uPortal 3 project workplan

Overview

The current document outlines the process and goals of the uPortal 3 development. The document is broken down in two major sections, first presenting a set of high-level functional goals, and second outlining the formal steps of the initial development.

Functional goals

The functional goals are listed in the rough order of relative importance to the uPortal project.

JSR168 portlet support

The support for the portlet standard will be accomplished by employing Pluto JSR168 container module. The uPortal will use the Pluto Portlet and the Portal Container APIs. Some functionality of the Pluto org.apache.pluto.portalImpl module will be used. The following interfaces from the portlet container should be implemented by the uPortal:

1) org.apache.pluto.services.information.PortletActionProvider – is responsible for changing the portlet modes and portlet window states based on the information obtained from the portal environment.
2) org.apache.pluto.services.information.PortletURLProvider – defines portlet URL syntax, and mediates construction of the dynamic URL targets by the portlets.
3) org.apache.pluto.services.information.ResourceURLProvider – provides the portlet container with the resource URL based on the dynamic information.
4) org.apache.pluto.services.information.PortalContextProvider – provides the portlet container with the static context information, such as supported portlet modes, window states and portal properties.
5) Dynamic information provider – the DynamicInformationProvider interface from org.apache.pluto.services.information package, provides the portlet container with dynamic information about the portlet states and the current request.
6) Static information provider – the StaticInformationProvider interface

from org.apache.pluto.services.information package, provides the portlet container with static information regarding the portlets and the portal itself.

7) org.apache.pluto.services.PortletContainerEnvironment – provides the portlet container with services existing in the portal.
8) org.apache.pluto.services.factory.FactoryManagerService – returns the service implementation for the given service class.
9) org.apache.pluto.services.log.LogService – provides the portal container with a logging mechanism.
10) Servlet request/response factories necessary to support the portlet request forwarding mechanism.
11) A basic utility to support deployment of portlet WAR files.

IChannel support

An adapter implementation will provide full support for the uPortal 2.x channels.

Customizable rendering pipeline

A separate, customizable and easily configurable rendering procedure will be associated with each portal context.

Flexible persistence layer implementation

A multi-layered persistence class architecture will be designed to provide flexibility with respect to the choice of the persistence mechanism, while providing long-term maintainability benefits.

Custom deployment mechanisms

The deployment subsystem will provide ability to deploy portlets using flexible automated mechanisms, as well as manually.

User priority ranges

Group-based configuration of user priority ranges, that determine the ability of the user to prioritize (move) existing layout fragments, and define restrictions on the layout fragments that will be created by the user.

Layout fragment constructor

The fragment constructor will provide a UI for creating, modifying and deleting layout fragments. Access to the fragment constructor will be restricted to the group of users authorized to manage pushed fragments.

Node restrictions editor

The restrictions editor will extend the layout fragment constructor by provide the interfaces necessary to define and modify the rich set of restrictions provided the Aggregated layouts.

Layout fragment subscription support.

The layout fragment will be extended to allow for publishing of fragments into the content categories used by the portlet subscription model. An ability to subscribe to the fragments will be added to the subscription subsystem.

User- modifiable layout fragments

An ability to modify the layout fragments present in the layout for the purposes of the individual user will be added. The restrictions, associated with each fragment, will determine the set of allowed modifications. Conflicts arising from the subsequent central modifications (by the fragment author) will be resolved favoring the user- requested configuration, while observing all of the restrictions defined by the fragment author.

WSRP support

The support for the WSRP provider/consumer implementations, currently present in the uPortal, will be carried over. It is likely, however, that the code base will change.

Intuitive back button support

The uPortal will react in an intuitive, predictable way to the request generated during or after the browser back button use.

Customizable layout generation

The architecture will provide an ability to maintain a choice of layout generation strategies at runtime. Aside from the traditional, persistence-

based layouts, it will be possible to generate layouts programmatically. The choice of the layout generator will be determined by the portal context, and can be changed dynamically during the timecourse of a user session by changing the portal context.

Basic publishing workflow support

Allow to specify approval process and basic scheduling aspects during the portlet publishing process.

Layout fragment support for flexible fragment depth

Extend the layout fragment support to allow for fragments other than top-level elements (i.e. tabs), allowing to specify fragments of any depth (i.e. column, or channel fragments). This fragment constructor utility and the relevant AL modules will be modified.

Customizable URL syntax

A separate URL syntax implementation can be associated with each portal context, providing an ability to employ a syntax that would satisfy individual restrictions of each context. For example, a context might use idempotent, logical URL syntax that can be bookmarked and easily understood by users.

Layout filtering

Ability to selectively view only portions of the layout, as determined by a group membership or publish- time node parameters (not clear what mechanism will be used at this point)

Layout fragment set support

Provide an ability to define sets of layout fragments that will act as a single unit, providing an ability to subscribe to a set of fragments at once.

Delayed authentication support

The uPortal will provide the facilities necessary for the portlets to preserve user interaction state across the authentication procedure.

Integration of authorization and authentication APIs from OKI

The support for the APIs defined by the OKI will be provided, either

natively, or in a form of an adapter.

Development process

Functional requirements

In this preliminary phase, the functional goals, described in the previous section, are translated into a set of requirements towards the business logic, architectural pattern and data. Each requirement is accompanied by a set of use cases, and, if necessary, activity diagrams.

Portlet container usage

Because the portlet container (Pluto) is the central piece of the portal, it is useful to determine the details of the container use patterns. At this stage, the APIs describing the points of interaction with the container will be determined, and the operations required for container use (such as portlet deployment) will be described in detail.

Architecture and portal interfaces

The next stage involves drafting of the architecture for the portal framework. A definition of the interfaces for the major components will be produced.

Persistence layer definition

Once the high-level requirements for the storage subsystems have been described, the necessary abstraction layers can be defined. The current step will formulate the set of interfaces necessary for the persistence of all portal framework data, and will define the database schema to be used by the reference implementation.

Design specification

During the last preparatory step, the patterns and interfaces defined by the previous steps will be integrated into the overall design document. This will include a set of technical requirements, accompanied by the necessary class, activity and sequence diagrams. The design specification will include a description of the test strategies for each major component.

Test case design

In parallel with the design specification, test cases for each major component will be defined. An testing testing environment will be constructed. It will include the test scripts themselves, definitions of the test scenarios and the required data.

Core implementation

The goal of this phase is to implement the basic pieces necessary for the primary development phase by the JASIG community. The set of core classes will be implemented, in coordination with the appropriate test classes.

Community implementation

With a core implementation formulating the main architectural pattern, and providing a workable environment, the development of the remaining components will be distributed among the active JASIG developers in a traditional manner.

Testing

The test phases will verify the compliance of the developed components with the requirements of the design specification (as defined by the test cases).

Documentation

The end- user documentation effort will start during, or immediately prior to the community implementation phase. It will proceed to described the aspects of the uPortal operation and management as the features become available throughout the development phase.