PROVENANCE Contract Number:
Enabling and Supporting Provenance in Grids for Complex Problems 511085

Title:	**Setup Protocol Implementation** Work Package 6
Authors:	Omer F. Rana, Arnaud Contes and Vikas Deora
Editor:	
Reviewers:	
Type:	Deliverable
Version:	0.1
Date:	January 24, 2006
Status:	Draft
Class:	Confidential

Summary
The purpose of this document is to define the setup protocol required to allow an application to submit and access the Provenance system. The stages involved in the protocol are described with reference to the components of a Provenance system. The setup protocol is coordinated by the tool suite administrator process and results in the generation of a configuration file that can subsequently be used by a client-side application.

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 für Luft- und Raumfahrt e.V.	Germany
Universitat Politecnica de Catalunya	Spain
Magyar Tudomanyos Akademia Szamitastechnikai es Automatizalasi Kutato Intezet	Hungary

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

Foreword

This document describes:

- Message order and format in the "setup protocol".
- Interactions supported in the Setup Protocol with the Provenance Store and the tool suite.
- User interface support provided for setup in the Portal.

The primary audience of this document include: administrators of the Provenance Store, application managers making use of the tool suite, and administrators responsible for the security sub-system.

Contents

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

List of Figures

List of Acronyms

- HTTP: HyperText Transfer Protocol.
- MIME: Multimedia Internet Mail Extension.
- URI: Uniform Resource Identifier.

1 Introduction

The aim of the setup protocol is to achieve a setup process. A protocol may generally be defined as a set of states, each of which represents a particular activity undertaken between one or more components of the system. The states in the setup protocol are therefore aimed at supporting this overall setup process.

The operations undertaken in the "setup protocol", the message formats used, and relationship to the application scenarios are described in this document. The setup protocol is divided into a number of stages/states – allowing a system administrator to add additional states if required in the future. This document is primarily intended for:

- *Provenance System Administrators*: These individuals would be responsible for identifying the set of states that would be necessary in the protocol, and in particular how these states relate to functionality already provided by the Provenance Store (PS). For instance, some aspects of the protocol would require use of the management interface at the PS. More details can be found in Sections 2 and 4.3.
- *Application Administrators*: These individuals are responsible for configuring the interaction between the application, the tool suite and the PS. Such users must therefore provide configuration parameters required within the setup process – in collaboration with application end users. The outcome of the setup process is the creation of a configuration file for use by application administrators to deploy one or more application instances. Under the application administrators definition, we aggregate two complementary functions: the application administrator who is responsible for configuring the application, and the application manager who is in charge of maintaining all the data associated with the application. In the rest of the document, we will use indifferently these two definitions.
- *Tool Suite Administrators*: These individuals are responsible for ensuring that all the tools (navigation, analysis, etc) are ready and available for use. They may be the same as application administrators – or may be part of the overall Provenance system administration team. Any errors produced during the setup process must be logged by the tool suite administrators. The error messages may subsequently be analyzed by application administrators.

The user classification is based on the description provided in Deliverable D6.1.1.

Based on further discussion, a new role has been identified by one of the application end users in the Provenance project (DLR), which is that of a *Project Manager*. A project manager is responsible for configuring the interaction between the application user that belongs to the project and the tool suite and the PS.

1.1 Purpose of the Document

The purpose of this document is to explain the setup protocol and its relationship to other components in the Provenance system. A distinction is made between setup and configuration:

- Setup involves identifying the operations that are necessary to enable an actor to make a provenance submission, or to retrieve data from a PS. Such operations will make use of functionality provided within a client application, the tool suite and the PS.
- Configuration involves identifying the set of parameters that may be associated with a setup process. Essentially, the setup protocol focuses on the set of activities (or processes), whereas configuration focuses on specific instances of these activities, with support for generating particular values for the parameters associated with each activity in the protocol.

It is useful to note here that such a protocol may include states which are application specific especially as resources (such as external data sources or external identity generation mechanisms) which do not directly constitute components in a Provenance system may be required. Currently, configuration parameters are assigned manually to evaluate the setup process additional support will be provided for an application manager (which allows configuration parameters to be defined, managed and shared within an application) as part of the next deliverable from WP6. The setup process includes a setup protocol (defined in this document) and configuration (which will be described in the next deliverable from WP6). Additional checks will be needed on the generated configuration parameters, these checks are not discussed in this document.

1.2 Document Overview

This document is structured as follows:

- Section 2 describes why a setup protocol is required and the likely operations that need to be supported within the protocol. In this section, reference is also made to the particular requirements identified in the application scenarios from WP7 (in particular).
- Section 3 contains a description of the protocol such as the types of actors, the state sequence, and message formats used in the protocol. This section also provides a description of the status codes that are generated if an error is produced during the execution of the protocol.
- Section 4 relates the setup protocol to the navigation (via the user portal) and analysis tool. Detail about the relationship between the user portal and the setup process is discussed here, along with how the portal may be customized by a user.
- Conclusion and further work is presented in section 5.

1.3 Links to other Provenance Documents

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

- WP2: Requirements identified in D2.2.1. In particular, the focus is on requirements that impact the setup of the Provenance system – such as SR-1-10, SR-1-11, SR-1-17, SR-1-18, SR-6-3, and SR-7-2. The last of these is particularly significant, as it states that the Provenance system should be loosely coupled to the application that makes use of it.
- WP3: The frozen architecture document D3.1.1. The setup protocol is discussed with reference to the architectural components discussed in section 3.3 of the Logical Architecture – such as the relation between the actor-side libraries and the PS.
- WP6: Tools Deliverable D6.1.1 – such as the operations necessary to support the navigation and analysis tools. The setup protocol also determines the number and types of portlets that should be provided to a given user by default.

2 Need for Setup Protocol

The setup protocol is employed to bootstrap an instance of an application and to allow it to submit or access data from one or more PSs. The setup protocol may therefore be viewed with reference to an application being initialized to interact with the Provenance system which primarily includes the tool suite and the PS. The setup protocol is initiated by the application manager based on a request from an end user starting up an instance of a Provenance-aware application, or an application end user deciding that it is necessary to start the recording of data in a PS. Changes in an application (such as a change in security access allocated to a particular end user) will necessitate the setup protocol to be run again. It is therefore envisioned that in a system that does not change very often i.e. where user privileges are not modified frequently, the setup protocol will only be needed once. The key aims of the protocol are:

- Identify actors involved in the Provenance system. Where an application is composed of multiple submitting actors, the application manager must separately register these actors with the tool suite.
- Verify credentials of actors, and based on these credentials determine access rights for each actor. It is likely that each component in a Provenance system (such as the PS and the Portal server) will provide its own access control mechanism.
- Verify that all components within the Provenance system are on-line and ready to receive (and process) requests.

- Verify access to any third party data servers that also need to be involved in retrieving of data. There may be multiple such servers involved.
- Verify that suitable plug-ins for user defined "documentation styles" are available. Plug-ins may be local or remote, and supported on the server hosting the application, the tool suite server or a third party server.

Additional states may be added to this protocol, depending on additional checks that may need to be performed by an application manager. Figure 1 defines the set of states necessary within the setup protocol. The initial state is activated by the application manager. All subsequent state transitions are managed by the tool suite. The final state generates configuration files that can be used by application's actors through the client side library. There are two possible reasons for the termination of the protocol: (1) one of the states in the process generates a failure; (2) a configuration file is produced. In case where one of the states reports a failure, an error code (see section 3.5) is generated and logged – and may subsequently be sent to the application manager. Depending on the importance of successfully completing a particular stage in the setup process, it may be possible for some states to fail and the setup protocol to still complete successfully. This is a policy decision that needs to be made by the application manager initiating the protocol. For instance, it may not be useful to successfully complete the setup process if no PSs are available; conversely, failure on a plug-in or portlet check may still allow an application making use of the provenance system to work successfully.

An example where additional states may be necessary is if p-assertions in one PS are moved to another – in this case a p-assertion that was previously resolved via a particular PS, may now need to be referenced to another PS. This issue has been discussed in Section 6.5 of the WP3 frozen architecture document. In this instance, prior to submission of a query to a particular PS, it would be necessary for the client side library to register the new PS location with the tool suite – as part of the setup protocol. The overall coordination of the setup process is achieved through the use of the tool suite.

It is expected that prior to the setup process being invoked, the following components are already available:

- Application manager process: responsible for initiating the setup protocol, the application manager process must be running prior to the setup protocol being called.
- Tool suite administrator process: responsible for coordinating the setup process. The application manager interacts with the tool suite to active the setup process.

2.1 Relationship to Logical Architecture

The logical architecture (D3.1.1, Section 3.3) discusses the use of *policies* to help configure the system. Such policies should: (i) identify the PS to use, (ii) the

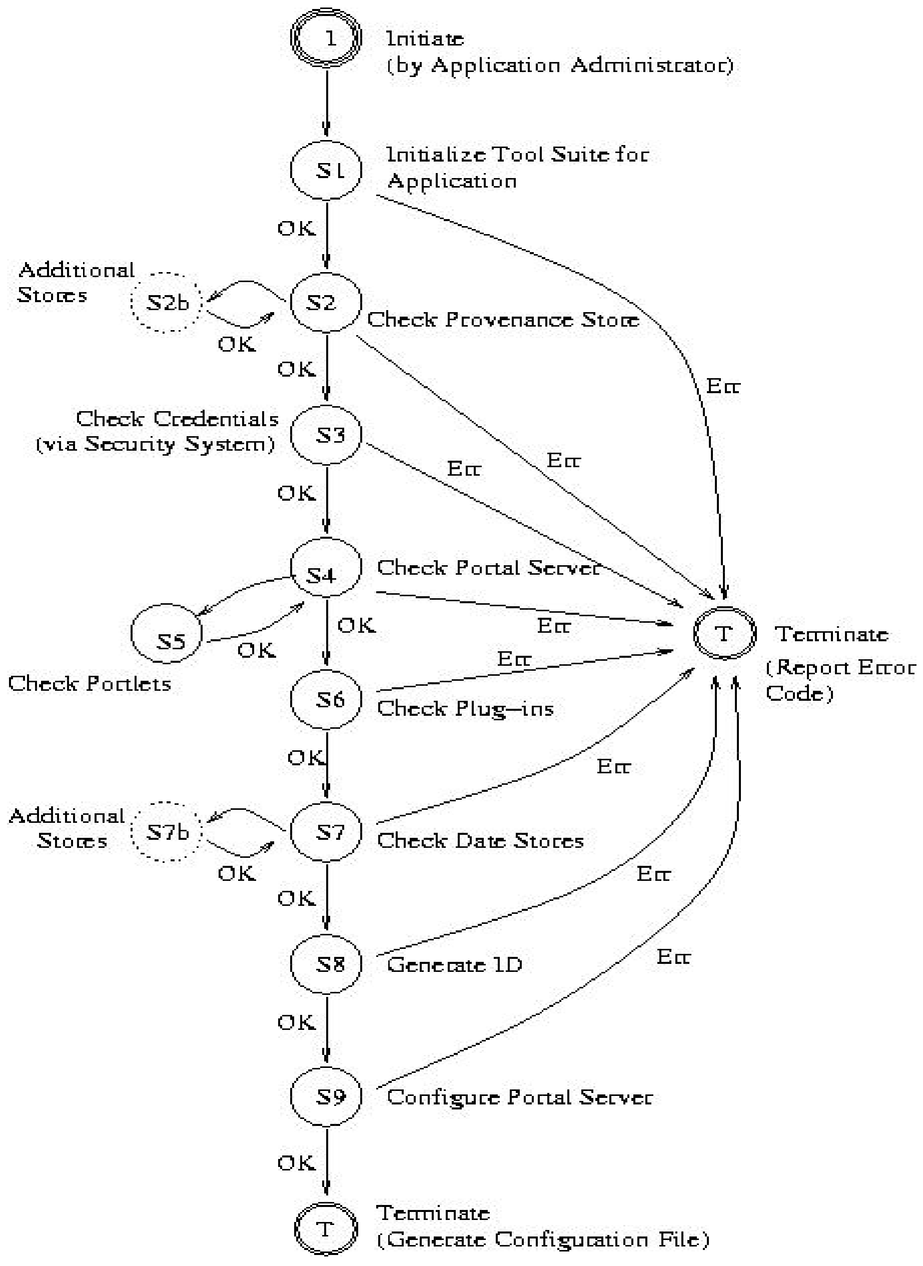


Figure 1: *States in the Setup Process*

level of documentation required by the user, (iii) security privileges that need to be associated with access. Other policies should be responsible for setting up the PS itself, and access to any third party data stores that may be involved in the recording or access process. The setup protocol relies on the existence of such policies – which need to be defined by different individuals, such as the application manager, the PS administrator and the tool suite administrator.

2.1.1 Documentation style modelling

As stated in the logical architecture document (deliverable D3.1.1), when an actor documents an interaction, it constructs an interaction p-assertion (specified in Section 6.4 of the logical architecture document), which states the content of a message received or sent by the asserting actor. We regard the activity of constructing a p-assertion from a message as an atomic transformation, which needs to be qualified by the asserting actor so that querying actors can understand the nature of the transformation that was applied to an application message. This is exactly the purpose of Documentation Style, which we now define.

Definition 1 *(Documentation Style) Documentation style is a representation of the transformation according to which the content of a message is asserted in an interaction p-assertion.*

Such a transformation is achieved at submission time by client-side libraries. A submitting actor may send a p-assertion with no transformation (referred to as a the verbatim style) to a PS. However, is is likely that an actor may transform a p-assertion prior to submission to a PS. In this latter case, the transformed p-assertion must contain a reference to a documentation style which primarily outlines how content contained in the p-assertion could be subsequently accessed in its original form (prior to the transformation).

The tool suite is now responsible for retrieving p-assertions from the PS, and using a documentation style provided in the p-assertion, responsible for identifying suitable plug-ins (explained in section 2.2.1) for transforming the p-assertion back to the original form. Given such an explicit representation of a message transformation, documentation style can be used for different purposes:

1. A documentation style is an explicit representation of the transformations that happened to a message, which enables a querying actor to navigate the contents of an assertion, through the use of corresponding schemas, and determine how the original message was transformed.

2. Once an application designer has identified the type of transformation that needs to be applied to a message, documentation style can be used by the asserting actor in order to construct a p-assertion from a message.

There are several types of documentation style that the provenance architecture may support, such as: verbatim, reference, anonymous, security-signing and security-encryption, etc. By default, the tool suite will support the verbatim style. Specialist plug-ins will be required for the other styles one example of which is the reference style for a WebDAV server. A description of these styles is as follows:

- Verbatim: denotes a null transformation applied to the contents of a message.
- Reference: denotes a transformation of a message by which a part of (or the whole of) its contents has been replaced by a reference to the location where the actual contents can be found.
- Reference-Digest: denotes a transformation of a message by which a part of (or the whole of) its contents has been replaced by a reference to the actual location where it can be found and a digest of the substituted data.
- Internal Reference: denotes a transformation of a message by which a part of (or the whole of) its contents has been replaced by a global p-assertion key, which refers to another p-assertion that contains the actual data.
- Anonymous: denotes a transformation of a message by which a part of (or the whole of) its contents has been replaced by an anonymous identifier. This identifier hides the actual data without losing the link to them.
- Security-signing: denotes a transformation of a message by which a part of (or the whole of) its contents has been signed.
- Security-encryption: denotes a transformation of a message by which a part of (or the whole of) its contents has been encrypted.
- Composite: denotes that more than one atomic documentation style has been applied to a message.

Additional details about these styles can be found in the logical architecture document (deliverable D3.1.1, Section 6.5 and 8.5).

2.2 Relationship to Application Scenarios

The DLR application from WP7 is used to illustrate various aspects of the setup protocol. The DLR application demonstrates how the documentation style approach may be used to access data in a server that is external to the Provenance system.

It should be noted however that the general concepts presented are also applicable to the OTM application, and not specifically restricted to the DLR application.

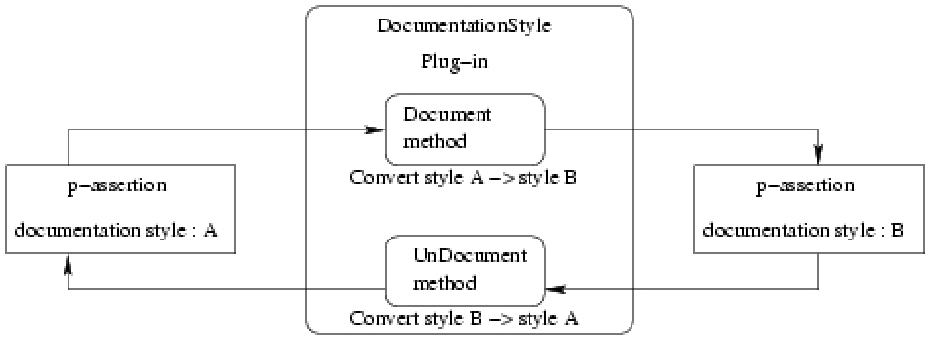


Figure 2: *Documentation style plug-in architecture : converting one style to an other*

2.2.1 Documentation style plug-in architecture

The transformation of a message into a styled p-assertion implies that to retrieve the original message, any p-assertion handler must understand the documentation style. There is no limitation on the types of documentation styles that may be supported currently, we provide support for a few standard styles, discussed in section 2.1.1. To be able to compare and navigate over contents in a PS, it is necessary for tools to provide a generic way to understand and manipulate various kinds of p-assertions. This is achieved by the introduction of a transformation plug-in architecture associated with the documentation style architecture.

In the general case, a p-assertion should contain all the data submitted by an actor to another during an interaction. However, it is possible that the size of this data is very large, or the data format cannot easily be interpreted at a later analysis. Indeed, the DLR application does not send contents during method calls but rather stores the data to be transferred onto an external storage location (a WebDAV server [7]) and only sends the location of such data as part of a p-assertion.

From a provenance point of view and considering the general case, we observe that the data will never be stored inside the PS – only a reference to such data will be stored. It is then difficult to ensure that the data will be always accessible. Indeed, if the location of the external storage location is changed, already recorded p-assertions may not be usable. In order to deal with this issue, which is similar in some ways to dealing with a loss of data, the concept of a documentation style plug-in has been introduced.

Hence, a plug-in designed for a particular documentation style is able to take a specific p-assertion as input and to produce an equivalent p-assertion, but one that is based on a different documentation style. Each plug-in is associated with a unique documentation style identifier. The process of documentation from one style to another one is presented in figure 2. We feel that the use of this plug-in approach provides significant advantage in making use of third

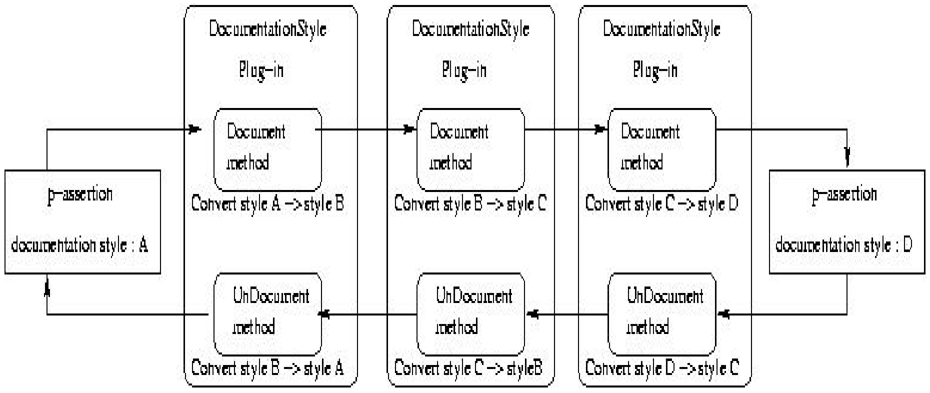

DocumentationStyle Plug-in
Document method
Convert style A -> style B
UnDocument method
Convert style B -> style A
DocumentationStyle Plug-in
Document method
Convert style B -> style C
UnDocument method
Convert style C -> styleB
DocumentationStyle Plug-in
Document method
Convert style C -> style D
UnDocument method
Convert style D -> style C
p-assertion
documentation style : A
p-assertion
documentation style : D

Figure 3: *Chain of documentation style plug-ins*

party data formats for which specialist access software may be necessary. This approach allows all the code linked to a particular documentation style to be kept in one place: the dedicated plug-in.

Hence an application specific documentation style can be handled by the tool suite in order to transform them into one of the standardized documentation style before being recorded in a PS. The generic design of a plug-in allows a user to create complex documentation style plug-ins based on a composition of several plug-ins as illustrated in figure 3.

From a programmer point of view, the creation of a plug-in for the documentation style architecture implies the implementation of at least one of the following interfaces:

- the `IDocument` is an interface for transforming a p-assertion with a particular documentation style into another one.

  ```
  public interface IDocument extends IDocumentationStylePlugin {

     public PAssertion document(PAssertion pAssertion);

  }
  ```

- the `IUnDocument` is an interface for extracting/reconstituting/understanding interaction p-assertions with a particular documentation style.

  ```
  public interface IUnDocument extends IDocumentationStylePlugin {

     public PAssertion unDocument(PAssertion pAssertion);

  }
  ```

The separation into two separate interfaces is induced by the fact that in some cases it could simplify the creation of a plug-in. For the WebDAV example

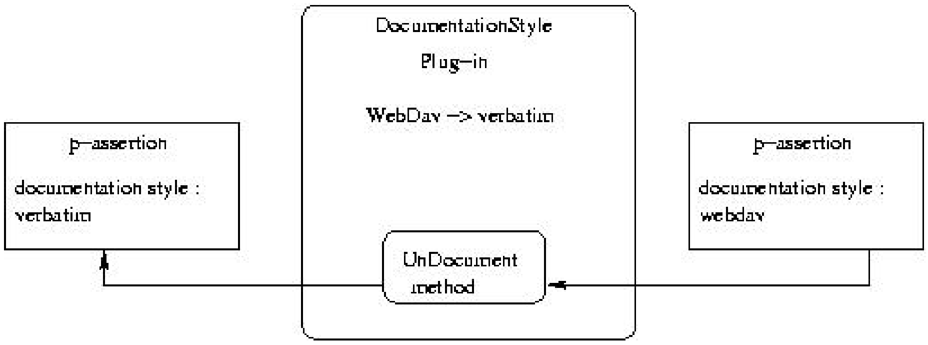


Figure 4: *Example using the WebDAV plug-in*

from the DLR application, the WebDAV plug-in only needs to *undocument* p-assertion (in order to retrieve data from the WebDAV server), this is described in the figure 4. The upload of data to the WebDAV server is undertaken by the application itself.

Moreover, this architecture can also be used by the comparison tool described in D6.1.1. Consider a scenario where the same application has been executed several times. In the first instance of the execution, p-assertions have been stored using the `verbatim` documentation style. Subsequently, in the second instance of application execution, all data contained in the p-assertions have been externalized into a `WebDav` server to solve a performance issue, for instance. In that case, it is not possible to directly compare two p-assertions where one has been recorded using the `verbatim` documentation style and the other using the `WebDav` documentation style – even if they represent the same interaction during the application workflow. The solution is to undocument the `WebDav` p-assertion to an equivalent `verbatim` p-assertion. Thus, the newly undocumented p-assertion and the other one will share the same documentation style, so now they can be passed to the comparison tool.

2.2.2 Relationship to the Portal

Various issues have been identified about how the Portal would be best suited for the DLR (WP7) application. Below we have listed some of the issues that have been identified to align the Portal with the DLR application that would provide better usability for DLR application users.

- Portal and Portlet design: A portlet by definition should contain a single (or limited) functionality. Hence, one can have a portlet that accesses the PS, or one that allows overall submission statistics to be viewed. Restricting the functionality offered by a portlet allows it to be reused, and makes it implementation and deployment simpler to manage. It has been identified by discussion with DLR (as application users) that this would be the right approach to Portlet design for their scenario. Creating smaller portlets that could be reusable in various portal pages was set as an effective design goal. For example, a PS selection portlet would allow a user to select which PS should be queried or the PS to which data may be copied as part of an archiving process. Hence, the PS selection functionality can be provided by a single portlet that can then use inter-portlet communication to achieve similar results.

Within the DLR scenario two main tools have been identified as being useful. These include the submission and query tools provided by the Portal. With relation to the usage of these tools by DLR, the actors involved in submission of p-assertions are software program components – it is therefore necessary to be able to embed some of this functionality within an existing client program. Hence, this scenario restricts the main usage of a submission portlet to an API-based access, and requires a mechanism to automatically select a PS for submission. A more detailed description of the submission portlet is covered later in this section. The Query portlets however are expected to be used by human application users rather than software components, thus making use of the portal and portlet interface. Our design of the Portal server and portlets would be achieved in relation to this usage scenario.

- Dynamic PS submission preference portlet and APIs: As described above, the primary usage of the submission portlet for the DLR scenario would be based on an API-based access, as the users are mainly software program components. In order to achieve a smooth automation of submission process, some configuration parameters need to be specified as part of the setup process. The aim of the submission preference portlet and APIs is to allow an application administrator to select the PS to which the submission should be made, and also check if the PS is online before a submission is made. The PS submission preference portlet also provides the capability to select multiple PSs for submission.

  As part of the setup process, a default preference list is defined. This list would allow one to configure two levels of submission preferences. The first would be at a general level that would specify preferences for submission of p-assertions by all the software program components, and the second level would be specific to each component – and one that would overwrite the general preferences if present.

  The idea behind creating the preference list is not just to provide increased control over submission, or for automating the submission process, but also for reducing the errors that might occur due to a PS not being available (for instance). This is achieved by creating an ordered preference list to which a p-assertion should be submitted – thereby supporting both multiple PSs, and to recover from a situation where a PS is not available and automatically making a submission to the next PS on the list. Based on the current DLR requirements, it is unlikely that a p-assertion would be stored in two different PSs. But the preferred order list of PS for submission would be within the DLR scenario requirements. The PS submission portlet would complete the process by providing an acknowledgement about where the p-assertion was submitted. A detailed list of specific operations that are covered as part of the setup process is explained in detail in section 4.1.

- Management Portlets: A set of portlets have been implemented that allow the management of PS data. These portlets allow functionality such as the ability to export and import data – with one portlet for each functionality. The export and import portlets may be tied with the navigation portlets to allow exporting the data returned from a query to the PS. For security, the export and import portlets would be made available to all users of a group. A deletion portlet would only be accessible by the application manager or project manager access groups.

- Comparison and Conflict detection Portlets: These portlets have been found to be useful in the DLR application, however a full list of scenarios is still under review. Thus, an initial basic implementation of the comparison and conflict detection portlets would be supported to evaluate their usefulness in the DLR application. It is intended, that the comparison and conflict detection portlets would work with the workflow visualisation portlet and would populate the text area with the results of the analysis performed. It must be noted that in this case one or more instances of the workflow tool would need to exist for the portlets to analyze the workflow. Some of the examples from the DLR scenario for the comparison and conflict tool include: identifying if anything has changed in a workflow, to check if a workflow exists with a similar configuration, and to check if a similar setup has been undertaken in the past. Many such scenarios involve analysis of relationships between multiple p-assertions. At the setup process it would be necessary to constrain the sets of p-assertions that would be returned in response to a query, to constrain the complexity of analysis that would be required subsequently.

- Configuration file: The configuration file marks the termination of the setup protocol. There are two possibilities for the generation of such a file: one would be to generate a configuration file per actor, another would be to generate one common configuration file for all the actors. Based on the DLR scenario, both of the above options do not completely fit the requirements. A DLR application-specific solution to the configuration file generation is to create a single configuration file for all the actors (software components) hosted on a computational cluster. Also this configuration file would be available to use only by the actors that are within the bound of a particular cluster. Hence, DLR is particularly interested in constraining the scope of a configuration file with reference to a group of actors that may within a particular platform.

3 Protocol Description

Protocols are classified based on the number of participants/principals they have – referred to as actors in this document, and the task they are trying to accomplish. Most protocols consist of an interaction between an "initiator" and a "requestor", although there are also protocols that involve the use of a "third party" (a trusted central server or a directory server). The setup protocol described in this document generally consists of message exchange between two parties. At the termination of the protocol, a configuration file must be generated, submitted to an application end user or application manager, to interact with a Provenance system. The setup protocol consists of three main stages:

1. Identification of actors involved in the protocol. These actors include human users and software processes/components. A human user may be responsible for initiating interaction with a component within the Provenance system.
2. Check whether all components that must either receive p-assertions, or that can generate and submit p-assertions are on-line and ready. This evaluation will also involve checking credentials of the submitting and receiving actors. Such a check may be undertaken via the management interface of the PS, or by interaction with a security system.
3. Generate a configuration file that can be used by each actor in the system. Such a configuration file, for instance, will be used by the client side library (used by an application) to submit and receive p-assertions. The configuration file will outline which PSs to interact with. Similarly, this state will also involve the set of default portlets that should be visible to a user when connecting to the Portal Server.

The stages specified above may be decomposed into a more detailed set of states, as outlined in figure 1, and include the following:

S1 This state is activated when the application manager sends a setup request to the tool suite. The request contains a number of parameters, such as the identity of the user requesting to access the Provenance system, the software processes managed by the user which will be submitting or accessing the Provenance system (in most cases, all processes created by a single user may inherit the access rights granted to the user), the PS involved – or reference to a default PS for a particular application (resolved via the tool suite), reference to a Portal server, the types of documentation styles needed, and reference to any external data stores that may be involved in the submission or access of a p-assertion. An error is generated if the server hosting the tool suite administrator process does not respond to a request generated by the application manager. This error is returned to the application manager and the protocol terminates.

S2 This state is activated just after the reception of the setup request to define the PS to use and to check whether the PS is on-line. This state addresses the requirement SR-1-10 identified in the document D2.2.1. There may be multiple possible PSs involved, requiring each one to be verified separately. If a particular PS does not respond within a timeout period (set by the tool suite administrator), an error code is logged, along with the name of the PS that did not respond. By default, only a single attempt is made to connect to the PS. However, the protocol may be configured to allow a number of retries before the PS is considered to be unreachable. During this stage, it is also necessary to verify how much storage space has been allocated to a particular user on a particular PS.

In the first instance, it is expected that there is a single PS, as indicated in the figure. However, it is possible to repeat this state for multiple PSs as indicated in state S2b.

S3 The credentials of the user or process making the request are verified. This involves interaction with the management interface at the PS, or a trusted third party system. Credential checking may be undertaken at the user level, or it may be undertaken at the process level. Once a user identity has been confirmed, this is then used to determine the level of access that should be granted to a particular user (or a particular category of user – user categories include, for instance: "user" or "administrator" and are application specific). An error is returned if the user cannot be verified, or if the credential checking server does not respond. The error code, along with the string identifying the error are logged.

S4 Check that the Portal server is on-line and ready to receive requests. It is possible that the Portal server is hosted on the same machine as the tool suite. In this case, this test primarily checks that a Portal server process exists and is contactable. A limitation with checking a co-located process is that it would not be able to verify the network connectivity between an external user and the Portal server. Therefore, any issues associated with a firewall or a proxy server-based access cannot be verified.

S5 Check that the default portlets that should be made available to a particular category of user are provided on the Portal server. Mapping of user name to category is maintained in a look-up table managed by the tool suite administrator – the same information as in state S3. This mapping is initially defined by the Provenance system administrator or the application manager. An error is generated if a given portlet cannot be found – achieved by undertaking a syntax check between the name of the portlet associated with a particular user category and all portlets that have been registered with the Portal server.

S6 Check that plug-ins that may be associated with a documentation style are installed and available for use. Such a check may involve verifying the existence of plug-ins within the local file system, or checking their existence via a URI. In both cases, an error is returned if a plug-in cannot be found, along with the name of the plug-in that led to the error. However, it may be impossible to check some plug-ins during the setup for several reasons like the unavailability of the plug-in on the current location, a limited access to remote server based on IP addresses, credentials kept secret or unknown at setup time. For all these reasons, a second check will occur during an actors' configuration at the *configuration* stage. The `webdav` plug-in described in section 4, page 16 is an example of such a plug-in.

S7 Check that any external data stores, some of which may contain data that needs to be accessed via the plug-ins, are available and on-line. A credential check could also be undertaken with each data store to ensure that a user has access rights to the data store. An error is returned if the data store is not on-line, or the user does have access privileges on the data store.

S8 An `ID` is generated to uniquely identify this particular execution of the application. It is composed of two different IDs: an *application ID* to uniquely identify the application among all other applications, and an *execution ID* to uniquely identify a particular execution of the application. It may be used in subsequent submissions made by a user for the same application. The uniqueness of the ID should be assured among all PSs used by the application. This could be achieved by enforcing the use of a particular ID generator service for all applications using a given set of PSs.

S9 This state involves the configuration of the Portal server, to ensure that user credentials have been mapped to the appropriate role – as required in the eXo Portal Framework [1]. An error is generated if a mapping of user credentials to role does not succeed.

T The protocol terminates with the creation of a configuration file which may subsequently be used by the application client API. An error is generated if there is not enough file space available to store the file on the tool suite server or the server hosting the application.

Figure 5 provides one instance of the setup protocol, initiated by the `register(p)` message from the application manager to the tool suite. It should be noted that the protocol is intended to be asynchronous. Essentially, the application administrator may launch a setup request and would need to be blocked. Termination of the initiated process would occur either as a result of an error, or with the creation of a configuration file. The following request and response messages are supported in the protocol:

Online(server_type) a request message to verify if `server_type` server is online and ready to accept further requests.

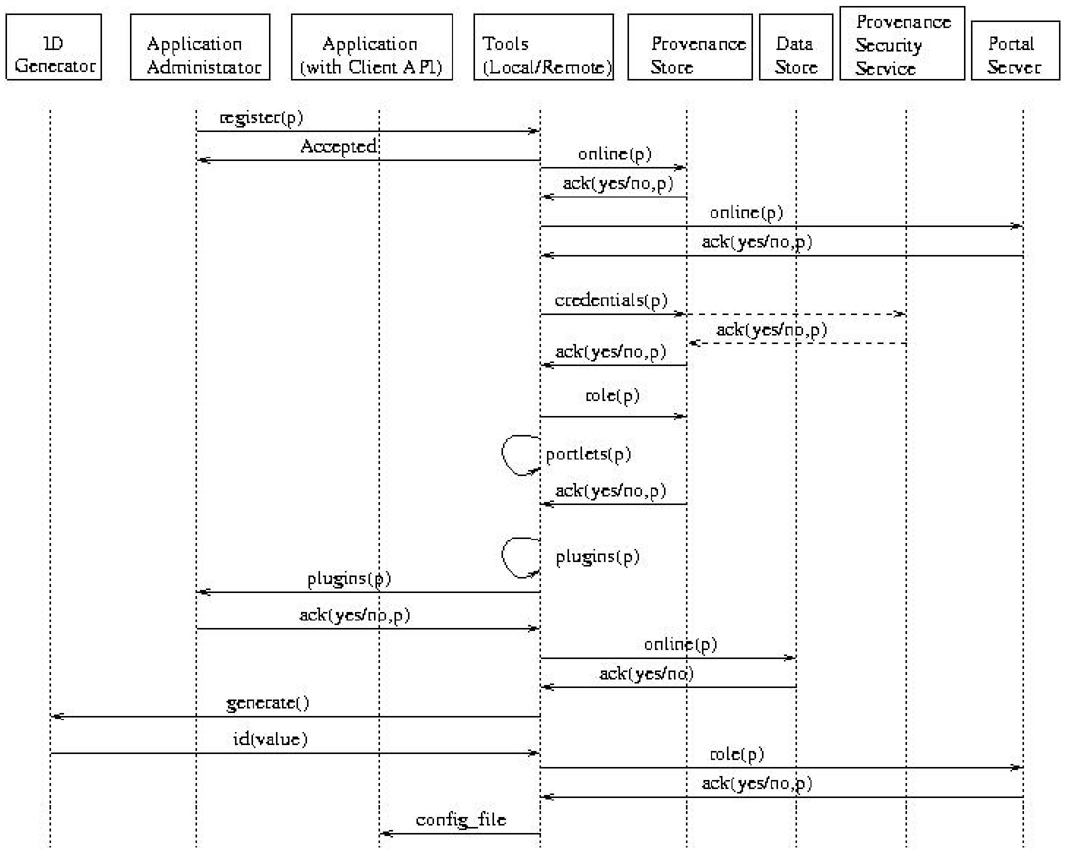


Figure 5: *Setup Protocol – message order*

Register(p) a request message to initiate the setup protocol. Here (`p`) represents an object containing parameters that are used by subsequent stages of the protocol. The parameters must be specified by the application manager.

Credentials(p) a request message used to verify that a particular user, process or role can be verified with an external server. Here (`p`) represents an object encoding properties that may be used to confirm user identity.

Portlets(p) a request message used to verify that portlets that have been identified by `p` exist on a given server.

Plugin(p) a request message used to verify that plug-in `p` exists on a given server (or the local file system).

Generate() a request message used to generate a unique identified for a given application instance. The software that provides this identifier may be co-located (on the same server) to the tool suite or accessed remotely.

Role(p) a request message used to check if a role, defined by parameter `p` can be found on a given server.

ID(value) a response message that contains a numeric `value` containing a unique identifier.

Accepted a response message that indicates that a Register(p) request has been accepted and is currently being processed. This message primarily contains a status code that is returned to a request initiator.

Ack(value,p) a response message sent as an acknowledgement to a request message – with value being a Boolean variable set to `yes` or `no`. Here (`p`) represents an object containing parameters that are used to reference parameters passed in the initial request – `p` may be a null object.

There is a need for a delegation of identity/access control between different components that make up the Provenance system. For instance, ideally a user connecting to the PS should not require a new identity/credential checking mechanism to connect to the Portal server, or any external server that may contain data associated with a p-assertion. This is being investigated in WP4, and will be adopted in a subsequent version of the protocol.

3.1 Identifying Actors

The first state of the setup protocol involves determining which actors will be involved in an interaction. The actors may be: application managers, provenance store managers, end users or portal administrators. An application manager will be a member of the system administration team or a specialist IT expert responsible for implementing or managing software applications at an end user site. 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. During the setup protocol, such individuals will be responsible for identifying:

- The location and types of PS that would be needed for submitting and accessing p-assertions. The location may be specified using a URI and "type" may include local/remote.
- The "documentation style" that would be required by a user within a particular application. A documentation style identifies the need for specialist plug-ins that may be necessary to view data associated with a particular p-assertion. A documentation style may involve the use of specialist plug-ins that are only relevant for a particular application, or may use commercial/shareware plug-ins – such as a PDF viewer – that are commonly accessible. The application manager is therefore responsible for specifying how such a plug-in can be located, and how a plug-in may be associated with the client side library made available to an application end user.
- Specify how the identity of application end users/software processes is defined. This depends on the security system being employed locally, or if an external security system (provided by the Provenance system) is to be used to check the credentials of an end user. Examples include the use of an LDAP-based string to identify a user – such a string may be associated with a digital certificate, the use of a username/password pair etc.
- Specify a policy that may be used to grant particular privileges to a user. We envision the need for two types of policies: (1) policies related to the tool suite; (2) policy related to the use of data associated with a particular application. We focus here on policies related to the tool suite – further discussion can be found in section 4.

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 or retrieve a p-assertion from a PS.

It is useful to distinguish between human actors and software process groups involved in generating and receiving a particular message. Such process groups may include: a process that submits a p-assertion, a process that receives a p-assertion (either through the tools or directly from the PS), the manager process responsible for the "Management Interface" at the PS, a "notified" process that has subscribed to listen to a particular event. The manager process and the notified process may interact, for instance if some modification is made to the PS, or a particular type of p-assertion is submitted to the PS.

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

3.2 Properties of the Setup Protocol

The setup protocol is an "application level protocol" designed to adhere to the following properties:

- Stateless: the order of message exchange in the protocol is not significant – hence it is possible to add additional states (if new checks are necessary) to the protocol described in figure 1.
- Terminates: the protocol always terminates in a finite period. Either an error is generated (and logged), or a configuration file is produced.
- Idempotence: this indicates that the the protocol has the same effect if used multiple times as it does when used only once. The use of multiple uses of the setup protocol may be necessary if modifications are made to the access privileges assigned to a particular user.
- Asynchronous: the protocol is asynchronous, which indicates that an application administrator may launch a request message and not have to block waiting for a configuration file to be produced.
- Safe: this indicates that the setup protocol cannot end up in an indeterminate state. The error checking mechanism ensures that a default state (indicated as the "Terminate" state in figure 1) is reached if no progress can be made within the setup process.

3.3 Status Codes and Error Management

During the setup protocol, response messages will be returned to the initiator of an activity. Such response messages are generally referred to as "Status Codes". There may also be a number of error messages generated as a consequence of a request to various components of the Provenance system. We identify error codes that may be associated with each stage of the protocol, and these may be used by an application manager or logged at the tool suite for analysis by the Provenance system administrator. As the setup protocol is initiated by an application manager, the first error message can arise if no interaction can be achieved with the tool suite (corresponding to S1 in figure 1).

3.4 Status: Success Messages

These codes indicate that a successful operation has been performed. The body section, if present, will contain an identity string sent with the request. If MIME format is being supported, only `text/plain` content-type will be supported. All success codes have the general format `2xx` (similar to the HTTP protocol):

OK 200 indicates that the request was successfully completed.

Accepted 202 indicates that a request has been accepted for processing, but the processing has not yet been completed. This request allows the tool suite to schedule a request made by an application manager for handling at a later time. No guarantees are provided that this request will be processed within a particular time period. The use of this message provides an asynchronous interaction between the tool suite and the application manager process.

No Response 204 indicates that no configuration file can be generated due to an error at the tool suite. This message specifies that although a request was received, it is not possible to totally fulfill this. This is the default message that is returned to the application manager if no configuration file can be built. A **No Response** message may be sent only after an **Accepted** message.

3.5 Status: Error Messages

As the tool suite coordinates the setup process, all error messages are logged at the tool suite. An application manager or the Provenance system administrator may request to view this log. Error codes use the format `4xx` (similar to client error codes in HTTP). The body of the message may contain an optional string that describes, in a human readable form, the basis for the error. If used with MIME format only `text/plain` content-type is supported. The following error messages are supported:

Bad Request 400 indicates that the request had incorrect syntax or could not be parsed effectively.

Credentials 401 indicates that a user identity cannot be verified based on information stored on a server. A credential check may be performed at the PS or at the Portal server.

Role 402 indicates that although a user credential has been confirmed, the user role cannot be verified based on information held at the server.

Forbidden 403 indicates that although a user credential has been confirmed, such a user does not have access to the requested resource. This could include a particular user requesting write privilege at a PS, or attempting to access a portlet on the Portal server.

Unavailable 404 indicates that the tool suite has reached a threshold number of requests, and cannot fulfill any additional requests at this stage. This message is generated by the tool suite to the application manager.

Plugin 405 indicates that a particular plug-in cannot be located. This error will also contain a string specifying the name of the plug-in that cannot be found. A separate error will be generated for each plug-in.

Portlet 406 indicates that a particular portlet cannot be located on the Portal server. This error will also contain a string specifying the name of the portlet that cannot be found. A separate error will be generated for each portlet. It is possible for portlets to be provided by remote servers – i.e. a server other than the one hosting the Portal framework. In the first instance, a user should treat all portlets are being co-located at the portlet server. If remote portlets are to be used, these should be independently verified by the application end user or the application manager.

No ID 407 indicates that a new identifier cannot be generated. The ID generator may be a process external to the tool suite – therefore the error will also contain a string `local` or `remote`.

No Space 408 indicates that not enough storage space is available to perform the requested operation. Examples include: (i) the creation of a configuration file was attempted on the tool suite or the application server, but could not be achieved due to lack of available file space; (ii) not enough space is available on the file system at the PS.

Internal Error 499 indicates that an unexpected condition has arisen on the server hosting the tool suite, and no further requests can be fulfilled. When this error is generated, it is necessary for the Provenance system administrator to take corrective action. It will be the default message when a request cannot be fulfilled, and no other error conditions are valid.

4 Tool Interaction

In this section we describe how the setup protocol interacts with the tool suite – especially the analysis engine, the navigation tool and the Portal server. Interaction between the setup protocol and the tool suite relies on the use of a policy – which needs to be specified by the application manager.

4.1 Portal

In this section, a description of setting up the Portal server, supporting access to various portlets, identifying which portlets are available for use by a particular user, is provided.

4.1.1 Setup Portal server

The operations undertaken by the Portal server as part of the setup process is described below. These operations are divided into various steps, each of which will be covered below:

- Authorization: The aim of this operation is to allow a user to authenticate with the eXo Portal server. This would be achieved by creating a credential object by the user.

- Group Membership: The aim of this operation is to setup the credentials for viewing and editing portal pages and access to the navigation capability provided by the portal. At present status of the implementation, such a restriction is managed by either allowing only the owner or all users within a particular group to access portlets. This form of management would however be replaced with a checking mechanism that would verify if a user has the correct membership in a group before a request for a given resource by that user is processed. By doing this, certain resources can only be made accessible by the user that have correct membership in a group. For example, when a user accesses the portal registry, it is necessary for the user to have the right type of group membership in order for him to add a new portlet to a portal page.

- Community Group: As discussed before, one stage of the setup process would include attaching application users to a particular group. This allows a useful mechanism for user authentication. A further extension to this setup is achieved by the introduction of a "community" group. Using the community group, a set of portal pages or a template that consists some default portlets – such as header or footer portlet for example – can be made available to a set of users that belong to a particular community. By doing so, a list of portlets can be made available automatically to all the users of a particular community.

  The setup configuration at this time can only be initiated by a user that belongs to the administrator group. A more practical approach however would be to allow one or more users from the community group to modify these preferences. This functionality would be provided in the future using the setup process implementation.

  Community groups in the DLR scenario may include the set of projects making use of the TENT framework. Each project will be provided with its own enforced homepage and a set of template portal pages. Members of a project manager group would be able to modify the portal preferences and perform the setup of the portal pages for particular project users.

- Submission Portlet Preference: The submission portlet could make a submission to multiple PS. A important configuration as part of this setup process would be to use the same interaction id for the p-assertion submitted to multiple PS.

- Portal configuration: The aim of this operation is to allow application users to have access to previously configured portlets that are ready to use. This process involves all the steps that are required in order for a user to start making use of the portlets provided by the group they belong to. This process involves steps starting from creation of a portlet category to store all the portlets in, applying access configuration so portlets are configured to be accessible by different user groups and finally to bind the default portlets to a group so all the default portlets are added to a users portal page automatically.

- User space: As part of the setup process, it is necessary to provide a user profile that contains all the relevant data associated with a given application user – such a profile must be defined by an application administrator and may be common to all users within a particular application domain. Some examples of the kind of relevant data that can be stored are information on past queries that the user has run, the PSs that have been used, visualization preferences, etc.

4.1.2 Accessing the Portal and Portlets

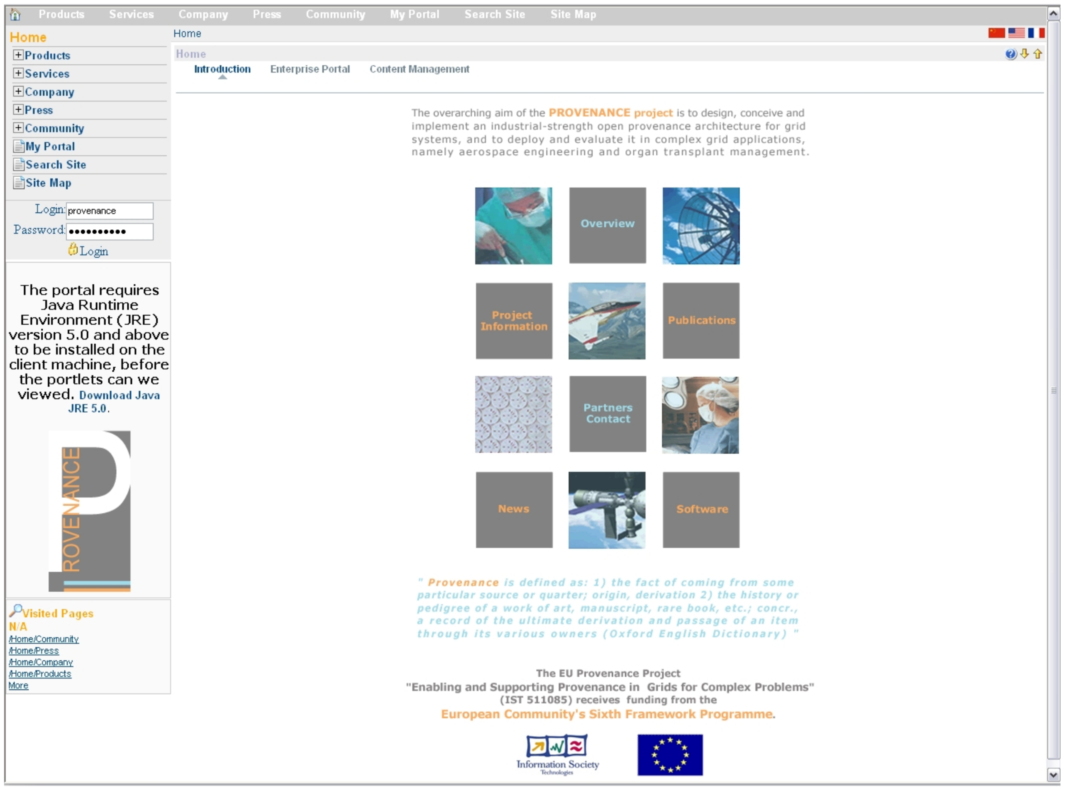
The steps undertaken by an application user to make use of the tools provided by the Portal server are explained using the help of screenshots. All portlets are deployed using the eXo portal [1, 2] framework and complies with the JSR168 Portlet specification [4]. The first portal page that an application user encounters is the login page as seen in figure 6. The task of the login page is to establish the credentials of the application user trying to connect to the Portal server. The credentials are checked using a username/password pair for now as seen from figure 7. In the future, it is intended that this credential check is mapped to the approach adopted in WP4. This page also allows application users to set language preferences – to enable instructions for use to be specified in different European languages. Implementation of the language preference option is currently limited and only acts as a placeholder which would in future be extended to cover portlet descriptions. The language option at current support Chinese and French as seen from figure 8 and figure 9 respectively. Once the application user has successfully entered the username and password, this will now enable the user to interact with the Portal server.

4.1.3 Interaction with the Portlets

The interaction between the application user and various portlets provided by the Portal framework is described below using the help of screenshots. The first portal page that is viewed by the application user once logged in is the users homepage. This is shown in figure 10 – the homepage provides a navigation menu with links to different tools and user related information. We discuss each of the tools in further detail below:

Once an application user selects the navigation tool link on the menu, the portal page containing the PS query tool portlet and the PS tree navigation tool portlet is displayed as seen in figure 11. The PS query tool accepts Xpath queries on the PS and displays the result. The PS tree navigation portlet displays the content of the PS in a tree form to help visualize the contents of the PS.

Once an application user selects the workflow tool, the portal page in figure 12 is displayed. The workflow tool page contains portlets such as workflow navigation tool, Java web start navigation tool, get provenance trace portlet and a JESS portlet. All of these portlets are accessible using the tab links on the workflow tool page as seen in figure 12. The workflow navigation tool portlet is used to reconstruct a workflow and display it visually. The workflow graph



Login: provenance
Password: ••••••••••

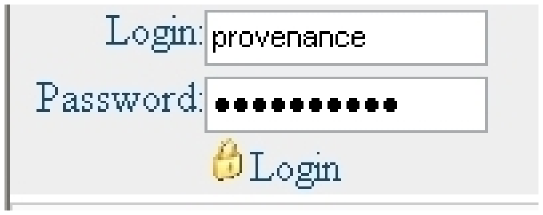
The portal requires Java Runtime Environment (JRE) version 5.0 and above to be installed on the client machine, before the portlets can we viewed. Download Java JRE 5.0.

The overarching aim of the PROVENANCE project is to design, conceive and implement an industrial-strength open provenance architecture for grid systems, and to deploy and evaluate it in complex grid applications, namely aerospace engineering and organ transplant management.

" Provenance is defined as: 1) the fact of coming from some particular source or quarter; origin, derivation 2) the history or pedigree of a work of art, manuscript, rare book, etc.; concr., a record of the ultimate derivation and passage of an item through its various owners (Oxford English Dictionary) "

The EU Provenance Project
"Enabling and Supporting Provenance in Grids for Complex Problems"
(IST 511085) receives funding from the
European Community's Sixth Framework Programme.

Figure 6: *Login Page for Provenance Portal*



Login: provenance
Password: •••••••••••

Figure 7: *Credential check in Login Page of Provenance Portal*

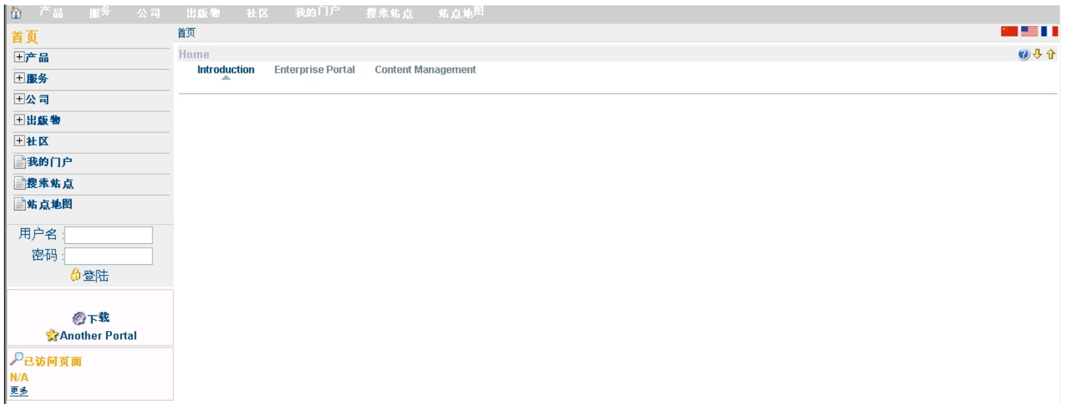
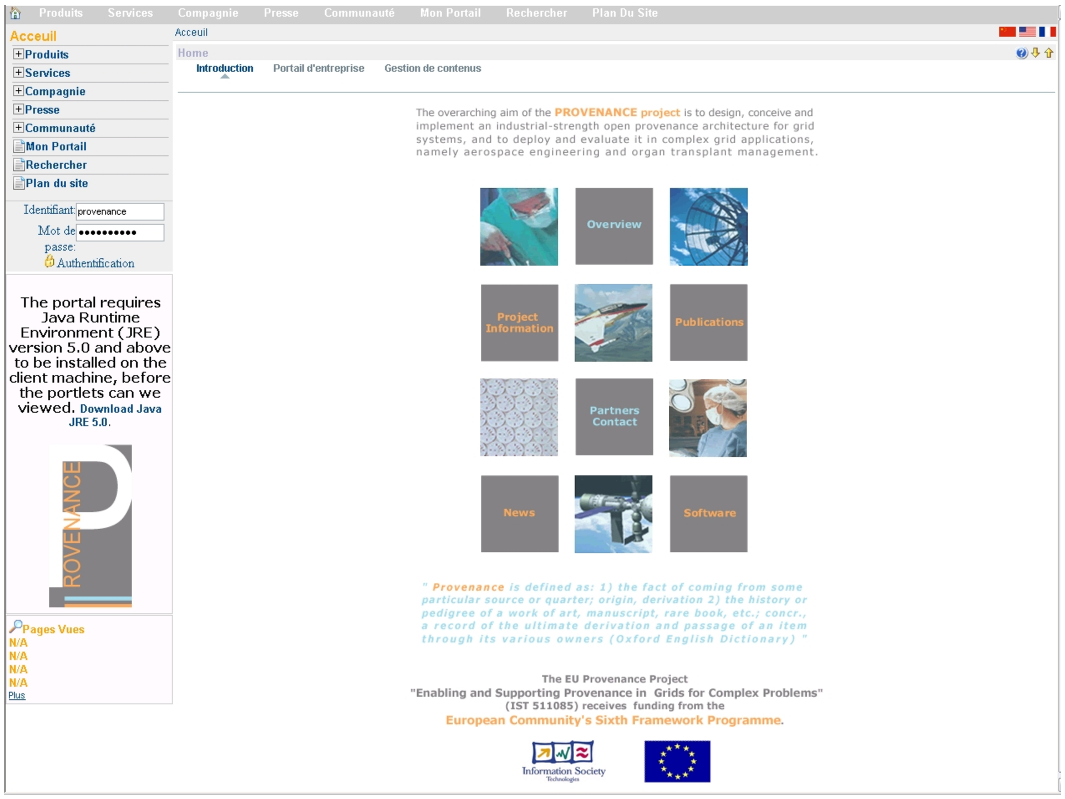


Figure 8: *Login page seen using Chinese language option*



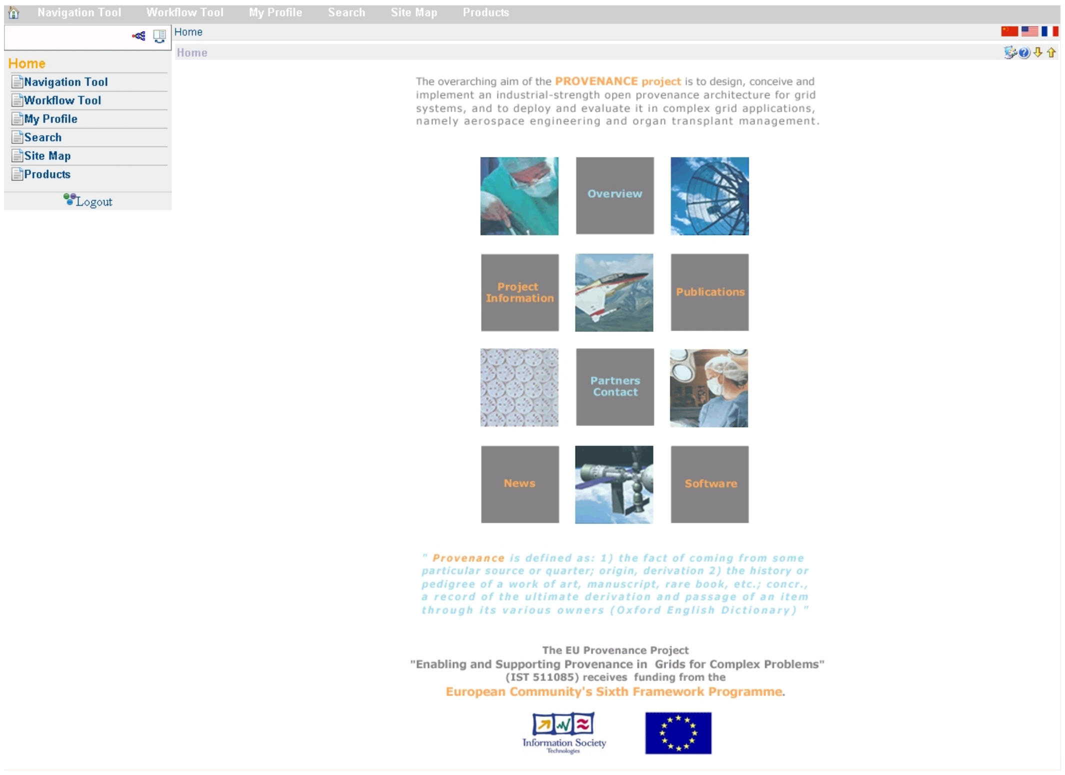
The portal requires Java Runtime Environment (JRE) version 5.0 and above to be installed on the client machine, before the portlets can we viewed. Download Java JRE 5.0.

The overarching aim of the PROVENANCE project is to design, conceive and implement an industrial-strength open provenance architecture for grid systems, and to deploy and evaluate it in complex grid applications, namely aerospace engineering and organ transplant management.

" *Provenance is defined as: 1) the fact of coming from some particular source or quarter; origin, derivation 2) the history or pedigree of a work of art, manuscript, rare book, etc.; concr., a record of the ultimate derivation and passage of an item through its various owners (Oxford English Dictionary)* "

The EU Provenance Project
"Enabling and Supporting Provenance in Grids for Complex Problems"
(IST 511085) receives funding from the
European Community's Sixth Framework Programme.

Figure 9: *Login page seen using French language option*

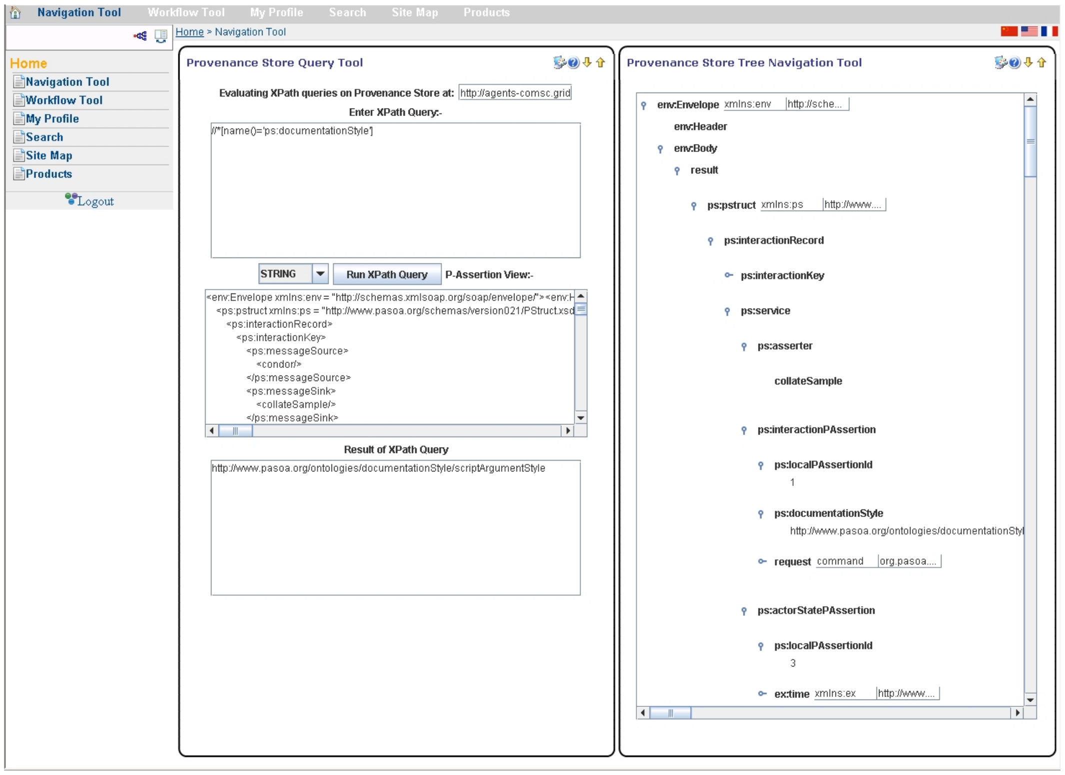


The overarching aim of the **PROVENANCE project** is to design, conceive and implement an industrial-strength open provenance architecture for grid systems, and to deploy and evaluate it in complex grid applications, namely aerospace engineering and organ transplant management.

*" **Provenance** is defined as: 1) the fact of coming from some particular source or quarter; origin, derivation 2) the history or pedigree of a work of art, manuscript, rare book, etc.; concr., a record of the ultimate derivation and passage of an item through its various owners (Oxford English Dictionary) "*

The EU Provenance Project
"Enabling and Supporting Provenance in Grids for Complex Problems"
(IST 511085) receives funding from the
European Community's Sixth Framework Programme.

Figure 10: *Homepage of a application user on the Portal*



Evaluating XPath queries on Provenance Store at: http://agents-comsc.grid

Enter XPath Query:-

```
//*[name()='ps:documentationStyle']
```

STRING

P-Assertion View:-

```
<env:Envelope xmlns:env = "http://schemas.xmlsoap.org/soap/envelope/"><env:H
  <ps:pstruct xmlns:ps = "http://www.pasoa.org/schemas/version021/PStruct.xsd
    <ps:interactionRecord>
      <ps:interactionKey>
        <ps:messageSource>
          <condor/>
        </ps:messageSource>
        <ps:messageSink>
          <collateSample/>
        </ps:messageSink>
```

Result of XPath Query

```
http://www.pasoa.org/ontologies/documentationStyle/scriptArgumentStyle
```

```
env:Envelope xmlns:env http://sche...
  env:Header
  env:Body
    result
      ps:pstruct xmlns:ps http://www....
        ps:interactionRecord
          ps:interactionKey
          ps:service
            ps:asserter
              collateSample
            ps:interactionPAssertion
              ps:localPAssertionId
                1
              ps:documentationStyle
                http://www.pasoa.org/ontologies/documentationStyl
              request command org.pasoa....
            ps:actorStatePAssertion
              ps:localPAssertionId
                3
              ex:time xmlns:ex http://www....
```

Figure 11: *Navigation tool containing PS query and tree navigation portlets*

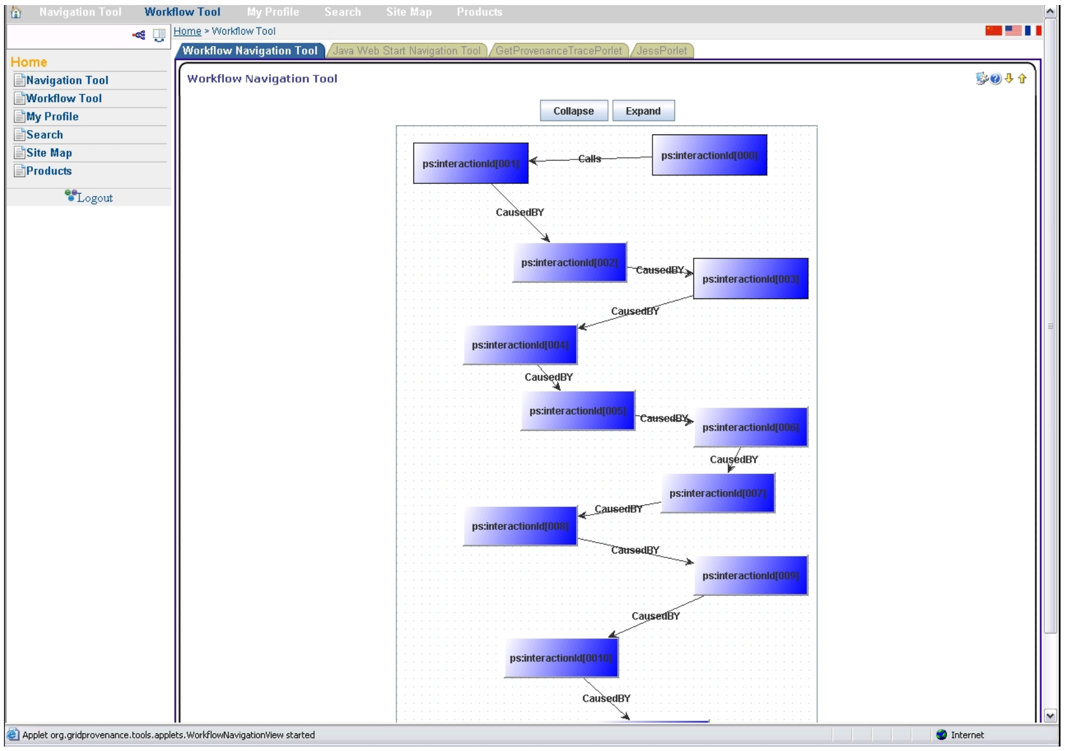
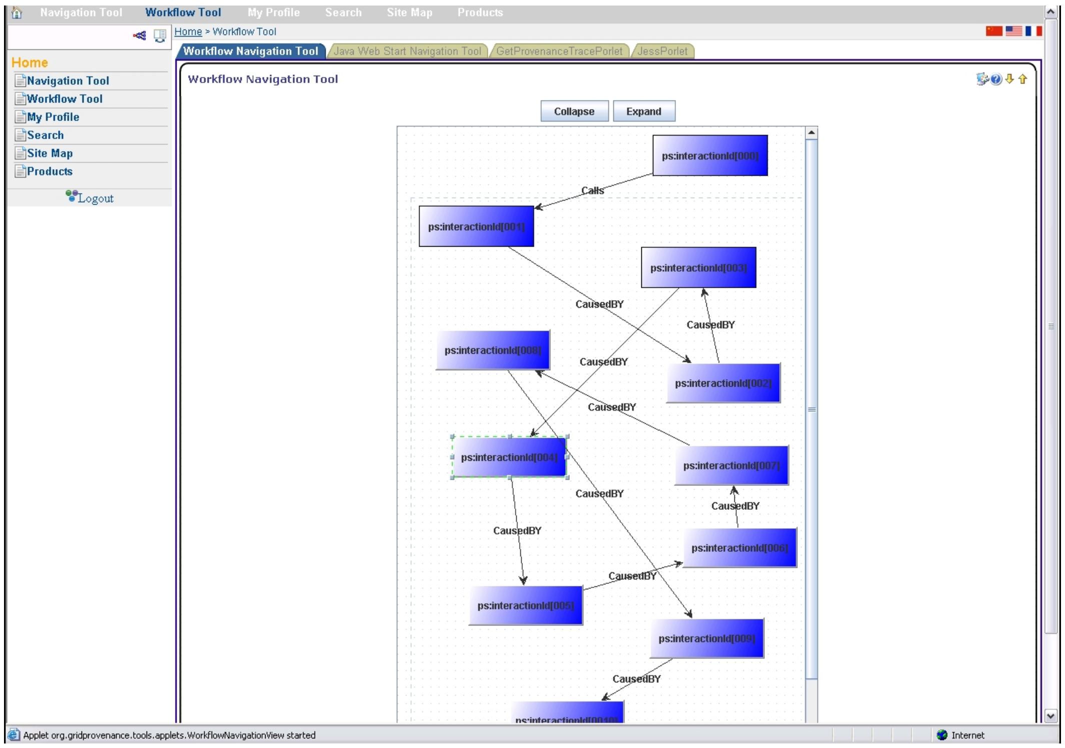


Figure 12: *Workflow tool portal page*

was created using the help of Jgraph tool [3]. The visualization option in the workflow tool portlet allows for drag and drop of different nodes in different work areas available on the workflow tool portlet window. This functionality allows a user to visualize the workflow in different manners. One instance of a re-arranged workflow is displayed in figure 13. The workflow tool also allows a user to group certain nodes and be able to collapse or expand them in order to get a better view of other nodes. This is especially useful in case of hundreds of node existing in a small display area. An example of a set of nodes collapsed in this fashion is displayed in figure 14.

An application user can add or remove portlets from the users portal page. The first step to adding or removing a portlet is to click the edit page mode link as shown in figure 15. Once in the edit page mode, a set of extra icons appear on the right side of the portlet container window as seen from figure 16. The 'cross' sign is used to delete either an individual portlet or to delete all the portlets that exist within a container. The 'three plus' sign in an inverse triangle form is used for adding new portlets to the container as seen from figure 16. Once an add portlet link is clicked, a portal registry page is displayed to the user by the portal server as seen in figure 17. The portal registry displays a list of all the portlets that are registered with the portal server and not just portlets that



Applet org.gridprovenance.tools.applets.WorkflowNavigationView started

Figure 13: *Workflow navigation tool re-arranged for different visualisation preference*

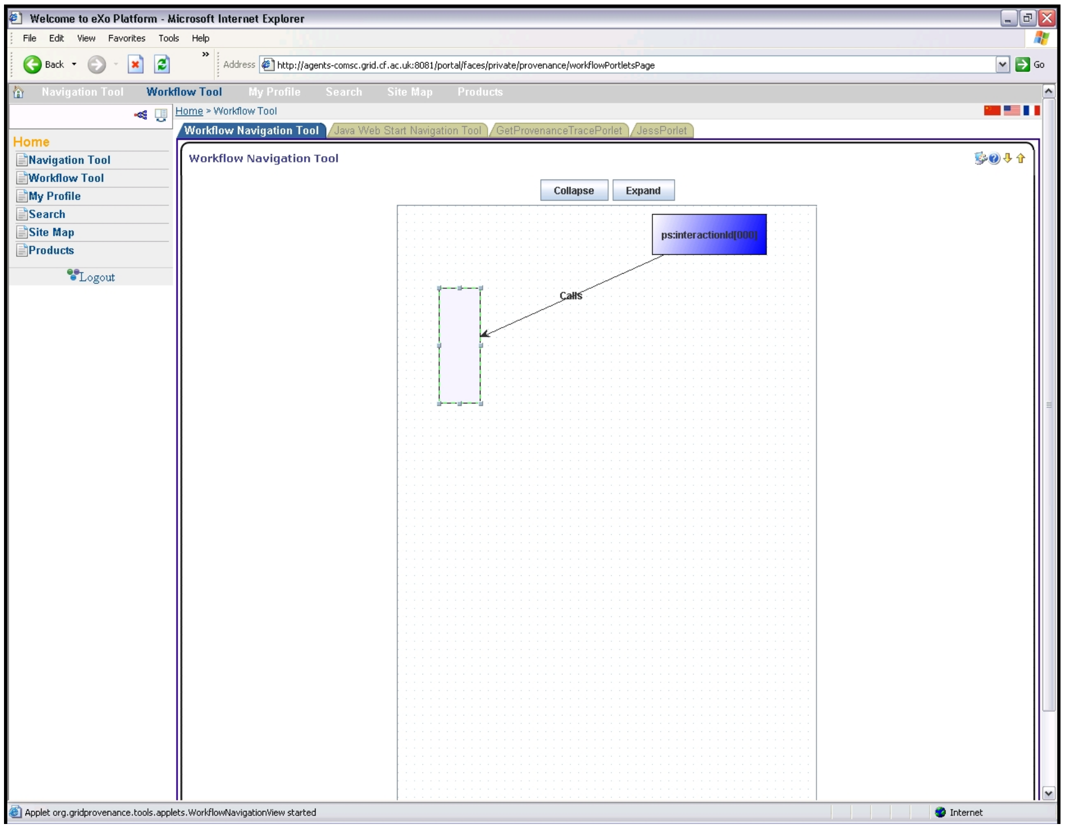

Welcome to eXo Platform - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address http://agents-comsc.grid.cf.ac.uk:8081/portal/faces/private/provenance/workflowPortletsPage
Go
Navigation Tool
Workflow Tool
My Profile
Search
Site Map
Products
Home > Workflow Tool
Home
Navigation Tool
Workflow Tool
My Profile
Search
Site Map
Products
Logout
Workflow Navigation Tool
Java Web Start Navigation Tool
GetProvenanceTracePorlet
JessPorlet
Workflow Navigation Tool
Collapse
Expand
ps:interactionId[000]
Calls
Applet org.gridprovenance.tools.applets.WorkflowNavigationView started
Internet

Figure 14: *Workflow navigation tool with group of nodes collapsed*

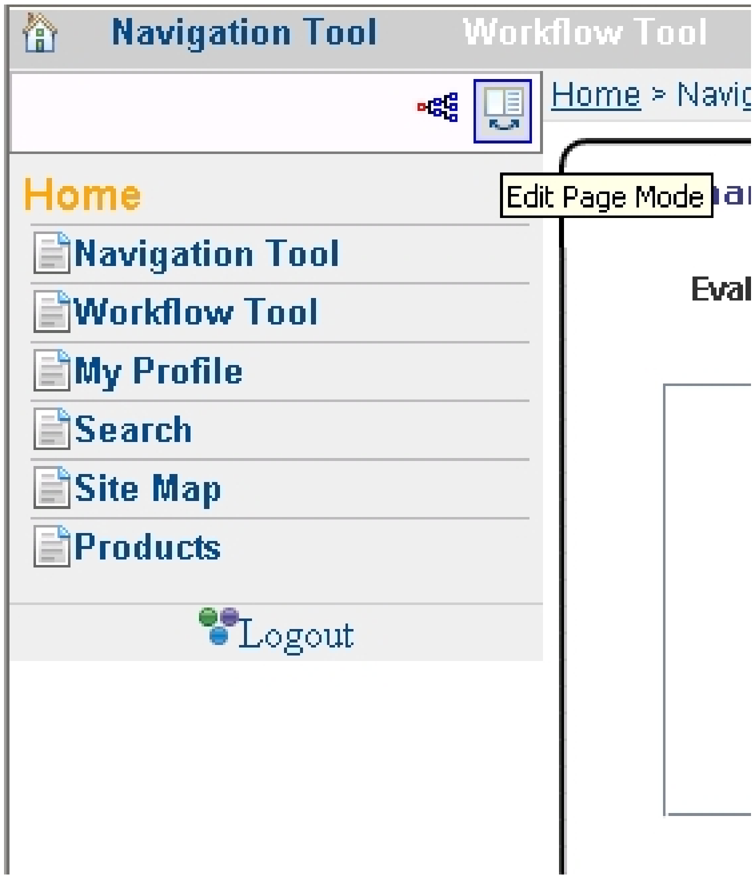


Figure 15: *Edit page link*

the user has access to. An application user can select any of the portlets from the registry. An instance with the user trying to add the workflow navigation portlet is displayed in figure 18. Once a user has accepted the selection, the new portlet is added to the portal page of the user as shown in figure 19. It can be noticed from figure 19 that the workflow navigation portlet is now added on the navigation tool page.

4.1.4 Customisation using the Portal server

The Portal facilitates application users to find information and access to portal contents quickly through customisation and personalisation of the portal content and layout. Some of the customisation feature available to the application users are explained below:

Firstly, container renderer on a portal page: How the portlets are rendered on the portal pages that are binded to one container can be customised by the

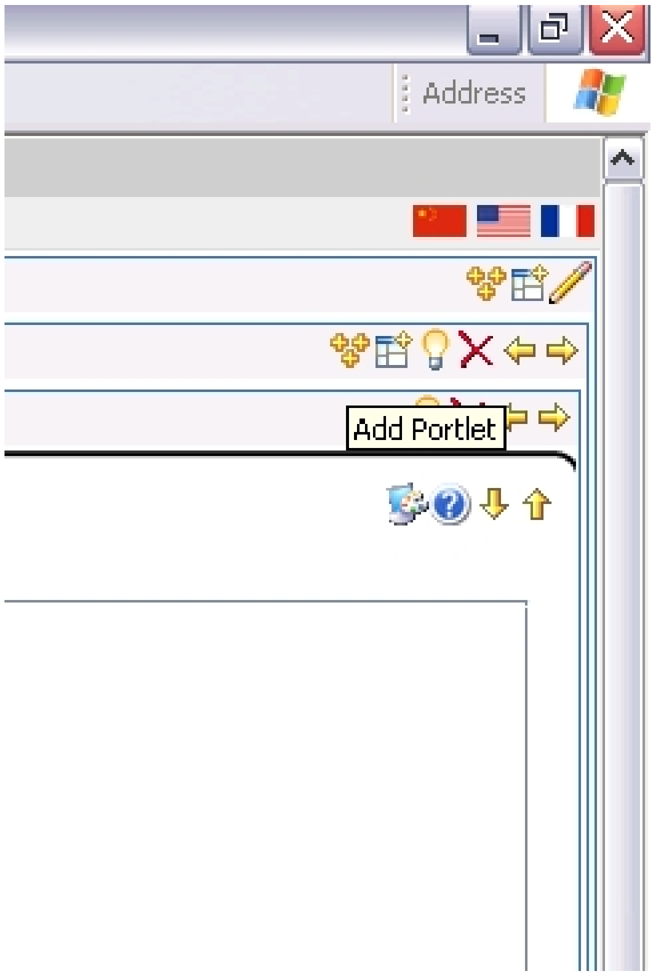
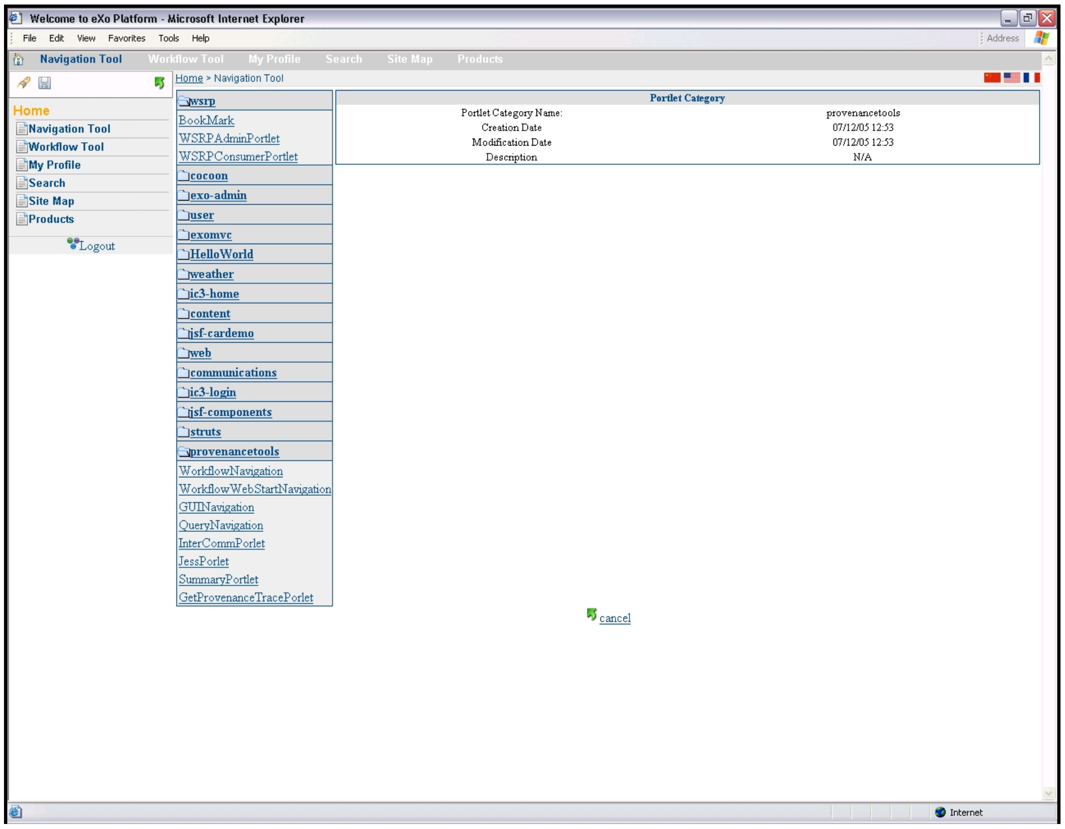


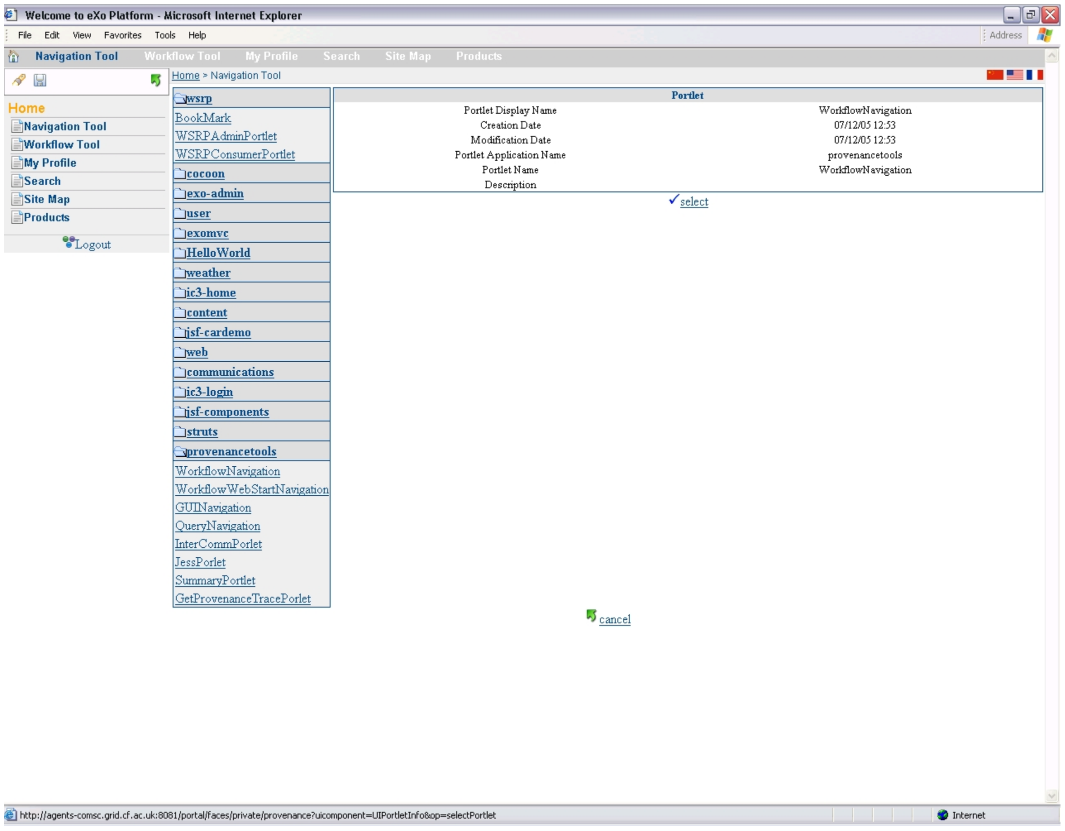
Figure 16: *Add portlet and delete portlet icons*



Portlet Category

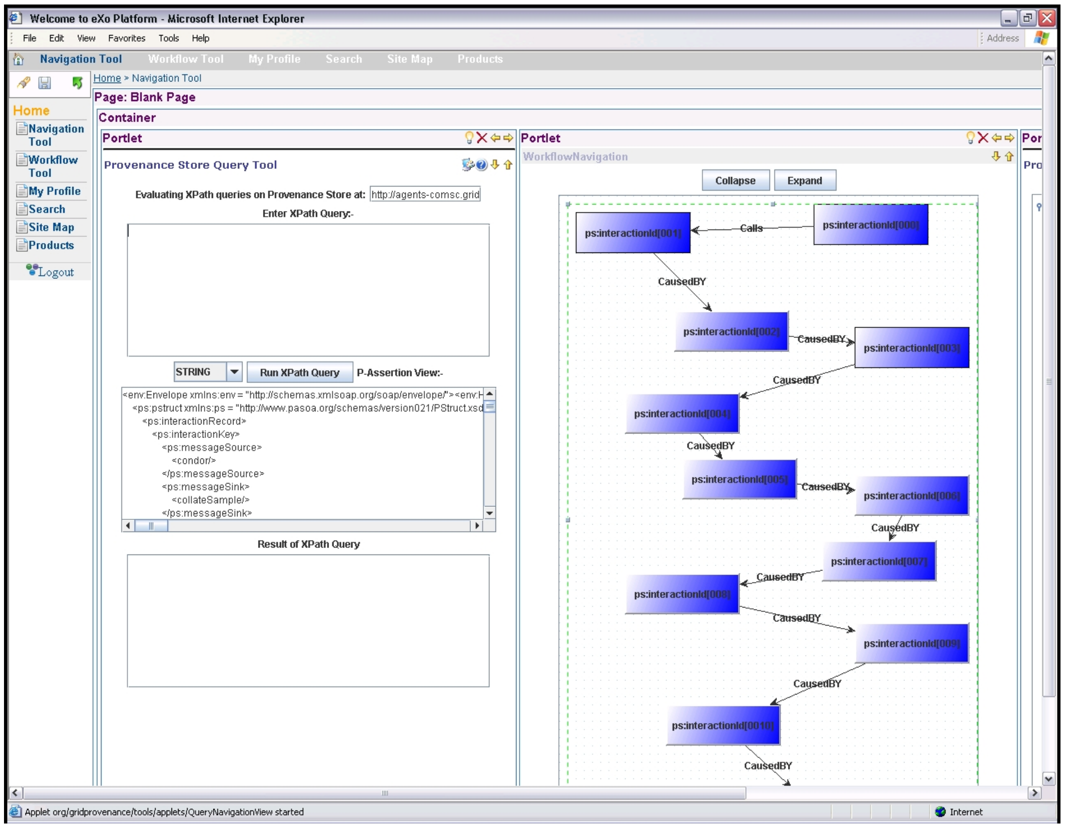
Portlet Category Name:	provenancetools
Creation Date	07/12/05 12:53
Modification Date	07/12/05 12:53
Description	N/A

Figure 17: *Portlet registry*



Portlet	
Portlet Display Name	WorkflowNavigation
Creation Date	07/12/05 12:53
Modification Date	07/12/05 12:53
Portlet Application Name	provenancetools
Portlet Name	WorkflowNavigation
Description	

Figure 18: *Selecting workflow navigation portlet to be added to the portal page*



Evaluating XPath queries on Provenance Store at: http://agents-comsc.grid

```
<env:Envelope xmlns:env = "http://schemas.xmlsoap.org/soap/envelope/"><env:H
  <ps:pstruct xmlns:ps = "http://www.pasoa.org/schemas/version021/PStruct.xsd
    <ps:interactionRecord>
      <ps:interactionKey>
        <ps:messageSource>
          <condor/>
        </ps:messageSource>
        <ps:messageSink>
          <collateSample/>
        </ps:messageSink>
```

Figure 19: *Workflow navigation portlet added to the navigation tool page*

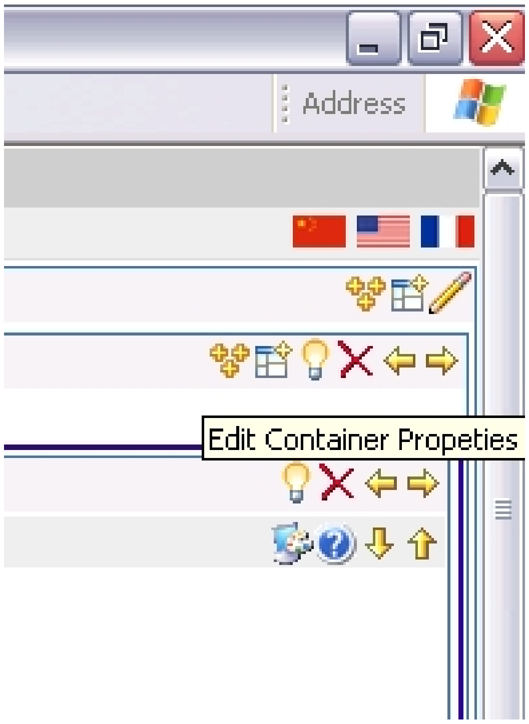
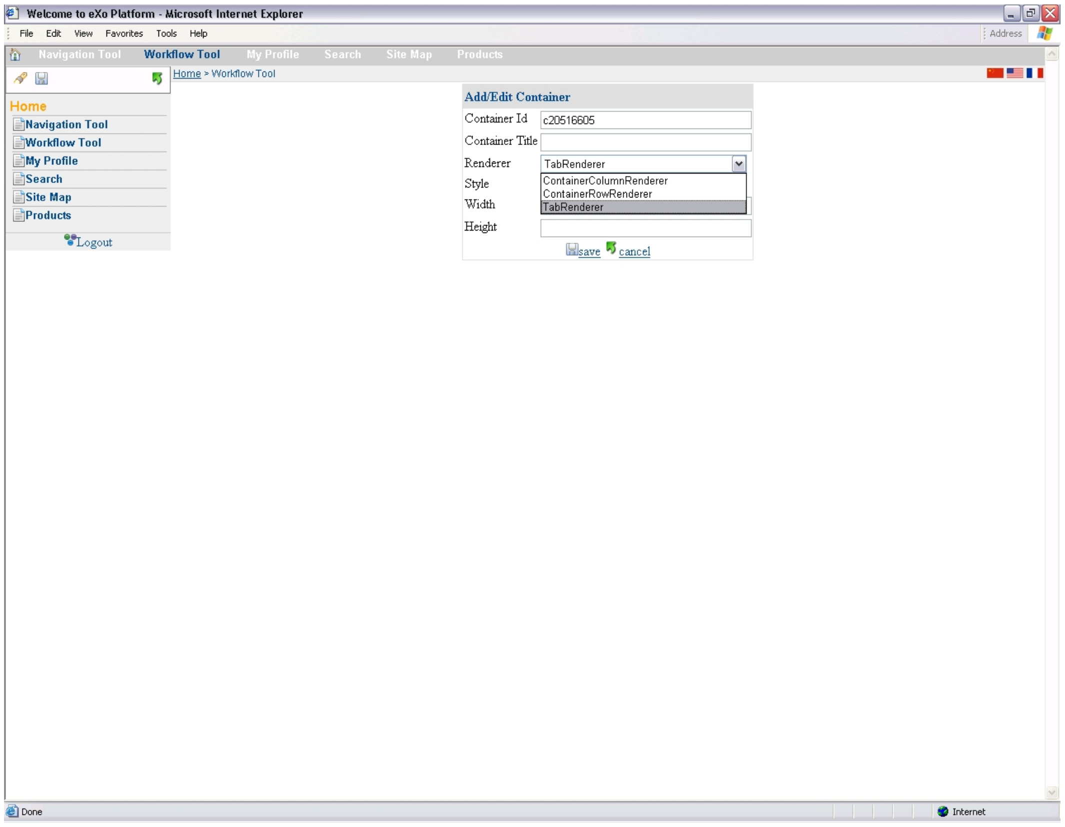


Figure 20: *Edit container link*

application users. This feature dictates how the portlets would appear on a portal page. In order to perform the customisation, the application user needs to first enter the edit page mode link as shown in figure 15. Once the user enters the edit page mode a set of extra icons would appear on the right side of the portlet container window. The application user than needs to select the 'bulb' sign as shown in figure 20 to enter the edit container properties. Once the user enters the container properties page as shown in figure 21 the user has three display patterns to choose from – 'Tab renderer', 'Column renderer' and 'Row renderer'. As the name suggest using the Tab renderer each portlet is displayed using tabbed interface as shown in figure 12. The Column Renderer divides a single page column wise, while rendering each portlet in each column. Figure 22 displays the workflow tool portal page using the column renderer.

Secondly, rearrange the viewing area on a portal page: The application user can very easily free up valuable space on a portal page by hiding content area of some portlets. This allows an application user quick access to critical portlets. In order to achieve this, the application user needs to click the minimised icon available on top right hand corner of each portlet as shown in figure 23. An application user can click the minimise icon for each portlet that needs to be collapsed, while only leaving the portlets that are required. Figure 24 displays

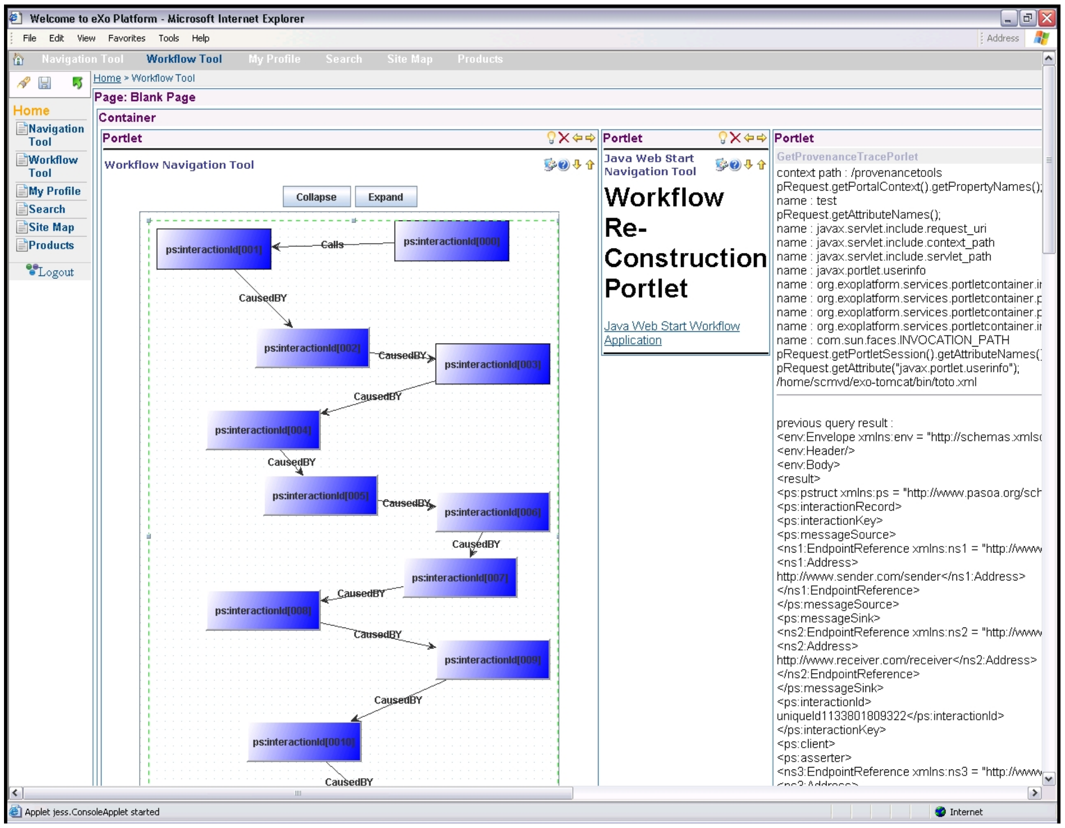


Add/Edit Container

Container Id	c20516605
Container Title	
Renderer	TabRenderer
Style	
Width	
Height	

ContainerColumnRenderer
ContainerRowRenderer
TabRenderer

Figure 21: *Edit container properties page*



```
context path : /provenancetools
pRequest.getPortalContext().getPropertyNames();
name : test
pRequest.getAttributeNames();
name : javax.servlet.include.request_uri
name : javax.servlet.include.context_path
name : javax.servlet.include.servlet_path
name : javax.portlet.userinfo
name : org.exoplatform.services.portletcontainer.i
name : org.exoplatform.services.portletcontainer.p
name : org.exoplatform.services.portletcontainer.p
name : org.exoplatform.services.portletcontainer.i
name : com.sun.faces.INVOCATION_PATH
pRequest.getPortletSession().getAttributeNames(
pRequest.getAttribute("javax.portlet.userinfo");
/home/scmvd/exo-tomcat/bin/toto.xml

previous query result :
<env:Envelope xmlns:env = "http://schemas.xmls
<env:Header/>
<env:Body>
<result>
<ps:pstruct xmlns:ps = "http://www.pasoa.org/sch
<ps:interactionRecord>
<ps:interactionKey>
<ps:messageSource>
<ns1:EndpointReference xmlns:ns1 = "http://www
<ns1:Address>
http://www.sender.com/sender</ns1:Address>
</ns1:EndpointReference>
</ps:messageSource>
<ps:messageSink>
<ns2:EndpointReference xmlns:ns2 = "http://www
<ns2:Address>
http://www.receiver.com/receiver</ns2:Address>
</ns2:EndpointReference>
</ps:messageSink>
<ps:interactionId>
uniqueId1133801809322</ps:interactionId>
</ps:interactionKey>
<ps:client>
<ps:asserter>
<ns3:EndpointReference xmlns:ns3 = "http://www
```

Figure 22: *Workflow tool portal page displayed using column renderer*

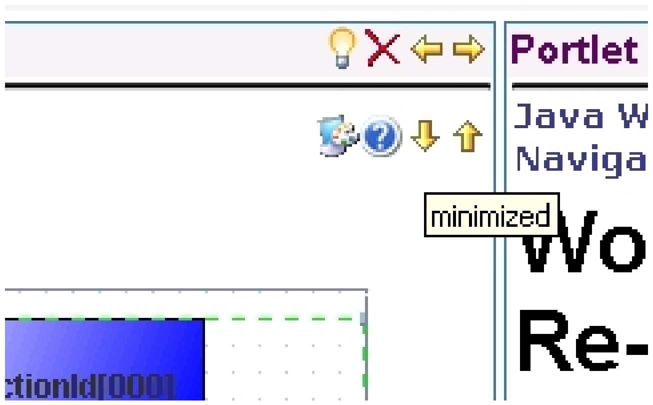


Figure 23: *Minimise icon attached to each portlet*

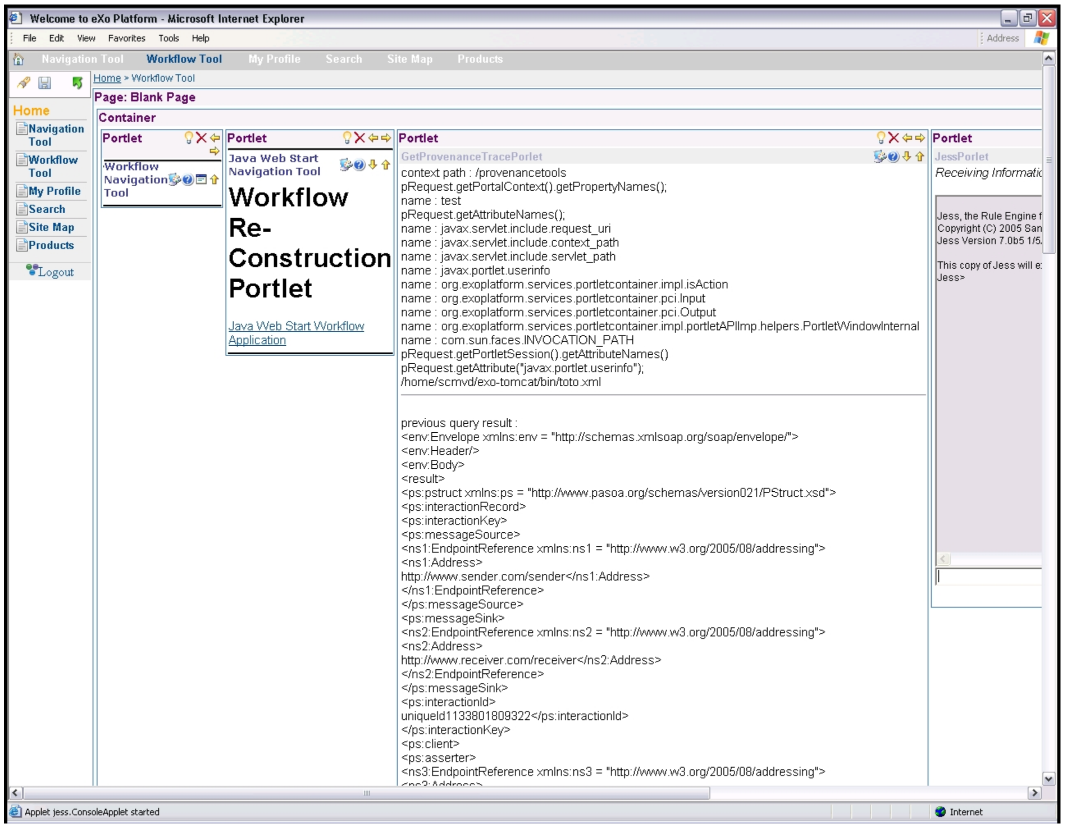
the workflow tool portal page with the workflow navigation portlet minimised and other portlets in the normal state. The benefit of this approach is that the user can recover valuable workspace on a portal page without having to add and remove portlets.

4.1.5 Issues and Trouble with Portal interaction

Some of the issues or problems that the application user might encounter during the interaction with the Portal server are discussed below using the help of screenshots. The first issue an application user may need to deal with is not being able to verify their credentials with the Portal server. One such instance is displayed in figure 25 where the user receives an error message that either a user is not registered or an invalid password has been entered. To correct this problem the user has to either verify that the password has been entered correctly or contact the tool suite administrator with the problem.

A second problem that an application user can encounter is not being able to add a new portlet from the portlet registry to the users' portal page. An instance of such a problem when adding a 'summary' portlet is displayed in figure 26. The error message occurs due to the application user not having the right role or authorization to add a particular portlet. Each portlet is configured by the tool suite administrator to only be accessible by users with a certain role. Any users that do not belong to the authorized list of roles are restricted from accessing the portlets.

The last and final common problem that a application user can face is 'time out'. After a certain time of inactivity the portal server requests the user to re-enter the username and password for security reasons. An instance of this is displayed in figure 27. By entering the username and password the user would be directed to the last portal page displayed before the time out occurred.



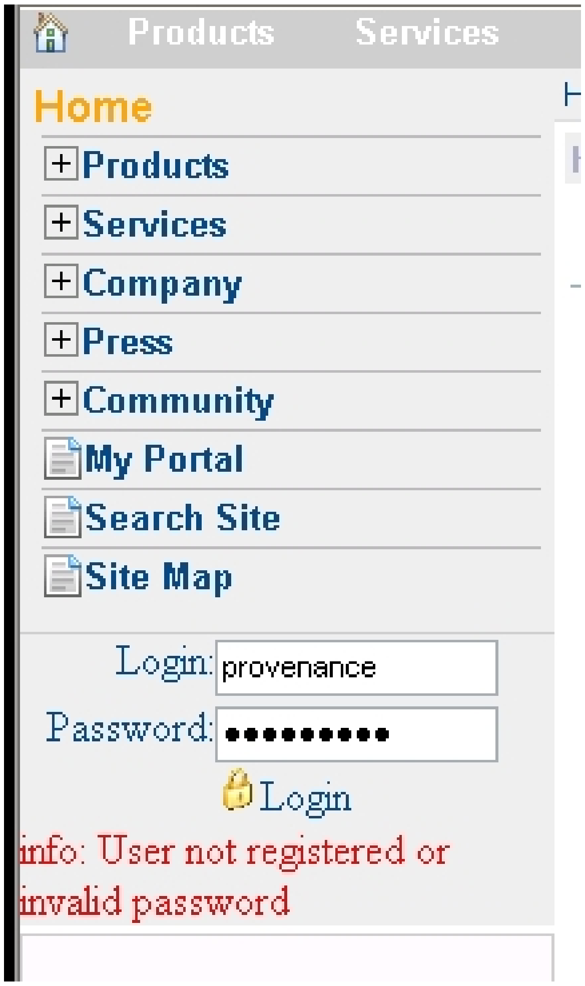
```
context path : /provenancetools
pRequest.getPortalContext().getPropertyNames();
name : test
pRequest.getAttributeNames();
name : javax.servlet.include.request_uri
name : javax.servlet.include.context_path
name : javax.servlet.include.servlet_path
name : javax.portlet.userinfo
name : org.exoplatform.services.portletcontainer.impl.isAction
name : org.exoplatform.services.portletcontainer.pci.Input
name : org.exoplatform.services.portletcontainer.pci.Output
name : org.exoplatform.services.portletcontainer.impl.portletAPIImp.helpers.PortletWindowInternal
name : com.sun.faces.INVOCATION_PATH
pRequest.getPortletSession().getAttributeNames()
pRequest.getAttribute("javax.portlet.userinfo");
/home/scmvd/exo-tomcat/bin/toto.xml

previous query result :
<env:Envelope xmlns:env = "http://schemas.xmlsoap.org/soap/envelope/">
<env:Header/>
<env:Body>
<result>
<ps:pstruct xmlns:ps = "http://www.pasoa.org/schemas/version021/PStruct.xsd">
<ps:interactionRecord>
<ps:interactionKey>
<ps:messageSource>
<ns1:EndpointReference xmlns:ns1 = "http://www.w3.org/2005/08/addressing">
<ns1:Address>
http://www.sender.com/sender</ns1:Address>
</ns1:EndpointReference>
</ps:messageSource>
<ps:messageSink>
<ns2:EndpointReference xmlns:ns2 = "http://www.w3.org/2005/08/addressing">
<ns2:Address>
http://www.receiver.com/receiver</ns2:Address>
</ns2:EndpointReference>
</ps:messageSink>
<ps:interactionId>
uniqueId1133801809322</ps:interactionId>
</ps:interactionKey>
<ps:client>
<ps:asserter>
<ns3:EndpointReference xmlns:ns3 = "http://www.w3.org/2005/08/addressing">
```

```
Jess, the Rule Engine f
Copyright (C) 2005 San
Jess Version 7.0b5 1/5.

This copy of Jess will e
Jess>
```

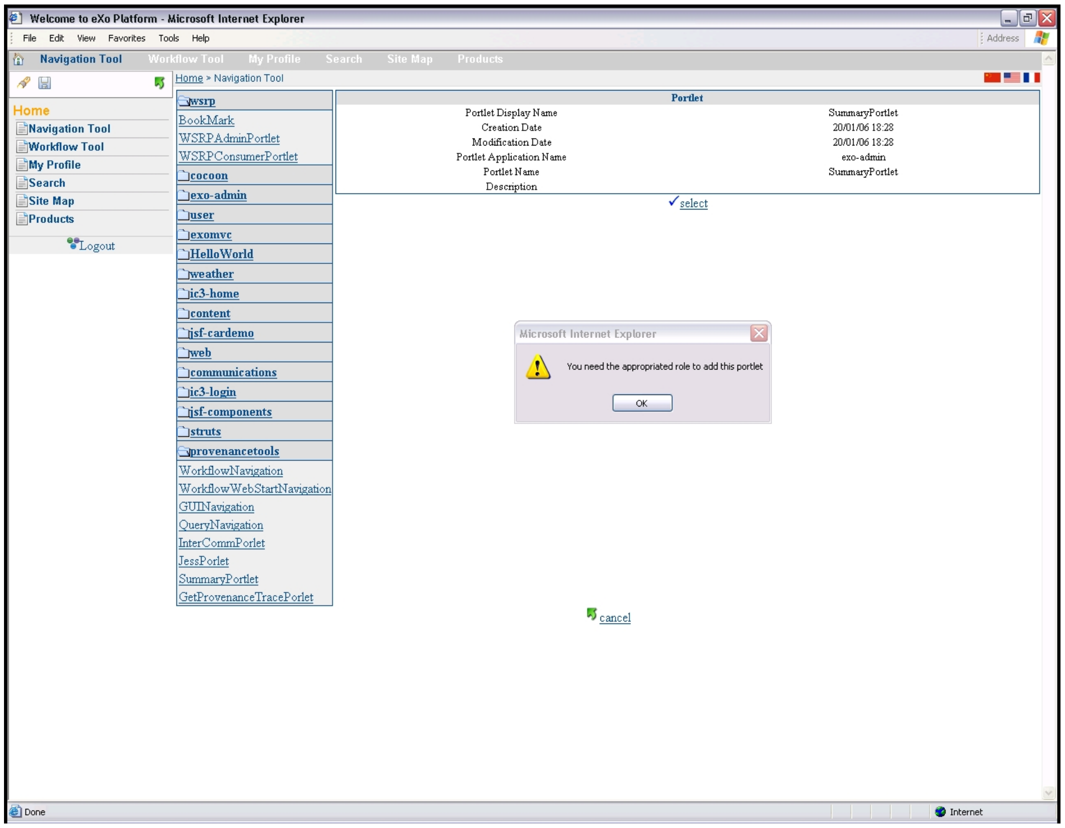
Figure 24: *Workflow tool portal page displayed with workflow navigation portlet minimised*



Login: provenance
Password: ●●●●●●●●●●

info: User not registered or invalid password

Figure 25: *Application user not being able to verify the credentials with the Portal server*



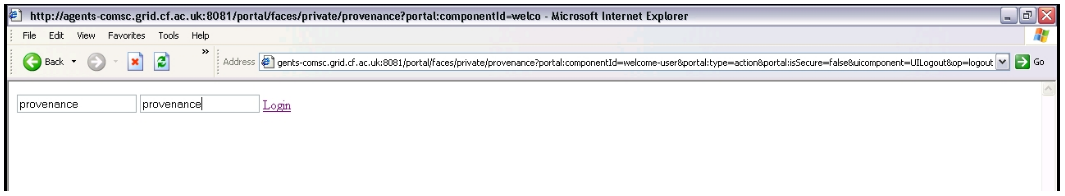
Portlet	
Portlet Display Name	SummaryPortlet
Creation Date	20/01/06 18:28
Modification Date	20/01/06 18:28
Portlet Application Name	exo-admin
Portlet Name	SummaryPortlet
Description	

Microsoft Internet Explorer

You need the appropriated role to add this portlet

OK

Figure 26: *Error message while trying to add new portlet from the portlet registry*



http://agents-comsc.grid.cf.ac.uk:8081/portal/faces/private/provenance?portal:componentId=welco - Microsoft Internet Explorer

Address gents-comsc.grid.cf.ac.uk:8081/portal/faces/private/provenance?portal:componentId=welcome-user&portal:type=action&portal:isSecure=false&uicomponent=UILogout&op=logout

provenance provenance Login

Figure 27: *Username and password request after a time out*

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

4.2 Analysis Tool

The analysis tool is dedicated to evaluate the application behaviour by querying all PSs used, in order to retrieve a set of p-assertions and by executing one or more rules on the result of the query (as discussed in requirement SR-1-7 of document D2.2.1).

Once a query has been submitted to the designated set of PSs and its result has been stored on the portal, this result can be loaded into the assertion engine in order to be evaluated. The format of the result cannot be directly exploited by the assertion engine. The data has to be preprocessed in order to be converted into a format understandable by the assertion engine. This part is handled by the tool suite and automatically done each time a result of a query is going to be loaded into the assertion engine.

It is also important to clearly define who has access to the analysis tool, and which rules a user can define and trigger. The current security model is based on a Role Based Access Control (RBAC) [8] where a user must belong to a particular role in order to allow access a particular feature or tool (in this instance, we intend to make use of the security subsystem discussed in deliverable D4.2.1 from WP4).

A user intending to start a new execution of an application must belong to the `ApplicationManagerRole` – created for the application he is trying to execute. This should have been done by the administrator of the portal server/tool suite. Once this has been checked and confirmed, the user is able to specify a list of rules for the analysis tool. The rules have to be encoded under a format understandable by the assertion engine used by the analysis tool. The current assertion engine, JESS [6], is based on the RETE algorithm [5]. It is able to load rules encoded under its default format which is a LISP-based encoding (presented in the listing JESS 1) or under an equivalent XML format JESSML (see listing JESS 2,page 50). Any one can create rules for the analysis tool, nevertheless these rules must be uploaded by a user belonging to the application's `ApplicationManagerRole`. As part of the configuration tool deliverable, we intend to provide a more user friendly rule encoding format than provided by JESS.

JESS 1 A JESS rule under the standard LISP encoding

```
;; display the interaction ID of all p-assertions loaded in the
;; working memory of the assertion engine
(defrule displayPAssertionID
   (Element (LocalName "interactionId") (Text ?interactionID))
    =>
    (printout t "interaction ID : " ?interactionID))
```

Obviously, analysis rules are only meaningful within a particular application and possibly only for a given instance of an application. So, we distinguish here

JESS 2 Same rule as in JESS 1 but encoded under the JESSML format

```
<rule>
  <name>
    MAIN::displayPAssertionID
  </name>
  <lhs>
    <group>
      <name>and</name>
      <pattern>
        <name>MAIN::Element</name>
        <slot>
          <name>LocalName</name>
          <test>
            <type>eq</type>
            <value><type>STRING</type>interactionId</value>
          </test>
        </slot>
        <slot>
          <name>Text</name>
          <test>
            <type>eq</type>
            <value><type>VARIABLE</type>interactionID</value>
          </test>
        </slot>
      </pattern>
    </group>
  </lhs>
  <rhs>
    <funcall>
      <name>
        printout
      </name>
      <value><type>SYMBOL</type>t</value>
      <value><type>STRING</type>interaction ID : </value>
      <value><type>VARIABLE</type>interactionID</value>
    </funcall>
  </rhs>
</rule>
```

two types of rules: application-wise and instance-wise rules. The scope of a rule is selected by the application manager inside the parameters given to the setup protocol.

To be allowed access to the analysis tool, a user should be associated with the role `AnalysisToolRole`. However, belonging to this role does not authorize the user to trigger rules for a given application. It is also necessary to belong to the role defining the set of users authorized to trigger a given rule of a particular application. As discussed previously, rules could be application-wise or instance-wise. The default role is `ApplicationWiseRuleRole`, a user belonging to that role can trigger any application-wise rule on any instance of the application. The second type of role is an `InstanceWiseRuleRole`. Users belonging to this role can only trigger rules on the ongoing execution of the application. Finally, the setup protocol allow the application manager to create additional roles by extending the `ApplicationWiseRuleRole` or `InstanceWiseRuleRole` in order to obtain a finer level of granularity. In order to set up the analysis tool, the following steps have to be performed during the setup protocol:

1. Allocate if needed an application space on the portal hosting the analysis tool. This space will contain both application-wise and instance-wise data related to the analysis tool.
2. Select the rules to upload, name them and for each of them, indicate if the rule is an application-wise or instance-wise rule. The location where the rule is stored on the portal server is dependant of the scope of the rule. If an application-wise rule already exists (same name) on the application space, it will be replaced by the new one. Consequently, all future usage of the rule will use the new version. There was no requirement for a rule versioning system. It is the responsibility of the application manager to take care of the implication of the upload of rule on the portal during the setup protocol.
3. Create or update roles required for the current execution of the application and fill them with selected users.

At the end of these steps, the analysis tool is ready to be used by designated users through the portal or by third party applications through the underlying API provided by the tools suite.

Even if this document mainly focuses on the setup protocol, it is worth to be mention that after the end of the setup protocol, an application manager still has access to the analysis tool configuration portlet. By the way, he is able to maintain the analysis tool by adding, modifying, deleting rules, adding user to a particular role, adding roles, ...

4.3 Management Interface at Provenance Store

It is expected that a security sub-system is provided to verify the credentials of a user, and define the types of valid operations that such a user can perform.

Such credential checking is undertaken via the the use of digital certificates. Security checking at a third party system – such as an external data store – needs to be undertaken prior to the use of the setup protocol.

The setup protocol interacts with the management interface at the PS by:

- Verify the credentials of a user wishing to access or submit one or more p-assertions. The actual mechanism used for undertaking this check is hidden from the tool suite and the application end user. The PS administrator is responsible for creating, modifying and deleting user names and passwords.
- Verify that the server hosting the PS and any associated data storage is online and accessible. This, however, does not guarantee that a subsequent request will be answered in a particular time period.
- It is assumed that the PS has a pre-defined amount of storage made available to each registered user (or application) to record p-assertions.
- Allow a mechanism to upload a notification policy – allowing a user to subscribe to particular events taking place at the PS.

5 Conclusion

The setup protocol has been described in this document, along with a description of all the actors involved in the protocol. The setup process is initiated by an application manager and subsequently coordinated by the tool suite administrator. The protocol terminates when either an error message (from which the tool suite administrator cannot recover) is produced, or a configuration file is generated. The next phase of work will involve:

This is a live document that will continue to evolve over the course of the EU Provenance project. As we progress through the project, it is possible that new requirements are identified, or we are better able to assess existing requirements. It is also possible that changes in Grid computing technologies would have a bearing on how we make use of some of the ideas expressed in this document (such as some of the techniques being employed within the Portal framework). Future updates are likely to include:

- Integrating the setup protocol with an existing application. This will also help us evaluate what additional stages are necessary within the protocol.
- Integrating the setup protocol with the security architecture being implemented in WP4.
- Better understanding of what specific configuration parameters will be required by a client-side library – in order to make use of the setup protocol.
- Implementation of a configuration tool that can be used alongside the setup protocol.

- Evaluating the setup protocol in the context of a particular application scenario – with recording of and access to p-assertions.

References

[1] *eXo Portal.* http://www.exoplatform.com/portal/.

[2] *eXo Portal Documentation v1.* http://forge.objectweb.org/projects/exoplatform.

[3] *JGraph.* http://www.jgraph.com/.

[4] *JSR 168 Portlet Specification.* http://jcp.org/aboutJava/communityprocess/final/jsr168/.

[5] Charles L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[6] Ernest J. Friedman-Hill. Jess, The Rule Engine for the Java Platform. http://herzberg.ca.sandia.gov/jess/.

[7] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. HTTP extensions for distributed authoring – WEBDAV, 1999.

[8] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *Computer*, 29(2):38–47, February 1996.

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