Advanced Topics in WebSphere Portal Development

Graham Harper
Application Architect
IBM Software Services for Collaboration

Ideas behind this session

- Broaden the discussion when considering what sort of solutions are possible using WebSphere Portal
- Introduce some of the lesser-known or used development facilities of the product
- Do this through showing some specific examples of non-standard solutions created previously

Agenda

- Introductions
- Portal development – unwrapping the layers
- Enforcing acceptance of terms and conditions
- Very long-running tasks in portlets
- Redirecting users on logout
- Questions and discussion

Introductions

I'll go first...

- Worked in IBM Software Group since the acquisition of Lotus in 1995
- Developing solutions for customers on WebSphere Portal for approximately 10 years
- Used many facilities of the product in that time
- Taught the WP 7 development course

Your turn – a quick survey

- **So, who here has:**
  - Developed JSR 286 portlets in RAD?
  - Developed portlets in Portlet / Experience Factory?
  - Created themes and skins?
  - Used the Portal APIs / SPIs?
    - e.g . PUMA, Login Service, Credential Vault, Selection Model

Portal development – unwrapping the layers

Layer One – Standard portlets

- **JSR 286 portlets**
  - Standardised contract between portlet and container
  - Usually Java and JavaServer Pages or Web Experience Factory
  - May add frameworks like JavaServer Faces or Struts to improve productivity and reusability
- **Portlets are aggregated into pages by the portal**
  - Navigation is provided
- **Inter-portlet communication is available**
  - Portlets work together to form complete applications
  - Events and public render parameters are standard mechanisms

Layer Two – Styling the portal

- **Themes and skins can be developed for your portal**
  - Consistency of interface across pages and applications
  - Compliance with organisation standards
  - Different look can be created where needed

- **Dojo toolkit provides rich client-side functionality**
  - To both themes and portlets

Layer Three – Additional portlet facilities

- **Less frequently used parts of JSR 286**
  - Resource-serving for Ajax requests
  - Complex event payloads with JAXB
  - Portlet filters
- **Common portlet services provided by WebSphere Portal**
  - Client-side aggregation APIs
  - Portal User Management Architecture (PUMA) API
  - Credential Vault Service

Layer Four – Portal facilities

- **Web Content Management (WCM)**
- **Personalisation**
  - Portlet and page visibility rules
- **Portal APIs and SPIs**
  - Model SPI (Navigation and Selection)
  - Dynamic UI API
  - Login Service
  - Content Access Service
  - Impersonation API
  - Tagging and Rating SPI

Layer Five – Portal plugin points

- **Piece Of Content (POC) URI resolvers**
- **Authentication Filters**
- **State Preprocessor**

Layer Six – WAS plugin points and facilities

- **Security**
  - Custom User Registry
  - Trust Association Interceptor
- **JEE features and extensions**
  - EJBs
  - Standard web applications
  - Scheduler
- **Be careful of your licence agreement, however**

Enforcing acceptance of terms and conditions

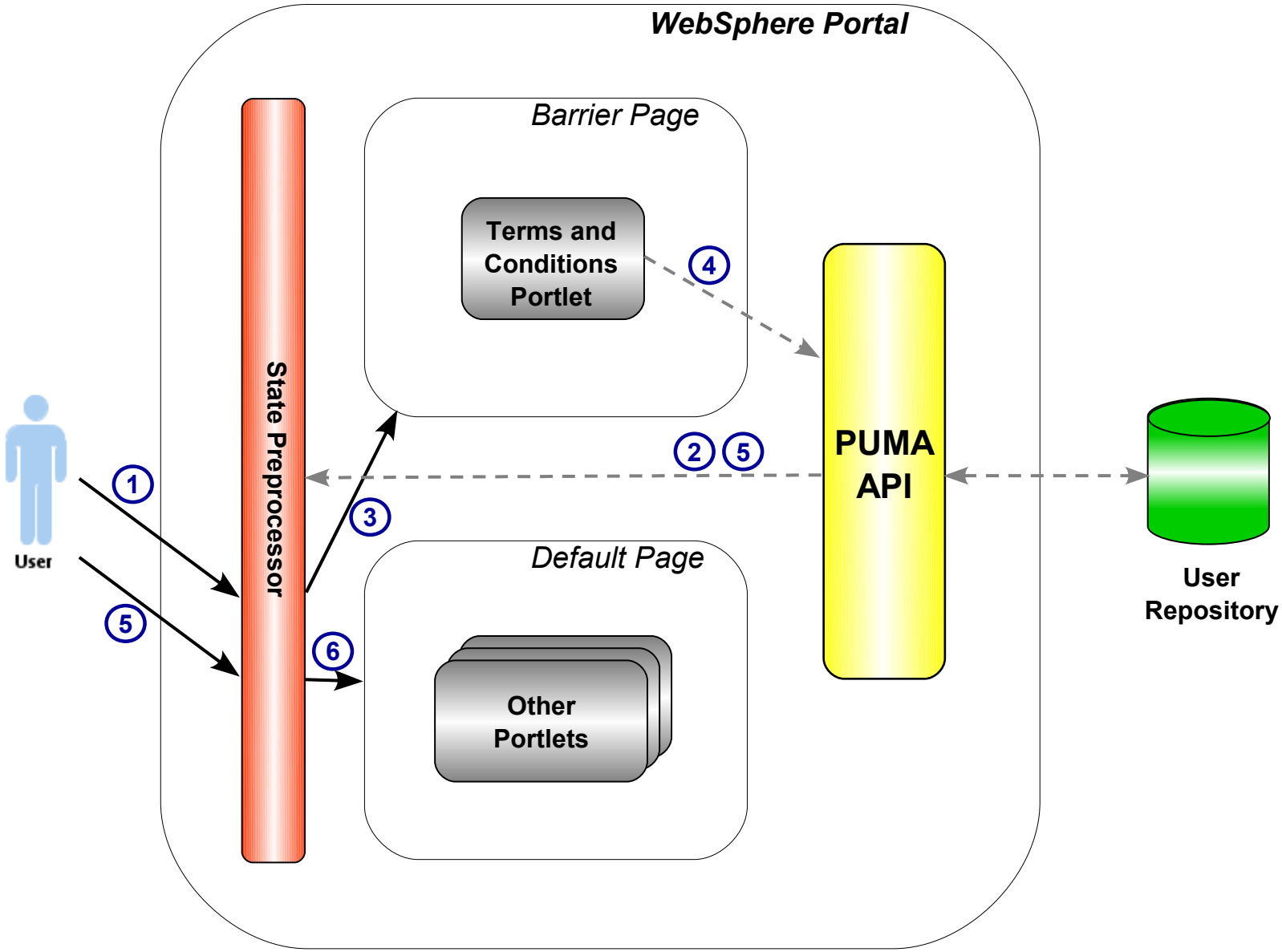
Business problem

- **Your organisation's legal team requires all users to accept a set of terms and conditions before using the portal for the first time**

- ***Variations:***
  - Must accept again every time the terms change
  - Must have different terms for different countries / legal systems
  - Perhaps the user has to enter other information instead:
    - Specify their department / contact information / shoe size

Featured solution

- Use a ***State Preprocessor*** to intercept each request for a portal page and change the target page to a barrier page if the user has not yet accepted the terms and conditions
- Store terms and conditions acceptance by each user in a user attribute, accessed via the PUMA API
- Have a portlet on the barrier page which shows the terms and conditions and sets the attribute if the user accepts


WebSphere Portal
Barrier Page
Terms and Conditions Portlet
State Preprocessor
PUMA API
User
User Repository
Default Page
Other Portlets
1
2
3
4
5
6
5

Solution flow

1) User requests a portal page
2) Preprocessor queries PUMA and finds that the user has not yet accepted the terms and conditions
3) Preprocessor changes the request destination to the barrier page
4) User submits acceptance of the terms and conditions; portlet writes this to PUMA
5) Attempted rendering of the barrier page after submission causes preprocessor to be invoked again; this time PUMA reports that the user **has** accepted the terms and conditions
6) Preprocessor changes the request destination to the default page

Demo

Components of the example

- A simple State Preprocessor that reads and, if necessary, modifies the navigation selection state
  - Uses Portal State Manager Service to manipulate state
  - Uses PUMA Service to check whether user has accepted terms and conditions
- A “Terms And Conditions” portlet on a barrier page
  - Uses PUMA Service to record when user accepts terms and conditions
- Repurpose the standard “departmentNumber” user attribute to store acceptance

State Preprocessor class details

- Must implement interface "*com.ibm.portal.state.PreProcessor*"
- Single method "*process*" taking request, response and current state parameters
- Can perform initialisation in a zero-argument constructor
  - e.g. looking up services
- Deployed as a standalone web application on the portal server
- Configured as a plugin to an extension point named "*com.ibm.portal.model.PreProcessor*"
  - Create a "*plugin.xml*" file and place in the "*WEB-INF*" subdirectory of the web application
- Be very aware of performance and retaining access to the portal
  - Preprocessor called for every page request

Updating the state to target a different page

```
final SelectionAccessorFactory selFactory = stateMgrService
                    .getAccessorFactory(SelectionAccessorFactory.class);

SelectionAccessorController selController = null;
try {

        SelController =
              selFactory.getSelectionAccessorController(stateController);
        selController.setSelection(targetPageUniqueName);

} finally {
        if (selController != null) {
                selController.dispose();
        }
}
```

State Preprocessor plugin configuration

```
<plugin id="com.ibm.issl.example.preprocessor.PUMABasedPreProcessor"
        name="PUMABasedPreProcessor" version="1.0.0"
        provider-name="IBM">

    <extension point="com.ibm.portal.model.PreProcessor"
               id="PUMABasedPreProcessor">

        <provider class=
             "com.ibm.issl.example.preprocessor.PUMABasedPreProcessor">
        </provider>

    </extension>

</plugin>
```

Portlet details

- For user to proceed to home page once terms and conditions accepted, requires the State Preprocessor to be invoked again when portlet form submitted
- This will not happen by default, we need to set the following parameters in "*portlet.xml*":

```
<init-param>
        <name>wps.enforce.redirect</name>
        <value>true</value>
</init-param>

<init-param>
        <name>wps.multiple.action.execution</name>
        <value>true</value>
</init-param>

<init-param>
        <name>com.ibm.portal.state.url.allowrelative</name>
        <value>true</value>
</init-param>
```

Alternative solutions

- **Use of page ordering and visibility rules**
  - Caching by the server means not really dynamic enough
  - What about bookmarked URLs?
- **Add terms and conditions to custom login portlet**
  - Doesn't work with desktop single sign-on
  - Doesn't work with WAS single sign-on if user goes somewhere else to log in first

Future refinements for this solution

- Exclude administration users from redirection to barrier page
  - Already implemented in example for convenience!
- Make the page unique names and user attribute details configurable
- Add a configurable list of pages to be excluded from redirection
  - Such as administration areas
- Use WCM to store terms and conditions text
  - Ease of update
  - Could vary based on some user attribute (e.g. country)
- Store version number of terms and conditions accepted (instead of just “PROCESSED”) so that users can be forced to accept again if updated

Other uses for a State Preprocessor

- Redirection based on factors such as User Agent
- Setting up a context for portlets
  - e.g. translating URL query string parameters to render parameters
- WCM multilingual solution uses it, I believe

Long-running portlets

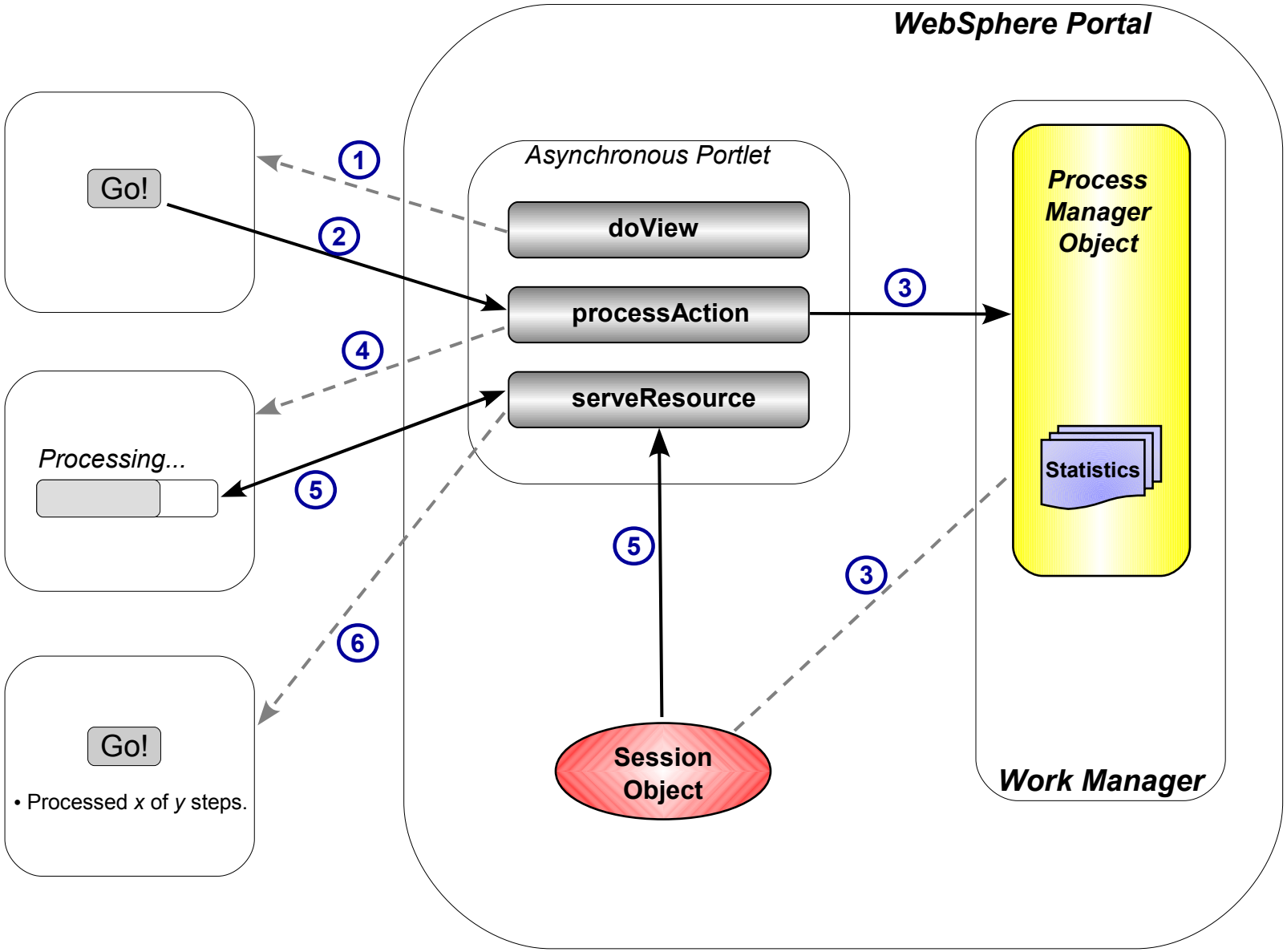
Business problem

- **Portlets need to execute lengthy operations on behalf of users, such as processing batches of updates**

- ***Specifics:***
  - Operation durations exceed server timeouts for user interface tasks
  - User needs to be able to see progress
    - See that the operation has not “hung” or failed
    - Predict the likely finish time
  - This is not something that happens for “normal” portal page accesses
    - Otherwise there is probably something wrong with your portal design!
    - It is likely only privileged users who can run these operations
  - Examples:
    - Register a batch of users
    - Upload content to database
    - Set terms and conditions flag in bulk for users in previous example

Featured solution

- Run the operation asynchronously in a separate thread
- Portlet kicks off execution in "*processAction*" and then returns page to user
- Returned markup includes JavaScript to make Ajax calls to portlet's "*serveResource*" method to poll operation progress
- Uses a WAS Work Manager to provide the worker thread
  - Security context is propagated which is useful if, for instance, accessing PUMA in the asynchronous operation


WebSphere Portal
Asynchronous Portlet
doView
processAction
serveResource
Process Manager Object
Statistics
Work Manager
Session Object
Go!
Processing...
Go!
• Processed x of y steps.
1
2
3
4
5
5
3
6

Solution flow

1) Portlet renders initial page containing a means to kick off the long operation (i.e. a button in this case)
2) Button sends action request to portlet
3) Portlet creates “process manager” object and submits to work manager for execution; reference stored in session object to allow progress tracking
4) Page returned with button hidden and progress bar shown instead
5) JavaScript in page makes Ajax calls to portlet to get current progress and update progress bar; portlet in turn queries statistics of process manager via reference in session object
6) When completion of the task is detected a message is displayed and the button unhidden again

Demo

Alternative solutions and variations

- **Extend necessary server timeouts**
  - Difficult, the server is reluctant to allow more than 180 seconds for user interface operation
  - No feedback on operation progress to user when synchronous
- **Create own threads in portlet**
  - Not recommended by JEE
  - No automatic security propagation
  - No automatic pooling to limit server load
- **Schedule batch work outside of portlets**
  - Appropriate in many instances, but:
    - Not user initiated
    - Requires additional technologies, e.g. WAS Scheduler, EJBs

Executing the asynchronous workload

```
// Get the work manager
InitialContext ctx = new InitialContext();
WorkManager wm  = (WorkManager) ctx.lookup(DEFAULT_WORK_MANAGER);


// Create an object to do the processing and store it in the
// session to be available to future calls to the portlet
AsynchProcessManager manager = new ExampleProcessManager();
sessionBean.setProcessManager(manager);


// Use the work manager to run the processing on a separate thread
WorkContainer updater = new WorkContainer(manager);
wm.startWork(updater);
```

Future refinements

- Have the operation actually do something ;-)
- Audit logging of operations
- Creation of a plugin architecture to allow many different operations to be configured into the framework

Redirecting users on logout

Business problem

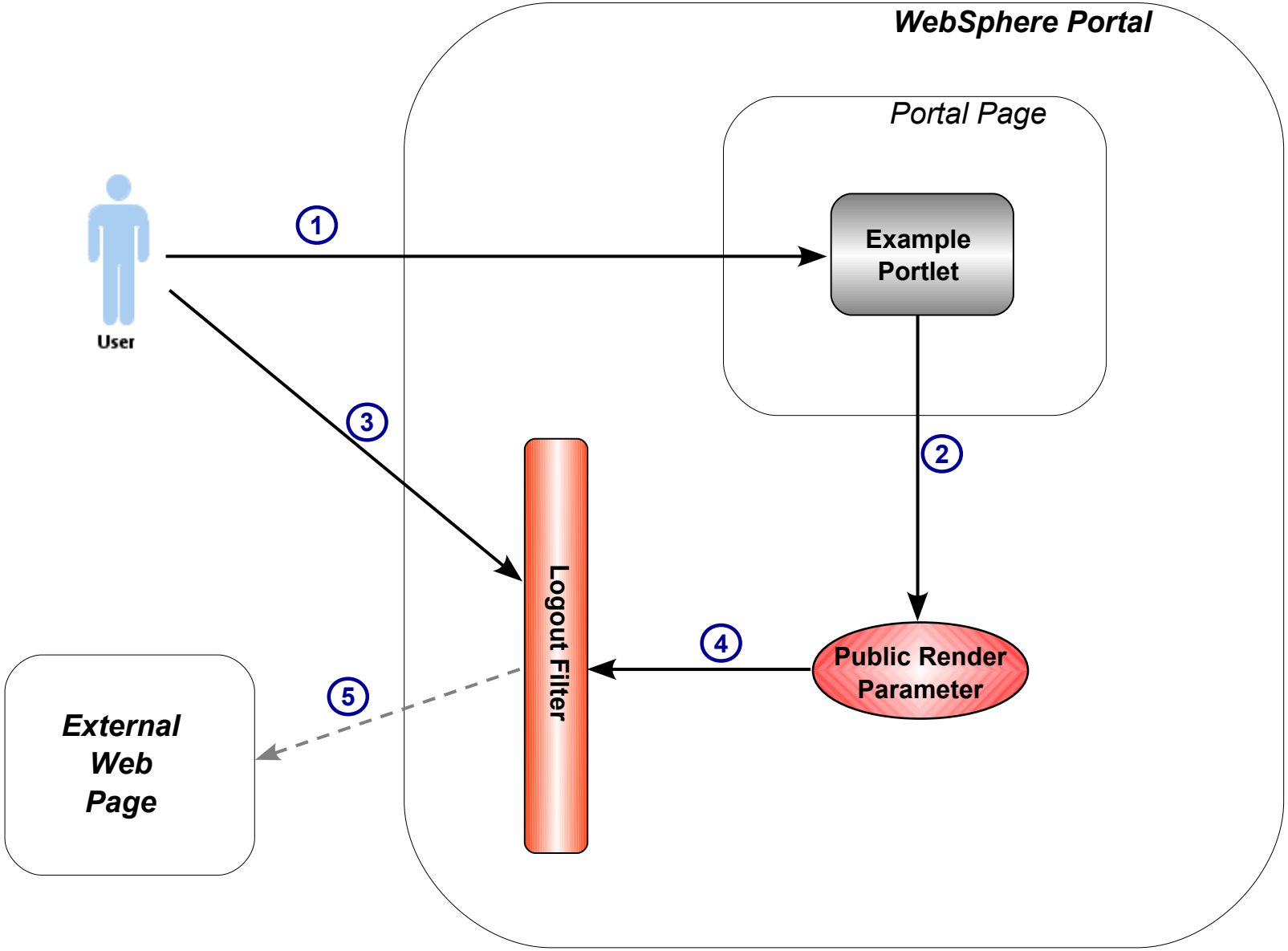
- **You need to send users somewhere other than the home page (or the login page or the "Your session has expired" screen) when they logout from Portal**

- ***Specifics:***
  - "Somewhere" is probably a page external to Portal
  - Can be independent of whether logout is explicit (user clicked the link) or implicit (session timed out) or different for the two scenarios
  - Destination may be dependent on user attributes or current state
  - Could be the login page for an external authentication manager, for example

Featured solution

- Create an *implicit* and / or *explicit logout filter* for Portal
- Reads a specified *public render parameter* using the Portal State Manager Service
- If parameter found, redirects the user to the URL it contains on logout

- Create a portlet ("*PRPLogoutPagePortlet*") that allows the user to enter a URL, which is then saved in the public render parameter


WebSphere Portal
Portal Page
Example Portlet
User
1
2
3
4
5
Logout Filter
Public Render Parameter
External Web Page

Solution flow

1) User indicates desired logout URL in portlet form
2) Portlet stores URL as public render parameter
3) User clicks logout link or accesses portal after session timeout; filter is invoked
4) Filter retrieves URL from public render parameter
5) Filter sends redirect URL to user's browser

Demo

Redirecting in the filter

```
if (logoutURLString != null && logoutURLString.length() > 0) {
        String redirectTarget = URLDecoder.decode(logoutURLString,
                                                  ENCODING_TYPE_UTF_8);
        // Set the redirect target
        if (redirectTarget != null) {
                portalLogoutContext.setRedirectURL(redirectTarget);
        }
}
```

Deploying and configuring the filter

- Filter is packaged as a JAR file and deployed to the Portal server's "*shared/app*" directory
- In the WAS console, navigate to:

  Resources – Resource Environment – Resource Environment Providers – WP Authentication Service – Custom Properties

  and create (or extend if they exist) the properties:

  logout.implicit.filterchain
  logout.explicit.filterchain

  each with the fully qualified class name:

  com.ibm.issl.example.portal.filter.PRPLogoutFilter

Future refinements

- Configure the redirection URL based on some characteristic of the user, rather than having them select it

Questions and discussion