Vývoj portletů – 2. část

František Hartman

Aleš Rybák

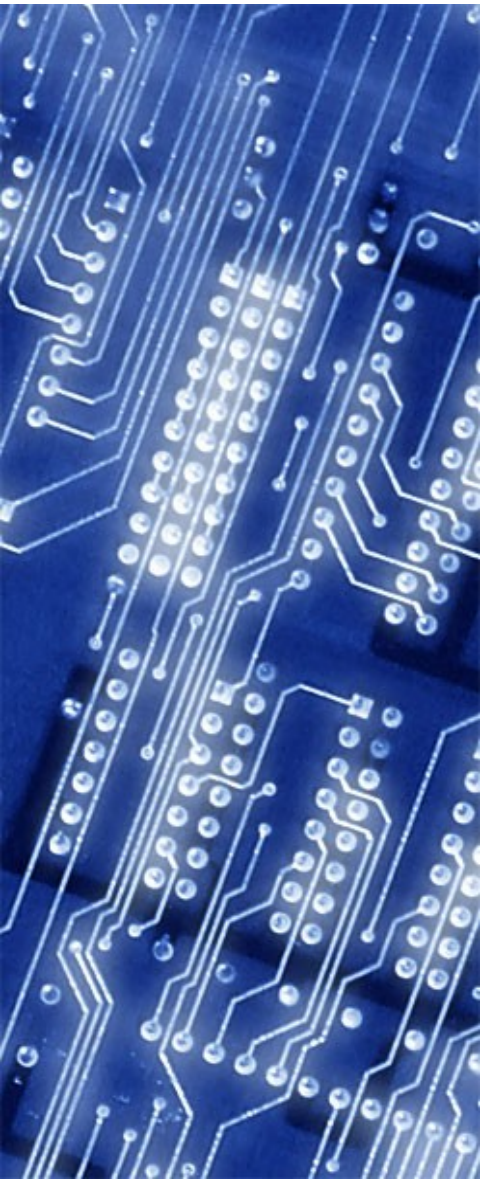
IBA CZ, s.r.o.

Vývoj portletů

Agenda

- Komplexní portlety
- Obsluha zdrojů (resource serving)
- Meziportletová komunikace (IPC)
  - Sdílená session
  - Veřejné parametry (public render parameters)
  - Události (events)

Komplexní portlety



- Více pohledů
- Více akcí
    - CRUD
    - Administrační aplikace
- Nastavení
- Velké množství vstupních dat
- Složitá aplikační logika

Více pohledů

- Využít metodu *doView()* k rozdělování požadavků
  - na základě příchozích parametrů
  - podle uživatelského nastavení
- Připravit zobrazovaná data v portletu
- Využit JSP (nebo jinou technologii) k důslednému oddělení view vrstvy

Více pohledů - ukázka

JSP:

```
<portlet:renderURL var="listURL">
   <portlet:param name="<%= PARAM_VIEW%>" value="<%=VIEW_LIST%>"/>
</portlet:renderURL>

<a href=“${listURL}“>Show list</a>

...

<portlet:renderURL var="detail">
   <portlet:param name="<%= PARAM_VIEW%>" value="<%= VIEW_DETAIL%>"/>
   <portlet:param name="<%= PARAM_ID%>" value="${product.id}"/>
</portlet:renderURL>

<a href=“${detailURL}“><c:out value=“${product.name}“/></a>
```

Více pohledů - ukázka

Portlet:

```
public static final String PARAM_VIEW = “view”;
public static final String VIEW_DETAIL = “detail”;

public void doView(RenderReq., RenderResp.).. {
    String view = req.getParameter(PARAM_VIEW);
    if (VIEW_DETAIL.equals(view)) {
        doViewDetail(request, response);
    } else {
        doViewMain(request, response);
    }
}
protected void doViewDetail(RenderReq., RenderResp.).. {
    ...
}
protected void doViewMain(RenderReq., RenderResp.).. {
    ...
}
```

Více akcí

- Využit metodu processAction() k rozdělování požadavků – JSR-168
- **Využit anotace k rozdělování požadavků** – JSR-286, Java 5+
- V akci
  - provést všechny změny stavu portletu
  - nastavit render parametry
  - odeslat události

Více akcí - ukázka

JSP:

```
<portlet:actionURL var="searchURL" name=“<%=ACTION_SEARCH %>“ />

<form action="${searchURL}" method="post"> .... </form>

...

<portlet:actionURL var="addToCartURL" name="<%= ACTION_ADD_TO_CART %>">
   <portlet:param name="<%= PARAM_ID %>" value="${product.id}" />
</portlet:actionURL>

<a href=“${addToCartURL}“><img src=“..“></a>
```

Více akcí - ukázka

Portlet:

```
...

@ProcessAction(name=ACTION_SEARCH)
public void actionSearch(ActionRequest request, ActionResponse response) throws
     PortletException, IOException {
    ...
    response.setRenderParameters(request.getParameterMap());
}

@ProcessAction(name=ACTION_ADD_TO_CART)
public void actionAddToCart(ActionRequest request, ActionResponse response)
     throws PortletException, IOException {
    ...
}
```

Konstanty, nastavení, pomocné metody

- Programové konstanty – PortletNameConstants (např. CatalogConstant)
- Nastavení
  - Init parameters
    - portlet.xml
    - web.xml
  - Preferences
- Pomocné metody
  - Odstranění duplikace kódu
  - Vede ke snadnější udržovatelnosti
  - Dědičnost vs. Util třídy

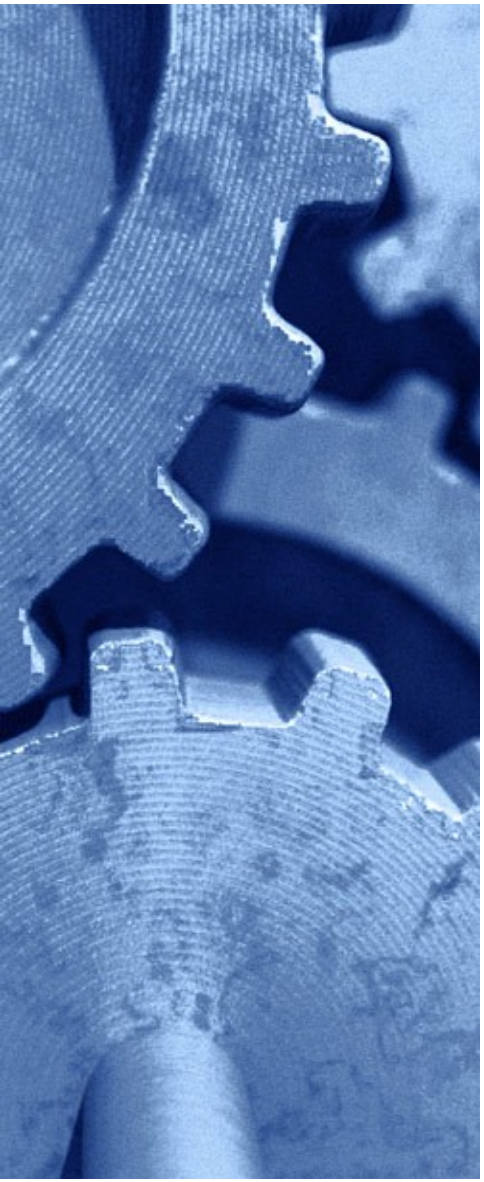
Použití MVC

- V MVC je
  - Portlet = Controller
  - JSP = View
  - Bussiness vrstva (DTO) = Model
- Velké množství vstupních dat
  - Validace není součástí specifikace
  - Zvážit použití MVC rámce, zjednoduší:
    - Validaci dat
    - Převod dat z formuláře na objekty
- Složitá aplikační logika
  - Patří do modelu
  - Portlet by měl obsahovat pouze logiku UI

Komplexní portlety

- **Velmi složitý portlet může naznačovat špatný návrh a portletovou dekompozici**
- Zkontrolovat návrh aplikace
  - Různé portlety pro různé role
  - Portlety podle případu užití
- Zkontrolovat duplikaci kódu
- Zvážit možnosti zjednodušení či rozložení funkcionality do více portletů
- Zvážit využití MVC rámce

Poskytování statických zdrojů



- PortletResponse.encodeURL()

```
<img src="<%= renderResponse.encodeURL(
    renderRequest.getContextPath() + “/img/cart.png“)%>" >
```

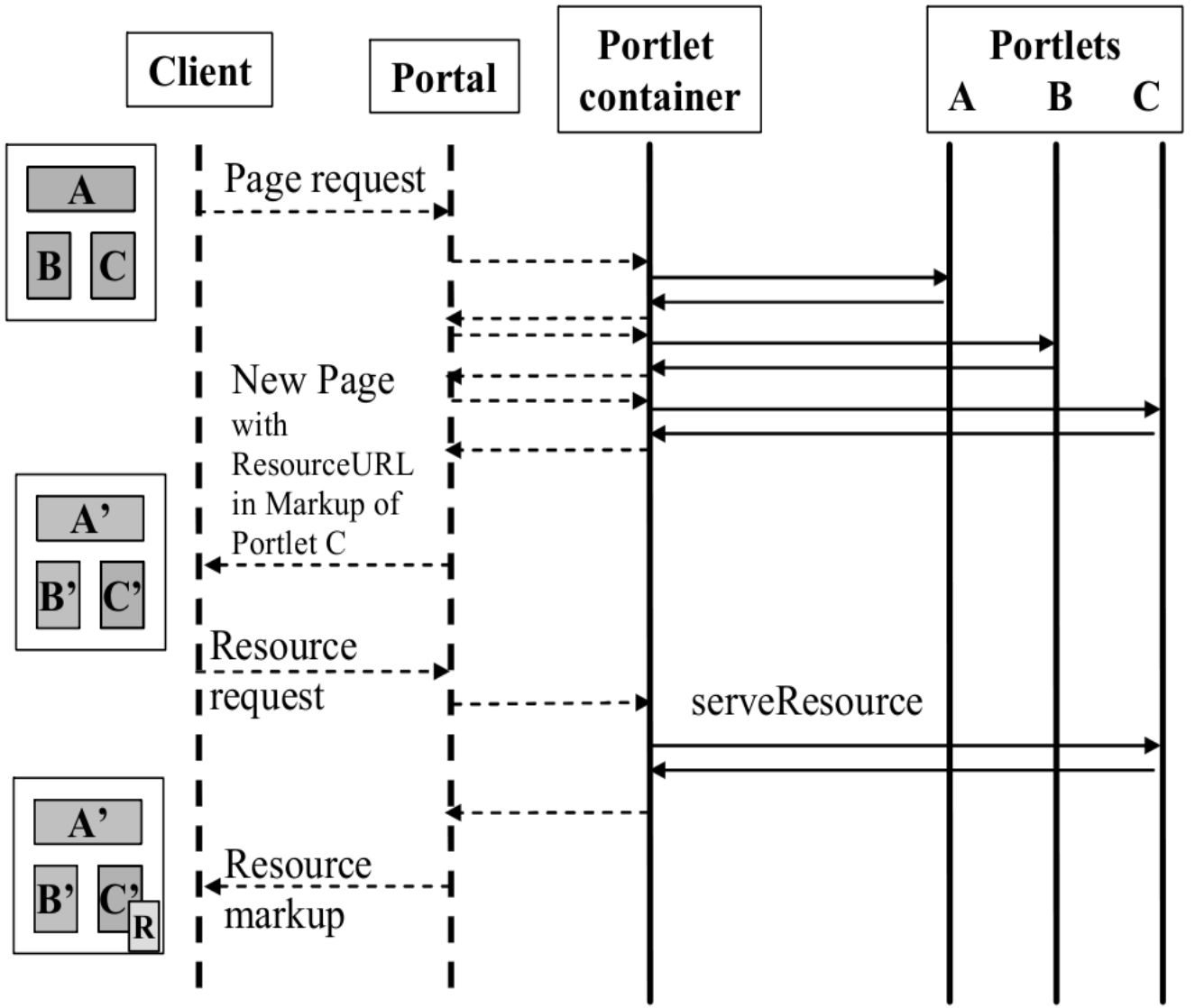
- Nemusí vygenerovat validní URL
- Vyšší výkon – nedochází k zpracování portálem
- Doporučený způsob pro statický obsah v JSR-168
- Cachované resourceURL v JSR-286

```
<portlet:resourceURL var="detailIconURL" id="/img/cart.png"
    cacheability="cacheLevelFull" />
```

Poskytování dynamických zdrojů

- JSR-168
  - Nemožné získat pouze fragment kódu nebo dynamická binární data
  - Řešení – servlet
    - Problémy s autentizací
    - Problémy se získáváním dat z portletu (preference, uživatel atd.)
- JSR-286
  - Rozhraní ResourceServingPortlet s metodou serveResource()
  - Klient může posílat požadavky na neagregovaná data
    - Binární data (obrázky, dokumenty, ...)
    - Serializovaná data (JSON, …)
    - HTML fragmenty
  - Často také ve spojení s technologií AJAX

Sekvence požadavků při poskytování zdroje



Dynamické zdroje - implementace

- Metoda serveResource()
  - Neměla by měnit stav portletu (databáze), pokud je volána metodou GET
- ResourceRequest
  - getResourceID()
  - getCacheability()
    - FULL, PAGE, PORTLET
- ResourceResponse
  - setContentType()
  - getWriter()
  - getPortletOutputStream()

Dynamické zdroje - ukázka

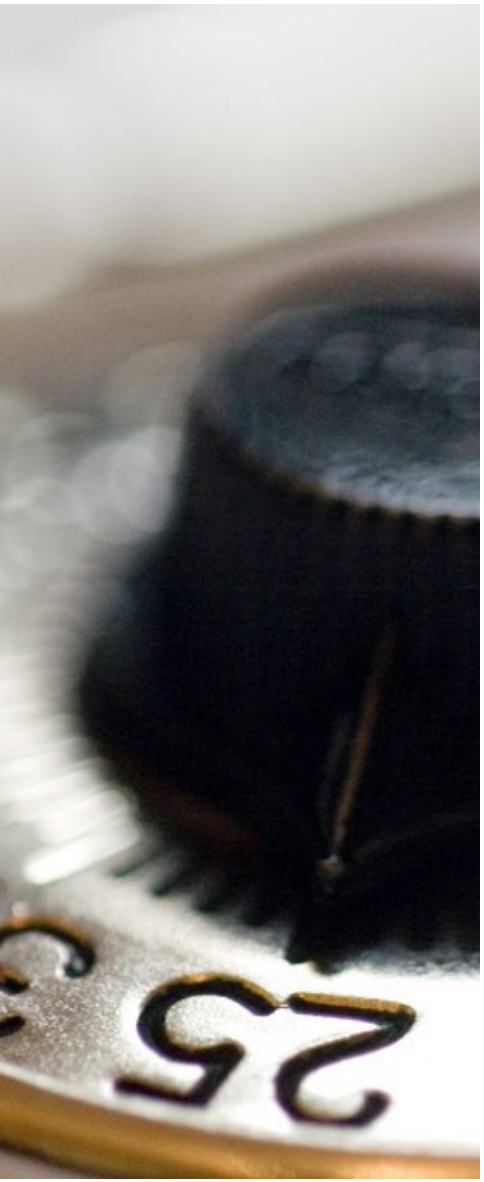
JSP:

```
<portlet:resourceURL var="queryURL" id="<%= RESOURCE_QUERY %>"/>

<portlet:resourceURL  var=“searchIconURL“ id="<%= “/img/detail.png“ %>"/>
<img src=“${searchIconURL}>

...
```

Portlet:

```
@Override
public void serveResource(ResourceRequest request, ResourceResponse response) throws
      PortletException, IOException {
    String resourceID = request.getResourceID();
    if (RESOURCE_QUERY.equals(resourceID)) {
        ...
    } else {
        super.serveResource(request, response);
    }
}
```

Meziportletová komunikace (IPC)



- JSR-168
  - Nemá standardizovanou cestu pro meziportletovou komunikaci
  - Řešení
    - Sdílení dat v „application scope“ session
    - JavaScript
    - Databáze
    - Proprietární řešení
- JSR-286
  - Veřejné parametry (public render parameters)
  - Události (events)
  - (Cookies)

Session – application scope

- Využívá se http session
  - PORTLET_SCOPE
  - APPLICATION_SCOPE
- Sdílení dat v rámci portletové aplikace
- Nelze sdílet mezi aplikacemi

- Změny provádět během zpracování akcí či událostí
- Pozor na změny dat session v render fázi
- Pozor na kolize jmen

Public render parameters

- Nejjednodušší standardní metoda meziportletové komunikace
- Parametry se definují v deskriptoru portlet.xml
  - Parametr je definován jako veřejný
  - Pro každý parametr je nutné definovat jedinečné jméno
  - Pro portlet se definuje který z veřejných parametrů využívá
- Veřejné parametry jsou dostupné ve všech částech životního cyklu požadavku
- Pouze řetězcové hodnoty

Public render parameters - ukázka

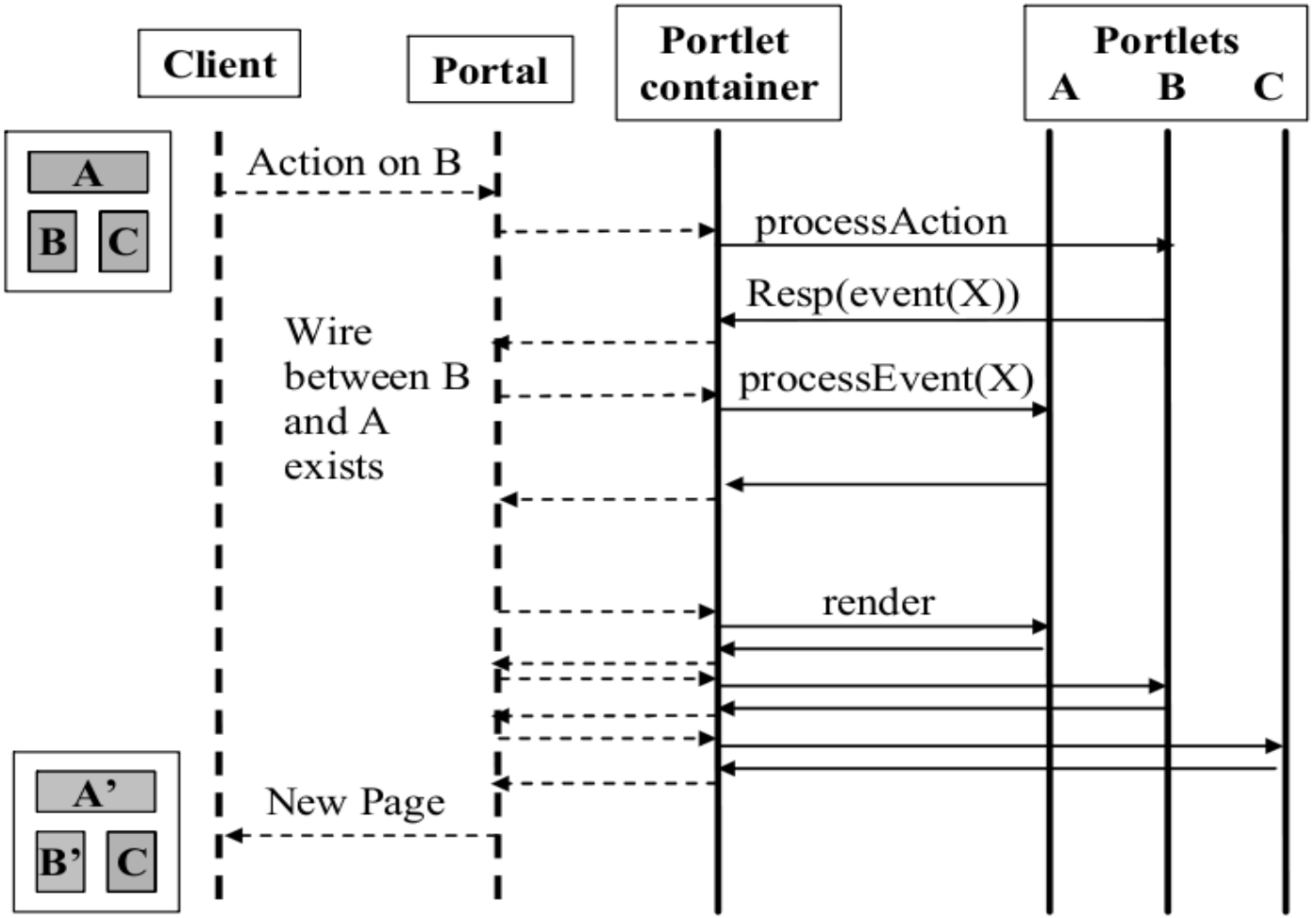
portlet.xml:

```
<portlet-app>

   <default-namespace>http://pv230.ibacz.eu/</default-namespace>

   <public-render-parameter>
      <identifier>param1</identifier>
      <name>p1</name>
   </public-render-parameter>

   <portlet>
      ...
      <supported-public-render-parameter>
         param1
      </supported-public-render-parameter>
   </portlet>
</portlet-app>
```

Události - events

- Události umožňují portletům reagovat na akce nebo změny stavu, které nesouvisí přímo s interakcí uživatele s portletem
- Portlety mohou reagovat na různé události a měnit svůj stav
  - Např. událost přidání položky do košíku
- Události mohou být vysílány jak portálem tak portlety
- Zpracování událostí probíhá před render fází

Události - events



Události - events

- Události jsou navrženy podle vzoru Producer – Listener
  - Portlety mohou vytvářet události
  - Portlety mohou události přijímat (portlet musí implementovat rozhraní EventPortlet)
- Události se mohou řetězit
- Událost je objekt
- Události je možné posílat mezi portlety různých portletových aplikací
  - Pozor na umístění třídy události – možné problémy s classloaderem

Události - events

- Nastavení se provádí v deskriptoru portlet.xml
  - Každá událost má jedinečné jméno
  - Pro portlety se nastavuje, které události vysílá a které přijímá
- Velmi mocné
  - Mohou zvýšit složitost projektu
  - Jedná se prakticky o interface, který portlet poskytuje a k němuž mohou přistupovat portlety třetích stran

Události - events - ukázka

portlet.xml:

```
<portlet-app>
   <default-namespace>http://pv230.ibacz.eu/</default-namespace>

   <event-definition>
      <name>event1</name>
      <value-type>java.lang.String</value-type>
   </event-definition>

   <portlet>
      ...
      <supported-publishing-event>
         <name>event1</name>
      </supported-publishing-event>

      <supported-processing-event>
         <name>event1</name>
      </supported-processing-event>
   </portlet>
</portlet-app>
```

Události - events - ukázka

portlet:

```
@ProcessAction(name=ACTION_ADD_TO_CART)
public void actionAddToCart(ActionRequest request, ActionResponse response) throws
      PortletException, IOException {
    response.setEvent("event1", "event value");
}


@ProcessEvent(name="event1")
public void processEvent1(EventRequest request, EventResponse response) throws
      PortletException, IOException {
    Event e = request.getEvent();
    String value = (String) e.getValue();
    ...
}
```

Události vs. Veřejné parametry

Veřejné parametry

- Výhody
  - Jednoduché a účinné
  - Pracuje s renderURL (záložky, caching)
- Použití
  - Zobrazení příbuzných informací v několika portletech na stejné stránce
  - První volba pro IPC
- Co nedělat?
  - Nastavit všechny parametry jako veřejné

Události

- Výhody
  - Velmi mocné
  - Volnost v jejich použití a možnost řetězení
- Použití
  - Když nestačí veřejné parametry
  - K propagování stavu napříč portlety
- Co nedělat?
  - Používat události jako message system