Content Management - uPortal Usability and Accessibility

Content Management - uPortal Usability and Accessibility

Two excellent articles (from an information architecture perspective) on this topic: The Challenge of Dashboards and Portals and Introduction to the Building Blocks by Joe Lamantia of Boxes and Arrows. More articles are planned for this series.

Part 1 - Modes, Behaviours and Content Explosions

In studying the usability and accessibility factors associated with content management in uPortal, we need to consider some of the modes and behaviours of both the portal (framework) and the applications (portlets). The rules of good design tell us that the system status must always be clear to user, and there are techniques for ensuring that this is the case when familiar elements - tabs, columns, application boundary boxes, icons - are consistently displayed. Where we have to take the most care is when the user is presented with a new mode, or a new display element, such as a new information window.

Modes and mode errors go back to the earliest days of human-computer interactions. ("Oops, I thought I was in data entry mode when I typed 'delete all files' to the command-line processor".) Some applications are designed to be modal. With Gmail for example, when you're in compose mode, you can't read messages, and you can't compose more than one message at a time. This makes it easy to control the user's workflow and indicate current status. Thunderbird is much less constraining. I can compose as many messages as I like, and at the same time search through folders, read new mail, work on my address book, or perform any other task that my latest interrupt directs me to. Getting back to the portal, we can ask ourselves a relevant usability question, with some profound accessibility implications:

How "modal" do we want to make our portal? That is, how much value is there on having the user always in a defined mode, with controlled transitions: E. g. "You can't go back to reading new messages until you decide what you're going to do with the one you're composing (send, delete, save, etc)".

Certainly, modes can help us with accessibility. If we know the exact state of the user's interaction and what her navigational options are, we can find ways of providing specific assistance and status indicators. On the other hand, allowing the user to have the portal spawn all sorts of dynamic and interactive agents, so that there is no overall mode or interaction focus can add tremendous flexibility and utility.

It appears that if we confine our workflow to mode changes, we can find ways to track and display system status. For example, if the "weather" portlet only displays the weather for one city at a time, with the user selecting the city, things are simple. If, however, clicking on the weather portlet causes it to clone itself in a new window, allowing recursive spawning of multiple instances of the application, things get more complicated. Some questions arise:

1. Is there any good way of making clear the overall state of things?

2. Can the user find her way back to the root window where the "real portal" resides?

3. Can the spawned windows be managed by the portal? Is it up to the user to close them individually?

4. What is the authentication state of the spawned windows? What is the effect if the user logs out in the root window?

Clearly there are issues here beyond usability. Componentry that seeks only to address usability will not solve all the problems. It is appropriate, however to use usability as a motivator and a standard for creating the best solution in the abstract. If we can contrive a design for the best user-centric solution, we have a target for technical implementation, which we can then examine for other issues such as security and spawned agent management.

The main point of all this is that usability issues often arise at transitions. Some transitions are characterized by a mode: e.g. the transition from public /guest mode to logged-in/home-page mode, or the transition from multi-portlet mode to single-portlet-focus-mode. Other transitions result in an explosion of new content, usually in new windows, each with its own modal status. These are worth examining because they are likely common causes for problems of both accessibility and usability.

Part 2 - Portlet Modes and Window States: A User-Centric View

Discussion

When a user enters the portal, she sees a set of *portlet windows*- usually rectangular boxes, each surmounted by a title bar. These bits of chrome aren't required, but they assist the user in identifying the display boundaries of each application hosted by the portal. The boxes are often collected under tabs and arranged in columns to aid navigation. Given that the user is being presented with content by a diverse variety of applications, all sharing her screen, we must consider how they can provide their information in a manner most useful to her. They should not compete for her attention, or for space on the screen. Rather, each should display information of most immediate interest to her, and not show excessive detail. The portlet window of the email service, for example, might only tell her whether or not she has new messages. If there *are* new messages, she can activate a clickpoint within the portlet to open up a display of her inbox contents and view her mail. If not, then she may want to leave the mail portlet alone and move on to some other service such as her calendar. In its initial state, the calendar may only show a summary of the current day's events. Again, activating a clickpoint expands the window and shows more detail.

Let's consider the email portlet and suppose that our user wants to look up a number of filed messages and compose some new ones. In this case, her attention is focused on her email interactions, and she will want to make effective use of the whole portal display. She will want the email application to take over the screen, hiding the portlet windows for other applications. The Java Portlet Specification 1.0 provides support for various levels of interaction through its defined window states: minimized, normal, and maximized. From the user's point of view these present:

1. Minimal useful information, shown by default, using minimal screen space. This is sometimes referred to as "cameo" presentation.
2. Detailed information of interest to the user, still perhaps in summary form, efficiently laid out to allow sharing of the display with other portlet windows.
3. Detailed information with the assumption that the user's attention is fully engaged with the application, and all other application content may be displaced.

There is reason to suggest that the three window states are insufficient to cover all the possibilities - window states of "hidden", "title only", and "detached" spring to mind - but even if we stick to just these three, we can ask ourselves a number of usability and accessibility questions. Let us imagine that our portal supports a catalog of several hundred portlets, and that our user wants to populate her main portal page with a dozen of her favourites. Here are some things to consider:

1. How much space should a portlet in cameo form occupy? How many elements of information should it present?
2. She may want some portlets to come up cameo form, and some in detail form. Can we provide customization support for this?
3. When a portlet displays in maximized (focused) form, how is the overall status conveyed to the user? What navigational aids appear/remain on the screen? How is the method for returning to multiple portlet windows made clear to the user?
4. How much can the portal do, and how much must be left to the application designer. What are the rules for designing the behaviour of an application that will be only one tile in a mosaic? Can we avoid the "billboards on the highway" effect?

Definitions

> *Quoted from http://dev2dev.bea.com/pub/a/2004/09/portal_webapps.html*
>
> ***Portlet Modes***
> *A portlet mode indicates the function a portlet is performing. Normally, portlets perform different tasks and create different content depending on the function they are currently performing. A portlet mode advises the portlet what task it should perform and what content it should generate. There are three defined modes: view, edit, and help. It is only when a portlet is changing a mode that the content being displayed within the portlet is modified.*
>
> - *View: The expected functionality for a portlet in VIEW portlet mode is to generate markup reflecting the current state of the portlet. For example, the VIEW portlet mode of a portlet may include one or more screens that the user can navigate and interact with, or it may consist of static content that does not require any user interaction.*
> - *Edit: Within the EDIT portlet mode, a portlet should provide content and logic that lets a user customize the behavior of the portlet. The EDIT portlet mode may include one or more screens among which users can navigate to enter their customization data.*
> - *Help: When in HELP portlet mode, a portlet should provide help information about the portlet. This help information could be a simple help screen explaining the entire portlet in coherent text or it could be context-sensitive help.*
> - *Custom Portlet Mode: These are modes that provide a specific piece of functionality, in addition to the standard modes. The portlet custom mode is an optional feature when implementing the Portlet specification. The decision to provide this feature is purely a vendor decision. A vendor can still be JSR 168 compliant even if this feature is not made available.*
>
> ***Portlet States***
> *A window state is an indicator of the amount of portal page space that will be assigned to the content generated by a portlet. When invoking a portlet, the portlet container provides the current window state to the portlet. The portlet may use the window state to decide how much information it should render. There are three portlet states defined: normal, maximized, and minimized. Therefore, when the portlet changes state it is only the view size that will be modified — not the contents of the portlet. The definitions given below are from the portlet specification. Remember: these are specifications, and the implementation is entirely up to the vendor.*
>
> - *Normal: The NORMAL window state indicates that a portlet may be sharing the page with other portlets. It may also indicate that the target device has limited display capabilities. Therefore, a portlet should restrict the size of its rendered output in this window state.*
> - *Maximized: The MAXIMIZED window state is an indication that a portlet may be the only portlet being rendered in the portal page, or that the portlet has more space compared to other portlets in the portal page. A portlet may generate richer content when its window state is MAXIMIZED.*
> - *Minimized: When a portlet is in MINIMIZED window state, the portlet should only render minimal output or no output at all.*
> - *Custom Portlet States: These are states that provide additional functionality in addition to the standard states. For instance, a state could be defined that forces a portlet to take up to 75 percent of the portal page when invoked. The portlet custom state is an optional feature when implementing the Portlet specification. The decision to provide this feature is purely a vendor decision. A vendor can still be JSR 168 compliant even if this feature is not made available.*

Part 3 - Institutional Channel Strategies

Institutions embarking on an enterprise portal implementation usually find themselves developing what has been called in the uPortal community an *Institutional Channel Strategy*. This is a recommended element of an enterprise portal implementation and sustainment plan. What it comes down to is assessing the institutional content providers to determine what they can step up to technically and what resources they are willing to commit to delivering their information and services through the portal. Conventionally in uPortal, an application delivers its services through what is called a channel interface, and there are a variety of mechanisms supported as different channel types. One type, known as WebProxy, just encapsulates an existing web site, while another handles RSS streams. There are also embedded channels - applications designed to work in close integration with uPortal (UBC Webmail is an example).

It is often the case, that content providers aren't in a position to create sophisticated, well-integrated portal channels, but they would like to present their services through the portal nonetheless. Using the WebProxy mechanism is often a good choice for them. They need to do little beyond setting up a web site, and hooking up an authentication mechanism if such is needed. Sometimes there is a desire to host major web applications in the portal, using it simply as a launching point and a source of web single signon authentication. These implementations are part of the landscape that Fluid will have to deal with. There may be more elegant solutions than simple launching or proxying, but they may not be feasible at many institutions.

The discussion of Channel Strategies has recently become less of a focus in the uPortal community. New development is focussed on creating portlets instead of channels, and there is an expectation that "the portlet" will be everyone's new channel strategy. Since there is now a standard interface (the portlet specification), there will be an opportunity to acquire portletized applications from other sites, or license commercial offerings. It seems likely, however, that the current approach of fire and forget application launches will be with us for some time to come, and will continue to present accessibility /usability challenges.

Part 4 - Personalization and Customization

The mechanisms by which portal content is tailored for the individual user fall into two categories: Personalization and Customization.

Personalization is the selection and manipulation of content based upon things the portal knows about the user. For example, if the portal knows that the user is a student, the SIS portlet may appear prominently in the default layout. *"I get personalized service at the Chateau Exclusionary. They know I require a north-facing room and a left-handed TV remote"*.

Customization refers to the user's ability to select and configure content and its presentation. For example, the user may decide move the mail portlet window to the default tab and select vermillion as its background colour. *"I customized my pickup truck with reflective mud-flaps and trout decals"*.

(When a user logs in to a portal, she moves from a public service to a personalized customized environment where her needs/interests are paramount.

Note:

- Give up the banner and maybe the footer
- Hand over the screen to the user
- Give portlets in focus mode all the screen you can - just leave navigation tabs.

Here's a very large question: Do Personalization and Customization add a useful dimension to accessibility? Can customizing help a user arrange content to meet particular accessibility requirements? Can personalization (modification based on profile) do the same thing?
)

Part 5 - Novice, Occasional, and Expert Users

(We must discuss the effects modal behaviour for each of these. What are the implications for accessibility when modal behaviour is enforced?)