Java Portlet 2.0 (JSR 286 Spec)

John A. Lewis
Chief Software Architect
Unicon, Inc.

JA-SIG Conference
28 April 2008

UNICON

Agenda

1. Portlet Specs and Features
2. JSR 286 Major Changes
3. JSR 286 Minor Changes
4. Questions and Answers

Special thanks to Cris Holdorph from Unicon for helping to prepare this material.

Portlet Specs & Features

Portlet 1.0 / JSR 168 History

- Java Community Process
  http://www.jcp.org/en/jsr/detail?id=168
- Led by Sun and IBM
- Started: 29 January 2002
- Released: 27 October 2003
- Reference Implementation: Apache Pluto
- Interoperability between Portlets / Portals
- Set of APIs defining Portlets
- Linked to WSRP 1.0 Specification

Portlet 2.0 / JSR 286 History

- Java Community Process
  http://jcp.org/en/jsr/detail?id=286
- Led by IBM
  - Steven Hepper (sthepper@de.ibm.com)
- Started: 29 November 2005
- Final Approval Ballot: 3 March 2008 (Passed)
- Waiting for TCK and Reference Implementation (Pluto 2.0)
- Linked to WSRP 2.0 Specification

JSR 168 Feature Summary

- Lifecycle (init, action, render, destroy)
- Portlet URLs (Render URL, Action URL)
- Portlet Mode (View, Edit, Help)
- Window States (Normal, Maximize, Minimize)
- Render Parameters
- Portlet Preferences
- Portlet Session
- Portlet Deployment Descriptor (portlet.xml)
  - expiration-cache

JSR 286 – Major Changes

- Portlet Events
- Public Render Parameters
- Resource Serving
- Portlet Filters
- Caching Changes

JSR 286 – Minor Changes

- Window ID
- Namespacing
- Lifecycle Phase Request Attribute
- RENDER_HEADERS Sub-phase
- Portlet Cookies
- Setting Markup Head Elements
- Next Portlet Modes
- Portlet Tag Library Changes
- Additional CSS Classes
- Portlet Request Dispatcher Changes
- Portlet Resource Bundle Changes
- Portlet Container Runtime Options

JSR 286 - Unchanged

- Portlet Modes
- Window States
- Portlet Preferences
- Portlet Security
- User Information

Portlet Events

Events and the Portlet Lifecycle

- New Lifecycle Phase: ***Event Processing***
- For each overall portal page request:
  - **Action Phase** – called on at most one portlet window
  - **Event Phase** – called on as many portlet windows as necessary
  - **Render Phase** – called on up to as many portlet windows that are displayed on current page
- Events may be generated during *Action Phase* or *Event Phase* – not during *Render Phase*

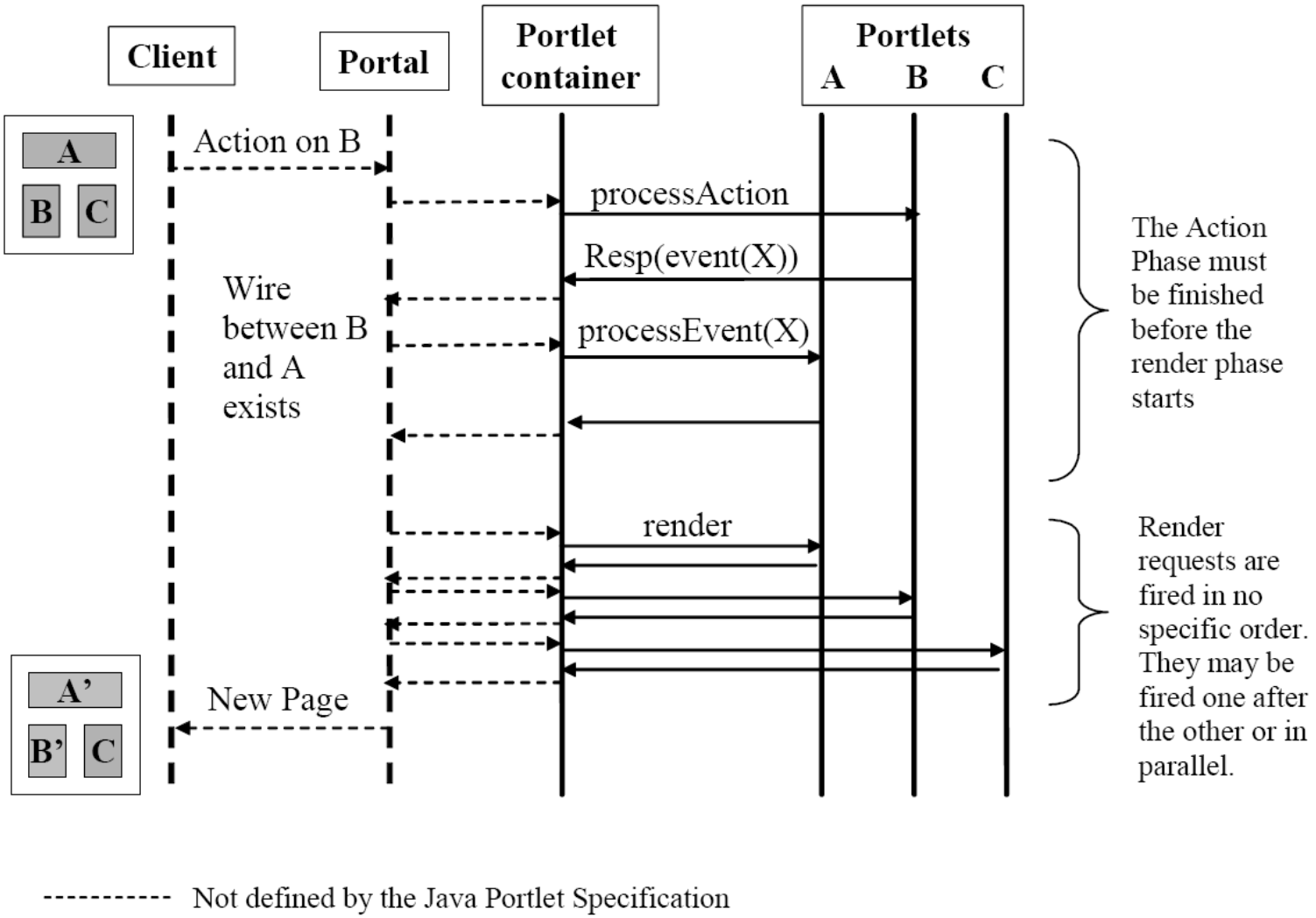

Diagram from Java™ Portlet Specification, Version 2.0 Public Draft , Rev. 19

EventPortlet Interface

- *javax.portlet.EventPortlet* Interface
  - May be implemented by a *Portlet*
  - Contains one method:

```
void processEvent(EventRequest, EventResponse)
```

  - *EventRequest* object provides event payload and other typical portlet info (mode, window state, etc)
  - *processEvent* is similar to *processAction* for copying *renderParameters*

Publishing Events

- Events may be published using methods on *ActionResponse* or *EventResponse*
  - *setEvent* or *setEvents*
  - Multiple calls to *setEvent* and *setEvents* are allowed
- Event delivery and processing order is **not guaranteed**

Event Definitions

- Events must be defined in *portlet.xml*
- After event definition, each portlet must declare what events it will publish or receive
- Portal-defined events do not have to be defined in *portlet.xml*
- Event naming:
  - Must use the W3C QName standard
  - Receiving events can end with a * wildcard
  - Can declare **`default-event-namespace`** in portlet.xml and just use local names

Events and JAXB

- JAXB 2.0 must be used to define the Event Payload
- JAXB is necessary for interoperability with WSRP events
- Implementing event payload class must be Serializable and annotated with JAXB annotations

Public Render Parameters

Public Render Parameters

- May be visible to multiple Portlets & Webapps
- Managed in *portlet.xml*
  - Defined in the **`<portlet-application>`**
  - Declared in each **`<portlet>`** that wants it
- Name must follow the W3C Qname spec – can declare a **`default-name-space`**
- A portal can decide which public render parameters will be shared by which portlets

Resource Serving

Resource Serving

- Portlets can create two types of Resource Links
  - Direct Links (not new)
  - Resource URL Links (new!)
- Direct Links
  - More efficient
  - Not guaranteed to go through Portal
  - Will not have portal context available
  - No portal access control
- Resource URL Links
  - Will go through the *ResourceServingPortlet* interface

ResourceServingPortlet

- *ResourceServingPortlet* Interface

```
void serveResource
  (ResourceRequest, ResourceResponse)
```

- Portlet can produce content with
  - *ResourceResponseWriter*
  - *OutputStream*
  - Delegate with a *RequestDispatcher* call
- Portal is not allowed to modify content
- Portlet should not use HTTP GET for state change use HTTP POST/PUT/DELETE instead

Resource URLs

- Portlet creates a *ResourceURL* to itself with *PortletResponse.createResourceURL()*
- *ResourceURL* only valid if a Portlet implements *ResourceServingPortlet*
- Does not cause *processAction* to be invoked
- Cannot change Portlet Mode or Window State
- All current render parameters will be included
- New parameters set do **not** become render parameters

Cacheability of Resources

- *ResourceURL* can control the "cacheability" of the resource via the *setCacheability* method:
  - **FULL** – The most cacheable – URL does not need to contain state of the page, the current render parameters, portlet mode, or window state
  - **PORTLET** – URL needs portlet state (render parameters, portlet mode, and window state), but does not need the state of the rest of the page
  - **PAGE** – The least cacheable – URL needs complete state of page and portlet
- Cannot create URLs with more detail in Resource requests from URLs with less detail

Other Resource Information

- *ResourceRequest* provides access to a mix of Portlet information and information unique to Resources:
  - Portlet Mode, Window State, and Render Parameters of the requesting portlet are provided
  - Full access to HTTP headers (can set on response as well)
  - HTTP Method of the request
  - The Resource ID set on the Resource URL (if any)
  - The ETAG for cache validation

Portlet Filters

Portlet Filters

- Modeled after Servlet Filters
- Modify request data by wrapping request
- Modify response data by wrapping response
- Intercept invocation of a portlet before and after it is called
- Filters may be chained

Portlet Filter Interface

- Must implement *javax.portlet.Filter* interface
- Must provide a public no-arg constructor
- *init()* method will be called on all Filters before being called on any Portlets
- *destroy()* will be called if Filter is removed from service
- *doFilter()* method called if *processAction()*, *processEvent()*, *render()*, or *serveResource()* would be called on Filtered Portlet

Declaring Portlet Filters

- Declared in *portlet.xml* in **`<filter>`** element
- **`<filter-mapping>`** element must specify the applicable portlets
- Restrict to specific lifecycle methods using the **`<lifecycle>`** element in **`<filter-mapping>`**
- Order in *portlet.xml* matters for multiple filters of the same portlet
- Portlet containers are expected/allowed to cache the “filter chain”

Wrapping Requests/Reponses

- New wrapper classes provided for all request and response objects for use with Filters:
  - ActionRequestWrapper
    ActionResponseWrapper
  - EventRequestWrapper
    EventResponseWrapper
  - RenderRequestWrapper
    RenderResponseWrapper
  - ResourceRequestWrapper
    ResourceResponseWrapper

Portlet Caching

Caching

- Two Types:
  - Expiration Caching
    - What existed before with some changes
  - Validation Caching
    - New for extension of expiration caching
- Caching is now applied to both the Render and Resource lifecycle phases

Expiration Caching

- If no **<expiration-cache>** value is specified then portlet will be treated as always expired
- New **<expiration-time>** sub-element
  - Previous time-in-seconds value goes here
- New **<expiration-scope>** sub element
  - **PUBLIC_SCOPE** may be shared across users
  - **PRIVATE_SCOPE** may NOT be shared (default)
- Action or Event request will expire cache
- **expiration-time** and **expiration-scope** can be changed programmatically

Validation Caching

- Portlet should set **`ETAG`** property (validation token) and **`expiration-time`** when rendering
- New render/resource requests will only be called after **`expiration-time`** is reached
- New request will be sent the **`ETAG`**
- Portlet should examine it and determine if cache is still good – if so, set a new **`expiration-time`** and *do not render*
- Must set the **`ETAG`**, expiration time, and caching scope before writing any output

JSR 286 Minor Changes

Window ID

- New *PortletRequest.getWindowID()* method must return the Portlet Window ID
- Review from JSR 168:
  - **Portlet Deployment** (not mentioned directly in specification): *portlet.xml* file information
  - **Portlet Definition**: Publish time information
  - **Portlet Entity**: Subscribe time information
  - **Portlet Window**: Login/Session time information
- Used for portlet-scoped session data

Namespacing

- *getNamespace()* method now available on all Portlet Request classes (previously only on *RenderRequest*)
- Provides a unique value for the current Portlet Window
- Value may be used to prefix Javascript functions / variables or other items within a portal page that must be unique
- Will return the same value for the lifetime of the Portlet Window

Lifecycle Phase Request Attribute

- **`LIFECYCLE_PHASE`** request attribute of the *PortletRequest* interface determines current phase:
  - **`ACTION_PHASE`** = *ActionRequest*
  - **`EVENT_PHASE`** = *EventRequest*
  - **`RENDER_PHASE`** = *RenderRequest*
  - **`RESOURCE_SERVING_PHASE`** = *ResourceRequest*
- Designed to let frameworks cast correctly

RENDER_HEADERS Sub-phase

- The Render Phase now has two sub-phases if the **renderHeaders** runtime option is set true
- Should be used when setting headers, cookies, the title, or next portlet modes
- Streaming portals will call *render* twice and set **RENDER_PART** portlet request attribute as follows:
  - **RENDER_HEADERS** on the first call, so perform appropriate header operations
  - **RENDER_MARKUP** on the second call, so now render the actual markup

Portlet Cookies

- Cookies can now be set on the *PortletResponse* and retrieved on the *PortletRequest*
- These cookies may be stored by the Portal and may not actually reach the client
- Cookies set in the response of one phase will be available in subsequent phases (e.g. a cookie set in the action phase will be available during the render phase)

Setting Markup Head Elements

- Use Response *addProperty* method with **`MARKUP_HEAD_ELEMENT`** constant as property name and an *org.w3c.dom.Element* value
- Provided DOM element should be added to the markup **`<head>`** section of the response to the client
- Support for this property is optional – verify via the **`MARKUP_HEAD_ELEMENT_SUPPORT`** property on the *PortalContext*
- For a Render Response, should be done in **`RENDER_HEADERS`** sub-phase

Next Possible Portlet Modes

- The *RenderResponse* can now indicate the next possible Portlet Modes and Window States
- Portals should limit available navigation controls accordingly
- To ensure this works in all portals, set them during the **`RENDER_HEADERS`** subphase

Portlet Request Dispatcher

- *PortletRequestDispatcher* may now be called from *processAction()* and *processEvent()*, as well as *render()*
- All non-render lifecycle methods will not be allowed to be write to any output stream
- *PortletRequestDispatcher* now has both an *include()* and a *forward()* method
- Portlet Request Dispatchers must follow any Servlet Filters set up

Portlet Resource Bundle

- Portlet Resource Bundle can now manage more information:
  - Portlet Info
    - title, short-title, keywords
    - display-name, description
  - Display Names / Descriptions
    - Public render parameters
    - Custom portlet modes and window states
    - Event definitions
    - User attributes

Portlet Container Runtime Options

- Define additional runtime behavior in *portlet.xml*
- Defined at portlet application level or the portlet level
- Use <*container-runtime-option*> element
- Current Options:
  - *javax.portlet.escapeXml*
  - *javax.portlet.renderHeaders*
  - *javax.portlet.includedPortletSessionScope*
  - *javax.portlet.actionScopedRequestAttributes*

Portlet Tag Library

- New **`resourceURL`** tag
- Existing **`namespace`** tag required to match the value of *PortletResponse.getNamespace()*
- New **`copyCurrentRenderParameters`** attribute on Action and Render URLs (default: false)
- New **`escapeXML`** attribute on Action, Render, and Resource URLs (default: true)
- New **`property`** tag for use in Action, Render, and Resource URLs to set request properties
- **`defineObjects`** tag now includes all new request/response objects and access to *PortletSession* and *PortletPreferences*

Additional CSS Classes

- Now includes the Table style definitions from WSRP 1.0
- Some additional Forms and Menus styles have been added
- *Note:* Some names in the draft still overlap and will need to be corrected before final release

Resources

Resources

- Main JSR 286 Website
  - http://jcp.org/en/jsr/detail?id=286
- WSRP 2.0 Specification
  - http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-spec.html
- Implementations
  - Pluto 2.0 (In Development)
    - http://portals.apache.org/pluto/
  - Jboss (Release Candidate)
    - http://www.jboss.org/portletcontainer/
  - OpenPortal (Release Candidate)
    - https://portlet-container.dev.java.net/
  - eXo (Release Candidate)
    - http://www.exoplatform.com/

Support in Spring Portlet MVC

- New annotations & parameters:
  - @RenderMapping
    - windowState
  - @ActionMapping
    - name (request parameter "javax.portlet.action")
  - @EventMapping
    - name (local part of the event name)
    - qname (full event qname)
  - @ResourceMapping
    - Id
  - All supporting the existing parameters for portlet mode and request parameters

Spring Portlet MVC Roadmap

- Portlet 2.0 support planned for Spring 3.0
- No idea when it will be released

  (the talk is sometime this summer, but you didn't hear it from me)
- Some design discussion in JIRA:
  - http://jira.springframework.org/browse/SPR-4259

Questions & Answers

John A. Lewis
Chief Software Architect
Unicon, Inc.

jlewis@unicon.net
www.unicon.net