TR-3-5-A, TR-3-5-B, TR-3-5-C
TSR-1-15 *essential*	The provenance architecture should support the storage of results of analysis and reasoning operations performed on the provenance data by tools that are not part of the generic architecture (3rd party tools on the application layer). Tools Requirement: Not applicable.	TR-4-3, TR-1-2

SRD ID, flags	***Text of Software Requirement*** ***Text of Architecture Requirement***	***Source (URD ID)***
TSR-1-16 *desirable*	The provenance architecture should provide support for maximum automation of the provenance recording mechanism. Tools Requirement: Not applicable.	TR-5-1, CR-5-9
TSR-1-17 *essential*	The provenance architecture should be deployable as an integrated part of a system, as a service within the same administrative domain as the client system and as a 3rd (external) party operated service, too. Tools Requirement: Some aspects of this are already covered by TSR-1-8 and TSR-1-9. In addition, the provenance architecture should not assume the existence of a particular set of services in the Tools Suite.	TR-5-3
TSR-1-18 *desirable*	Client side components of the provenance architecture should not block an executing workflow if any provenance services are unavailable. Tools Requirement: The provenance architecture should not necessitate the submission of a PA from a particular type of client. No requirements on synchronous or asynchronous behaviour should be assumed.	TR-5-4
Performance requirements		
TSR-2-1 *essential*	The additional execution overhead for an application recording provenance information should be kept to a minimum. Tools Requirement: Not applicable.	CR-1-1-A, CR-1-1-B, CR-1-1-C, CR-1-1-D
TSR-2-2 *essential*	Storage space requirements of the provenance architecture for provenance information recording should be kept at a reasonably low level. Tools Requirement: Not applicable.	CR-1-2-A, CR-1-2-B, CR-1-2-C
TSR-2-3 *desirable*	The provenance architecture should guarantee reliable once-and-once-only delivery of provenance information to and from a PS. Tools Requirement: Not applicable.	CR-2-1, CR-2-2
TSR-2-4 *essential, critical*	The provenance architecture should be capable of handling large amounts of provenance data submitted frequently by user applications. The provenance architecture should not be the cause of any bottlenecks in the overall system due to the processing of provenance data. Tools Requirement: Not applicable.	CR-5-3

SRD ID, flags	*Text of Software Requirement* *Text of Architecture Requirement*	*Source (URD ID)*
Interface requirements		
TSR-3-1-1 *essential, critical*	All of the functions of the provenance architecture should be accessible through its API so it can be used as an embedded component in a system. Tools Requirement: The Provenance architecture should make no assumptions about the contents of this API, or the particular protocol that is used to access services made available through this API.	CR-5-5, CR-5-1, TR-6-4-B
TSR-3-1-2 *essential, critical*	The provenance architecture should support a rich set of published, generic APIs that allow application specific analysis and reasoning tools to be built upon. Tools Requirement: As TR-3-1-1.	TR-6-1, CR-5-1, CR-5-7
TSR-3-1-3 *essential, critical (eDiamond)*	The provenance architecture should provide a programmatic interface for the administration of the system. Tools Requirement: Not applicable.	TR-6-4-A, TR-6-4-B
TSR-3-1-4 *desirable*	The provenance architecture should support an XML-based API format for provenance data. Tools Requirement: Not applicable.	TR-2-1-B
TSR-3-2-1 *essential, critical*	Export formats for provenance data should be non-proprietary to allow tools and applications to be built without violating IPR rules. A format based on an existing data representation standard (with special focus on XML defined by XML schema) would be highly preferred. Tools Requirement: Not applicable.	TR-2-1-A, TR-2-1-C, TR-2-1-D
Operational requirements		
TSR-4-1 *essential*	Provenance information displayed by the provenance architecture on a HCI should be updatable on user request. Tools Requirement: The provenance architecture should make no assumption about the types or modes of user interfaces being supported.	TR-6-6-A

SRD ID, flags	***Text of Software Requirement*** ***Text of Architecture Requirement***	***Source (URD ID)***
TSR-4-2 *essential*	HCIs presented by the provenance architecture for displaying the contents of a PS should support continuous monitoring, i.e. the displayed information should be updated automatically on every change as soon as possible. Tools Requirement: Such human accessible interfaces are to be provided as visualisation tools in the Tool Suite. Additional visualisation "portlets" may be added over time by an application user.	TR-6-6-B
TSR-4-3 *essential*	The update frequency of provenance information displayed by the system on a HCI should be configurable based on policies. Tools Requirement: The configuration facility made available in the Tool Suite should support configuration of the update frequency.	TR-6-6-D
TSR-4-4 *essential*	Human-computer interfaces presented by the provenance tools should be designed to allow multilingual support. Tools Requirement: Tools should allow application users to adapt interfaces or add additional portlets.	TR-6-2
Documentation requirements		
TSR-5-1 *essential*	Detailed documentation of the provenance architecture public interfaces should be produced both for application programming interfaces (APIs) and Human Computer Interfaces (HCIs). Tools Requirement: Not applicable.	TR-7-1
TSR-5-2 *essential*	A detailed description of the administrative interface of the provenance architecture should be produced. Tools Requirement: Not applicable.	TR-7-1
Security requirements		
TSR-6-1 *essential, critical (myGrid)*	The provenance architecture should have a configurable access control system over the resources it provides, with a granularity that is sufficient to protect these resources. Tools Requirement: Tools should allow interaction with the security components on the architecture, and enable initialisation of such security properties as part of the "setup protocol".	CR-4-1

SRD ID, flags	*Text of Software Requirement* *Text of Architecture Requirement*	*Source (URD ID)*
TSR-6-2 *essential*	The provenance architecture should allow both automated and manual determination of access control rights. Tools Requirement: The provenance architecture should allow such access control to be supported (and configured) through the Management interface of the PS, and a specialist credentials management service (or similar).	CR-4-2
TSR-6-3 *essential*	The provenance architecture should allow a service or user to request the level of security they wish to be associated with the recording process. The level of security can range from no security through encrypted data transfer to more complex security mechanisms. Tools Requirement: As TSR-6-2.	Omer Rana
TSR-6-4 *essential*	The provenance architecture should provide a way to map access rights information of embedding systems into its security subsystem. *Note:* For examples on user groups and their required access rights in two application scenarios refer to user requirements CR-4-4-A and CR-4-4-B. Tools Requirement: As TSR-6-2.	CR-4-3, TR-1-1-B-5, CR-4-4-A, CR-4-4-B
TSR-6-5 *desirable*	Security related procedures for accessing the provenance system should be subsumed under the existing security related procedures for the embedding system if possible, so that changes or additions to the existing procedures are minimized. Tools Requirement: Not applicable.	CR-4-6, CR-4-5
TSR-6-6 *desirable*	The provenance architecture should provide a mechanism for recording provenance data in an unmodifiable form and also ensuring that the party responsible for the recording process cannot deny having recorded that provenance data. Tools Requirement: The provenance architecture should make no assumption about the recording format used.	CR-4-7, CR-3-2, AR-2-3, AR-7-2, AR-4-1, AR-4-2, AR-5-2, AR-5-10
TSR-6-7 *desirable*	The provenance architecture should provide a mechanism for the authentic timestamping of provenance records. Authenticity should be guaranteed by the mechanism on a level that is enough even for the use in legal procedures. Tools Requirement: Covered by TSR-1-6.	CR-3-2, AR-4-1
Other requirements		

SRD ID, flags	***Text of Software Requirement*** ***Text of Architecture Requirement***	***Source (URD ID)***
TSR-7-1 *essential, critical*	The provenance architecture should have the properties of cost efficiency and robustness versus an in-application hand-engineered logging system. Tools Requirement: Not applicable.	CR-5-2
TSR-7-2 *desirable, critical*	The provenance architecture should be loosely coupled and independent from the applications as much as possible. Integration costs for existing systems should be minimal, ideally existing system components should remain unaffected. Tools Requirement: Not applicable.	CR-5-6, CR-5-8

Table 1

2.3 Types of Tools and Scope

2.3.1. Tools for Accessing and Analysing PS Contents

This first category of tools only read data stored in the PS, primarily through the use of the query interface available at the PS. A PT in this instance is defined as a collection of PAs that have been retrieved as a result of a query. Two extreme conditions include: (1) only a single PA matches the query, (2) all PAs match the query. The following capabilities are provided:

Navigating a Provenance Trace

Navigation may involve simply retrieving all the PAs based on a query specified by the user in the navigation tool, or it may involve displaying PAs using a graphical format that demonstrates some relationships between the PAs visually. A user may specify some constraints on the number of PAs to be retrieved, or the type of PAs of interest.

A query in this instance is expected to be specified according to the schema made public through the Query API of the PS. Queries may be specified using XPath expressions, for instance. Retrieval of PAs from the PS may involve issuing a set of repeated queries via the PS Query API.

Analysing a Provenance Trace

Analysis in this instance may involve specifying:

- a relationship between PAs, and using this as a constraint for determining what should be retrieved,
- a constraint on the type of data that must be contained within a PA,
- a constraint on schema elements that may form part of a PA. This is to be undertake in collaboration with WP7 and WP8,
- a relationship between schema elements that form part of a PA (defined with reference to the P-Structure of the PA),
- a relationship between PAs based on their schema elements,
- a relationship between PAs based on their data.

Analysis in this instance would be undertaken based on the definition of a set of rules to specify "relationships" or "constraints" mentioned above. Such an analysis will also form the basis for comparing two traces, for detecting conflicts between traces, and for checking whether a trace is up-to-date.

Analysis in this instance would correspond to verifying a set of "assertions" on the PS, using the navigation tool also developed in the WP. An assertion can evaluate to true or false, depending on the contents of the PS. Each of the six analysis modes defined above should be specified through this assertion mechanism. Furthermore, assertions may be applied over elements of a PA schema, or over the data contained within these elements.

A key aspect will be to identify how assertions may be specified using a language that is easy to use and adapt for the two application scenarios outlined in WP7 and WP8. As an example, consider the analysis of two traces which encode a particular "process". In this instance, a user may be interested to know whether a particular activity, or set of activities, have been undertaken in a particular order. The analysis in this instance will involve identifying a set of assertions, which if found to be valid, indicate that either a particular activity has occurred or not. Some examples are provided in Section 6.1.

The tool will therefore only provide the capability to specify a set of assertions that can be verified, in some order, by issuing queries to the PS. Describing an application specific requirement as a set of assertions will not be undertaken by the tool. It will therefore be necessary to work alongside WP7 and WP8 to identify some mechanism whereby a user can translate their higher level, application specific requirement, into a set of assertions that can then be confirmed using this tool.

Assertion definitions are currently being investigated. Two candidate technologies are being investigated: (1) The Java Expert System Shell, (2) DROOLS, or (3) OpenRules. The choice of these technologies are based on their suitability to the particular application scenarios outlined in WP2 (as part of Deliverable D2.2.1). Alternative query mechanisms such as those based on databases (using the SQL language) do not easily allow representation of analysis rules. Furthermore, such approaches do not enable such rules to be executed over an XML document easily. A number of alternative approaches, such as those based on Web Services Relationships Language [WSRL] do not have suitable implementations, and are often very restrictive in the type of reasoning support that they provide.

2.3.2. Comparing Provenance Traces

The comparison tool makes use of the analysis capability described above. Comparison in this instance involves verifying if two PTs are (1) identical, or (2) "similar". Considering traces T1 and T2, the comparison tool will perform the following operations:

- Loosely identical: T1 and T2 are identical if they contain the same document structure (defined with reference to a PA schema).
- Exactly identical: T1 and T2 are exactly identical if they contain the same document structure (defined with reference to a PA schema), and the same data associated with these elements.
- Similar: T1 and T2 are similar if some "semantic" similarity can be found between the elements contained in T1 and T2. Semantic similarity in this instance must be defined with reference to a domain specific ontology, for instance (interaction with WP7 and WP8 will be necessary to describe this).

  A comparison performed between PAs assumes that they are defined according to the same schema. It is possible that a single PS may contain PAs that are specified with reference to different schemas, or make use of different formats for recording data. In

this instance, it is necessary for the user to encode the contents of a PA in a uniform way to allow comparison to be supported.

2.3.3.Detecting Conflicts in Provenance Traces

Conflict detection in this instance is used to determine if two PAs submitted either by a client and a service provider, for the same interaction, contain differences. A conflict in this instance is therefore used to identify differences observed in the recording of the same interaction by different actors (such as the client requesting a service, and the service provider). For instance, if the interaction consists of exchange of a float value, the sender might record a float value in its PA, and send it to the PS. However if the recipient is not designed to receive float value but only integer value, in this case, it will store within its PA an integer value. As these two PAs are recorded for the same interaction, the conflict detection mechanism is able to detect this discrepancy. The conflict detection mechanism therefore needs to evaluate the contents of a minimum of two PA – each coming from a different actor.

This example was rather simple and the conflict detection was easy mainly because of the type of the data we have chosen. However, data type stored can be application specific types, so the conflict detection tool also uses the data comparison architecture provided by the analysis tool being developed in this WP.

2.3.4. Checking that a trace is up-to-date

This check will be undertaken by examining the time stamps associated with a PA – how such a time stamp is generated is dependent on the particular application submitting the PA. This tool will therefore identify when a PA was last recorded for a given actor, and use this as a basis to determine if a submission has been made by an actor recently. The tool may also enable a user to determine how many submissions have been made by an actor over a given time period, or the time interval between a given number of submissions. This tool will also make use of the analysis tool being developed in this WP. Such an analysis assumes the existence of some time information with a PA – further discussion about this issue is provided in Section 3.6.

Figure 1 illustrates the interaction between the Tool Suite (WP6) with other components within a provenance system. A key component within the Tool Suite is the analysis tool, which is made use of by all the other tools. Interaction with the PS is via the Query, Management and Submission APIs. The Tool Suite may contain both an API to allow other application to interact with the PS, or it may contain a graphical interface (in the case of the navigation tool, for instance). An application may submit data to the PS, but must make use of the the capability provided in the Tool Suite to interact with the PS.

3 Tool Suite Description and Implementation

The two main tools that form the core of WP6 are the navigation and analysis tool. Others include the graph tool, comparison tool, and conflict detection tool. Each of these tools plus their relationship with each other is described below. Figure 4 lists the methods available within each tool.

3.1 Navigation

The main objective of the navigation tool is to provide interfaces to retrieve and display PAs. The getAll() public method accepts XPath query as one of the inputs and provides a link between the application and the navigation interfaces. The getAll() method is supplemented with either an option to retrieve a graphical representation of the resultant PAs or to restrict the total number of PAs retrieved from the query. The tool offers capability to perform repeated queries on the PS to retrieve specific PAs that meet the constraints defined by an application user. The navigation tool is linked to a graph tool that provides visualization capability, and the analysis tool that allows application users to test relationships between the retrieved PAs.

The analysis tool interacts with the navigation tool to also retrieve PAs. This interaction however represents two different aims: (i) the analysis tool formulates rules that can be directly converted to XPath queries – thus being able to evaluate the rules based on the query results returned by the navigation tool; (ii) the analysis tool interacts with the navigation tool repeatedly to retrieve results for multiple XPath queries, which are then processed by the analysis tool using the help of an assertion engine.

3.2 Relationship Definitions

Data recorded within a PA can be either a simple data type like string or integer, or can also represent application specific data types like a sound or an image. In the former case, data type are well-know and supported in virtually all programming languages, thereby allowing us to compare them easily. However, in the latter case, we cannot make any supposition how to compare these data. Hence, the aim of this tool is to:

- provide a mechanism to compare arbitrary data types by providing an interface that may be used by other tools;
- allow the comparison of application specific data by providing a plug-in enabled architecture. An application should provide one or more plug-ins according the type of data it handles. Each plug-in provides a data type specific mechanism to compare between two entities that are of that type.

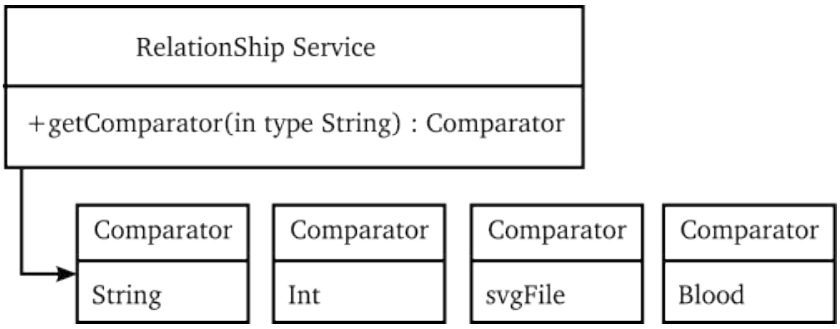

Figure 2 Comparison Plug-in Architecture

When a new data type needs to be handled, the corresponding plug-in must be registered using the relationship tool. The relationship tool stores all plug-ins in a comparison store. At registration time, a plug-in provides the comparison tool with the type of data it can handle. The name associated with a given data type must be defined within the application schema.

Each plug-in contains only one method with the following signature:

int compare(in String data1, in String data2)

The two parameters (data1 and data2) represent the data recorded within the PA we want to compare. According to the application and the plug-in logic, this data could represent a URI (a file on a local hard disk, on a Web page, on a FTP server), or the actual data object that needs to be compared. The String in the Comparator description above corresponds to the type of data being analyzed.

The returned value type is an integer (because this data type is widely supported). The semantics of the returned value are (in this instance, <, > and = are assumed to have some meaning based on the plug-in being used):

- -1 if data1 < data2
- 0 if data1 == data2
- 1 if data1 > data2

3.3 Analysis

The main objective of the analysis tool is to provide an assertion checking interface for use by all the other tools. The analysis tool returns a boolean outcome, indicating whether the particular condition being analysed holds or not. The analysis tool provides two main methods recordMatch() and recordSchemaMatch() to the comparison and the conflict tools. The first method provides an interface to check if two PAs are similar, and the other checks if a PA conforms to a given schema. The first method will be used via the navigation interface, while the second method would primarily be used through the application interface. All of the methods in this tool assume that the reference to the PA is sent as part of the string argument – indicating therefore that each of the methods here apply to a PA that has already been defined. Other methods in the analysis tool provide the following functionality:

a) check(in element: String) - checks if the element exists within a PA schema. For example, checks if an element “temperature” exists within a PA schema.

b) check(in data: String) - checks if a particular data value exists within a PA. For example, checks if a data value of "34" exists within a PA.

c) check(in element: String, in data: String) - checks if the element and data exist within a PA. For example, checks if an element "temperature" consisting of a data value "34" exists within a PA.

d) checkRelation(in data: String, in relation: String) - checks if data exists within a PA that meets the required relationship constraint. For example, checks if a data value of greater than "34" exists within a PA. In this example the data string would consist of "34" and the relation string would consist of "greater than".

e) checkRelation(in element: String, in data: String, in relation: String) - checks if an element and data exist within a PA that meets the required relationship constraint. For example, checks if an element "temperature" consisting of a data value greater than 34 exists within a PA.

f) checkSubsumption(in element1: String, in element2: String) - checks if two elements within a PA schema are linked by a "subsumption" relationship. A subsumption relation in this instances relates child and parent elements in a schema. A child element in this instance subsumes the properties of the parent element. This type of analysis is used for similarity assessment between a query and one or more PAs.

3.3.1 PAAM : Provenance Assertion Abstraction Mechanism

This mechanism provides a generic set of rules that will be manipulated by an assertion engine through specific wrappers. An assertion engine here represents an interpreter that can process rules applied on a PA. These rules allow us to verify a set of constraints on a PA. A rule must return true or false when evaluated against a PA.

Several kinds of rules can be used :

- Basic rules validate a condition on a particular data located in the application specific part of a PA. These condition are: equality, greatherThan, lesserThan, exists, notNull.
- Meta rules check a condition on the provenance specific part of a PA, such as date and invocation time, date of received results, actor name, client ID, service ID, etc.
- Operator rules combine basic rules and meta rules using operators like AND, OR, NOT.

An example of an assertion combining user- and provenance-defined rules is shown in figure 3. This assertion requires that for application data, data val1 equals 3, data val2 equals 1.xx.E34 (val2 is an application specific data) – and for provenance related data, the actor of this record could be either "Arnaud" or "Omer".

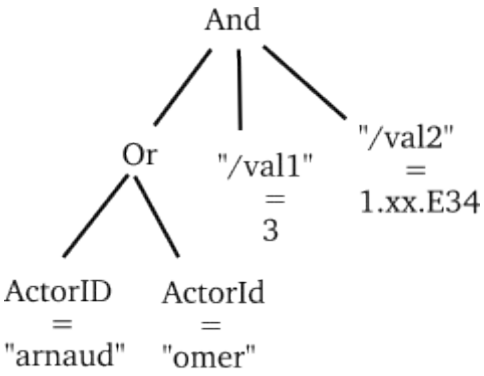

Figure 3 Assertion combining User and Provenance defined data

As we want to create a mechanism that has a general applicability, a given rule should be able to handle various kinds of data. For example, the rule which checks the equality of two data values is not designed to know how to compare these data. For this action, PAAM relies on the relationship tools we introduced previously.

Hence, when a rule is instantiated with an XPath string, the first thing it does is to look for the type of the value intended to be associated with the string inside the schema. Next, it asks the relationship service for a comparator tool matching the type. Finally it checks the value according its own behavior and returns a boolean.

3.3.2 Assertion Engine

The assertion engine provides input to the analysis tool. Hence, the main task of the assertion engine is to provide an interface to accept assertions obtained from the navigation tool and provide an appropriate boolean output. Many candidate technologies are currently being investigated for assertion definitions: (1) The Java Expert System Shell [JESS], (2) DROOLS [DROOLS], or (3) OpenRules [ORULES]. A comparison between these different assertion engines has been conducted based on their ease of use in defining and using rules, and support for the Java Specification Request 94 [JSR94].

	JESS	DROOLS	Openrules
Language	Java	Java	Java
JSR 94 compliant	Yes	Yes	Yes
Rule :			
- format	Clips	XML + code	Excel
- readability	+++	---	---
Engine :			
- Flexibility	+++	+	+
- Adaptability	+++	+	+
- Extensibility	+++	++	+
Documentation	+	++	++
Community	+++	+++	++
License	Commercial	GPL	GPL
Result	1st	2nd	3rd

JSR94 defines a simple API to access a rule engine from a Java client. It provides APIs to (i) register and unregister rules; (ii) parse rules; (iii) inspect rule metadata (this specifies the format used to define the rules); (iv) execute rules; (v) retrieve results, and (vi) filter results.

JSR94 does not standardize the following: the rule engine itself, the execution flow for rules, the language used to describe the rules, and the particular deployment mechanism being used for the rules. JSR94 therefore does not standardize the semantics of rule execution. This is particularly useful as rules defined in JESS (for instance) which are executed using the Rete algorithm may be ported to another engine. Based on this comparison, the JESS engine was selected. Currently, rules are designed as standard Java classes.

3.4 Comparison

The comparison tool makes use of the analysis capability described in Section 3.3. Comparison in this instance involves verifying if two PAs are (1) identical, or (2) “similar”. The comparison tool consists of two main types of analysis: one that accepts a particular type of match and the PAs to provide a boolean result, and the second that accepts PAs and compares them to return a string that specifies what type of match exists between the PAs. A method is also provided within the comparison tool that allows an application user to specify a plug-in algorithm using the navigation interface. The plug-in algorithm will represent an application users' understanding of similarity, and based on this the analysis will be performed for PAs.

To illustrate the use of a comparison tool, consider the following example:

- A user issues a query, which returns a number of PAs from one or more PSs.
- A user may store a subset of these PAs in his/her file space, or may select one PA that may be of interest and only record this.
- The selected PA is then used to undertake a comparison with other PAs that have been retrieved as a result of the query, or PAs that have been recorded previously. It is also possible for a user to generate a PA and use this to compare with PAs returned as a response to a query.

The use of such a comparison technique may be beneficial to reduce the number of PAs that a user may wish to analyze further.

3.5 Conflict

Conflict tool interfaces with the analysis tool to provide a conflict detection mechanism. Conflict detection in this instance is used to determine if two PAs submitted by a client and a service provider, for the same interaction, contain differences. A conflict in this instance is therefore used to identify differences observed in the recording of the same interaction by different actors. For instance, an actor A may send a message to actor B, and both record this interaction. The conflict detection tool could be used to identify if the contents of this message are the same (based on what A has sent, and what B has received). To support such detection, it would be necessary to ensure that PAs corresponding to the same interaction in one or more PSs can somehow be related. Conflict tool provides similar functionality as mentioned in the comparison tool in terms of analysing either an (1) identical, or (2) “similar” conflict has occurred and providing plug-in mechanism for user defined conflict detection. This assumes that the recording format used by the two actors is the same. If the recording format is different, an additional step may be necessary to convert the contents of the message for comparison. A user would be responsible for performing the conversion prior to the use of the conflict tool.

3.6 Temporal

This is a particular tool that may be used by the analysis and conflict tools, and deals with providing methods that can analyse information on time relationships between PAs. This check will be undertaken by examining the time stamps associated with a PA. This tool will therefore identify when a PA was last recorded for a given actor, and use this as a basis to determine if a submission has been made by an actor recently. The tool will also enable a user to determine how many submissions have been made by an actor over a given time period, or the time interval between a given number of submissions. No assumptions are being made here about who generates the time stamp, or whether a time stamp provides a true reflection about when a PA was generated. The temporal tool primarily makes use of a time stamp if one is already provided.

It is useful to note that time stamps may be provided by the PS – although in some instances it may also be possible for actors submitting a PA to generate a time stamp. This issue is still being discussed in the project, and no consensus has been reached. The reason for this is that it cannot be assumed that actors making PA to a given PS have synchronised clocks. Also, due to the asynchronous nature of the recording protocol, a time stamp may not indicate when a PA was actually produced and submitted by an actor. A current point of discussion is whether it would be useful to assume, in the context of a restrictive application domain, that clocks are synchronized and some deduction can be made based on time stamps generated by an actor. It is more likely, therefore that a PA is only acknowledged after it has been recorded within the PS, and has been given a time stamp. However, this also leads to concerns similar to those with actor generated time stamps, if submissions may be made by a given actor into multiple PSs (each of which attaches its own time stamp).

3.7 Graphical Interface and Portlets

The graph tool is used for displaying PAs using a graphical format that demonstrates relationships between the PAs visually. The GUI tool provides an interface that is used by the navigation tool to retrieve the graphical object. In the first instance, the graph tool will simply provide a mechanism to view the structure of a PA – by plotting the tree associated with a PA schema, and mapping data values associated with each PA to this schema. Subsequent versions will also contain additional methods, that would allow a navigation tool to send in a “graph type” request (the type corresponding to different visual), which would then allow application users to visualize different types of graphs.

A portal is a Web application which typically provides personalization, single sign-on, content aggregation from different sources, and hosts the presentation layer of different backend systems. The main task of a portal is to allow different applications to be aggregated into a single Web page that may be accessed by an application user. A portal may also provide personalization features which provide customized content to users. The degree to which such customization is supported can vary – from the ability to simply change the colour or sizes of fonts, to the ability to configure interaction that takes place between two portlets. Portal pages may have different sets of portlets to create content for different users. Portlets perform different tasks and create content according to their current function. A portlet “mode” indicates the function a portlet is performing, at a particular point in time. When invoking a portlet, the portlet container provides the mode for the current request to the portlet. Portlets can programmatically change their portlet mode while processing an action request. The notion of a container is very significant in the context of portal development, as a container provides the necessary interaction infrastructure allowing portlets to communicate with each other.

There are a number of different ways in which workflow may be displayed to a user. Each makes use of a process graph that demonstrates how a collection of processes are executed in a sequence. An example of such an approach for displaying workflow can be found at: http://www.ilog.com/products/jviews/workflow/

A PT can be displayed graphically. Two main representations are available:

- Workflow hierarchy (default) displays all the PAs contained within a PT. If one PA has been generated after another (based on some relationship assessment), an arrow is displayed from the ancestor to the child. This description is also dependent on the workflow graph that has been supplied by a user (Section 6).
- An alternative representation technique would make use of a time line. This representation may be used to compare two similar PTs. The duration of each step can also be compared using this approach. A user may interact with this time line representation by clicking on a trace and deriving information about the PAs associated with a particular trace. This assumes that a time stamp has been provided with a PA.

To support the navigation tool, we have undertaken a comparison of different Portal Frameworks based on a set of criteria that represent requirements identified in the SRD, and also taking account of the ease of deployment when making use of such a framework. To enable as wide a user base as possible for the navigation tool, it is necessary to ensure that the graphical interface capabilities conform to existing standards. This is also important to allow users to add their own portlets, or to connect portlets with user interfaces that have already been implemented as part of an existing application. Although the later is unlikely to be followed in this project, the adoption of standards such as the Java Specification Request (JSR) 168 [JSR168] and the Web Services for Remote Portlets (WSRP) [WSRP] provides a useful basis to enable such interoperability in the future.

The JSR 168 defines a portlet specification, including a contract between the portlet container and the portlet. JSR 168 is defined by the Java Community Process (JCP). The JSR 168 was co-led by IBM and Sun and had a large Expert Group that helped to create the final version which is now available. This Expert Group consisted of Apache Software Foundation, Art Technology Group Inc.(ATG), BEA, Boeing, Borland, Citrix Systems, Fujitsu, Hitachi, IBM, Novell, Oracle, SAP, SAS Institute, Sun, Sybase, Tibco, Vignette. JSR 168 supports three modes for a portlets: view mode – allows displaying of portlet output; help mode – allows viewing of help files associated with a portlet; and edit mode – this allows personalization of a portlet to be undertaken by a user. The editing capability is a particularly strong point of a portal framework. JSR 168 defines different mechanisms for the portlet to access transient and persistent data [JSR168-IBM]:

The portlet can set and get transient data in the following scopes:

Request	The request has attached data, such as request parameters and attributes, similar to a Java servlet request. The request can contain properties to allow extension, and client header fields being transported from the portal to the portlet and vice versa.
Session	The portlet can store data in the session with either global scope, to let other components of this Web application access the data, or portlet scope, which is private to the portlet.
Context	The portlet can store data in the Web application context, similar to servlets

The portlet can access persistent data with these scopes:

Per portlet	The portlet can store configuration and personalization data in the portlet preferences to enable the portlet to create personalized output. The portlet can define which data the user is allowed to change in the edit mode (for example, stock quotes), and which data are configuration settings can only be changed by an administrator in configuration mode (for example, the stock quote server).
Per user	User profile information can be read by the portlet to tailor its output towards the user (for example, show the weather of the city where the user lives).

JSR 168 therefore provides a basis for developing portlets that support a particular set of functionality – allowing portlets to interact with each other and support for user personalization. WSRP [WSRP] specification on the other hand is aimed at providing a standard for embedding plug-in capability into portlets that can be supported through specialist Web Services. The aim of this specification is to allow Web Services to interact with portlets. This standard allows a portlet to be published as a WSRP producer, which can then be consumed by one or many compliant portals (WSRP consumers). The standard also provides ease of adding remote portlets on a portal page just like adding any other local portlet.

Based on these two standard specifications, we have compared a number of portal frameworks to assess their validity with reference to the SRD. The comparison (and the criteria used) are described in the table below. Based on this investigation, we have chosen the eXo Portal Framework [eXo].

Criteria	eXo	Liferay	Gridsphere	Jetspeed1	uPortal	JBoss
1. JSR 168 and WSRP compliance	☺☺☺	☺☺☺	☺ (no WSRP compliance)	☺ (no WSRP compliance)	☺☺☺	☺ (not truly compliant)
2. Ease of initial setup and documentation – ease of initial start-up from download to first running of portal	☺☺☺	☺ Difficult to setup initially and lack of documentation	☺☺ Average	☺☺ Average	☺ Difficult to setup	☺ Setup is easy but lack of documentation
3. Community – how popular the portal tool is in various forums and blogs to share knowledge	☺☺☺	☺☺	☺☺	☺☺☺	☺☺	☺
4. IDE integration – this allows ease of development and debugging	☺☺☺	☺	☺	☺	☺	☺☺
5. Sample portlets – this is quite essential due to lack of documentation linked to most free portal software	☺	☺☺☺	☺	☺	☺	☺☺

Average - ☺ Good - ☺☺ Very Good - ☺☺☺

Criteria	eXo	Liferay	Gridsphere	Jetspeed 1	uPortal	JBoss
6. Ease of developing new portal pages and linking to a navigation portlet	☺☺☺	☺☺☺	☺☺	☺☺☺	☺	☺☺
7. Ease of portlet deployment – A interface to achieve this can considerably save development time, specially during debugging stage	☺☺☺	☺	☺	☺	☺	☺
8. Dependency on servlet container – compatibly with only single server environment is very restrictive for web deployment	☺☺	☺☺☺	☺☺	☺☺	☺	☺
9. Performance	☺☺	☺ (crashes a lot)	☺	☺☺	☺☺	☺☺
10. Security setup	☺☺☺	☺☺☺	☺☺☺	☺☺☺	☺☺☺	☺☺☺
11. Portal management tasks	☺☺☺	☺☺	☺☺	☺☺☺	☺☺	☺☺
Final Conclusion	**29**	**23**	**18**	**22**	**18**	**18**

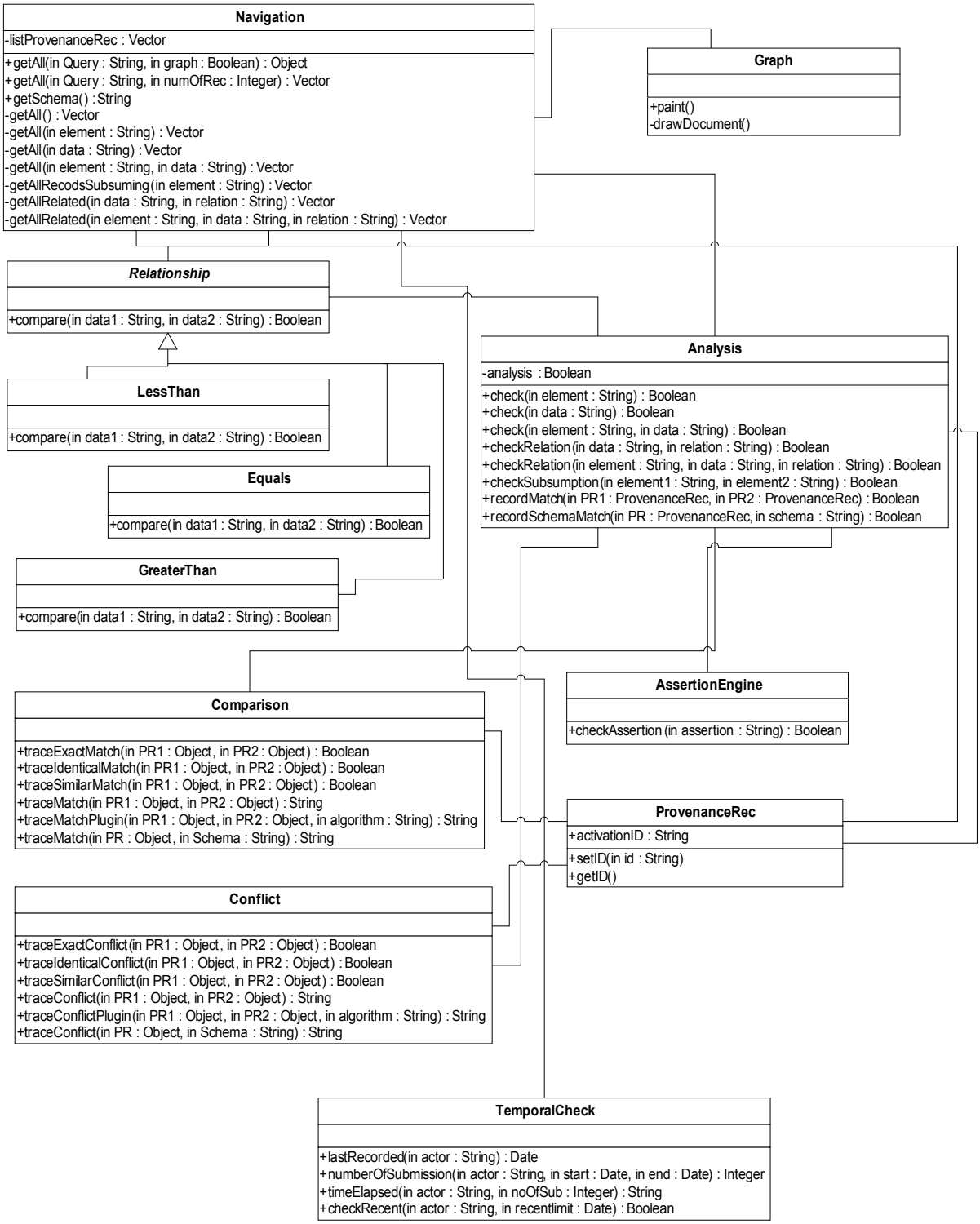

Figure 4: UML Class Diagram of the Tool Suite

4 Tools and PS Interaction

The sequence diagram in Figure 5 illustrates the communication between the Tool Suite (WP6) and the PS. Interactions between the tools and the PS consist of either: (1) query for

data contained within a PA, or (2) query to determine the schema for a PA. Below we detail different interactions taking place between the Tool Suite and PS.

Navigation – PS interaction: The navigation tool provides browsing capabilities for content maintained in the PS. This capability extends that already provided by the query API of the PS. For instance, the Query API is generally intended to manage XPath queries, whereas the navigation tool can allow visualization of results from such XPath query, or specialist conditions (generally specified as a set of rules) that need to hold over the results that have been returned.

Once a query has been received by the navigation tool, it is divided into sub-queries, converts each into an XPath query, and submits these to one or more PSs. The number of PSs involved is based on the configuration provided by a user during the setup process. Subsequently, the navigation tool interacts with the PS Query API to retrieve the required results for each of the sub-queries. This process would be repeated several times until all the sub-queries have been processed.. Once all results have been received, a final processing is performed by the navigation tool to generate a response based on the collected information from the PS. The two methods that may be used by the navigation tool to query the PS are: a) getAll(in Query String, in graph boolean) and b) getAll(in Query String, in numOfRec Integer). Both of the these accept queries in different formats from the application, these are then converted to the XPath query format, so that the available Query API at the PS can be used from this point onwards.

Most of the interactions taking place between the Tool Suite and the PS are therefore through the navigation tool. The outcome of a query generated to the PS may be therefore be passed on to other tools for analysis, or may be stored for delivery to the end user.

Analysis – PS interaction: The analysis tool provides capabilities to compare relationships between a PA and the current schema used to specify the structure of content within one or more PSs. In most cases, the analysis tool may also use results generated through the navigation tool. However, some specialist interactions are also provided between the analysis tool and the PS to enable faster operation. One such interaction involves retrieval of the schema of a PS in order to perform either similarity checks or conflict detection.

The interaction between the PS and the Tool Suite are limited to just the above two interaction for now, but in future new tools may be added to the Tool Suite.

5 Tools and Application Interaction

The sequence diagram in Figure 5 also illustrate the communication between the Tool Suite (WP6) and other components within a provenance system. Key components are the analysis and the navigation tools, which are made use of by all the other tools, namely the comparison conflict and temporal tools. Below we describe interactions taking place between these tools.

Application-Navigation interaction: Communication starts between the two components with the application tool issuing a query to the navigation tool. This query may include retrieving PAs with or without a graphical representation option. One of the graph type would include the option of “no graph”, this would result in retrieval of PAs only. The navigation tool interacts with the PS Query API using the query preference to retrieve the required results. The navigation tool then interacts with the graph creation tool, to create appropriate visualisation of these returned result.

Application-Analysis interaction: An application could interact directly with the analysis tool to check assertions. These assertions can either be on an XML element, a data or a combination of an XML element and data. Assertions can also include relationships based on some constraints over the data contained within the XML element. Once a method call is initiated by the application with appropriate input parameters, the analysis tool uses the assertion engine to perform the analysis. Once results are retrieved on a group of assertions, the analysis tool may need to aggregate the results to formulate the final boolean result that needs to be returned to an application.

Application-Comparison interaction: The interaction between the application tool and the comparison tool could either be initiated directly if the comparison is taking place between a PA and a schema or could be via the navigation tool, once the user has acquired appropriate PAs to compare. Interaction between the application and the comparison tool for each of the above two cases are displayed in the sequence diagram in figure 5. The comparison tool upon receipt of a call for either of the two cases, initiates a further call to the analysis tool to analyse the relationship. Analysis tool as always uses the assertion engine to analyse relationships between the PAs, upon completion, the analysis tool sends the result to the comparison tool which in turn forwards the result to the application.

Application-Conflict interaction: The interaction between the application and the conflict detection tool is exactly the same as the interaction between the application tool and the comparison tool, except that in case of comparison tool the aim of the analysis tool is to compare similarity, while in the later case it is to detect conflicts.

Application-Temporal interaction: The temporal tool allows two main types of interactions. One that retrieves temporal information about an actor and a second to ascertain some facts regarding time information that an actor has submitted. In the first case, the temporal tool interacts with the PS using the navigation tool to retrieve the required information, which is then sent to the application making the initial call. For the later case, the temporal tool uses the analysis tool which in turn interacts with the assertion tool to analyse a fact. The temporal tool sends the results obtained from the analysis tool back to the application initiating the call.

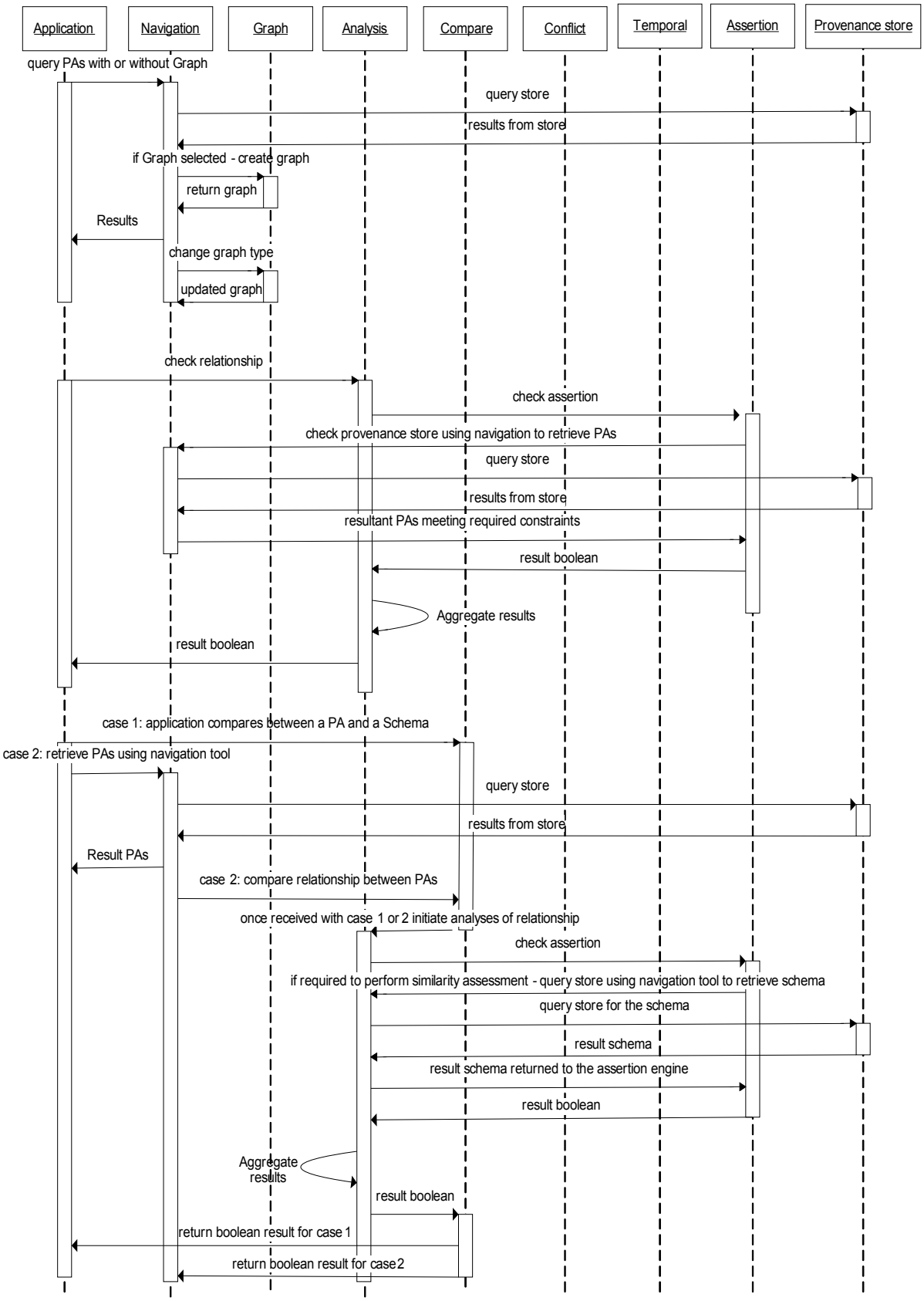

Figure 5: UML Sequence Diagram of the Tool Suite

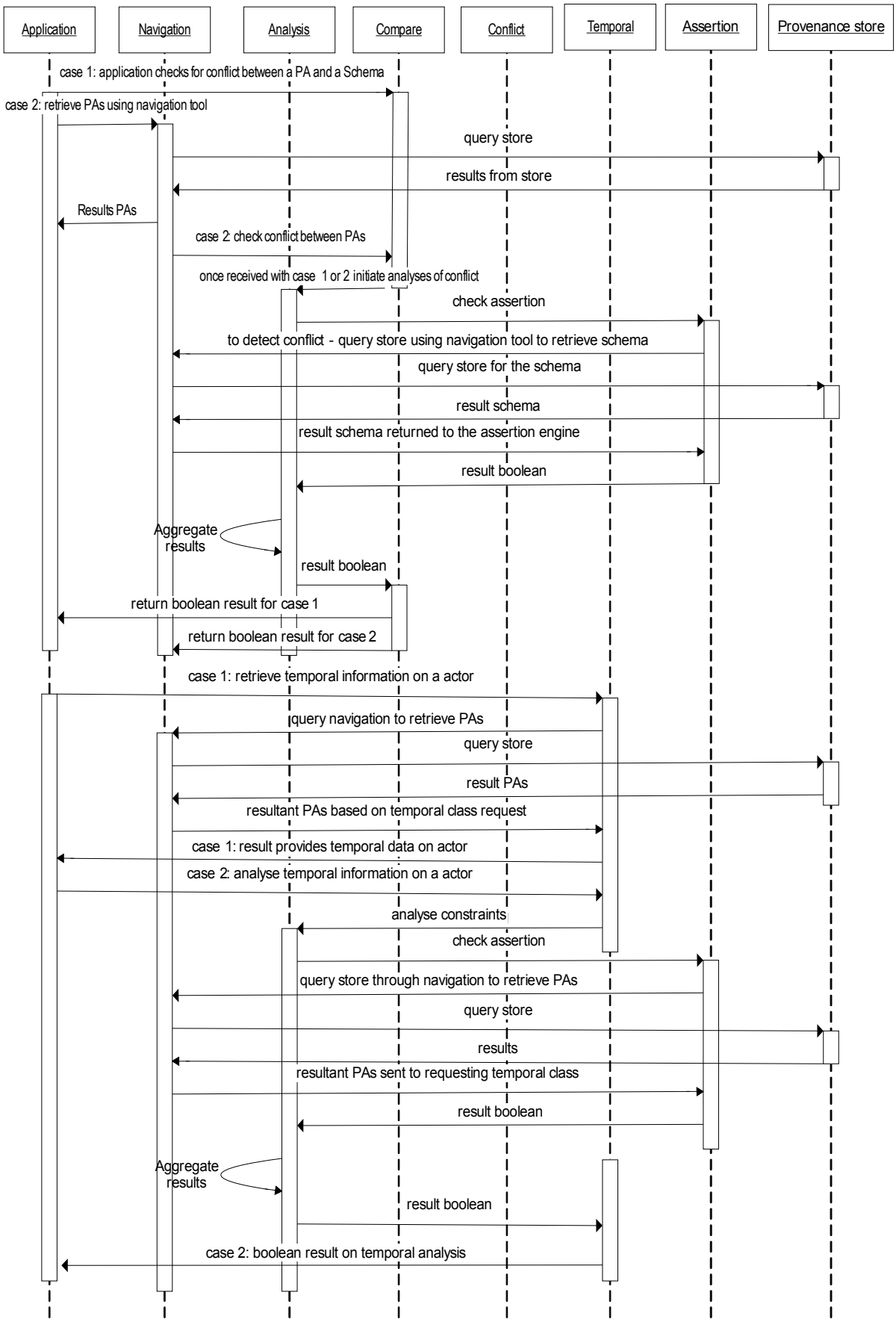

Figure 5 - continued: UML Sequence Diagram of the Tool Suite

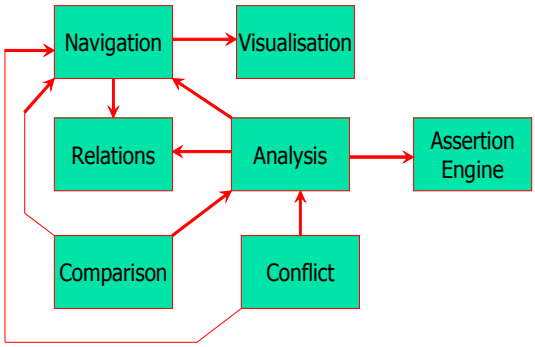

Figure 6: Tool Interaction

Figure 6 illustrates the interaction between the tools. Each arrow illustrates a "makes use of" relationship. Hence, the navigation tool makes use of a visualization (or graphing) tool. The analysis tool is a significant component within the system as explained previously.

6 Types of Users

It is useful to provide some discussion about the potential user base for the tools. The intention of such a discussion is to provide some context for the tool operations discussed in Section 3 above. Tools are primarily intended for following types of users:

- **Provenance System Administrators**: These individuals are responsible for configuring, deploying and managing the overall provenance system. For instance, they may be responsible for configuring a PS and manage the security policy associated with reading or submitting to a PS.

  Provenance System Administrators are likely to make use of the setup protocol and the configuration tool.

- **Application Administrators**: These individuals would be responsible for configuring the tools for use within a particular application. We consider such individuals to be members of the systems administration team at a particular end user site, or specialist IT experts who are responsible for implementing or managing software applications at an end user site. Configuration parameters would include identifying the location of the PS, identification of any particular visualization capability that would be required by users of a particular application, checking security policy and identifying suitable security checking mechanisms that need to be in place before an end user can submit a PA or retrieve a PA from a PS. Such users are likely to have access to the application end users at a particular site, and therefore must understand constraints on access control within their particular organization. It is expected that where security domains have already been specified and defined for information access, these are made use of also for accessing provenance data.

  Application administrators are also responsible for specifying queries that must be submitted to the PS. Such queries should make use of terms used within an application specific schema, but formatted in a form that the tool could then execute over a PS. If a set of commonly used queries can be found, these may be bundled into a library for use by an application end user. An application administrator would also be responsible for generating specialist plug-ins that could be used for comparing different type of data. This may be

achieved by identifying how specialist "documentation styles" may be specified that are relevant for a particular application.

Application administrator could make use of all the functionality in the Tools Suite, the Setup protocol and the Configuration tool. The software deliverables from WP6 are primarily intended for these types of users.

- **Application End Users**: We consider "end users" to be the decision makers or evaluators who either need to record PAs into the PS, or analyze existing PAs that have been submitted previously. In the DLR application, these individuals may be computational scientists responsible for running a particular workflow scenario. In the OTM application these may be particular clinicians or administrators responsible for tracking a particular case (identified as the Implant Team, the Duty/Consultant Transplant Surgeon or members of the Regional Organ Transplant Authority for instance – from WP8 deliverable D8.1.1). It is also envisioned that application end users may make use of tools that have been configured by Application Administrators. Such individuals may already be using existing specialist application specific interfaces to undertake their work. For instance, in the DLR application, computational scientists are currently presented with the interface provided by the TENT system (these users are identified as Application Actors in WP7 Deliverable D7.1.1). These interfaces or access mechanisms will not be modified – instead additional "portlets" will be provided that can be accessed through a Web browser or as a standalone Java application to allow navigation over the PS. The portlet client may however be embedded within an existing graphical interface (as outlined in Section 4.2.2.2 of Deliverable D7.1.1), or it could be invoked by a call that launches an external application. (There are a number of reasons for supporting this separation: (1) the application interface does not have to be modified – this is particularly useful if the interface is tightly coupled with the existing application implementation; (2) access to the PS can be made by end users who may not be involved in the experiment that led to the PAs; (3) interfaces to applications may not allow remote access (such as in the existing TENT application). The portal framework allows the provision of remote navigation support for data contained in the PS; and (4) the types and language for defining queries to the PS may be different to those being made within the application interface.

  Application End Users are likely to make use of the navigation, comparison, conflict detection and analysis tools.

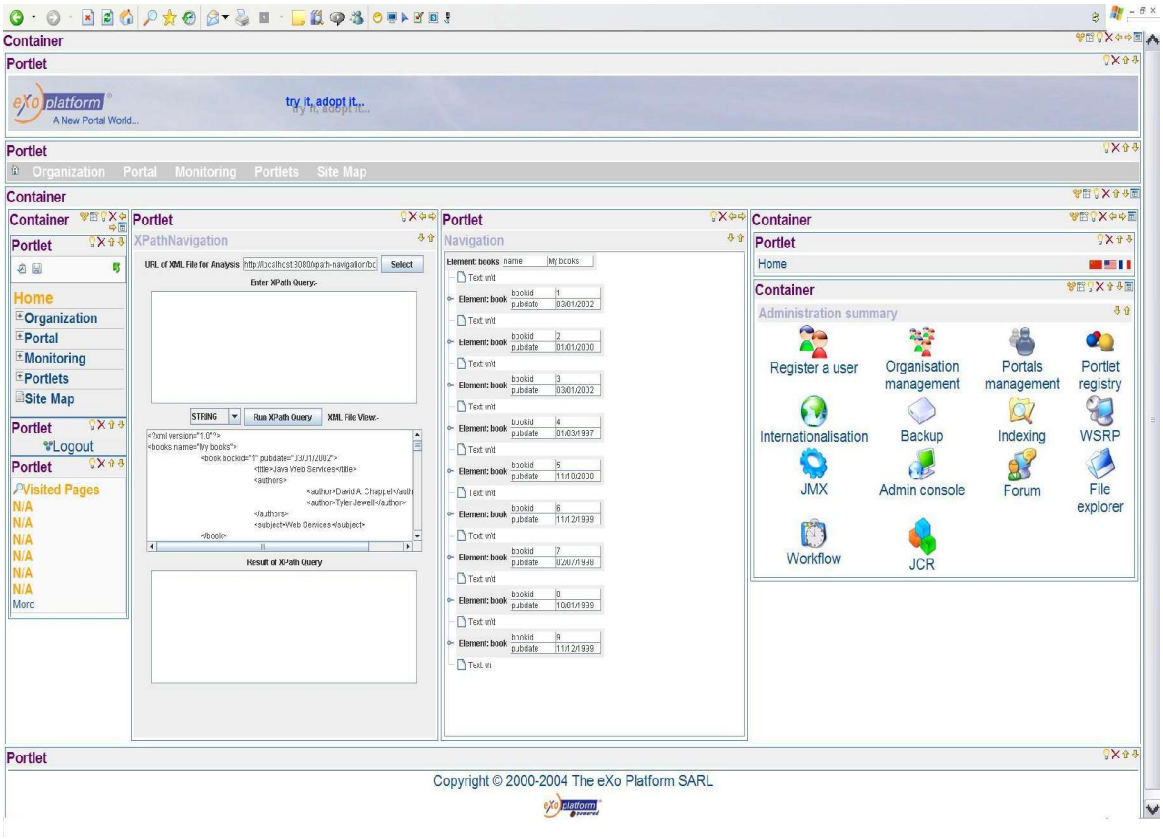

Figure 7: eXo Portal Framework

An example of the portal interface is provided in Figure 7. This demonstrates the interface that would be provided to Application Administrators. The portlets on the the right hand side of the figure provide support for registering new portlets, registering a user to access existing portlets etc. The middle frames demonstrate application specific portlets that provide a navigation portlet providing a tree view of a PA – while the other portlets allow visualization of a PA as an XML document.

6.1 Examples of Usage

Some queries that may be handled by the tools include the following (in this description <e> refers to an element in the PA, and d:<e> refers to data value associated with element <e>) are discussed below. It is assumed that the PA schema already exists and is specified as part of the setup protocol by the application user. The examples below are based on query descriptions in D8.1.1 (Section 5.1.4).

- Find all PAs submitted by actor <a>
  - The result of this query is a PT containing all the PA that contain the element <a>. The query makes use of the navigation tool.

- Find a PA submitted at/over time <t>
  - The result is a PA or a PT which may be ordered based on the some start time also specified by an application user. This query makes use of the navigation tool, and the results can then be ordered using the analysis tool.

- Find n PAs containing element <e>
  - Returns the first "n" PA that can be found in the PS. This is to restrict the total number of PAs returned. Makes use of the navigation tool.

- Find n PAs containing data d:<e>
  - Returns the first "n" PAs. This is to restrict the total number of PAs returned. Makes use of the navigation tool.

- Is condition <X> on PS valid. Where <X> is defined using an application schema and the set of relationships that must hold on <X>.
  - Returns True/False. Makes use of the navigation and the analysis tools.

Based on the Organ Transplant Management (OTM) application, it is possible to specify the following queries:

Query 1: **Retrieve metadata and references to all actions/events associated with a case**.

In this instance, the navigation tool is used, with the query <e>=CaseID, with the subsequent retrieval of all PAs which have CaseID as a schema element. It may be necessary to combine results from different case identifiers (as described in Section 5.1.4.3 of Deliverable D8.1.1) – in which case the associations between different CaseIDs need to be made by the Application End User.

In some instances, it is possible that CaseIDs may be encoded, or perhaps need to be extracted by referral to a particular format for recording the data contained in a PA. In this instance, it would be necessary to extract a PA, apply the appropriate plug-in, and then return results back to the end user.

Query 2: **Determine decision tree for a particular case.**

In this instance, the navigation tool would be used with the query: Given <e>=CaseID, retrieve all PAs associated with <e>. The results of the navigation tool are then compared based on a <time stamp> on each PA, which may be used to order the PA. Elements of the P-Structure identified as part of WP3 may be used to inform which PAs are associated with a given case. Using the workflow portlet in the navigation tool, a workflow graph may be constructed as PAs are retrieved, and by making use of "relationship" information contained within each PA.

The level of detail presented within the workflow graph may be configured by a user. For instance, in the first instance, a user may not need to expand and see all the data contained within a PA, only that a PA had been recorded. A user may then interact with the workflow graph to either retrieve additional PAs, or retrieve the full data contained within a PA.

Query 3: **Determine if a particular medical staff member was involved in a particular decision.**

In this instance, the navigation tool is used to first retrieve PAs that contain the CaseID and the particular staff member. The analysis tool is then used to check whether a given PA contains both the CaseID and the mentioned individual. Hence, Given (<e1>=CaseID && <e2>=PersonX) does this evaluate to True? This would be equivalent to undertaking a validity check on a subset of PAs that have been retrieved a response to a query.

With reference to the DLR application, a tracer mechanism (Deliverable D7.1.1, Section 4.2.2.2) is being used to refer to a "simulation set". In this instance, a query would reference a "tracer" rather than a CaseID (as in the OTM application). There is also a requirement in the DLR application to present the retrieved PA graphically (as a tree structure) or as a text file. The use of portlets that allow different views on the same data would provide a useful basis for achieving this. Similar to the OTM Query example 2 above, it is also required to manage how much information must be presented to a user in response to a query. This would correspond to a configuration parameter that a user needs to specify to limit the level of detail to be presented.

7 Interaction with other Work Packages

The Tools and Setup WP has interacted with the other WPs in the following way:

- WP2: Understanding of requirements from the applications surveyed, and subsequent production of the SRD. The division of requirements into "essential" and "desirable" has provided the basis for the development of the navigation tool. In the first instance, the intention is to develop portlets that primarily address the "essential" requirements. The URD and SRD have formed the basis for identifying tools in WP6, and an assessment is provided in Section 2.1.
- WP3: The tools being implemented can be related directly to the logical architecture (the relationship as understood at this stage has been described in Section 2.2). Determining exactly how workflow mechanisms should be described and subsequent reconstructed, is still being evaluated through discussion between WP3, WP6, WP7 and WP8. No consensus on this has yet been reached, although it is clear that support for workflow remains an important requirement. It is also outlined here how tools place particular requirements on the architecture itself (based on interpretation of SRD).
- WP4: Security support being implemented in this WP is being used in the Setup protocol.
- WP7 and WP8: Application scenarios identified in these WPs have formed the basis for defining the navigation tool (such as support for workflow derivation from submitted PAs). The choice of the Portal framework (described in Section 3.7) has also been informed based on requirements from the DLR and OTM application scenarios.

The overall interaction with WP3 has been to better understand the PA structure and possible query formats that could be supported at the PS based on the PA structure. One example of this is the need to support a particular "documentation style" to allow encoding of data stored in the PS through the use of an application-specific plug-in. The types of plug-ins are being identified by WP7 and WP8, and WP6 is providing an API to specify application specific plug-ins in a common way.

8 Summary and Future Plan

The focus of this WP has been on the design and development of tools to support Application Administrators (Section 6) in the first instance. Such tools may be adapted for use by an Application End User, but this has not been the aim in the first instance. The tools being provided include: (1) navigation tool: that allows retrieval of PAs based on user defined queries, and subsequent visualization of these PAs through a set of portlets. An important role of the navigation tool would be to assess how to retrieve all PAs that have been made about the processes leading to a particular result. In the same context, it would be useful to determine how much detail should be retrieved and presented to a user – for instance, would be sufficient to simply return a list of PAs that have been made, or would the full content of a PA also need to be made visible. This is clearly a configuration parameter that needs to be specified; (2) analysis tool that makes use of an assertion engine: primarily for defining relationships between PAs and for checking for the validity of certain data in a particular set of PAs. Such validity checking is supported through a set of rules that are application specific; (3) the comparison tool: allows two or more PAs to be compared, and identifies the degree of similarity (exactly or loosely similar, or similar to a certain "distance"); (4) the conflict tool: evaluates whether submissions being made about the same event by two different actors differ in some way.

Significant work has been undertaken so far on identifying the requirements that the tools need to address (with reference to WP7 and WP8), and particular architecture components that are needed to support tools (WP3). This analysis has informed the design of the Tool Suite. A comparison between different portal frameworks has been undertaken, with the selection of "eXo". Similarly, evaluation has been undertaken between different rule engines, with the selection of "JESS".

Work in the immediately future is to focus on how the currently developed tools can be used in mini-workflow scenarios identified by the DLR and OTM applications. Initial work has already started on how DLR queries for access to recorded PAs can be mapped to rules, and the identification of suitable wrappers for DLR applications that allow submission of PAs. A possible schedule to Summer 2006 is as follows:

December 2005

- Establish setup protocol
- Complete comparison and analysis tool

March 2006

- Implement setup protocol
- Implement workflow visualization portlet. As part of this effort, it will also be necessary to better understand how an application specific workflow should be described, and subsequently how it should be reconstructed. For instance, given a particular PA, it may be possible to look back to all the PAs that were submitted before it. Identifying how a workflow graph could be constructed as new PAs are retrieved and related to PAs that have already been processed by the tools is an important part of this activity.
- Implement example of a relationship plug-in. This will be undertaken in collaboration with WP7.
- Integrate tools with OTM and DLR application mini-workflow examples.
- Investigate any constraints on use of Portal Framework.
- Design of configuration tool.

July 2006

- Integrate Tools with OTM and DLR application
- Integrate Tools with Security Framework (for Setup Protocol)

References

[D2.2.1] Project Deliverable D2.2, "Software Requirements Document". Version 1.0, 2005, http://twiki.gridprovenance.org/pub/Restricted/DeliverableD2dot2dot1/SRD-v1-ofr-JBI10-03-05.sxw (Project internal web site).

[Moreau05] "Logical Architecture Strawman for Provenance Systems". Version 1.11, 2005, http://twiki.gridprovenance.org/pub/Restricted/LogicalArchitecture/strawman1-11.pdf (Project internal web site).

[D8.1.1] Project Deliverable D8.1.1, "Specification of mapping to provenance architecture, and domain specific provenance handling". Version 1.0, 2005, http://twiki.gridprovenance.org/pub/Restricted/DeliverableD8dot1dot1/OTM_Mapping_and_Provenance_Handling_Document_v0.62.sxw (Project internal web site).

[DROOLS] http://drools.org, November 2005.

[WP3] "An Architecture for Provenance Systems". Version 0.3, 2005, http://twiki.gridprovenance.org/pub/Restricted/LogicalArchitectureFrozen/logarch-v0.3.pdf (Project internal web site).

[Jetspeed] http://portals.apache.org/jetspeed-1/, September 2005.

[JESS] Java Expert System Shell -- http://herzberg.ca.sandia.gov/jess/, November 2005.

[JSR168] "Introduction to JSR 168 - The Portlet Specification", Sun Microsystems Developer Network, July 2003. Available at: http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168/

[JSR168-IBM] "Comparing the JSR 168 Java Portlet Specification with the IBM Portlet API", Stefan Hepper, IBM Developer Works, December 2003. Available at: http://www-128.ibm.com/developerworks/websphere/library/techarticles/0312_hepper/hepper.html

[JSR94] Java Rule Engine API Specification. Available at: http://www.jcp.org/aboutJava/communityprocess/review/jsr094/.

[Gridsphere] http://www.gridsphere.org/gridsphere/gridsphere, September 2005.

[Liferay] http://www.liferay.com/web/guest/home, September 2005.

[eXo] http://www.exoplatform.com/portal/faces/public/exo, September 2005.

[ORULES] Rule-based Web Development (Open Rules) – http://openrules.com/, November 2005.

[WSRL] http://webservices.sys-con.com/read/39555.htm

[WSRP] OASIS – Web Services for Remote Portlets Specification. Available at: http://www.oasis-open.org/committees/ tc_home.php?wg_abbrev=wsrp