Vývoj portletů

Miroslav Ligas <miroslav.ligas@ibacz.eu>

Důvěrné

Veškeré materiály získané v rámci školení jsou klasifikovány jako obchodní tajemství společnosti IBA CZ, s.r.o. Jsou důvěrné a určené pouze pro účastníky školení. S informacemi v nich obsažených je nutné nakládat ve smyslu ochrany obchodního tajemství a uzavřené dohody o ochraně důvěrných informací (NDA).

Materiály jsou duševním vlastnictvím společnosti IBA CZ, s.r.o. Veškerá práva jsou vyhrazena.

Organizační záležitosti

- Časový rozvrh
- Motivace
  - Body
- Kooperace a komunikace
- Otázky a odpovědi

Agenda

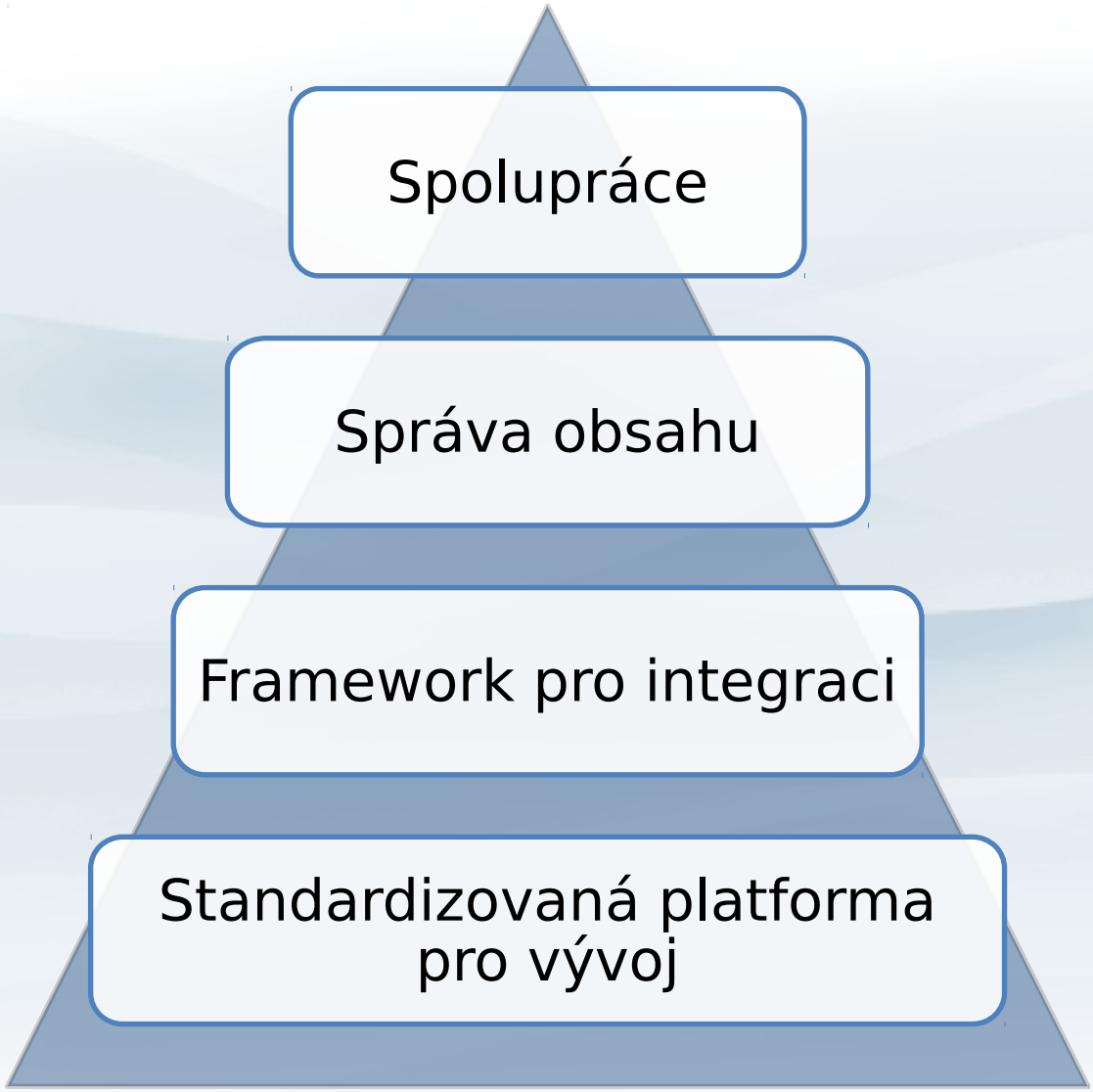
- Úvod do vývoje portletů
- Portletové specifikace
- JSR-168 / JSR-286
  - API objects
  - Tag Library
  - Inter-portlet communication
  - Resource serving
- Spring Portlet MVC
- Portal development best practices

Handshake

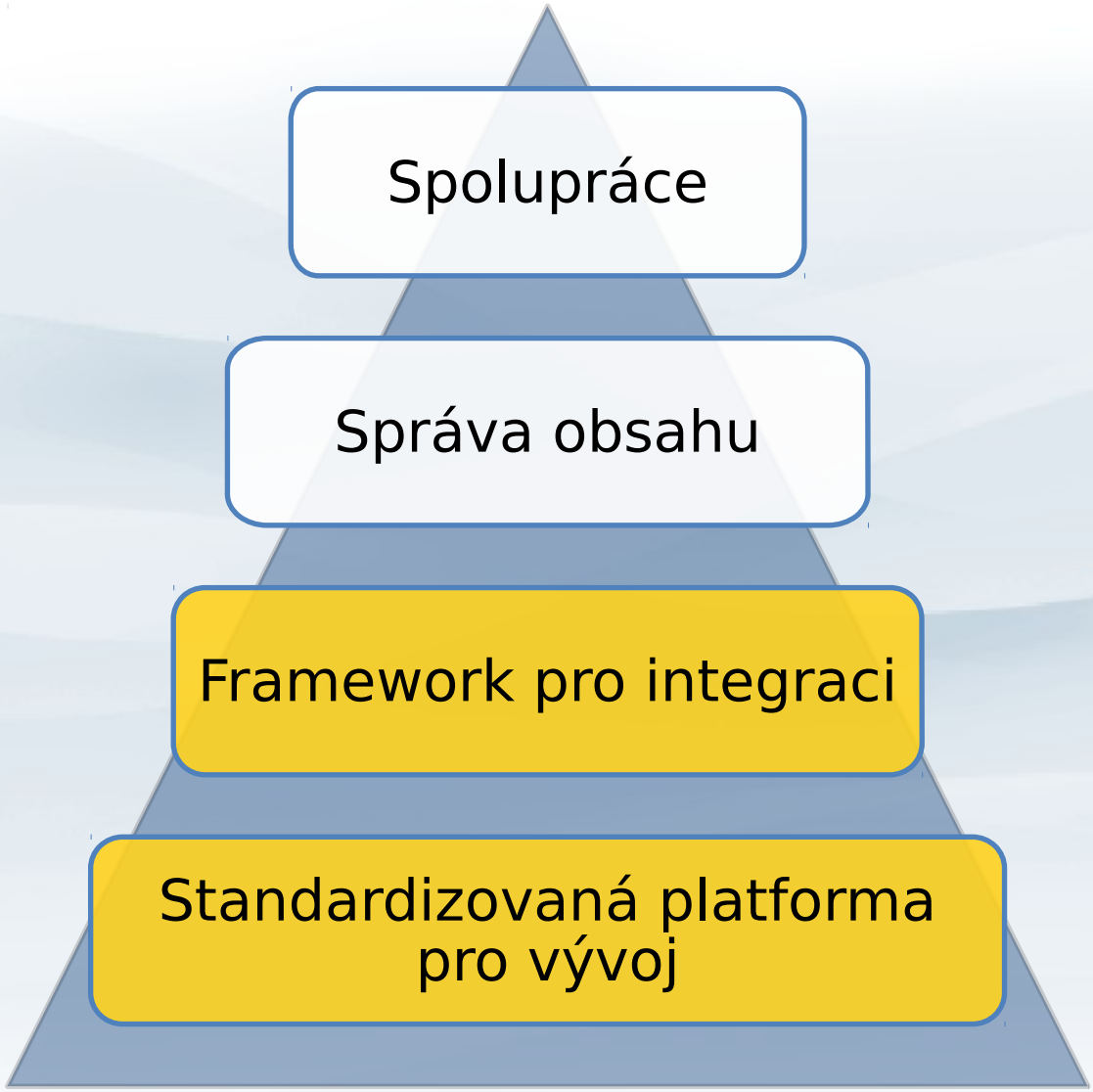
- Oblíbené vývojové prostředí

Úvod do vývoje portletů

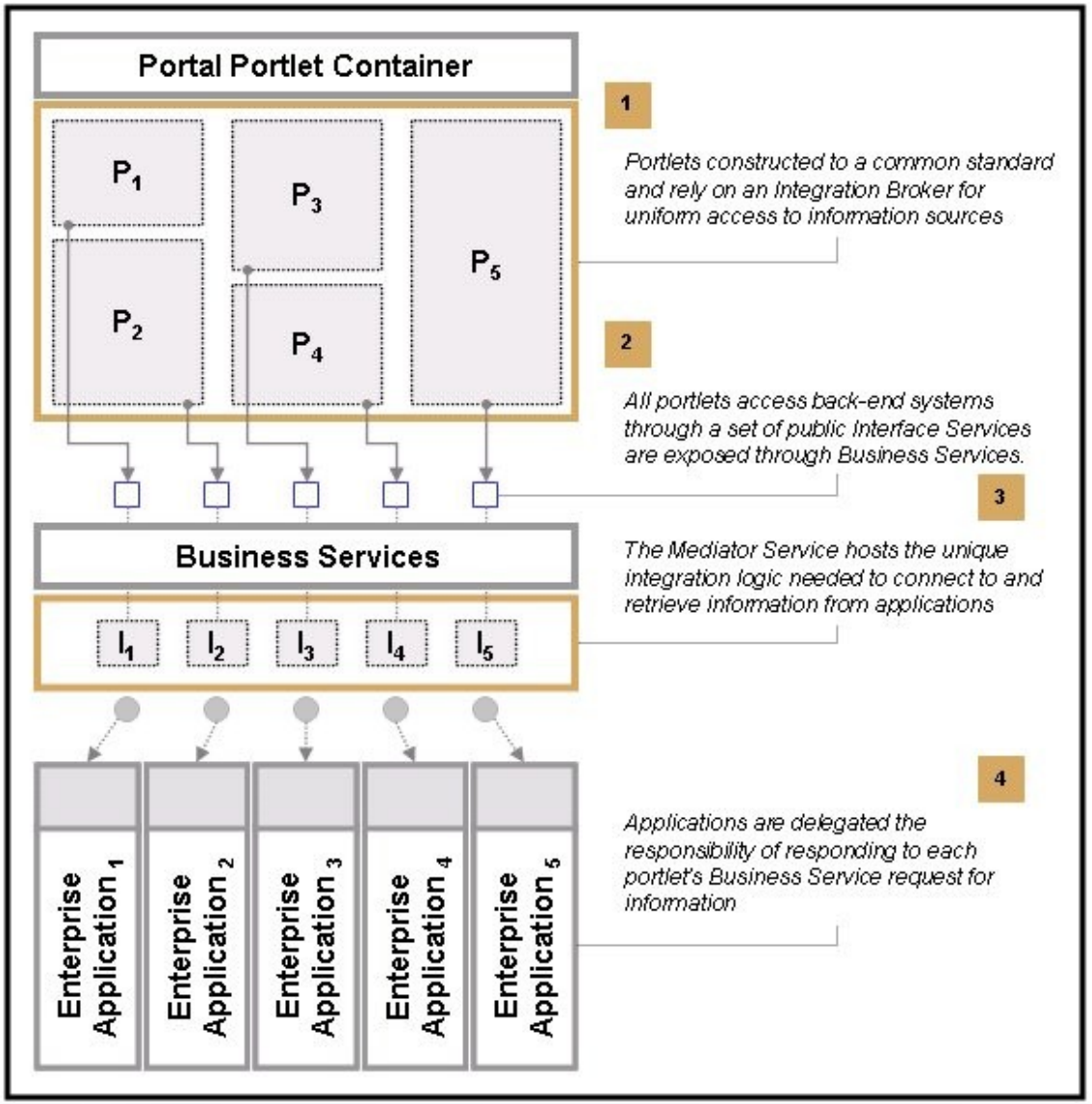
Jak lze portál využít?



Co nás dnes zajímá?



Integrační platforma



Vývojová platforma

- Vývoj nových aplikací
- Integrace existujících aplikací
- Rozšiřování funkcionality

Vývojová platforma

Portálová platforma obstarává pro vývojáře standardizované prostředí

- Správa uživatelů, rolí, oprávnění, autorizace, autentikace
- Správa vzhledu, rozložení
- Hierarchie prostorů a stránek
- Cache
- Personalizace stránek
- Ukládání nastavení
- Další API (většinou proprietární)
  - Vyhledávání
  - Tagy
  - Social

Co zbývá?

Portletové standardy JSR-168 a JSR-286

Proč je máme?

”Doba kamenná”

- Různí dodavatelé s různými proprietárními API
  - Odlišné termíny a chování komponent
- Poskytovatelé jsou nuceni implementovat různé ”portlety” pro různé portály.
- Zákazníci, kteří vyvinuli většinu svých aplikací pro konkrétní portál jsou ”chyceni” touto technologií.
- Neexistuje jednotná, jednoduchá cesta jak vytvořit plug & play komponenty pro portály.
- Není možná spolupráce mezi portlety a různými portálovými servery

Portlety

- Portletový standard pro lokální java portlety
- Java Portlet Specification – JSR 168
- IBM, Sun
- Cíle
  - Jednoduchý programovací model
  - Přenositelnost
  - Využití a kooperace s technologiemi J2EE
  - Interoperabilita s dalšími Java technologiemi

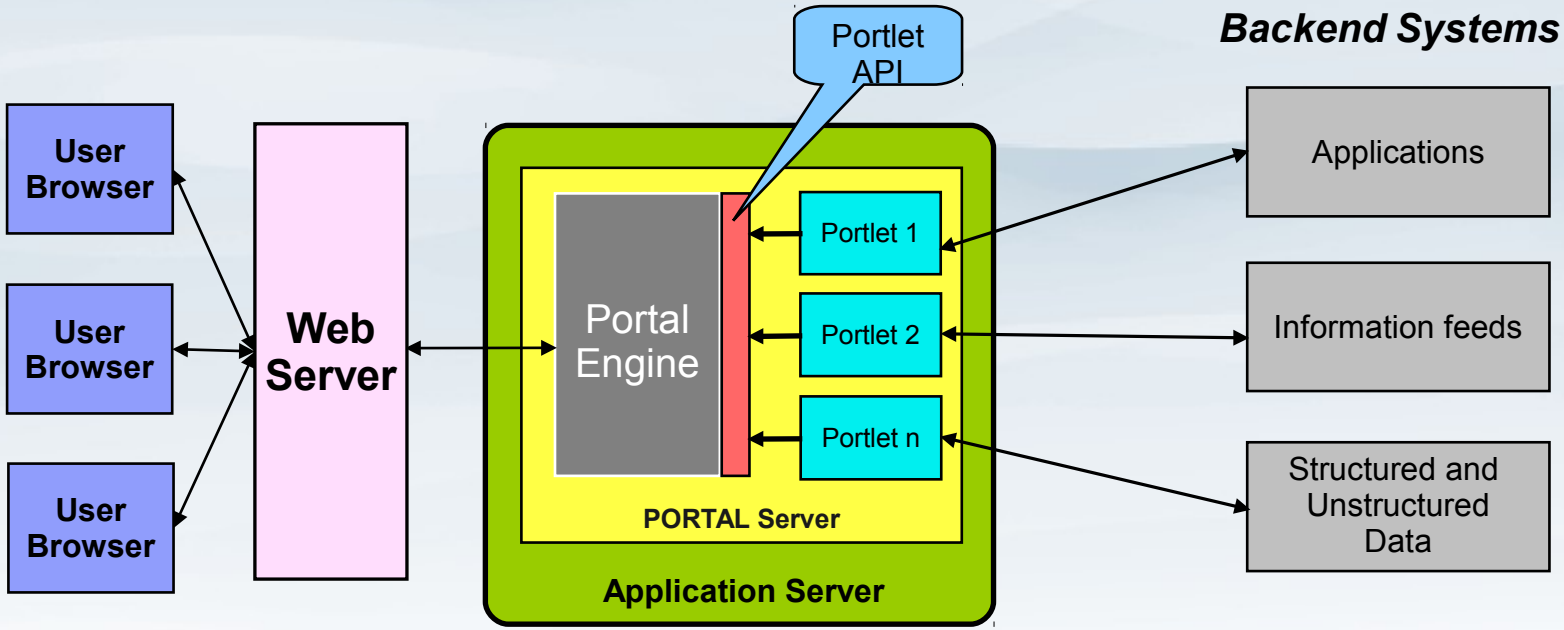
Portlety 2.0

- Definovány v JSR-286
- Rozšíření JSR-168 a zpětná kompatibilita
  - Meziportletová komunikace
    - Veřejné parametry (public render parameters)
    - Události (events)
  - Poskytování obsahu
    - Obrázky
    - AJAX
  - Filtry
  - Zjednodušení itegrace webových rámců

Rozsah specifikací

- Portlet API
- Portlet container
- Kontrakt mezi API a kontejnerem

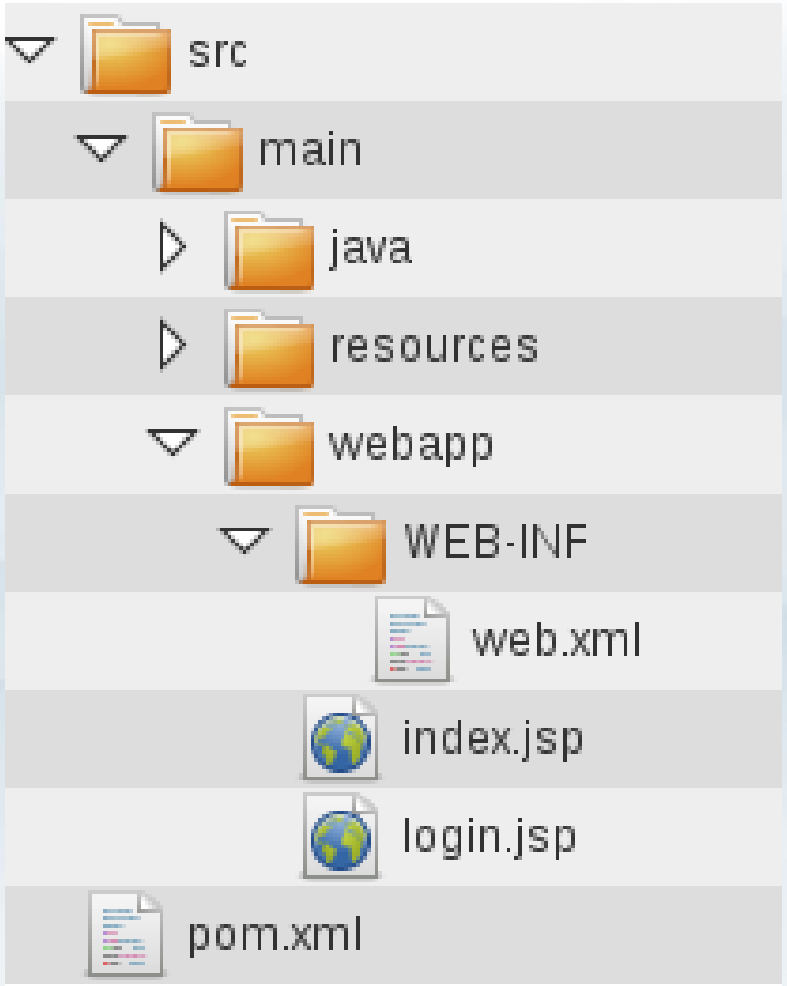
- Struktura portletové aplikace
- Artefakty pro nasazení (WAR)

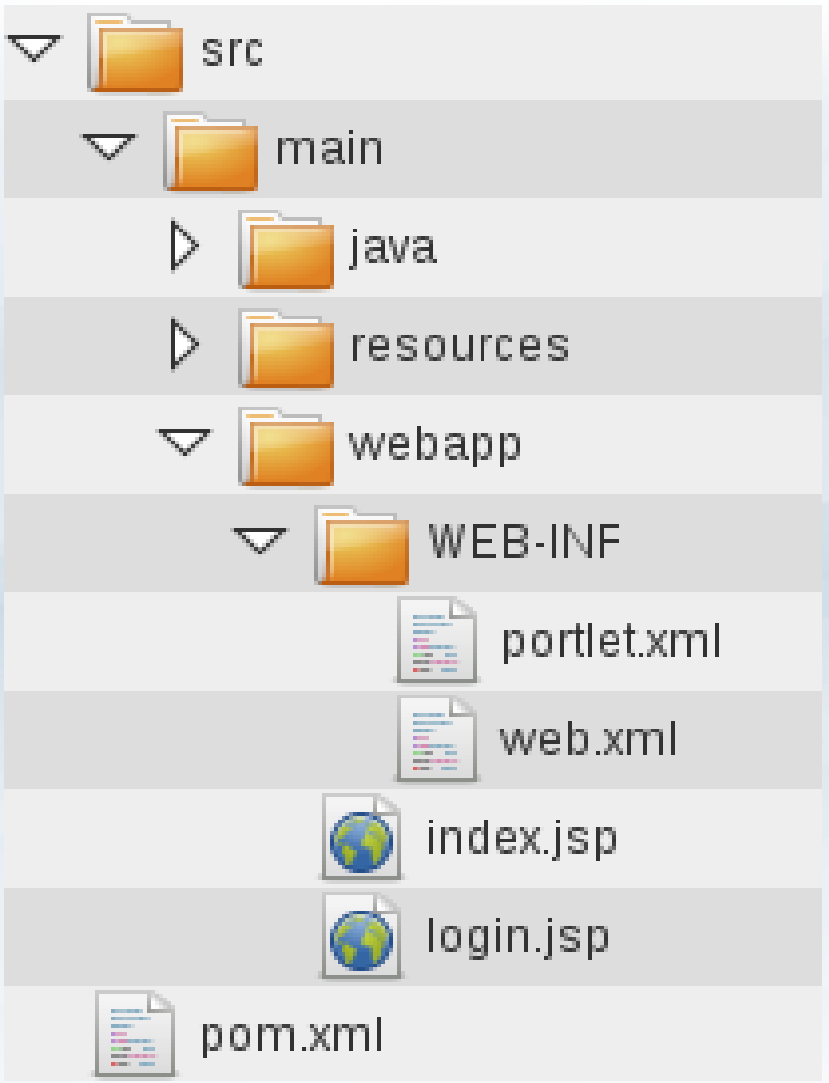


Portlety vs. Servlety

- Portlety jsou webové komponenty
- Portlet API je vytvořeno na základě Servlet API
- Podobné životní cykly (inicializace, zpracovávání požadavků, zničení)
- Portlety oddělují zpracovávání a rendering
- Portlety mají navíc stav:
  - Portlet mode
  - Portlet state
  - Render parameters
  - Portlet preferences
- Portlety poskytují jen část obsahu stránky, servlety jsou tradičně zodpovědné za mark-up na celé stránce.

Struktura portletové aplikace





Portletové aplikace

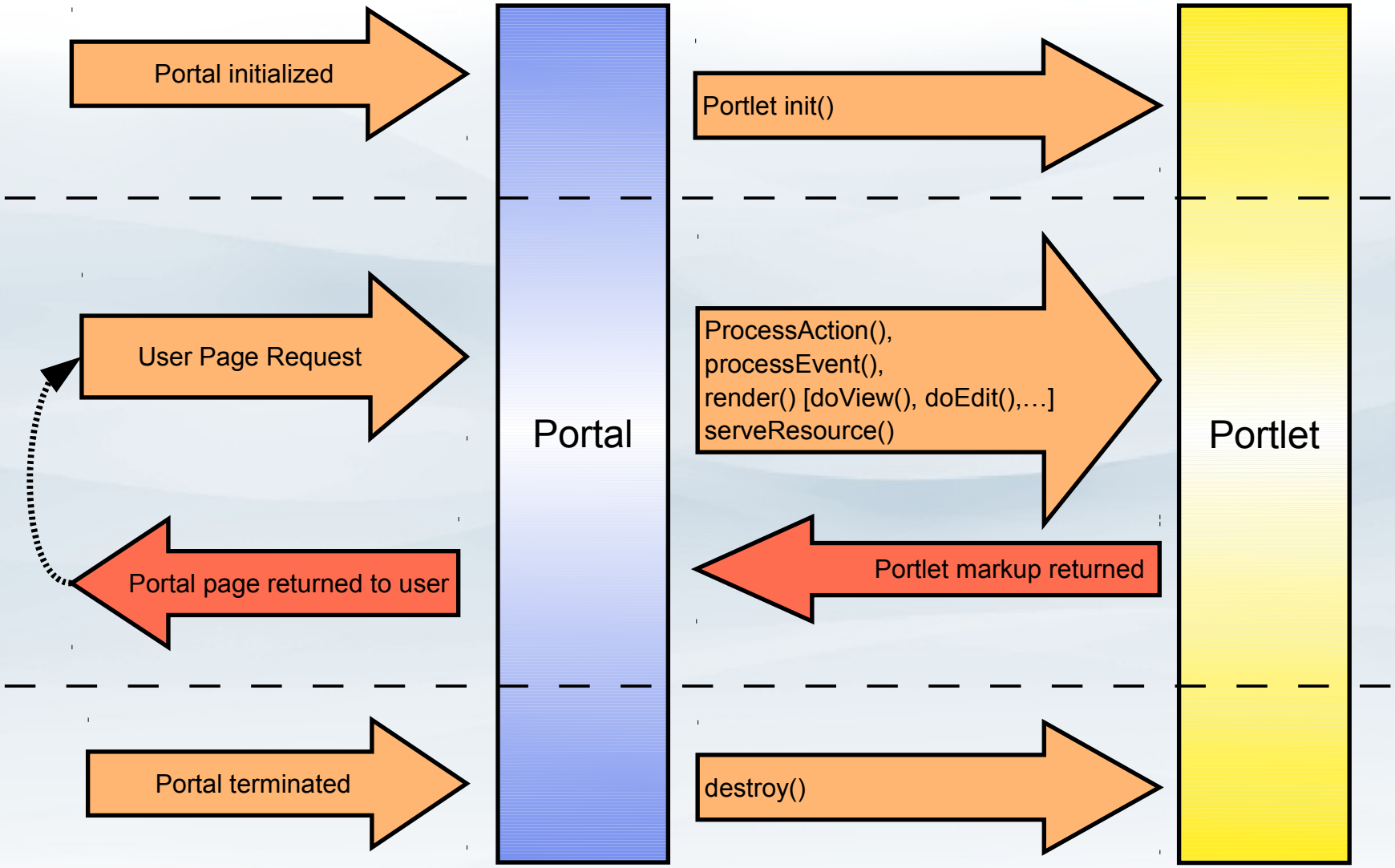
- Portletová aplikace
  - Sada portletů a dalších zrdojů sdílejících stejný *context*
  - Zabalená jako WAR soubor
  - Instalovaná prostřednictvím portálu
- Deskriptory
  - ***web.xml***
    - Nastavení webové aplikace
    - Servlety a JSP stránky
    - Bezpečnostní role
  - ***portlet.xml***
    - Obsahuje všechny informace týkající se portletů
    - Formální definice pomocí XML schématu
  - Další „vendor specific“ deskriptory

portlet.xml

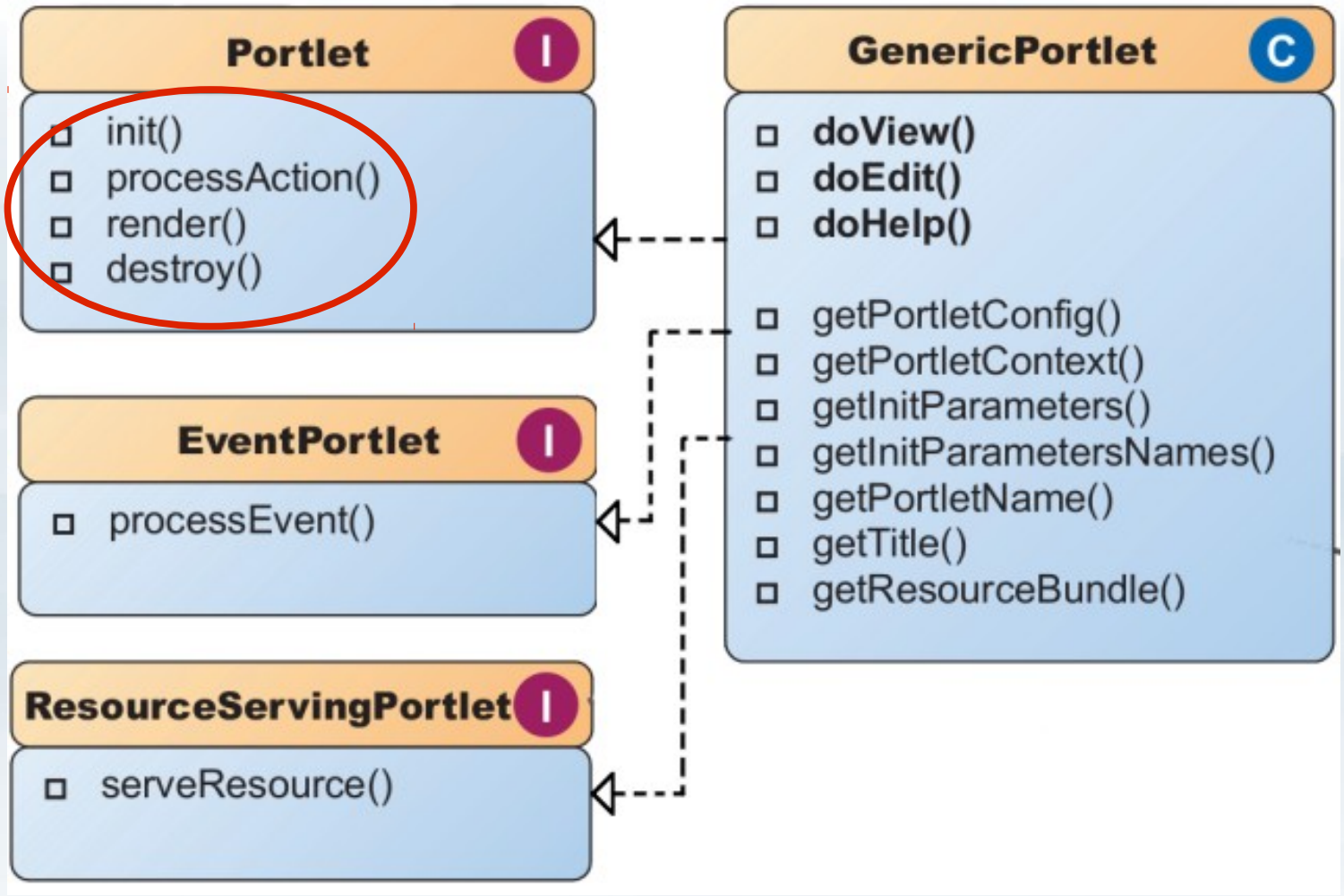
- Obsahuje konfigurační informace přímo pro potrál
  - Typy značkovacích jazyků podporavoné portletem
  - Podporované módy a stavy oken
  - Podporované jazyky
  - Popis a označení portletu
  - Konfigurace preferencí
  - Veřejné parametry
  - Události
  - Filtry
  - A další

Životní cyklus portletu a požadavků

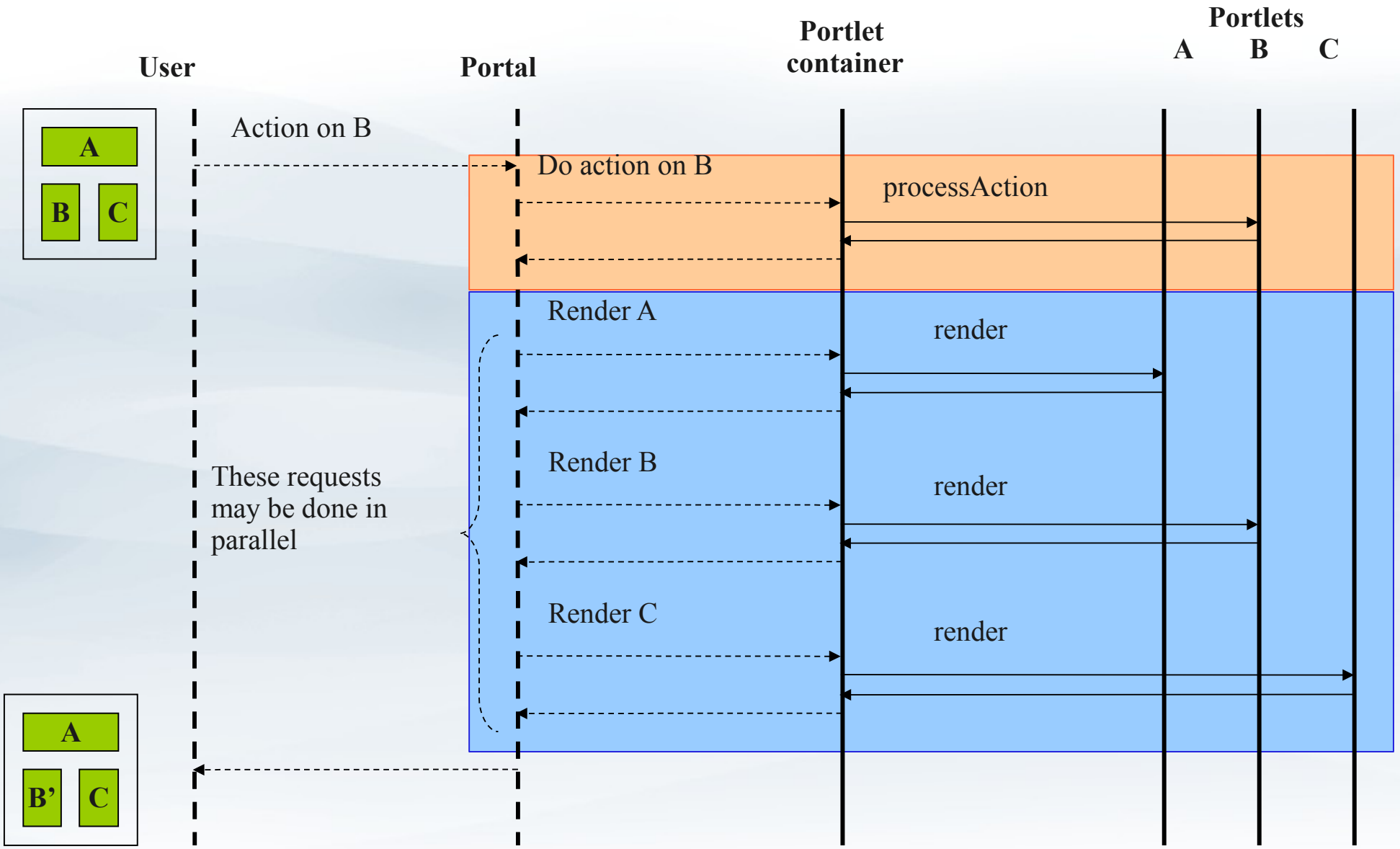
Životní cyklus portletu



Životní cyklus portletu



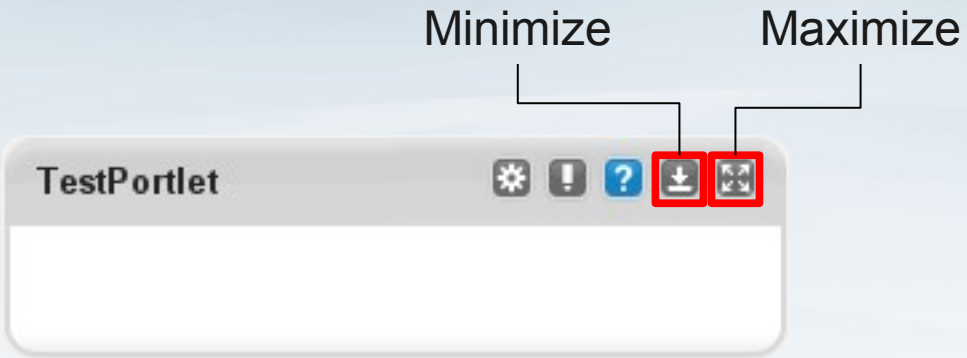
Zpracování požadavku



Použití portletu

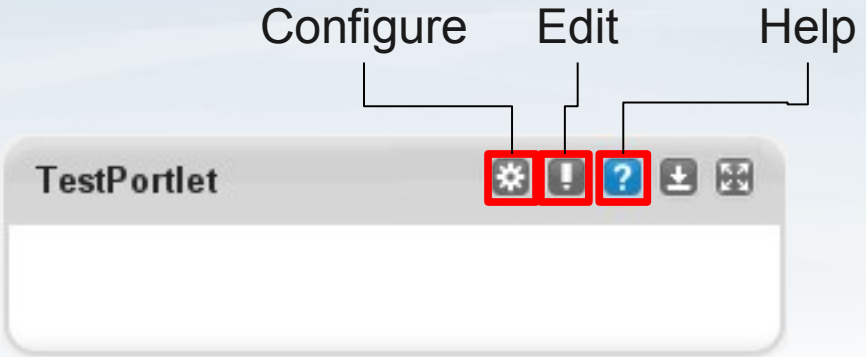
Stav okna portletu (window state)

- Stav okna definuje, jak bude portlet zobrazen na stránce
  - *Normal* – jako součást stránky
  - *Maximized* – je vykreslen pouze jeden daný portlet
  - *Minimized* – je zobrazeno pouze záhlaví portletu
  - Existují další stavy, které jsou většinou specifické pro konkrétní portály (např. *Solo*)



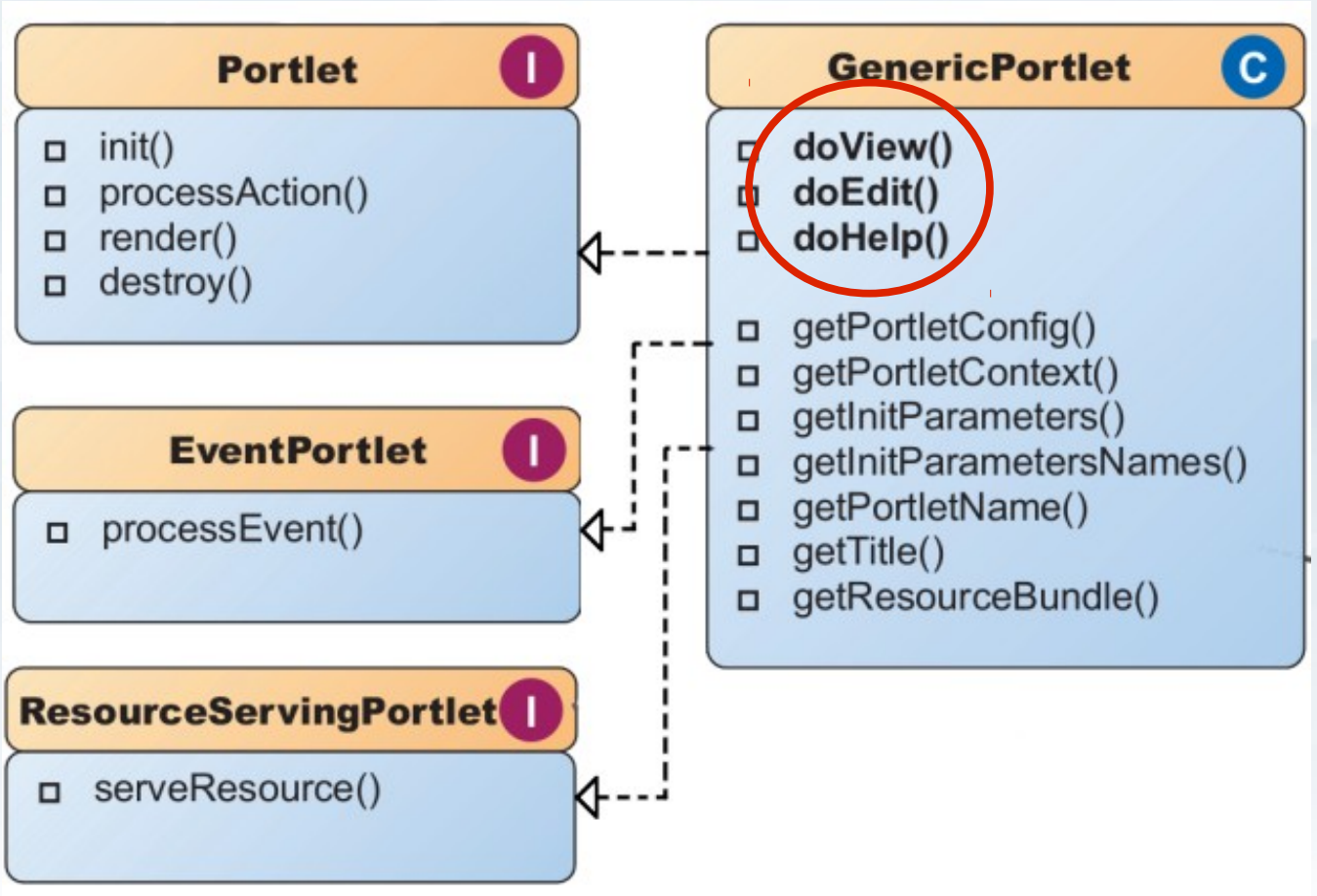
Portletové módy (portlet mode)

- Portletové módy umožňují portletu zobrazení různých uživatelských rozhraní
  - *View* – základní mód
  - *Edit* – většinou pro úpravu uživatelských nastavení
  - *Help* – zobrazení nápovědy k portletu
  - Existují portálově specifické módy a je možné definovat i módy další
    - *Configure* – administrace portletu

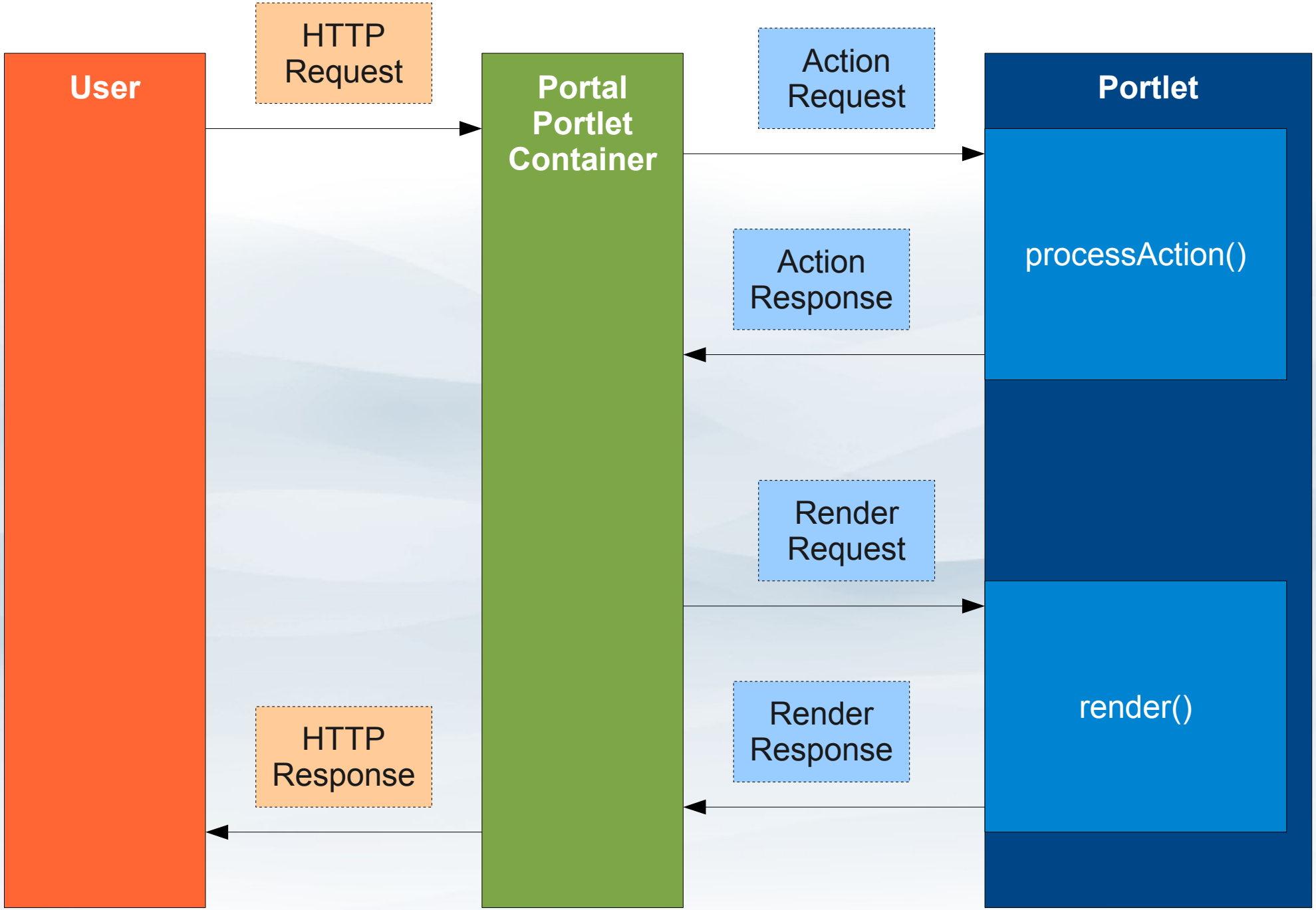


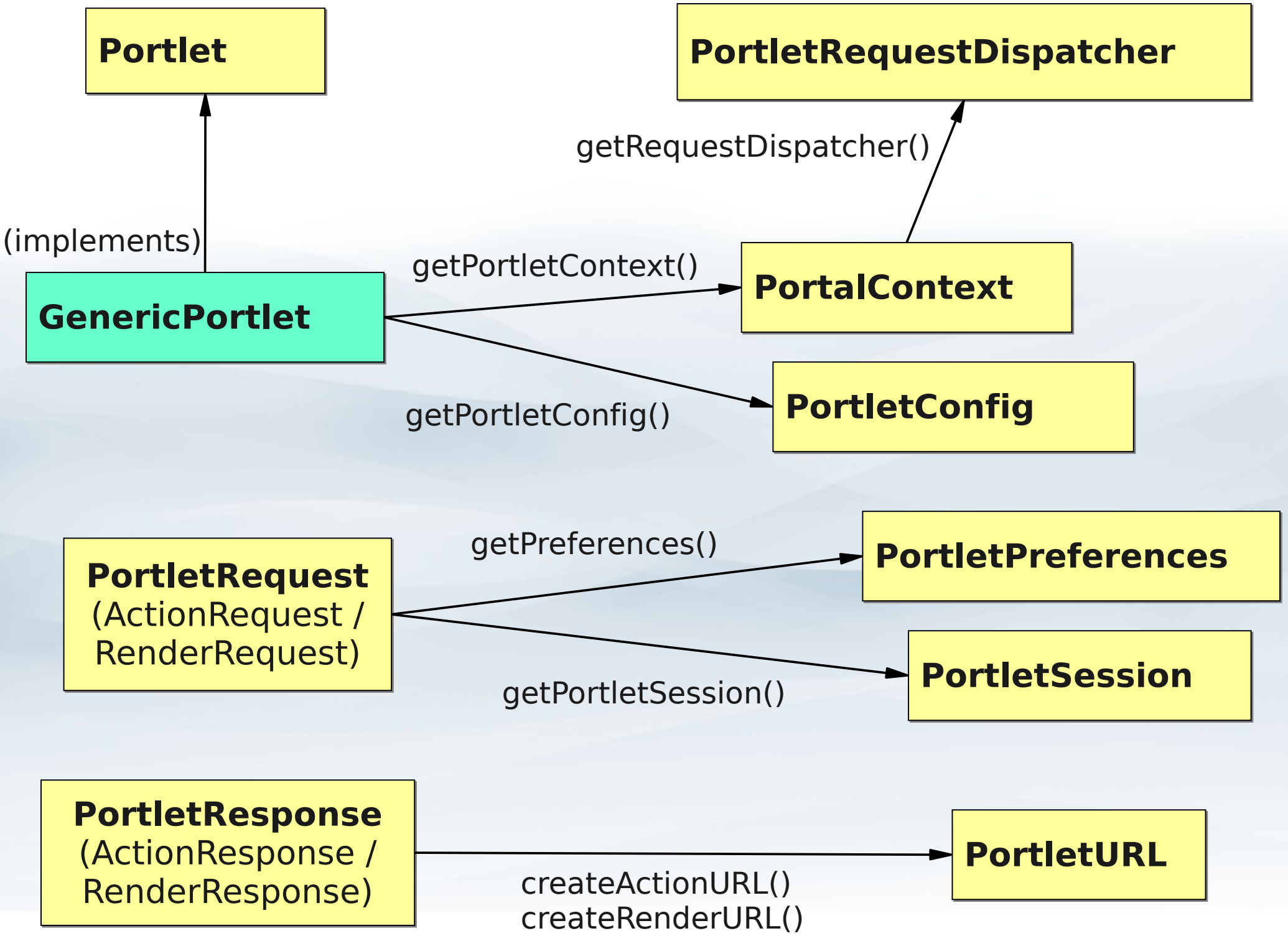
Životní cyklus portletu

- GenericPortlet automaticky vyvolá *render* metodu podle portletového módu



Významné objekty v JSR-168 API


User
HTTP Request
Portal Portlet Container
Action Request
Portlet
processAction()
Action Response
Render Request
render()
Render Response
HTTP Response


Portlet
PortletRequestDispatcher
getRequestDispatcher()
(implements)
GenericPortlet
getPortletContext()
PortalContext
PortletConfig
getPortletConfig()
PortletRequest
(ActionRequest /
RenderRequest)
getPreferences()
PortletPreferences
PortletSession
getPortletSession()
PortletResponse
(ActionResponse /
RenderResponse)
PortletURL
createActionURL()
createRenderURL()

GenericPortlet

- Standardní implementace rozhraní Portlet
- Abstraktní třída, jenž je nejčastěji rozšiřována
- Uživatelské portlety by měly přepisovat:
  - *processAction()* pro zpracování požadavků a změnu stavu portletu
  - *doView()* pro zpracování požadavků ve *VIEW* módu
    - Portlet by měl vždy definovat metodu *doView()*
  - *doEdit()* pro zpracování požadavků v editačním módu
  - *doHelp()* pro zobrazení nápovědy
  - *init()* a *destroy()* pro práci se zdroji
- Běžně se nepřepisují metody
  - *render() a*
  - *doDispatch()*, která volá *doView()*, *doEdit()* nebo *doHelp()* na základě požadovaného portletového módu

GenericPortlet

- **`getInitParameter`**`(java.lang.String name)`
  - Returns a String containing the an initialization parameter, or null
- **`getInitParameterNames`**`()`
  - Returns the names of initialization parameters as an Enumeration of String objects
- **`getPortletName`**`()`
  - Returns the name of this portlet
- **`getResourceBundle`**`(java.util.Locale locale)`
  - Gets the resource bundle for the given locale based on the resource bundle defined in the deployment descriptor

PortletConfig

- Poskytuje portletu jeho nastavení z deskriptoru *portlet.xml*
- Důležité metody jsou zpřístupněny přímo z třídy GenericPortlet

PortletContext

- Definuje pohled portletu na portletový kontejner
- PortletContext zpřístupňuje portletu další zdroje
  - Získávání odkazů na zdroje
  - Předávání řízení dalším částem webové aplikace
- Inicializační parametry jsou pouze ke čtení a nastavují se deskriptoru web.xml
- Důležité metody
  - **`getRequestDispatcher`**`(java.lang.String path)`
  - **`getInitParameter`**`(java.lang.String name)`
  - **`getInitParameterNames`**`()`

PortletRequest

- Objekty PortletRequest
  - Vytvářeny a spravovány portletovým kontejnerem
  - Předávány jako argument portletovým metodám
  - Jsou využívány pro komunikaci mezi portletovým kontejnerem a portletem a zároveň k předávání dat v rámci zpracování požadavku
- Data
  - Parametry (parameters)
  - Atributy (attributes)
  - Atributy sezení (session attributes)
  - Preference (preferences)
  - Portletový mód a stav okna
  - Informace o uživateli

PortletRequest - Parameters

- Hodnoty většinou nastavené v URL
- Hodnoty typu String
- Pouze ke čtení
- Velmi časté např. pro navigaci „uvnitř“ portletu
- Metody
  - **`getParameter`**`(java.lang.String name)`
  - **`getParameterNames`**`()`
  - **`getParameterValues`**`(java.lang.String name)`

PortletRequest - Attributes

- Hodnoty typu Object
- Atributy se využívají k předávání větších objemů dat v rámci požadavku

- Metody
  - **`getAttribute`**`(java.lang.String name)`
  - **`setAttribute`**`(java.lang.String name, java.lang.Object o)`
  - **`removeAttribute`**`(java.lang.String name)`
  - **`getAttributeNames`**`()`

PortletRequest – Informace o uživateli

- *getRemoteUser()*
  - Vrátí ID přihlášeného uživatele nebo *NULL*
- Atribut *USER_INFO*
  - Informace o uživateli uložené do objektu Map
  - Požadované informace je třeba definovat v portlet.xml

```
Map userInfo =
    (Map)request.getAttribute(PortletRequest.USER_INFO);
String givenName =
    (String)userInfo.get("user.name.given");
```


PortletPreferences

- Perzistentní úložiště dat spravované kontejnerem
- Výchozí hodnoty jsou definované v *portlet.xml*
- Pro každé portletové okno se nastavují samostatně
- Pro uložení dat je zapotřebí volat metodu *store()*
  - Nelze volat během render fáze
  - Pro data lze vynutit validaci před uložením pomocí PreferenceValidator

PortletSession

- Úložiště pro uživatelské informace déle než na dobu jednoho požadavku
- Session se vytváří pro uživatele pro každou portletovou aplikaci
- PortletSession má jeden z rozsahů:
  - *APPLICATION_SCOPE* – objekty jsou přístupné všem portletům a JSP stránkám v rámci jedné portletové aplikace.
  - *PORTLET_SCOPE* – objekty jsou přístupné pouze portletu, který je do session umístil.
- Pozor na velikost objektů, které do session ukládáte – velké objekty mají dopad na výkonnost

PortletSession

- **`getAttribute`**`(String name)`
- **`getAttribute`**`(String name, int scope)`
  - Returns the object in this session, or null
- **`getAttributeNames`**`()`
- **`getAttributeNames`**`(int scope)`
- **`setAttribute`**`(String name, Object value)`
- **`setAttribute`**`(String name, Object value, int scope)`
- **`removeAttribute`**`(String name)`
- **`removeAttribute`**`(String name, int scope)`
  - Removes the object bound with the specified name and the given scope
- If scope is ommited, then PORTLET_SCOPE is used

PortletResponse

- Rozhraní pro odesílání odpovědi na požadavek

- *encodeUrl()*
  - Vrací zakódované URL požadovaného zdroje (např. servlet, JSP, obrázky nebo jiný statický obsah)
- Další metody jsou dostupné ve specifických rozhraních pro danou fázi zpracování požadavku

RenderResponse

- Odpověď na render požadavek
- Vytváření URL
  - *createActionURL()*
  - *createRenderURL()*
  - *createResourceURL()*

PortletURL

- Portletová URL umožňují portletům vytvářet odkazy takové, které ukazují na ně samé (přes portál)
- URL je zpracováno portálem tak, že odkazuje na portlet i v případě, že na stránce je několik instancí stejného portletu.

ActionResponse

- **`sendRedirect`**`(java.lang.String location)`
  - Instrukce pro portletový kontejner, že má odeslat přesměrování na zadanou lokaci
- **`setPortletMode`**`(PortletMode portletMode)`
- **`setWindowState`**`(WindowState windowState)`
- Nastavení parametrů vykreslování
  - **`setRenderParameter`**`(String key, String value)`
  - **`setRenderParameter`**`(String key, String[] values)`
  - **`setRenderParameters`**`(Map parameters)`
    - Parametry viditelné v processAction se automaticky nepropagují do render – často se používá pro předání parametrů beze změn

Data s nimiž portlety pracují

Umístění	Typ	Velikost	R/W	Doba života
Init parameters(PortletConfig)	String	KB	RO	Jako portlet
Init parameters(PortletContext)	String	KB	RO	Jako portlet
Attributtes	Object	MB	RW	Request
Parameters	String	KB	RO	Request
Session Attributes	Object	KB	RW	Session
Preferences	String	KB	RW/RO	Persistent

Knihovna značek

Knihovna značek (Tag Library)

- Knihovna obsahuje značky pro
  - Přístup k objektům v *request* a *response*
  - Generování odkazů
- JSP direktiva pro přidání knihovny

```
<%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet" %>
```

- Definované značky
  - *defineObjects*
  - *actionURL*
  - *renderURL*
  - *resourceURL*
  - *namespace*
  - *param*

Značka <portlet:defineObjects>

- Umožní přístup k
  - *renderRequest*
  - *renderResponse*
  - *portletConfig*
- Po značce defineObjects je možné proměnné využívat v JSP pomocí scriptletů

```
<portlet:defineObjects/>
<% Employee emp =
  renderRequest.getAttribute("Employee");%>
```

Vytváření odkazů

- Využití značek
  - **<portlet:renderURL>**
  - **<portlet:actionURL>**
  - **<portlet:resourceURL>**
- Pro přidání parametrů odkazům lze využít značky **<portlet:param>**

```
<portlet:renderURL var="invoiceRenderUrl">
    <portlet:param name="view" value="invoice"/>
    <portlet:param name="id" value="42"/>
</portlet:renderURL>

<a href="${invoiceRenderUrl}">Show invoice 42</a>
```

Značka <portlet:namespace>

- Vrací jednoznačné označení portletu
- Používá se pro odlišení entit na stránce v případě, že je na ní více stejných portletů
- Využití
  - Javascript
  - Element ID
  - Proměnné

```
<a href="javascript:<portlet:namespace/>doJob()">Job</A>
```

První portlet

Hands on!

Příprava vývojového prostředí

Úkol:

Připravte si vývojové prostředí

- Vývojový portál
- Nastavení IDE
- Zásuvné moduly IDE
- Provázání IDE a serveru

Hands on!

Kostra projektu

Úkol:

Sestavte si knihovnu *SimpleShopBackend*

- Dodaný balíček
  - *SimpleShopBackend*
  - *SimpleShopPortlet*
- Sestavení aplikace
- Nasazení aplikace

Hands on!

První portlet

Úkol:

Vytvořte portlet *Catalog*, který zobrazí seznam produktů získaný pomocí dodané knihovny.

- Třídu implementující `GenericPortlet`
- JSP stránku pro zobrazení seznamu produktů
- Záznamy v deskriptorech
- Lokalizaci pro JSP

Hands on!

Třída implementující `GenericPortlet`

Implementuje pouze metodu **`doView(...)`**

- Získá informace z DB

```
ServiceProvider.getCatalog().getAllProducts();
```

- Připraví je pro zobrazení
- Volá *RequestDispatcher*, který nechá informace zobrazit JSP stránku

Umístění	Typ	Velikost	R/W	Doba života
Init parameters(PortletConfig)	String	KB	RO	Jako portlet
Init parameters(PortletContext)	String	KB	RO	Jako portlet
Attributtes	Object	MB	RW	Request
Parameters	String	KB	RO	Request
Session Attributes	Object	KB	RW	Session
Preferences	String	KB	RW/RO	Persistent

Hands on!

JSP stránka

- JSP – view v rámci MVC
- Pokud to jde, nepužíváme *scriptlets* → JSTL
- Vše potřebné je již definováno ve fragmentu *init.jspf*
  - Podívejme se, co tam je

- Obsah JSP:
  - Include *init.jspf*
  - Výpis produktů pomocí JSTL
    - Potřebná data jsou již v atributech připravená z doView(..)

Hands on!

Portletové desktiptory

- Podle specifikace
  - ***portlet.xml***
    - portlety
    - veřejné parametry
    - portletové události
- Liferay specifické
  - ***liferay-portlet.xml***
    - Konfigurace portletů specifickoá pro portál Liferay
  - ***liferay-display.xml***
    - Definuje jak bude portlet zobrazen v katalogu
- Všechny najdeme v adresáři WEB-INF

Hands on!

Lokalizace

- Vytvoření resource bundle
- Navázání JSP na resource bundle
- Vložení lokalizovaných hlášek do JSP

Hands on!

JSP s formulářem

- Formuláře se musí odesílat pomocí metody POST
- Uložení odkazu do proměnné v rámci JSP

```
<portlet:actionURL var="xxx" name="YYY" />
<form action="${xxx}" method="post">...
```

Hands on!

Zpracování požadavku a zobrazení informace

- Zpracování akce v portletu pomocí anotací

```
@ProcessAction(name = "YYY")
public void actionSearch(..., ...) …
```

- Během zpracování akcí se nastavuje výsledný pohled pomocí parametrů
  - **`response.setRenderParameter(...)`**
  - **`response.setRenderParameters(`**
    `request.getParameterMap());`

Hands on!

Komplexní portlety

Komplexní portlety

- Více pohledů
  - CRUD, administrační aplikace, formuláře
- Více akcí
- Nastavení a konstanty
- Velké množství vstupních dat
  - Formuláře
- Složitá aplikační logika

Více pohledů

- Využití metody *doView()* k rozdělování požadavků
  - Nejlépe na základě příchozích parametrů
    - Možnost vytvářet záložky
  - Podle uživatelského nastavení
- Využití JSPs nebo jiné technologie k oddělení view vrstvy
- Příprava zobrazovaných dat v portletu – důsledné oddělení view vrstvy

Více pohledů - ukázka

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

protected void doViewMain(RenderReq., RenderResp.).. {
    ...
}

protected void doViewDetail(RenderReq., RenderResp.).. {
    ...
}

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


Více akcí

- Využití metody *processAction()* k rozdělování požadavků
- **Využití anotací k rozdělování požadavků**
  - JSR-286, Java 5+

- Nastavení render parametrů

Více akcí - ukázka

- JSP

```
<form
    action="<portlet:actionURL name="formSendAction"/>"
    method="post">
        ...
        ...
</form>
```

- Portlet

```
@ProcessAction(name="formSendAction")
public void processFormSendAction(
    ActionRequest req,
    ActionResponse res)
    throws PortletException
{
    ...
}
```

Nastavení a konstanty

- Programové konstanty
- Init parameters
  - *portlet.xml*
  - *web.xml*
- Preferences
  - Výchozí hodnoty
  - Read-only preferences

Nastavení a konstanty

- Pomocné metody
  - Vhodné pro vícekrát použité položky
  - Snadnější udržovatelnost kódu
  - Typová kontrola
- Pomocné třídy
  - Vhodné pro více položek
    - Pomocné metody
    - Programové konstanty

Pomocné třídy - ukázka

```
public class Constants {

    // Parameters
    public static final String PARAM_VIEW = "view";
    public static final String PARAM_VIEW_DETAIL = "detail";
    /* ...more parameters here... */

    // Preference keys
    public static final String PREF_ITEMS_PER_PAGE =
            "eu.ibacz.test.itemsPerPage";
    /* ...more preferences here... */

    // Attribute keys
    public static final String ATTR_ITEMS_PER_PAGE = "itemsPerPage";
    public static final String ATTR_ITEMS_LIST = "itemsList";
    /* ...more attributes here... */

    /* ... */

}
```

Pomocné metody - ukázka

```
public class DataHelper {

    public static int getItemsPerPage(PortletRequest req) {
        int result = 0;
        String defaultValue = "10";

        PortletPreferences prefs = req.getPreferences();
        String value = prefs.getValue(
                Constants.PREF_ITEMS_PER_PAGE,
                defaultValue);
        result = Integer.parseInt(value);

        return result;
    }

    /* ... more methods here ... */

}
```


Pomocné třídy a metody - ukázka

```
public class SupremePortlet extends GenericPortlet {

    protected void doViewMain(
            RenderRequest request, RenderResponse response)
            throws IOException, PortletException
    {
        // Prepare data
        request.setAttribute(Constants.ATTR_ITEMS_PER_PAGE,
                DataHelper.getItemsPerPage(request));
        request.setAttribute(Constants.ATTR_ITEMS_LIST,
                SupremeEntityServiceUtil.getAllEntities());

        // Dispatch to JSP
        String viewJspName =
                getPortletConfig().getInitParameter("viewJsp");
        PortletRequestDispatcher dispatcher =
                GetPortletContext()
                .getRequestDispatcher(viewJspName);
        dispatcher.include(request, response);
    }

    /* ... more methods here ... */
}
```

Velké množství vstupních dat

- Hodně vstupních hodnot
- Vícekrokové formuláře

- Validace dat není součástí specifikace
- Zvážit použití MVC rámce
  - Validace dat
  - Převod dat z formuláře na objekty

Složitá aplikační logika

- V rámci MVC je portlet Controller
- Aplikační logika patří do modelu
- Portlet by měl obsahovat pouze logiku UI

Komplexní portlety

Velmi složitý portlet může naznačovat špatný návrh a portletovou dekompozici

- Překontrolovat návrh aplikace
  - Různé portlety pro různé role
  - Portlety podle případu použití
- Zvážit možnosti zjednodušení či rozložení fukcionality do více portletů
- Zvážit využití MVC rámce

První portlet +

Úkol:

Rozšiřte portlet *Catalog* tak, aby umožňoval vyhledávat v katalogu produktů.

- Pomocná třída pro konstanty
- Rozšíření v JSP
  - Statický import konstant
  - Formulář pro odeslání požadavku
- Podpora v portletu
  - Zpracování požadavku
  - Zobrazení informace

Hands on!

Konstanty

```
<%@page import="static eu.ibacz.swsc.MyConstants.*" %>
<%@page import="static eu.ibacz.swsc.Const.ACTION_X"
%>

<portlet:actionURL ... name="<%= ACTION_X %>"/>
```

Formuláře

- Formuláře se musí odesílat pomocí metody POST
- Uložení odkazu do proměnné v rámci JSP

```
<portlet:actionURL var="xxx" name="YYY" />
<form action="${xxx}" method="post">...
```

Hands on!

Zpracování požadavku a zobrazení informace

- Zpracování akce v portletu pomocí anotací

```
@ProcessAction(name = "YYY")
public void actionSearch(..., ...) …
```

- Během zpracování akcí se nastavuje výsledný pohled pomocí parametrů
  - **`response.setRenderParameter(...)`**
  - **`response.setRenderParameters(`**
    `request.getParameterMap());`

Hands on!

Meziportletová komunikace
Interportlet communication (IPC)

Meziportletová komunikace

- JSR-168
  - Nemá standardizovanou cestu pro meziportletovou komunikaci
  - Řešení
    - Sdílení dat v ”application scope” session
    - JavaScript
    - Databáze
    - Proprietární řešení (IBM Click2Action)
- JSR-286
  - Public Render Parameters
  - Events
  - (Cookies)

Session – Application Scope

- Využívá se HTTP session
  - *PORTLET_SCOPE* – prefixy
  - *APPLICATION_SCOPE*
- Sdílení dat v rámci portletové aplikace
- Nelze sdílet mezi aplikacemi

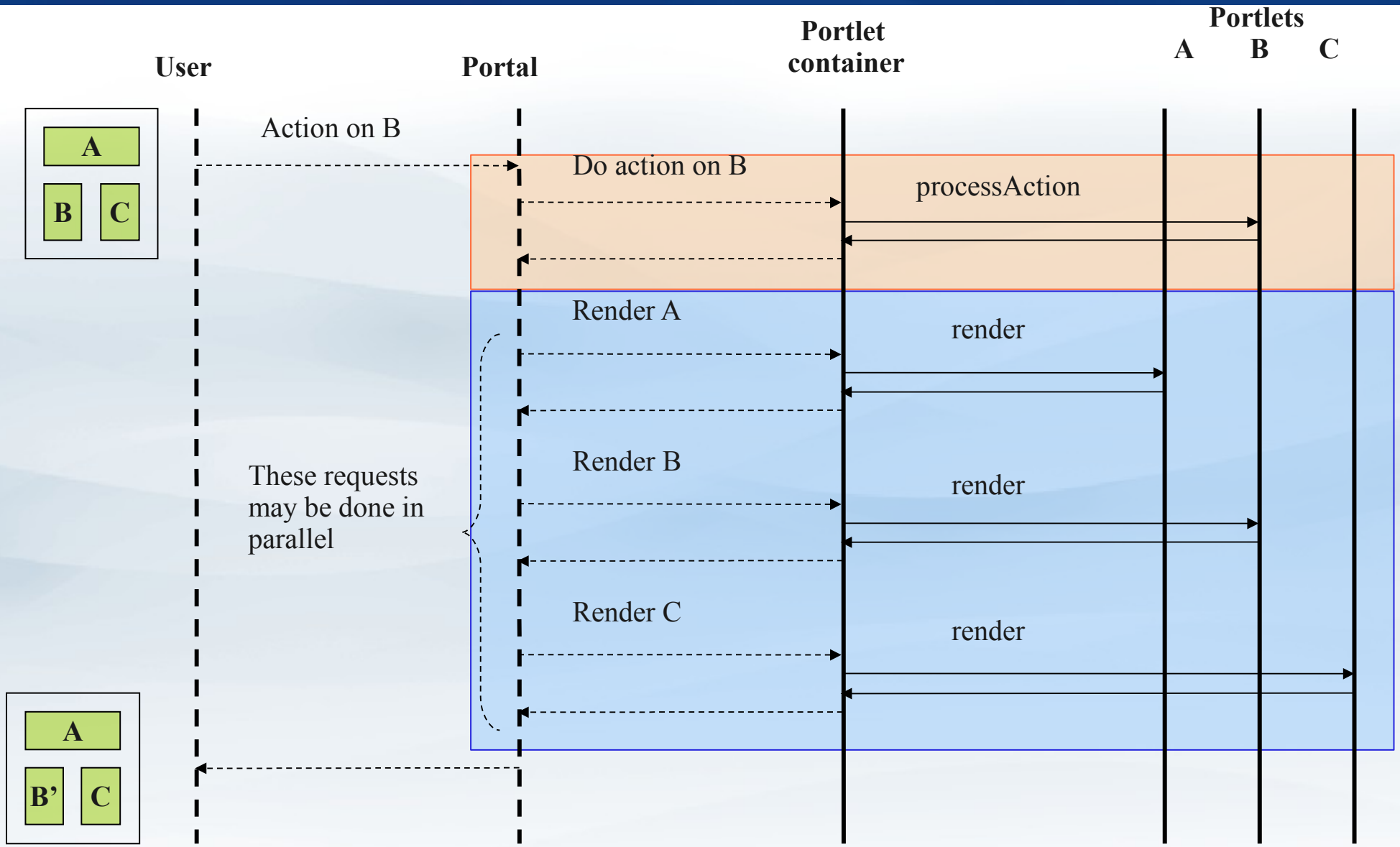
- Změny provádět během zpracování akcí či událostí
- Pozor na změny dat session v render fázi
- Pozor na kolize jmen

Veřejné parametry
Public Render Parameters

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, které z veřejných parametrů využívá
- Veřejné parametry jsou dostupné ve všech částech životního cyklu požadavku
- Pouze řetězcové hodnoty

Public Render Parameters



Public Render Parameters – ukázka

```
<portlet-app>
    ...

    <public-render-parameter>
        <identifier>param1</identifier>
        <qname
            xmlns:x="http://enablement.ibacz.eu/params">x:p1</qname>
    </public-render-parameter>

    ...

    <portlet>
        <portlet-name>portletA</portlet-name>
        ...
        <supported-public-render-parameter>
            param1
        </supported-public-render-parameter>
    </portlet>

    ...
</portlet-app>
```

QName

- Qualified name
  - Namespace + name
  - Minimalizuje možnost kolizí ve jménech
  - http://en.wikipedia.org/wiki/QName
- Použití pro veřejné parametry i události
  - Umožňuje jejich používání napříč aplikacemi
- V *portlet.xml* jsou 2 možnosti definice
  - Celé QName
  - Použití výchozího jmeného prostoru

Kostra projektu

Úkol:

- **Definujte veřejný parametr *productId***
- **Vytvořte portlet *Detail***
- **Propojte jej s portletem *Catalog* pomocí *productId***
  - **Portlet *Catalog* bude po kliknutí na link u produktu zveřejňovat parametr *productId***
  - **Portlet *Detail* bude reagovat právě na veřejný parametr *productId***

Hands on!

Vytvoření veřejného parametru

```
<portlet-app>

    <portlet>
    ...
    </portlet>

    <default-namespace>
        http://swsc.ibacz.eu/params
    </default-namespace>

    <public-render-parameter>
        <identifier>productId</identifier>
        <name>productId</name>
    </public-render-parameter>

</portlet-app>
```

Hands on!

Vytvoření portletu *Detail*

- Portlet vytvoříte podobně jako portlet *Catalog*
  - Třída rozšiřující *GenericPortlet*
  - JSP stránka
  - Záznamy v porletových deskriptorech
    - Nezapomenout na ***<supported-public-render-parameter>***

Hands on!

Rozšíření portletu *Catalog*

- Ke každému produktu přidáme link
  - RenderURL na sebe sama
  - Parametr *productId*
- Do *portlet.xml* přidáme *<supported-public-render-parameter>*

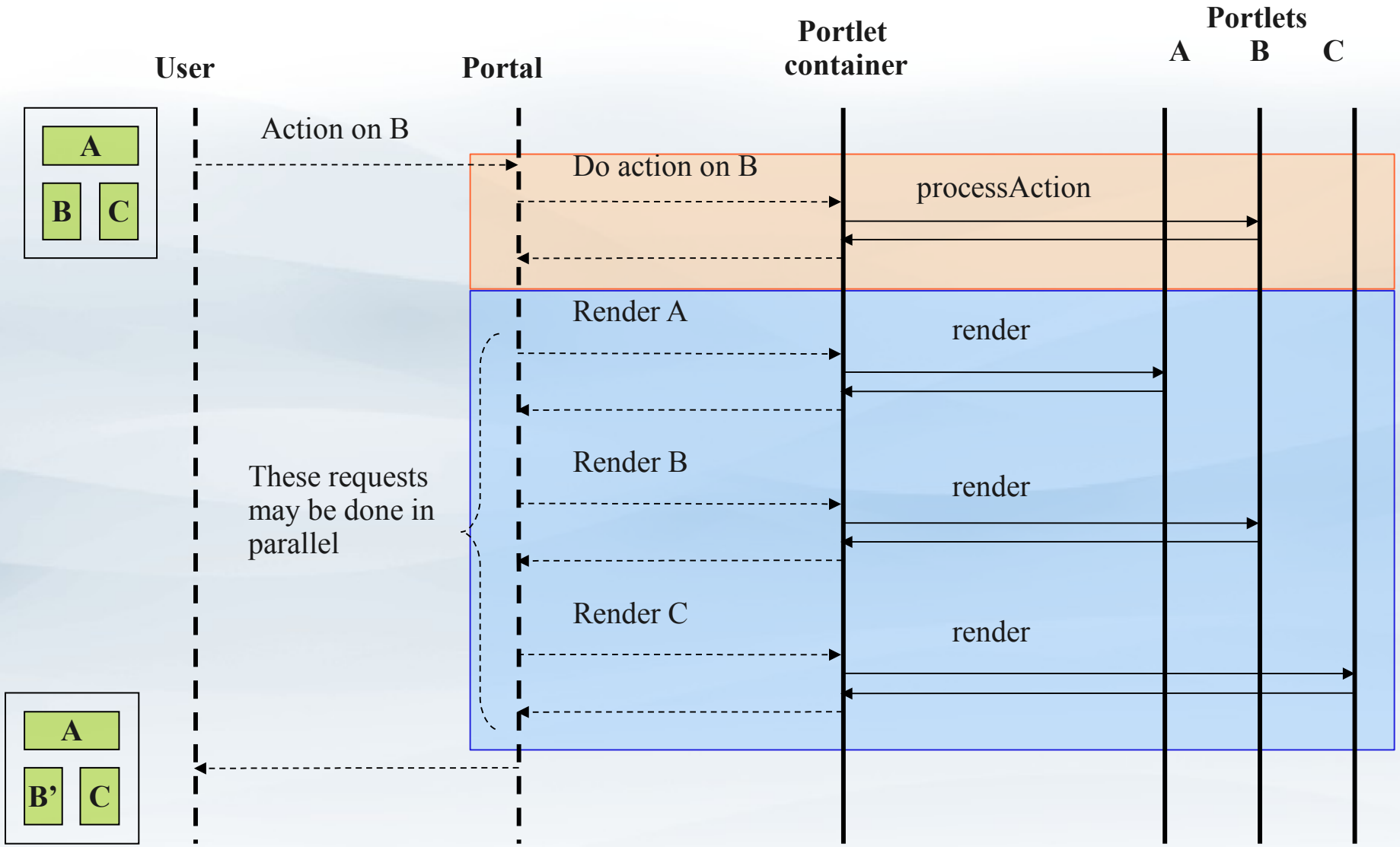
Hands on!

Události
Events

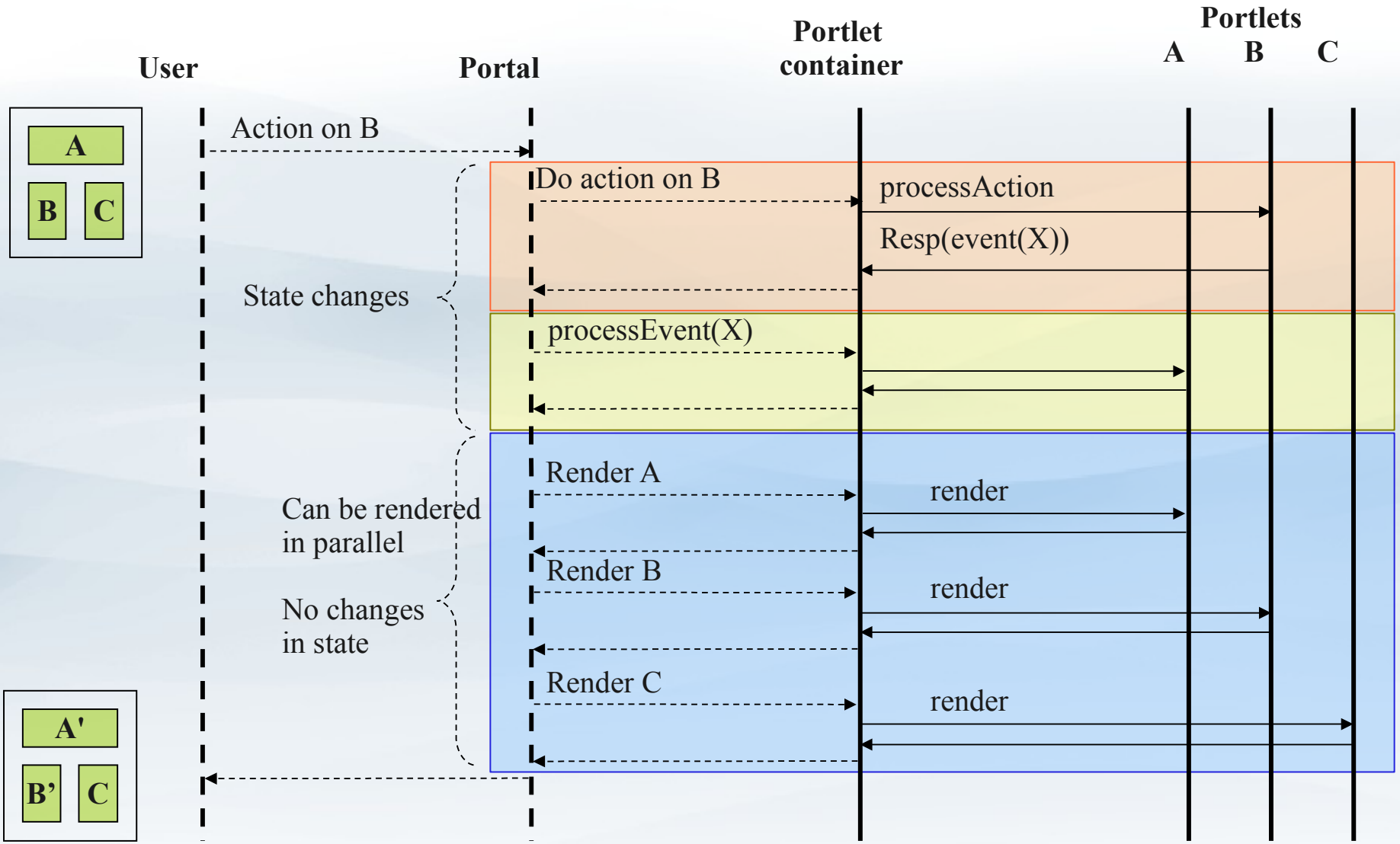
Události

- Události umožňují portletům reagovat na akce nebo změny stavu, které nemusí přímo souviset 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

JSR-286 Request



JSR-286 Request with Events



Události

- Události jsou navrženy podle vzoru Producer–Listener
  - Portlety mohou vytvářet události
  - Portlety mohou události přijímat
    - Portlet musí implementovat rozhraní *EventPortlet*
- Události se mohou ”řetězit”
- S událostí je možné posílat objekt
- Události je možné posílat mezi portlety různých portletových aplikací

Události

- Nastavení se provádí v deskriptoru *portlet.xml*
  - Každá událost má jedinečné jméno (Qname)
  - 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 – definice událostí

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

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

    <event-definition>
        <qname xmlns:x="http://enablement.ibacz.eu/events">
            x:event2
        </qname>
        <value-type>java.lang.Integer</value-type>
    </event-definition>

    ...
</portlet-app>
```


Události – generování událostí

```
<portlet-app>
    ...

    <portlet>
        ...

        <supported-publishing-event>
            <qname xmlns:x="http://enablement.ibacz.eu/events">
                x:event2
            </qname>
        </supported-publishing-event>

        ...
    </portlet>

    ...
</portlet-app>
```

Události – generování událostí

```
void processEvent(EventRequest req, EventResponse resp) {
    ...

    Integer message = new Integer(42); // THE number
    QName name = new Qname(
            "http://enablement.ibacz.eu/events",
            "event2");
    resp.setEvent(name, message);

    ...
}
```

Události – příjímání událostí

```
<portlet-app>
    ...

    <portlet>
        ...

        <supported-processing-event>
            <qname xmlns:x="http://enablement.ibacz.eu/events">
                x:event2
            </qname>
        </supported-processing-event>

        ...
    </portlet>

    ...
</portlet-app>
```

Události – příjímání událostí

```
@ProcessEvent(qname="{http://enablement.ibacz.eu/events}event2")
public void processEvent2(
        EventRequest request,
        EventResponse response)
        throws PortletException, IOException
{
    ...

    Event e = request.getEvent();
    Integer id = (Integer)e.getValue();

    ...
}
```


Události vs. veřejné parametry

Veřejné parametry

- Výhody
  - Jednoduché a účinné
  - Pracuje s *render* URL (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 řetězení
- Použití
  - Když nestačí veřejné parametry
  - K propagování stavu napříč portlety
- Co nedělat?
  - Používat události jako message system

Poskytování prostředků
Resources serving

Poskytování zdrojů – JSR-168

- Portletová URL (actionURL, renderURL) odkazují na portálovou stránku
  - Nemožné získat pouze fragment kódu nebo dynamická binární data
- Řešení
  - Servlet
    - Problémy s autentizací
    - Problémy se získáváním dat z portletu (inicializační parametry, preference atd.)
  - Speciální portletový mód (např. *Solo*)
    - Proprietární řešení

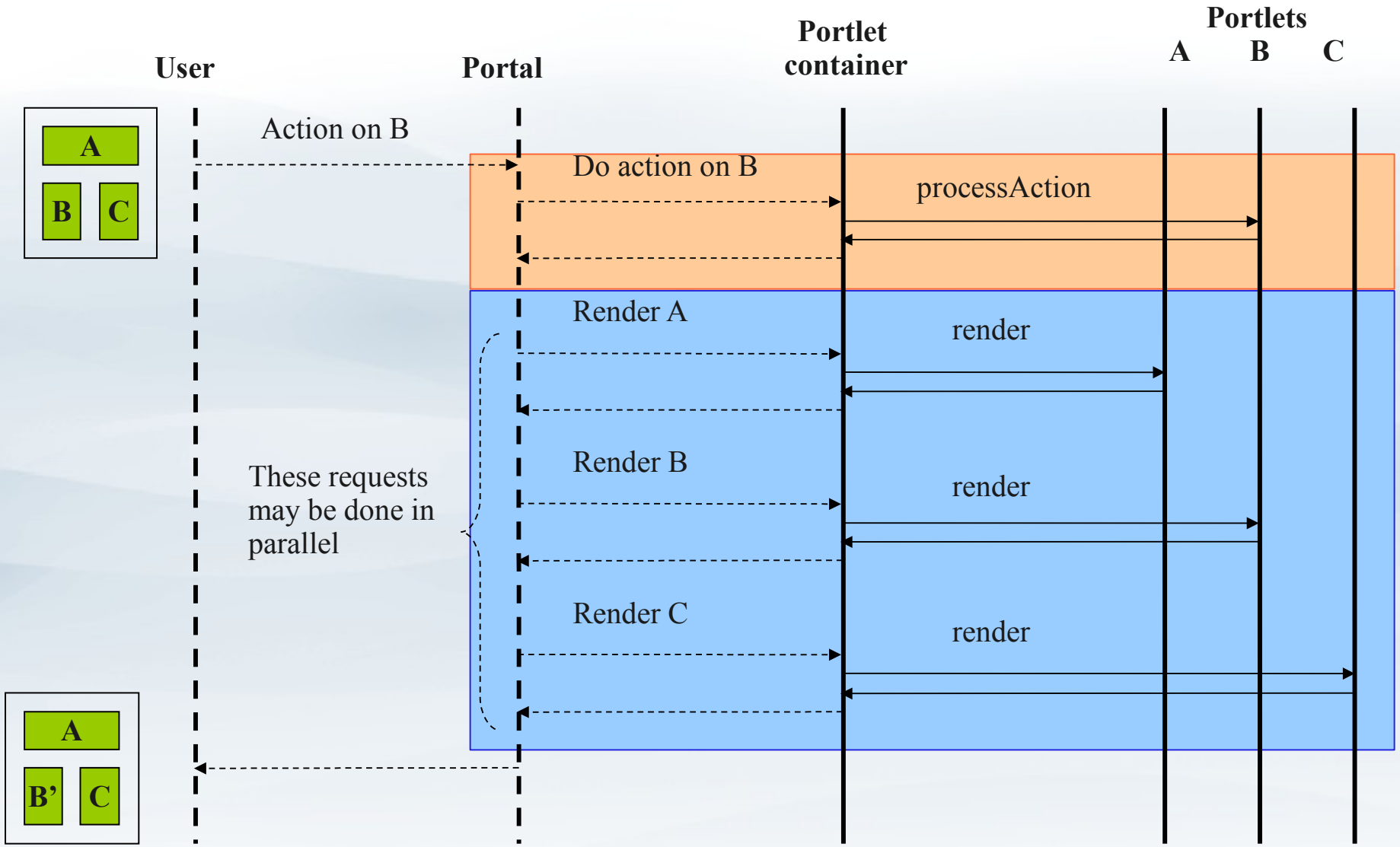
Poskytování zdrojů – JSR-286

- Rozhraní *ResourceServingPortlet*
- Klient může posílat požadavky na neagregovaná data
  - Binární data
  - AJAX
  - HTML fragmenty
  - …

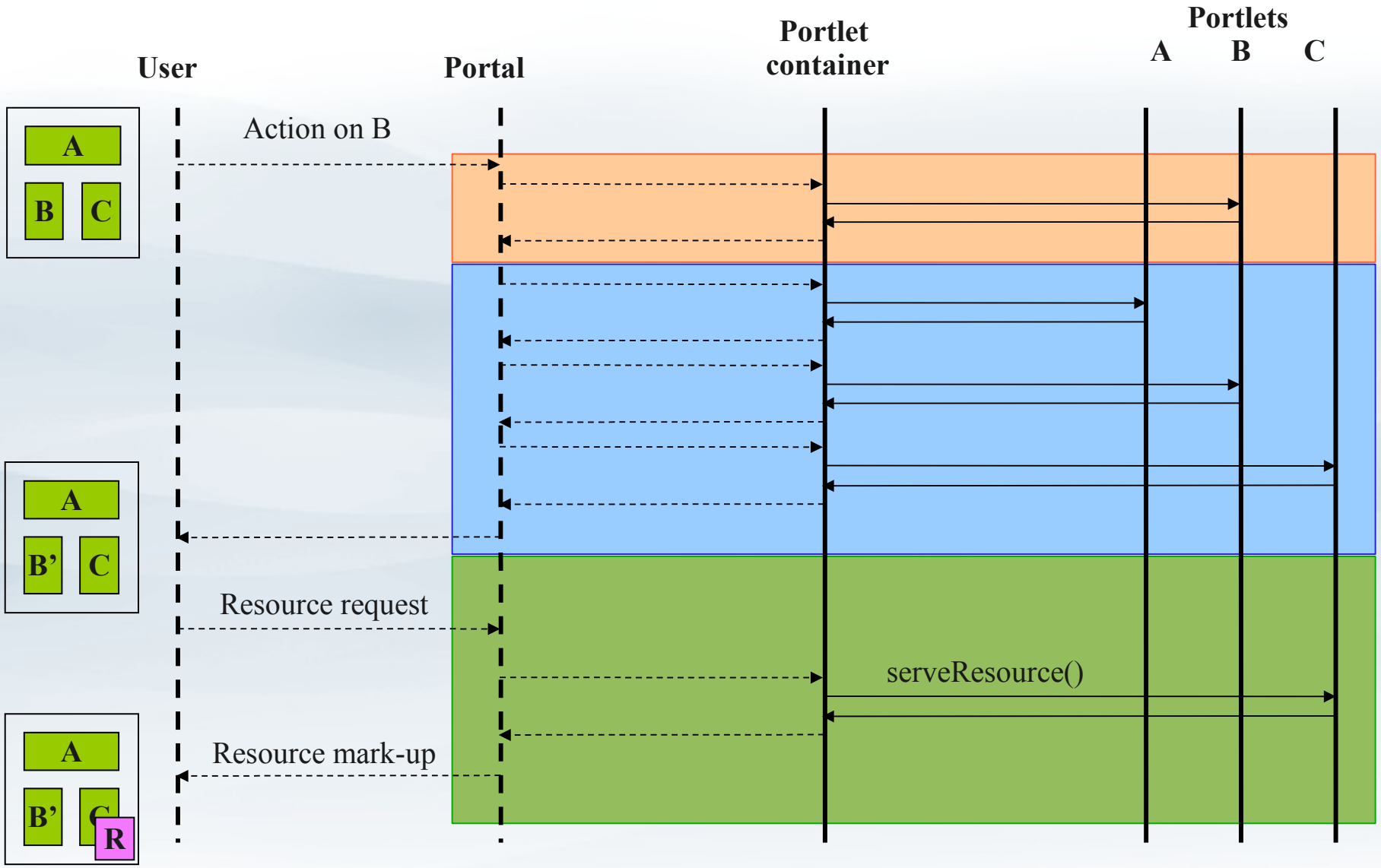
Portlet Resource vs. Servlet

- Portlet Resource
  - Created by `createResourceURL()` in `MimeResponse`
  - URL pointing back to the portlet
  - Can be protected by portal security
  - Can leverage portlet context
- Direct links
  - Created by `encodeURL()` in `PortletResponse`
  - Might not be valid URL
  - Portlet context not available
  - More efficient
  - Recommended for serving static content

Resource Serving Request Sequence



Resource Serving Request Sequence



Resource Serving

- Pro poskytování prostředků portlet musí implementovat rozhraní `ResourceServingPortlet` s metodou `serveResource()`
- Volání metody `serveResource()` obvykle následuje až po volání `render()` a může být považováno za logické rozšíření render fáze
- Portlet by neměl během volání `serveResource()` měnit svůj stav, pokud byl požadavek odeslán pomocí HTTP metody *GET*

ResourceRequest Interface

- `ResourceRequest` rozšiřuje `ClientDataRequest`, který dále rozšiřuje `PortletRequest`
- `ClientDataRequest` – požadavek může obsahovat uživatelská data
  - `GetMethod()` - vrátí metodu, pomocí které byl požadavek odeslán
  - `getPortletInputStream()`, `getReader()`
  - `GetCharacterEncoding()`, `getContentType()`, `getLength()`
- Další důležité funkce
  - **`getResourceID`**`()`
    - Vrací ID požadovaného prostředku (nastavuje se v `ResourceURL`)
  - **`getCacheability`**`()`
    - Vrací informaci o tom, jak je prostředek cachován

Autocomplete

Úkol:

Uprave portlet *Catalog* tak, aby pole pro vyhledávání automaticky našeptávalo položky k hledání.

- Podpora v portletu
  - Poskytování seznamu odpovídajících položek pro našeptávání
- Rozšíření v JSP
  - Jednoznačné označení pole
  - JavaScript pro našeptávání

Hands on!

Poskytování prostředků z portletu 1 - rozcestník

```
@Override
public void serveResource(resRequest, resResponse) ... {

    String resourceID = request.getResourceID();

    if (MY_SUPER_RESOURCE.equals(resourceID)) {
        // call my method to serve my resource
        serveMySuperResource(resRequest, resResponse);
    } else {
        // let GenericPortlet to handle other requests
        super.serveResource(resRequest, resResponse);
    }
}
```

Hands on!

Poskytování prostředků z portletu 2

- Metoda pro poskytování prostředků
  - Získá parametr o aktualním řetězci v poli
    - a defenzivně si jej ověří ;-)
  - Nastaví v response *contentType* (v tomto případě *text/plain*)
  - *Z response* získá *PrintWriter* pro textový výstup
  - Najde v katalogu odpovídající položky
  - Pro každou položku zapíše její jméno
  - Uzavře *PrintWriter*

Hands on!

Jednoznačné označení pole

- Využití `portlet:namespace` (viz. *init.jspf*)

```
<c:set var="ns"><portlet:namespace/></c:set>
```

- Jednoznačné označení pole na stránce

```
<input type="text"
        name="..." id="${ns}query" />
```

Hands on!

IBA CZ