Neil Griffen - Software Architect, Liferay, Inc.

Martin Scott Nicklous. – Specification Lead, JSR 362 Portlet Specification 3.0, IBM Corp

29 September 2014

What's New in Portlet 3.0 [BOF2210]

JSR 362 Portlet Specification 3.0

Important Disclaimers

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.

WHILST EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

ALL PERFORMANCE DATA INCLUDED IN THIS PRESENTATION HAVE BEEN GATHERED IN A CONTROLLED ENVIRONMENT. YOUR OWN TEST RESULTS MAY VARY BASED ON HARDWARE, SOFTWARE OR INFRASTRUCTURE DIFFERENCES.

ALL DATA INCLUDED IN THIS PRESENTATION ARE MEANT TO BE USED ONLY AS A GUIDE.

IN ADDITION, THE INFORMATION CONTAINED IN THIS PRESENTATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM, WITHOUT NOTICE.

IBM AND ITS AFFILIATED COMPANIES SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.

NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:

- CREATING ANY WARRANT OR REPRESENTATION FROM IBM, ITS AFFILIATED COMPANIES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS

Presenter: Scott Nicklous

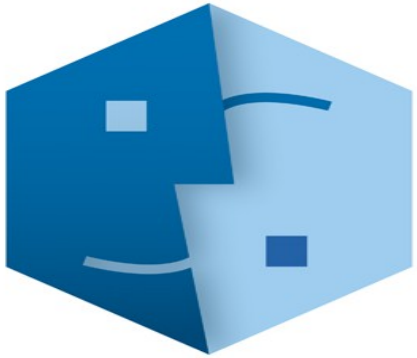
- Joined IBM in 1984 as a software developer for projects in the financial sector
  - Have held various roles: developer, architect, team lead, manager, consultant
  - Lead OpenCard Framework effort – standardized Java framework for Smart Card access
  - Security-related work in IBM Pervasive Computing Division

- Recent work focus:
  - WebSphere Portal Development
  - Manager responsible for portlet container & WSRP development
  - Specification Lead for JSR 362 Portlet Specification 3.0

- My contact information:
  - Scott.Nicklous@de.ibm.com

Presenter: Neil Griffin

- Software Architect, Liferay Inc.
- Liferay Faces Project Lead
- JSR 362 (Portlet 3.0) Expert Group Member
- JSR 344 (JSF 2.2) Expert Group Member
- Contributing Author: JSF Complete Reference



Liferay Faces

Agenda

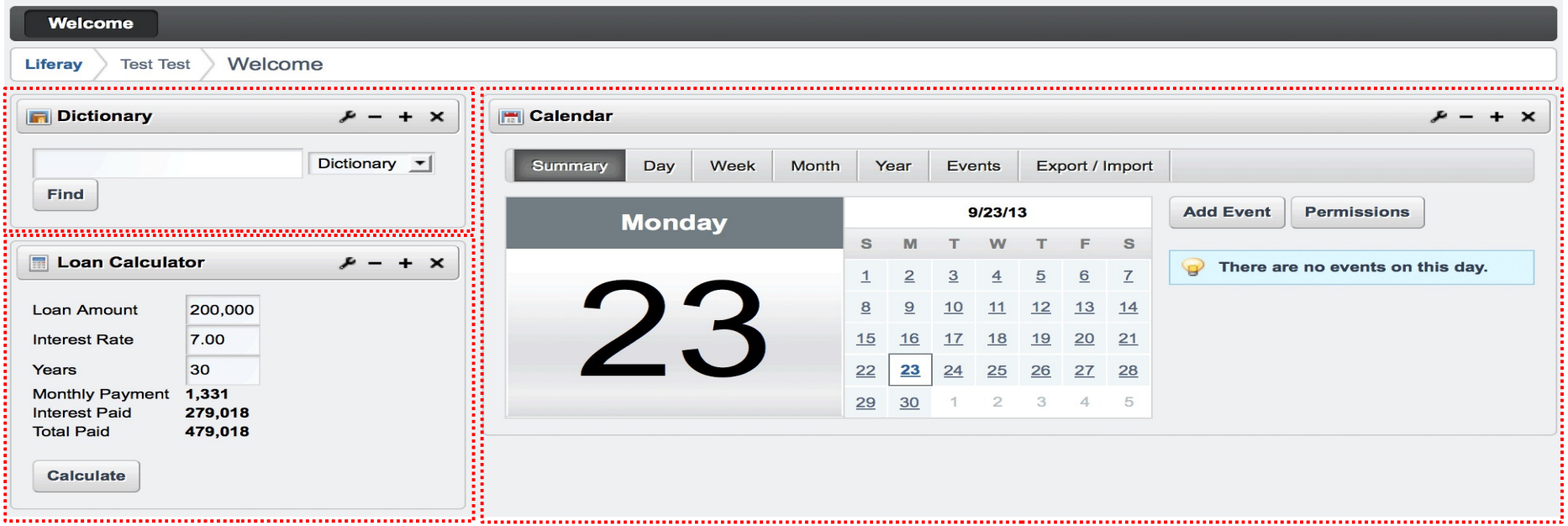
- Introduction to Portlets
- JSR 362: Where we're at
- Portlet State / Parameter Handling Improvements
- Ajax Support
- Outlook

Introduction to Portlets

Portlet Standard

- The Portlet Specification is a JCP standard that defines requirements for portlet containers and behavior of portlet applications
  - JSR 168 – Portlet 1.0
  - JSR 286 – Portlet 2.0
  - JSR 362 – Portlet 3.0

Portlet Examples



Servlet Lifecycle

- Single phase associated with HTTP GET/POST/PUT/DELETE operations:

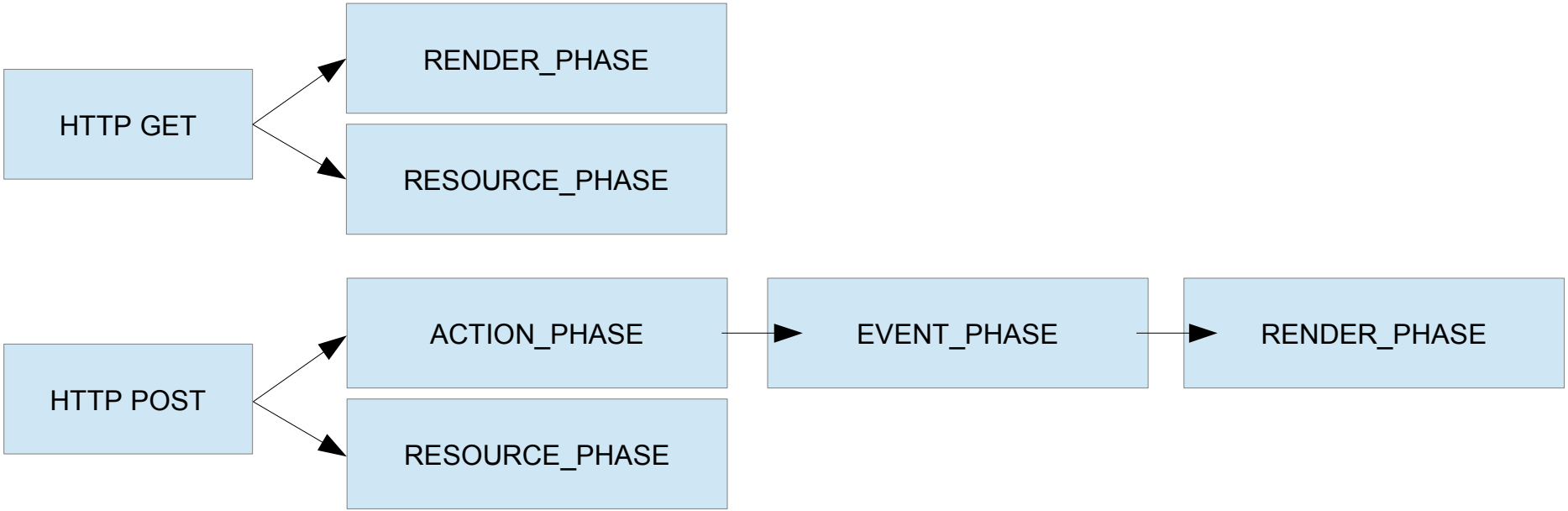
```
public class MyServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    @Override
    public void service(ServletRequest servletRequest, ServletResponse servletResponse)
            throws ServletException, IOException {

        // Single phase that handles HTTP GET/PUT/POST/DELETE
    }

}
```

Portlet Lifecycle

- Multiple phases associated with HTTP GET/POST operations:

```
public class MyPortlet extends GenericPortlet {

    @Override
    public void processAction(ActionRequest actionRequest, ActionResponse actionResponse)
            throws PortletException, IOException {
    }

    @Override
    public void processEvent(EventRequest eventRequest, EventResponse eventResponse)
            throws PortletException, IOException {
    }

    @Override
    public void render(RenderRequest renderRequest, RenderResponse renderResponse)
            throws PortletException, IOException {
    }

    @Override
    public void serveResource(ResourceRequest resourceRequest, ResourceResponse resourceResponse)
            throws PortletException, IOException {
    }
}
```

Portlet Lifecycle (Cont.)



- Inter-Portlet Communication (IPC) is a way of sharing data between two or more portlets
- Standard Mechanisms:
  - Events IPC
  - Public Render Parameters IPC

JSR 362: Where we're at

JSR 362 Portlet Specification 3.0

- The JSR 362 Expert Group was formed in April 2013
  - Participants: Apache Foundation, IBM, Liferay, Oracle, RedHat, Vaadin, Independents
- In accordance with the JCP process version 2.9, our work is public
  - JCP page for JSR 362: http://www.jcp.org/en/jsr/detail?id=362
  - Project web site: http://java.net/projects/portletspec3
    - Interested parties can subscribe to an observers mailing list at the site
  - JIRA Issue tracker: http://java.net/jira/browse/PORTLETSPEC3
  - Portlet API working docs: http://msnicklous.github.io/portletspec3/apidocs/index.html
- The Portlet Specification 3.0 Reference Implementation
  - Work began in 2014
  - Apache Pluto subproject of Apache Portals project
  - Volunteers welcome!

Status

- Discussed and accepted numerous small improvements / errata changes to the JSR 286 spec
  - See JIRA issue tracker
- Adopted a proposal to support idea of portlet state in portlet API definitions
  - Improves usability from developer point of view
  - Clarifies use of parameter handling APIs
- Ajax Support
  - Basic agreement on Portlet Hub JavaScript API
  - JSdoc documentation of JavaScript API 1st draft complete
  - Portlet Hub mockup 1st draft complete
  - Implemented many Jasmine JavaScript test cases to verify mockup behavior
- Have begun work on the JSR 362 TCK – Apache Pluto project
- Ready to begin prototyping for portlet state & Ajax proposals
  - JavaScript & Java APIs available for public review:
    - http://msnicklous.github.io/portletspec3/apidocs/index.html

Portlet State -
API Usability Improvements

Idea of Portlet State

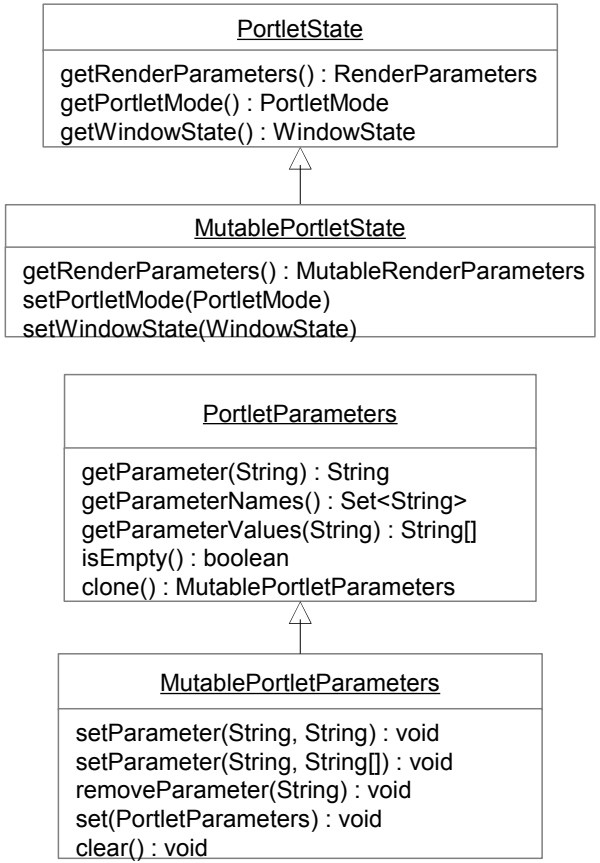
- Portlet State
  - Consists of render parameters, portlet mode, & window state
  - Render parameters may be either private or public
  - Set during Action and Event processing
  - Used during render phase to generate content
  - Used during resource phase to serve resources (depending on cacheability)
  - Stored on the URL (conceptually, at least)
  - Enables bookmarking & back-button support
- Action Parameters
  - Provided through ActionRequest to allow portlet to set up next render parameters
  - May be set by the portlet on the Action URL
  - May be provided by client through form submission
- Resource Parameters
  - Provided through ResourceRequest to allow the portlet to discern between different resource requests
  - Set by the portlet on the Resource URL
- So far, nothing new …
  - Concept is implicitly defined by JSR 286 Portlet Specification 2.0
  - Will be explicitly defined by JSR 362 Portlet Specification 3.0

Where's the Problem? - JSR 286 API

- The portlet state is **implied**
  - Portlet request API designed to make portlets look like servlets
  - From the point of view of the API, all parameters are just “parameters”
  - Meaning of parameters must be interpreted by context (action, event, render, or resource request)
- Action processing
  - No “official” idea of portlet state – all parameters are action parameters
  - But often, developers must store render parameters on an action URL in order to add them again as render parameters in the processAction method
  - Real action parameters including form parameters must be separated out by code logic
- Resource serving
  - Conceptually the idea of portlet state is present – there are render parameters and resource parameters
  - However, the API defines only “parameters”
  - Rule: if resource & render parameter have same name, resource parameter comes first in values array
  - Danger of name clashes
  - Must be handled by logic
- No common interface for parameter handling between URLs and Responses
  - Render parameters, portlet mode, and window state have to be set on responses as well as on URLs
  - However, there is no common interface, so that state handling code must be written twice

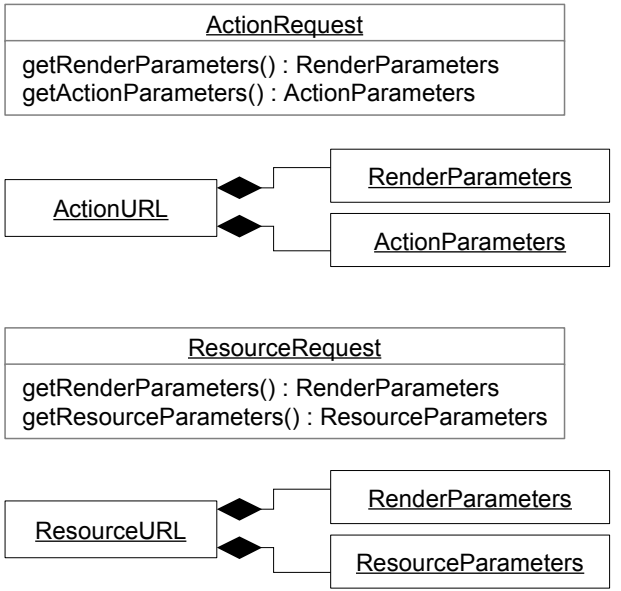
What's the Solution? - JSR 362 API

- The portlet state defined **explicitly in the API**
  - PortletState object: render parameters, portlet mode, window state
    - always present and is available from every portlet request
  - MutablePortletState object allows state to be updated
    - Available from portlet responses & URLs as appropriate
  - REST paradigm: portlet state associated with URL
- Parameter handling
  - Parameters are encapsulated in dedicated objects for better handling
  - Appropriate mutable and non-mutable objects
  - Clean, regular API for all types of parameters
- Common API for state handling on URLs and Responses
  - Handling state within the portlet becomes more regular
  - State handling code can be the same regardless of whether
    - State is being set on a response for next render phase
    - State is being set on a new URL during rendering



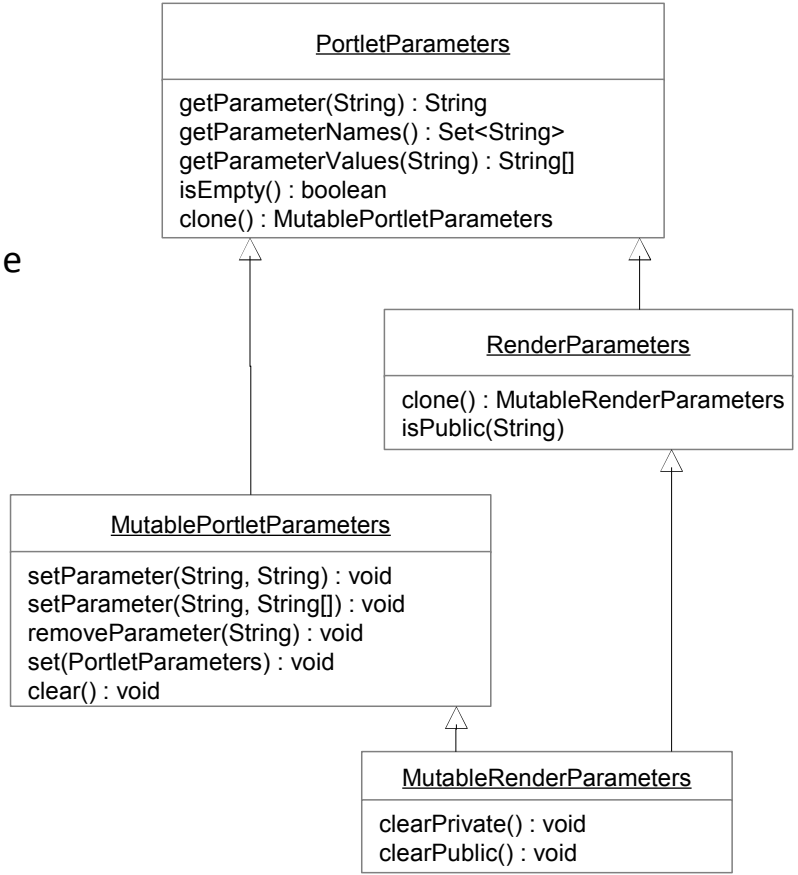
What's the Solution? - JSR 362 API

- Action processing
  - Action Parameters
    - Separate from portlet state (render parameters)
    - Can be set on the action URL explicitly
    - Can be provided by client through form parameters
  - Portlet state can be automatically added when action URL is created
- Resource serving
  - Resource Parameters
    - Separate from portlet state (render parameters)
    - Can be set on the resource URL explicitly
  - Portlet state added when resource URL created (like JSR286)
- Render URL
  - Portlet state can be automatically added when render URL is created
- Separation of action & resource parameters from portlet state
  - No possibility of parameter name clashes
  - Improves clarity
  - Makes API more intuitive



What's the Solution? - JSR 362 API

- Public render parameters
  - Handled in RenderParameters object
  - Method for determining if a parameter is public
  - But setting / removing public render parameter uses same API as regular parameters



Ajax Support

Ajax Support – Main Goals

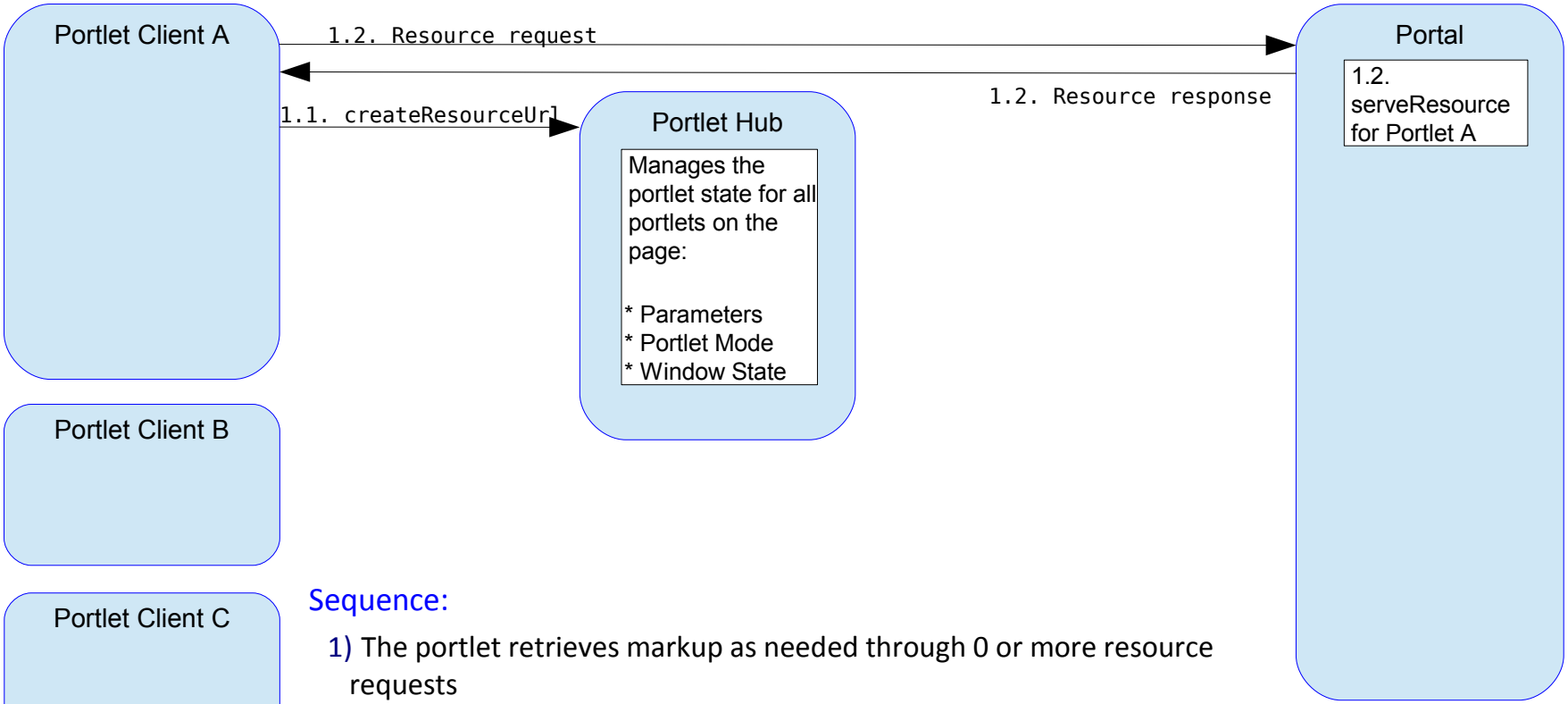
- Background: JSR 286 added resource serving concept
  - Enabled basic Ajax support by allowing portlets to retrieve content taking into account the portlet state
  - But did not provide for portlets to change their state within an Ajax paradigm
- JSR 362 Portlet Specification 3.0 adds stateful Ajax support
  - Portlets can update their public & private render parameters while remaining within an Ajax paradigm
  - Portlets can perform an action (with HTTP POST method) while remaining within an Ajax paradigm
  - Allow portlets to obtain resource URLs on the client that reflect the current page state
- JSR 362 defines a JavaScript API to accomplish these goals
  - Provides for a JavaScript component that manages state for portlets on the page

Ajax Support – Main Goals

- Terminology
  - The Portlet Client is JavaScript code provided by the portlet that represents the portlet on the client.
  - The Portlet Hub is JavaScript code running on the client.
    - Communicates with both the Portal and the Portlet Client.
    - Manages the portlet state (render params, portlet mode, window state) for all portlets on the page
    - Part of the portal implementation; implementation is vendor specific.
    - must implement the API methods described by JSR 362
    - The goal of the Portlet Hub API is to define its external semantics while allowing each portal vendor freedom to implement in the manner most suitable for the Portal implementation.
    - Covered by TCK tests
  - Communication between the Portal and its Portlet Hub implementation is unspecified and is vendor specific.

Scenario: Portlet retrieves data through resource request

Portlet Client obtains resource URL for current page / portlet state in order to retrieve markup from the server. Free to use any JS library to perform Ajax request.

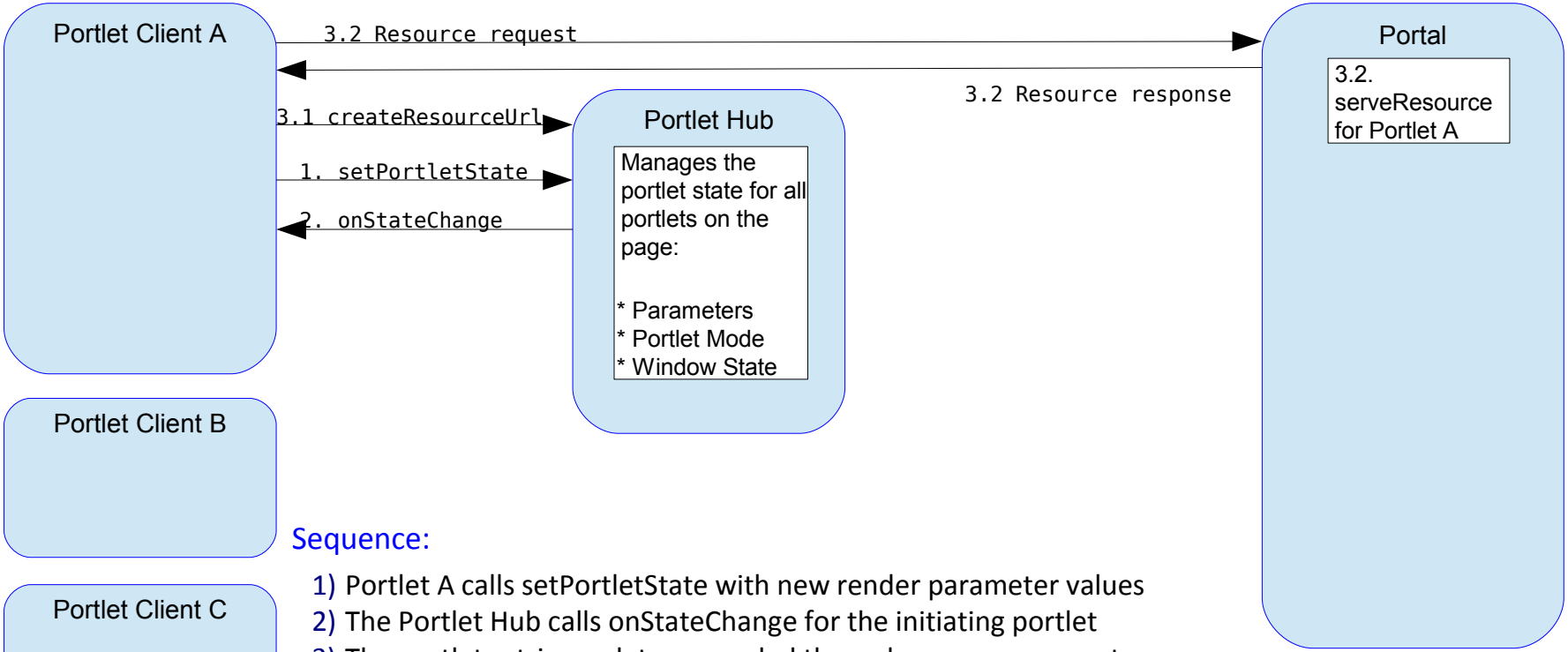


Sequence:

1) The portlet retrieves markup as needed through 0 or more resource requests
    1) The portlet gets a resource URL for current portlet / page state
    2) The portlet retrieves markup from the server

Set portlet state – only private render parameters

Moves the portlet to a new navigational state by changing private render parameters. Example: view different tab or new page of data within portlet. Affects only a single portlet.

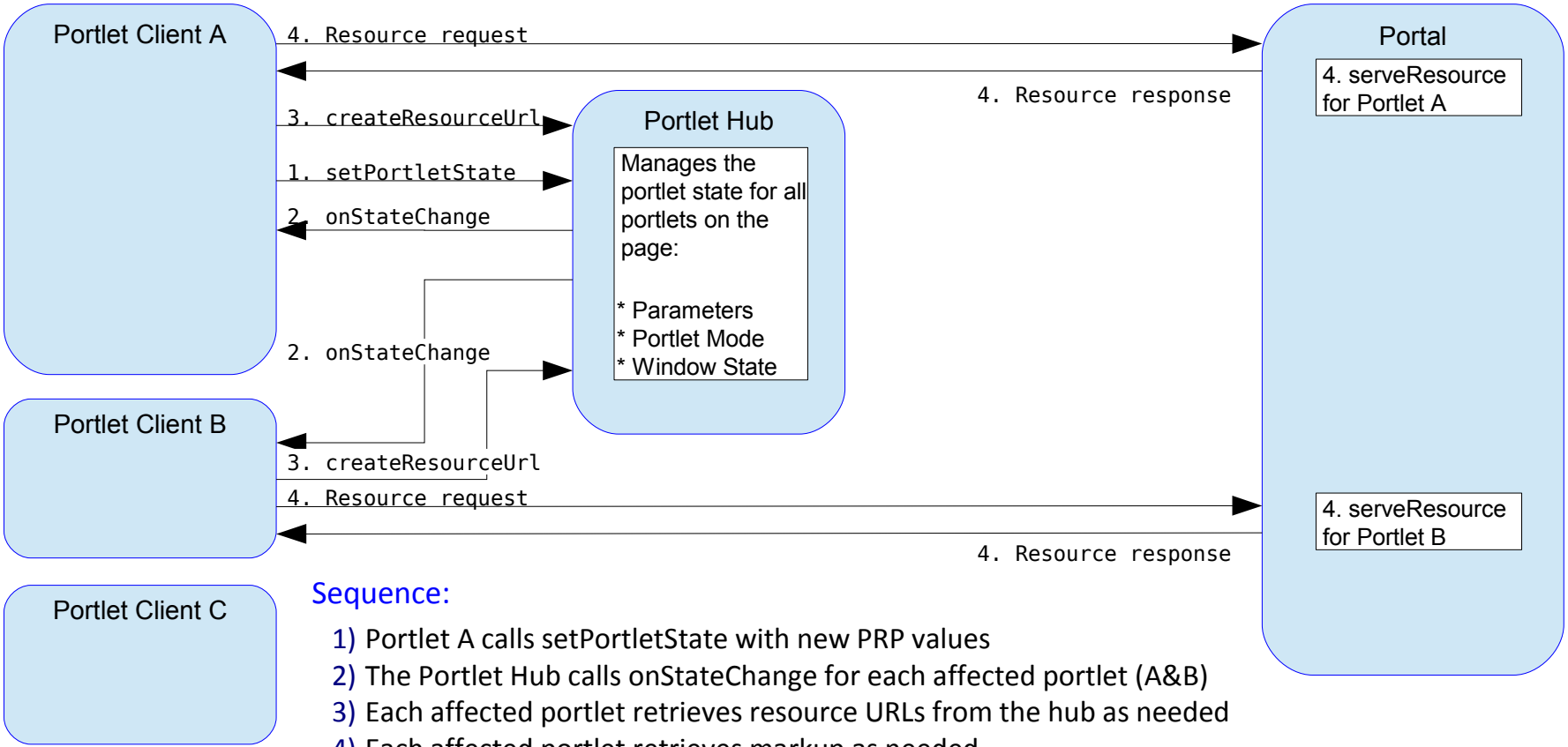


Sequence:

1) Portlet A calls setPortletState with new render parameter values
2) The Portlet Hub calls onStateChange for the initiating portlet
3) The portlet retrieves data as needed through resource requests
    1) The portlet gets a resource URL corresponding to the new portlet / page state
    2) The portlet retrieves markup from the server

Set portlet state with public render parameters

setPortletState with PRPs acts like a simple eventing mechanism, since other affected portlets on the page are updated with the new PRP values.

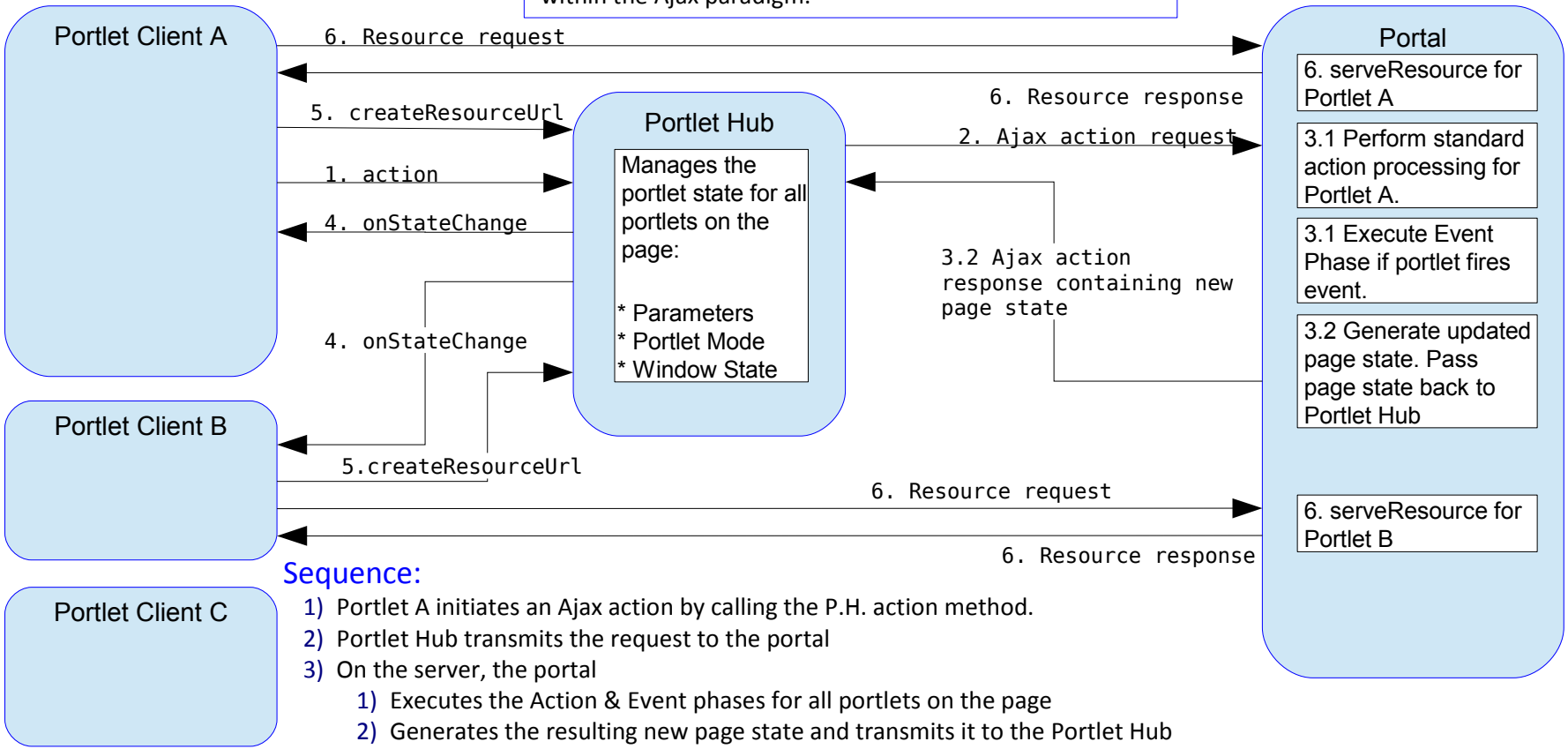


Sequence:

1) Portlet A calls setPortletState with new PRP values
2) The Portlet Hub calls onStateChange for each affected portlet (A&B)
3) Each affected portlet retrieves resource URLs from the hub as needed
4) Each affected portlet retrieves markup as needed

Action Processing Sequence

Basic idea is to allow an action request to be executed within the Ajax paradigm.

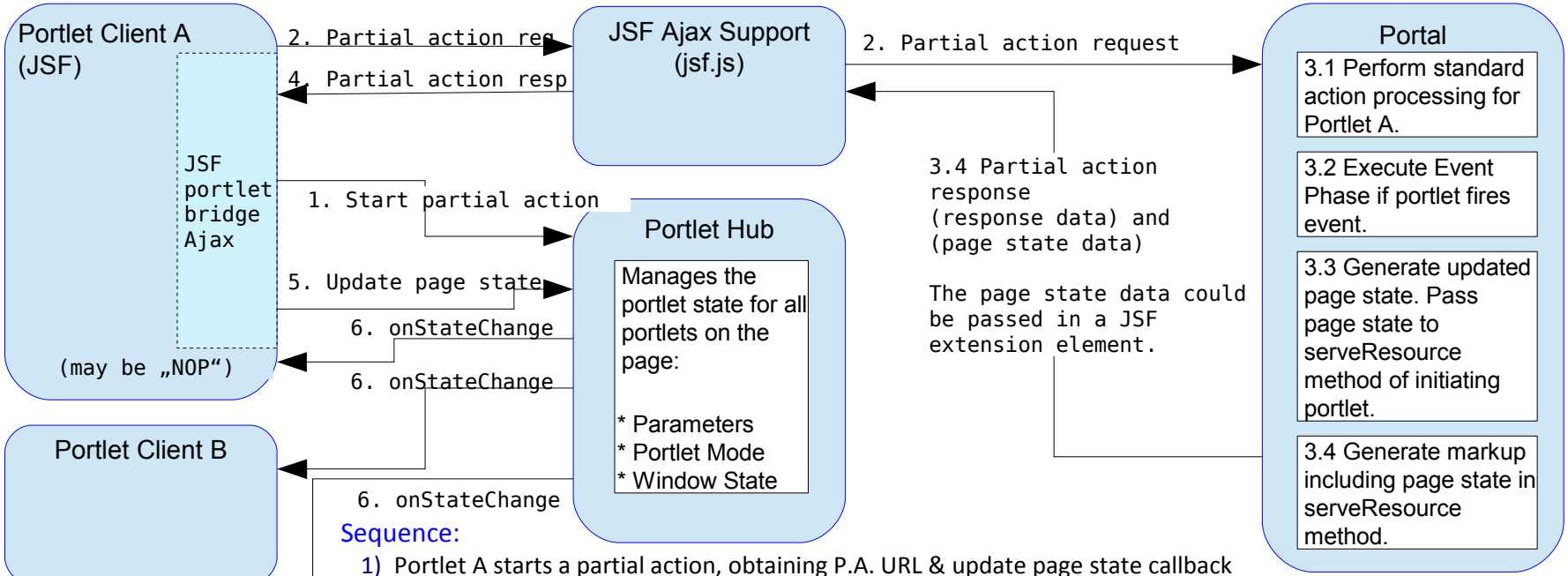


Sequence:

1) Portlet A initiates an Ajax action by calling the P.H. action method.
2) Portlet Hub transmits the request to the portal
3) On the server, the portal
    1) Executes the Action & Event phases for all portlets on the page
    2) Generates the resulting new page state and transmits it to the Portlet Hub
4) The Portlet Hub decodes the new page state and calls onStateChange for each affected portlet (A&B)
5) Each affected portlet retrieves a resource URL from the hub
6) Each affected portlet retrieves markup as needed

Partial Action Overview

This is a special execution path to enable UI framework (JSF) support. (Resource requests left out for clarity)



Sequence:

1) Portlet A starts a partial action, obtaining P.A. URL & update page state callback
2) Portlet A initiates partial action through JSF.js module
3) On the server, the Portlet Conainer:
    1) Drives the Action phase for portlet A, allowing events & parameter updates
    2) Drives the Event Phase as req'd.
    3) Generates new page state & passes it to Portlet A serveResource via reserved parameter.
    4) Portlet A generates content including page state and transmits it to jsf.js FW code
4) Jsf.js handles the response, passing response data & state info to Portlet A
5) Portlet A updates the Portlet Hub with the new page state data
6) The Portlet Hub decodes the state data, calls onStateChange for affected portlets

Additional features under consideration

Portlet 3.0 Features

- CDI Scopes and the Portlet Lifecycle
  - CDI scopes to support portlet lifecycle
- Improvements to Portlet Events
  - Programmatic event definition, Broadcast events
- Improvements to Public Render Parameters
  - Programmatic PRP definition
- WebSocket Support
- Mobile Device Support
  - Allow Portlets to declare supported devices / user agents
  - Dedicated window state / portlet mode for mobile
- Resource sharing / dependency declaration
- Offline Support - allow portlets to sync data to client
- Servlet 3.1 Alignment
  - Async support
  - Multipart form support
  - Componentization (Portlet Fragments)

Discussion

Copyright and Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., and registered in many jurisdictions worldwide.

Other product and service names might be trademarks of IBM or other companies.

A current list of IBM trademarks is available on the Web – see the IBM “Copyright and trademark information” page at URL: www.ibm.com/legal/copytrade.shtml