Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Dokumentace frameworku pro OpenCms moduly www.kiv.zcu.cz

Miroslav Král(A08N0088P)
miika@students.zcu.cz

Plzeň,2009

11.11.2009

Obsah

1 Důležité odkazy

Domácí stránka projektu se nachází na adrese:

- `http://wiki.kiv.zcu.cz/WebKiv/HomePage`

a domácí stránka pro vývoj je na adrese:

- `http://wiki.kiv.zcu.cz/WebKiv/RedSys`.

Většinou je nutné nainstalovat vývojové prostředí na lokální stroj. Postup a seznam potřebných technologií je na adresách:

- `http://wiki.kiv.zcu.cz/WebKiv/PouzivaneTechnologie`
- `http://wiki.kiv.zcu.cz/WebKiv/JakVytvoritTestovaciProstredi`.

2 Použité technologie

2.1 Spring

Pro více informací o Springu využijte dobře zpracovaný tutoriál:

- `http://www.kiv.zcu.cz/~brada/vyuka/files/pia/ppp/spring/`.

2.1.1 Definice datového zdroje

Spring je konfigurován přes xml soubor *redsys-servlet.xml*, který se nachází v:

- `<instalační-adresář-tomcatu>\webapps\opencms\WEB-INF\`

Zde se obvykle definuje navázání databázových objektů na konkrétní třídy v aplikaci. Tato možnost není v projektu využita, zde se používá *SimpleJdbcTemplate*, který je zmíněný v následující podkapitole. V redsys-servlet.xml je ale definováno přímo datové úložiště, tedy databáze kiv_zcu_cz a všechna nastavení, která jsou potřebná k připojení do databáze. První fragment ukazuje, z jakého konfiguračního souboru se budou brát údaje pro připojení k databázi:

```
<bean id="propertyConfigurer"
      class="org.springframework.beans.factory.
      config.PropertyPlaceholderConfigurer">
   <property name="location" value="/WEB-INF/webkiv.properties"/>
</bean>
```

Potřebné údaje pro připojení k databázi, ze souboru *webkiv.properties*, jsou:

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/kiv_zcu_cz
jdbc.username=root
jdbc.password=
```

Údaje *url*, *username* a *password* jsou závislé na instalaci prostředí na lokálním stroji, zde konkrétně MySQL. V ukázce jsou použity výchozí instalační hodnoty.
Poslední fragment ukazuje využití informací z konfiguračního souboru pro vytvoření objektu *dataSource*, který si Spring bude uchovávat ve svém IOC kontejneru a je využit ve `webkiv-db` knihovně.

```
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.
      DriverManagerDataSource">
   <property name="driverClassName"
             value="${jdbc.driverClassName}"/>
   <property name="url" value="${jdbc.url}"/>
   <property name="username" value="${jdbc.username}"/>
   <property name="password" value="${jdbc.password}"/>
   <property name="connectionProperties">
      <value>
         useUnicode=true
         characterEncoding=utf8
      </value>
   </property>
</bean>
```

2.1.2 Použité části Spring Framework

Na webu je Spring použit pro operace prováděné nad databází. Základem je knihovna *SimpleJdbcTemplate* z balíčku:

- `org.springframework.jdbc.core.simple`.

Pro Spring jsou důležité anotace pro automatické instanciování a provázání databázových objektů uložených v IOC kontejneru Springu. Použity jsou následující:

- `org.springframework.stereotype.Repository`
- `org.springframework.beans.factory.annotation.Autowired`

3 Datová vrstva

Databáze pro doménové aplikace webu KIV je uložena v MySql 5, pracuje se s ní pouze přes DAO (Data Access Objects) vrstvu.

3.1 Datový model

Celkový náhled na datový model je na adrese:

- `http://wiki.kiv.zcu.cz/WebKiv/DatovyModel`.

Na této adrese se také nachází aktuální verze SQL scriptu pro vytvoření databáze pro vývoj na locale.
Pro ERA model a vytvářené tabulky platí následující pravidla:

- Všechny tabulky mají názvy VELKÝMI_PÍSMENY a prefix KIV_ následovaný zkratkou oblasti a jménem tabulky (např. KIV_PERS_AKTIVITY nebo KIV_PUBL_CLANKY).
- Jako primární klíč se používá `INT id`, atributy cizích klíčů začínají `FK_`.
- Každá ne-číselníková tabulka musí obsahovat atributy `zaznam_datum` (timestamp), `zaznam_autor` (FK do tabulky KIV_PERS_OSOBY) a `zaznam_aktivni` (boolean).
- Jmenné a popisné entity (viz níže) musí obsahovat atributy `nazev_cz|en` respektive `popis_cz|en`.
- Rozlišení lokalizovaných atributů se děje pomocí postfixů `_cz` a `_en`.

3.2 DAO vrstva

Třídy pro operace nad databází jsou umístěny v knihovně `webkiv-db.jar` a rozděleny do dvou balíčků:

- `webkiv-db\src\cz.zcu.kiv.db.dao`
- `webkiv-db\src\cz.zcu.kiv.db.dao.jdbc`

První balíček obsahuje rozhraní, která jsou následně implementována třídami v druhém balíčku. Následující ukázky budou ze souboru *ProduktDaoImpl.java* z druhého balíčku.
Každá DAO třída z balíčku *jdbc* obsahuje u názvu třídy anotaci *@Repository*, zde konkrétně:

```
@Repository("produktDao")
public class ProduktDaoImpl implements ProduktDao { ... }
```

Anotace @Repository(”produktDao”) uloží instanci této třídy do IOC kontejneru Springu pod identifikátorem ”produktDao”. Pod ním je k dispozici ostatním třídám, jak ukazuje následující kód z třídy *LicenceMediator.java*:

```
ProduktDaoImpl produktDao =
(ProduktDaoImpl) super.appContext.getBean("produktDao");
```

Související anotací je *@Autowired*, která je použita jak u instančních proměnných, tak speciální metody *init()*. Pokud je anotace použitá u deklarace instanční proměnné uvnitř třídy *ProduktDaoImpl.java*, bude zajištěno automatické instanciování příslušného objektu z IOC kontejneru:

```
@Autowired
private OsobaDao _osobaDao;
```

To znamená že instanční proměnné *_osobaDao* bude v impl třídy *ProduktDaoImpl* automaticky přiřazena instance třídy *OsobaDaoImpl*, která je již uložena v IOC kontejneru za pomoci anotace *@Repository(”osobaDao”)*.
Dále se anotace *@Autowired* používá u metody *init()*:

```
@Autowired
public void init(DataSource dataSource) {
   this.simpleJdbcTemplate =
      new SimpleJdbcTemplate(dataSource);
   this.licenceInsert = new SimpleJdbcInsert(dataSource).
      withTableName("KIV_PROD_LICENCE").
      usingGeneratedKeyColumns("id");
   ...
}
```

Anotace vloží beanu *dataSource*, nadefinovanou v konfiguračním souboru *redsys-servlet.xml*, z IOC kontejneru do volání metody jako parametr.

3.3 Práce s tabulkami v DAO třídách

V metodě na předchozí ukázce je vidět navázání DAO třídy na konkrétní tabulku v databázi (pozor na psaní názvů tabulek – je třeba počítat s tím,

že MySql server je case sensitive, a proto názvy ve zdrojovém kódu tříd musí být velkými písmeny v souladu s datovým modelem).
Použitá třída typu *SimpleJdbcInsert* se nachází v balíčku:

- `org.springframework.jdbc.core.simple.SimpleJdbcInsert`

Ta umožňuje snadné vkládání dat do databáze a následné získání automaticky generovaného primárního klíče (pokud takový je). S touto třídou dále v našem projektu úzce souvisí třída *MapSqlParameterSource* z balíčku:

- `org.springframework.jdbc.core.namedparam.MapSqlParameterSource`

Tato třída dokáže namapovat sloupce tabulky na konkrétní proměnné instance objektu. Tuto možnost ukazuje následující kód pro instanci typu *Licence* a tabulku *KIV_PROD_LICENCE*:

```
private
MapSqlParameterSource getLicenceParametrSource(
   Licence licence) {
   return new MapSqlParameterSource()
      .addValue("id", licence.getId())
      .addValue("nazev_cz", licence.getNazevCz())
      .addValue("nazev_en", licence.getNazevEn())
      .addValue("text", licence.getText())
      .addValue("url", licence.getUrl())
      .addValue("zkratka", licence.getZkratka())
      .addValue("zaznam_editor", licence.getZaznamEditor())
      .addValue("zaznam_datum", licence.getZaznamDatum())
      .addValue("zaznam_aktivni", licence.isZaznamAktivni());
}
```

Toto mapování se využívá při operacích update a insert. Následující kód ukazuje update záznamu v tabulce *KIV_PROD_LICENCE* a využití tříd: *SimpleJdbcTemplate*, *SimpleJdbcInsert* a *MapSqlParameterSource*:

```
public void ulozLicenci(Licence licence) {
   if (licence.isNovyZaznam()) {
      licence.setId(licenceInsert.executeAndReturnKey(
         this.getLicenceParametrSource(licence)).intValue());
   }
   else {
      String sql = "update KIV_PROD_LICENCE set " +
```

```
                    "nazev_cz=:nazev_cz, " +
                    "nazev_en=:nazev_en, " +
                    "text=:text, " +
                    "url=:url, " +
                    "zkratka=:zkratka, " +
                    "zaznam_editor=:zaznam_editor, " +
                    "zaznam_datum=:zaznam_datum, " +
                    "zaznam_aktivni=:zaznam_aktivni " +
                    "where id=:id";
      this.simpleJdbcTemplate.update(sql,
         this.getLicenceParametrSource(licence));
   }
}
```

V kódu je dále vidět, jak se získá automaticky generovaný klíč, který se zároveň použije.

3.4 Doménové třídy

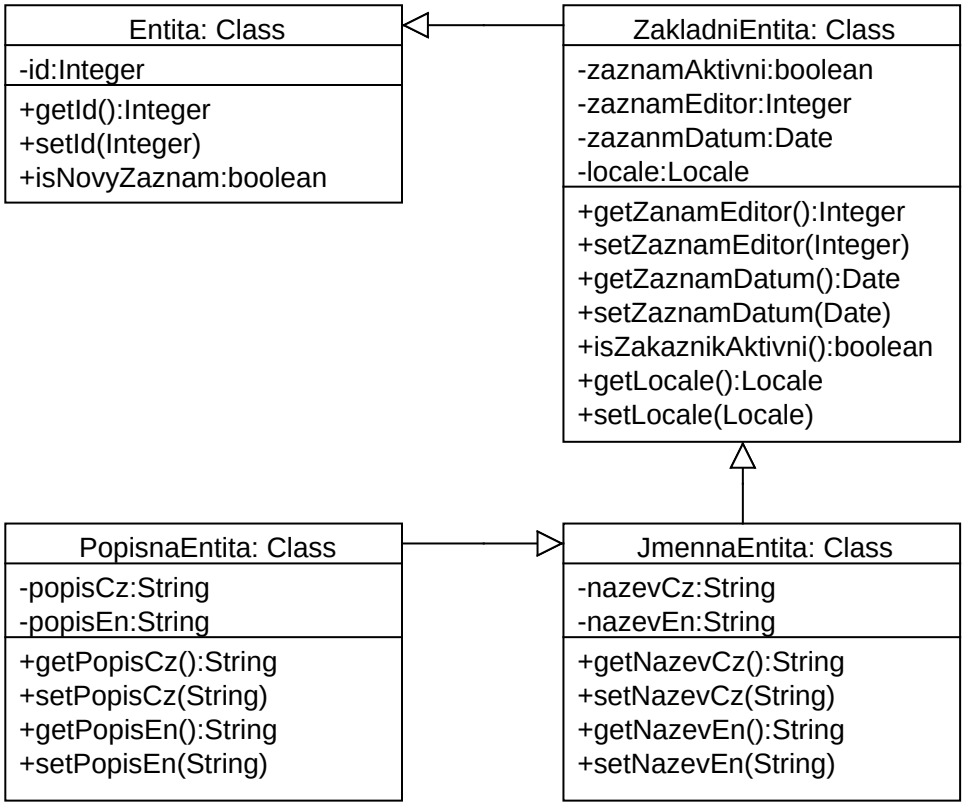
Třídy reprezentující entity datového modelu v aplikační logice jsou odvozeny od bázových tříd pro zajištění jednotné funkčnosti základních atributů.

3.4.1 Bázové třídy

Všechny bázové třídy se nachází v modulu *Common* a knihovně `webkiv-db` v balíku `cz.zcu.kiv.db.model`. Slouží jako rodičovské třídy pro všechny doménové třídy a k nim přidružené třídy. Vztahy mezi bázovými třídami je ukázán na diagramu tříd na obrázku 1.

Entita.java Poskytuje *id* jako jednoznačný identifikátor do databáze. Této vlastnosti využívají všechny doménové třídy a většina přidružených tříd.

ZakladniEntita.java Bázová třída pro tabulky v databázi které obsahují sloupce pro administrativní položky *zaznam_aktivni*, *zaznam_datum* a *zaznam_editor*. Položka *zaznam_aktivni* určuje, zda je záznam používán při načítání z databáze (deaktivace, tj. nastavení na *false*, znamená "smazání" hodnoty z pohledu aplikační vrstvy). Většinou se načítají pouze záznamy, které jsou aktivní, ale v implementaci se vždy zavádí i metody, které načtou současně i neaktivní záznamy.



Obrázek 1: Diagram tříd bázových tříd

JmennaEntita.java Bázová třída pro tabulky které obsahují sloupce *nazev_cz* a *nazev_en*. Tyto vlastnosti například obsahuje doménová třídy *Produkt.java* a k ní přidružená tabulka KIV_PROD_PRODUKTY. Obsah těchto sloupců se vypisuje na JSP stránkách, kde jsou přítomné seznamy daných položek, jako například na stránce se seznamem produktů (*seznam.jsp* v modulu *Produkty*).

PopisnaEntita.java Bázová třída pro tabulky které obsahují sloupce *popis_cz* a *popis_en*. Tyto vlastnosti například obsahuje doménová třídy *Produkt.java* a k ní přidružená tabulka *kiv_prod_produkty*. Obsah těchto sloupců se vypisuje na JSP stránkách, kde se zobrazuje detail nějaké položky jako například na stránce s detailem produktu *detail.jsp* v modulu *Produkty*. Tyto položky se dále zobrazují i u některých seznamů a při editaci konkrétní položky.

3.4.2 Doménové třídy

Všechny doménové třídy se nacházejí v modulu *Common*, knihovně `webkiv-db`. K níže uvedeným třídám existují další pomocné třídy navázané na tabulky v databázi. Tyto třídy jsou ale většinou navázané na číselníky nebo rozkladové tabulky.
Základní doménové třídy jsou:

- **Osoba.java** je svázána s tabulkou KIV_PERS_OSOBY a nachází se v balíčku `cz.zcu.kiv.db.model.osoby`
- **Publikace.java** je svázána s tabulkou KIV_PUBL_ZAZNAMY a nachází se v balíčku `cz.zcu.kiv.db.model.publikace`
- **Produkt.java** je svázána s tabulkou KIV_PROD_PRODUKTY a nachází se v balíčku `cz.zcu.kiv.db.model.produkty`
- **Soubor.java** je svázána s tabulkou KIV_PROD_SOUBORY a nachází se v balíčku `cz.zcu.kiv.db.model.produkty`
- **Predmet.java** je svázána s tabulkou KIV_STUD_PREDMETY a nachází se v balíčku `cz.zcu.kiv.db.model.studium`
- **Obor.java** je svázána s tabulkou KIV_STUD_OBORY a nachází se v balíčku `cz.zcu.kiv.db.model.studium`

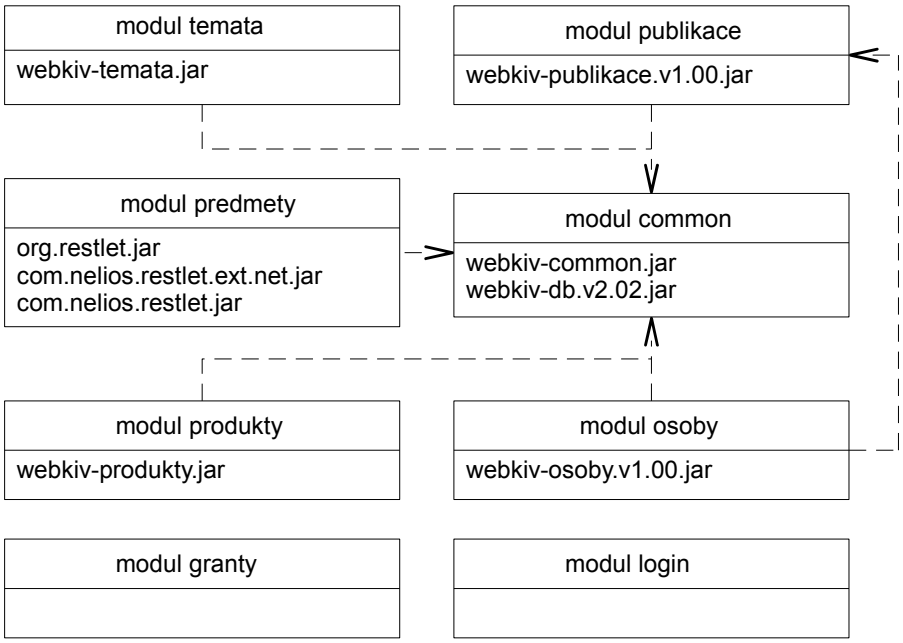
4 Zavedené moduly

Systém obsahuje několik základních modulů pro jednotlivé oblasti webu, které budou popsány v následujících částech. Na obrázku 2 jsou znázorněny vazby mezi moduly ve stavu k 11/2009:
Přehled použitých a nasazených verzí modulů se nachází na adrese:

- `http://wiki.kiv.zcu.cz/WebKiv/FunkcniKombinaceModulu`.

4.1 Modul Common

Modul *Common* je základní modul pro všechny ostatní moduly v systému. Modul obsahuje třídu *PermissionsAdapter*, která udržuje přístupová práva a oprávnění pro přihlášeného uživatele. Tyto údaje získává přímo z OpenCMS z nastavení uživatelů a uživatelských skupin.



Obrázek 2: Vazby mezi moduly

Nejdůležitější součástí modulu je knihovna *webkiv-db*, která poskytuje operace nad databází pro všechny závislé moduly (viz kap 3). Pomocí této knihovny se přistupuje do databáze, získávají se potřebné informace, úpravy uložených dat a vkládání nových dat.

4.2 Modul Produkty

Modul *Produkty* má na starosti správu a zveřejňování výzkumné činnosti katedry v oblasti softwaru. Náhled poslední nasazené funkčnosti je zde:

- `http://www.kiv.zcu.cz/vyzkum/software/`.

Pokud má přihlášený uživatel administrátorská práva, může spravovat licence a také jednotlivé uložené produkty. Než je produkt možno zveřejnit, musí projít schválením od vybraných uživatelů, kteří jsou součástí skupiny *Schvalovatel* definované v properties adresáře s front-endovými JSP stránkami modulu. Schvalovatel je na nově přidaný produkt upozorněn e-mailem.

Součástí modulu je také verzování jednotlivých produktů.

4.3 Modul Osoby

Modul *Osoby* slouží ke správě všech zaměstnanců uložených v databázi. Modul dovoluje spravovat osobní údaje, pozici zaměstnance, aktivity, členství v určité organizaci, vyučované předměty a publikace, které daný zaměstnanec napsal.
Na obrázku 2 je vidět závislost na modulu *Publikace*. Tato vazba je způsobena následujícím fragmentem kódu z JSP stránky *detail.jsp* z modulu osoby:

```
<cms:include page="/system/modules/
cz.zcu.kiv.publikace/pages/publikaceProOsobu.jsp">
   <cms:param name="login">
      <c:out value="\${osoba.login}"/>
   </cms:param>
</cms:include>
```

Zde je vidět vložení speciální JSP stránky z modulu *Publikace* pro zobrazení publikací konkrétní osoby[1].

4.4 Modul predmety

Modul slouží k zobrazování předmětů vyučovaných členy KIV. Předměty se aktualizují ze STAGu přes webové služby. Blížší informace o webových službách jsou na adrese:

- `https://stag-ws.zcu.cz/ws/help/list`.

4.5 Modul publikace

Modul slouží k zobrazování publikací členů KIV. Dostupná je editace a také se zde nachází JSP stránka *publikaceProOsobu.jsp*, která je použita v modulu *osoby*, jak ukazuje vazba na obrázku 2.

[1]Podobný mechanismus by měl být použit i pro ostatní vkládané funkčnosti, např. z modulu *Předměty*.

5 Vzory řešení

Framework pro web KIV používá MVC architekturu s DAO vrstvou a oddělením logiky od prezentace v modulech. Logika je vždy v pomocných třídách uložených v `lib/nazev-modulu.jar`, prezentační vrstva jsou JSP používající JSTL uložené v adresáři `pages/` modulu.

5.1 Návrhový vzor Mediátor

Architektura modulů je založena na návrhovém vzoru Mediátor, který slouží k obsluze požadavků přidružených JSP stránek. Pro bližší informace:

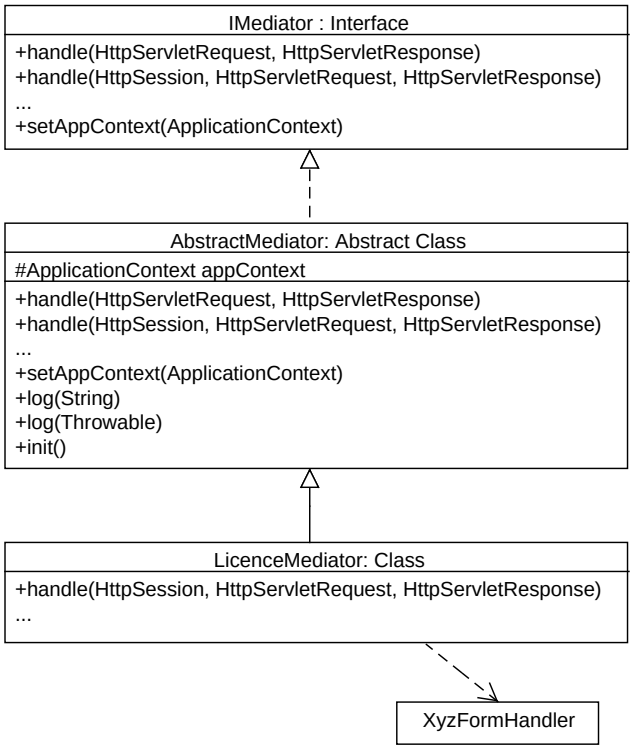
- `http://www.vincehuston.org/dp/mediator.html`
- `http://objekty.vse.cz/Objekty/Vzory-Mediator`.

Tento vzor řídí vazby mezi částmi aplikace tak, že se odpovídací funkčnost zapouzdří do řídícího objektu. Ten odděluje interní zprávy od zpráv, které je třeba předat dalším částem aplikace. Vzor Mediátor výrazně redukuje počet komunikačních cest mezi jednotlivými komponentami, kdy by JSP stránky přímo komunikovaly s DAO třídami. (Mediátory v obecném případě navíc umožňují vytvářet architekturu, ve které se komponenty starají jen o zprávy, které chtějí posílat, a nemusí se start o to, které objekty jsou cíli těchto zpráv.)

5.2 Implementace vzoru Mediátor

V systému používáme pro mediátory rozhraní *IMediator*, ve kterém jsou definované jednotlivé řídící metody *handle* pro obsluhu požadavků z jednotlivých JSP stránek. Metoda *handle* správně vyhodnotí druh požadavku a tomu přizpůsobí další činnost. Toto rozhraní je implementováno abstraktní třídou *AbstractMediator*, která obsahuje základní utility metody (viz Obrázek 3). V současné podobě implementace má každý modul své rozhraní *IMediator* a to z důvodu různorodých potřeb[2] (může si zavést vlastní metody *handle* s potřebným počtem a typem parametrů). Je ale samozřejmě důležité nezavádět zbytečné metody *handle* a raději lépe analyzovat situaci a pokusit se využít již zavedené metody *handle*.

[2]Ideálně bychom měli dospět k tomu že IMediator a AbstractMediator jsou společné (v modulu Common) a každý modul má svou třídu/třídy *XyzMediator* odděděnou od *AbstractMediator*.



Obrázek 3: Diagram tříd Mediátoru

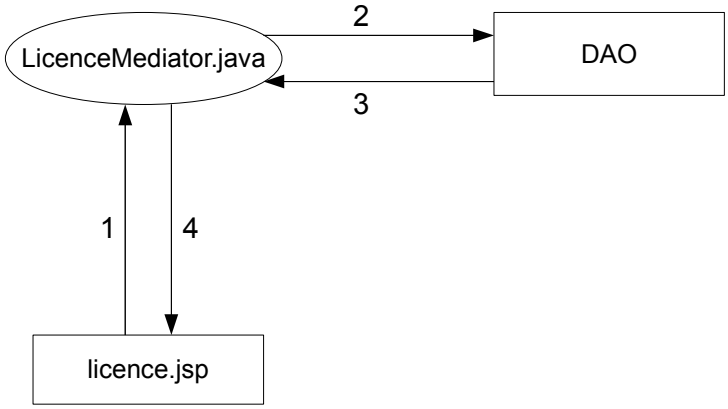
Pokud si to situace vyžádá, je lepší pro přehlednost a související funkčnost zavést obslužné třídy (nazývané např. *XyzFormHandler*) s vyšší soudržností, které jsou mediátorem využívány. Tento postup by měl být dodržen u všech nově vytvářených mediátorů. Popsané vztahy jsou znázorněny pomocí diagramu tříd na obrázku 3.

Poznámka: Dřívější implementace používala k obsluze požadavků třídy *FormHandler* a *Mediátor* bez striktního oddělení. Ve FormHandlerech byla většina funkčnosti pro obsluhu požadavku a to není správné využití návrhového vzoru Mediátor. U některých tříd nebylo možno původní implementaci převést na "čisté" mediátory z důvodu rozsahu implementace a provázání jednotlivých tříd.

5.3 Příklad

Implementace bude demonstrována na řešení z modulu *Produkty* a konkrétně JSP stránce *licence.jsp* a mediátoru *LicenceMediator.java*. V tomto vzorovém příkladu není použit FormHandler, protože implementace není tak rozsáhlá, aby bylo nutné vyčlenit obslužné metody do další třídy.

Obrázek 4 ukazuje pořadí akcí při vyvolání požadavku z JSP stránky a jeho následnou obsluhu Mediátorem.



Obrázek 4: Prace mediátoru

5.3.1 View stránka – licence.jsp

JSP stránky by měly obsahovat minimální množství scriptletů s vloženým Java kódem. Většinou dostačující kód pro obsluhu JSP stránky za použití Mediátoru ukazuje následující fragment ze stránky *licence.jsp*:

```
<%
// Získání Spring kontextu: jen je-li potřeba
ApplicationContext ctx = WebApplicationContextUtils.
        getWebApplicationContext(config.getServletContext());

IMediator mediator = new LicenceMediator(pageContext, request,
                                         response, ctx);

mediator.handle(session, request, response);
%>
```

Zde nejdříve získáme aplikační kontext pro případy, kdy mediátor potřebuje získávat DAO objekty ze Spring kontextu "ručně", viz kapitola 2. Poté pro rozhraní *IMediator* vytvoříme objekt s konkrétní implementací - *LicenceMediator*. Poslední řádka kódu odpovídá bodu *1* v obrázku 4 a slouží k obsluze požadavku, který vznikl otevřením stránky nebo jejím opětovným načtením. Následující fragment pak ukazuje vlastní prezentační část JSP, která následuje po scriptletu a v JSTL výrazech používá hodnoty nastavené v mediátoru. Zároveň je v něm vidět navázání na lokalizační *resource bundle* uložený v modulu a použití formátovacích značek pro výpis lokalizovaných řetězců (elementy `<fmt:`).

```
[frame=single, fontsize=\small]
<div id="content">
<fmt:setLocale value="${locale}" />
<fmt:bundle basename="cz/zcu/kiv/produkty/produkty">
  <h1><fmt:message key="licence.seznam" /></h1>
...
  <form action="<cms:link><c:out value="${folderUri}" />licence.\-
html</cms:link>" method="post" accept-charset="UTF-8">
...
  <c:when test="${!empty aktivniLicence}">
    <table>
    <c:set var="iterator" value="1" />
    <c:forEach items="${aktivniLicence}" var="licence">
      <tr><td><c:out value="${iterator}" />.</td>
...
```

5.3.2 Aplikační logika – LicenceMediator.java

Následující fragment kódu obsahuje samotnou metodu *handle* uvnitř třídy *LicenceMediator.java*.

```
public void handle(HttpSession session,
                   HttpServletRequest request,
                   HttpServletResponse response)
{
   ...

   if (this.request.getParameter("ulozLicenci") != null)
   {
      this.ulozLiceniDoDatabaze();
   }
```

```
   else if (this.request.getParameter("vyraditLicence") != null)
   {
      try
      {
        this.vyradLicenceZDatabaze(request);
        // Zde by také mohl být použit form handler nebo jiná
        // pomocná třída.
      }

   ...

   else if (this.request.getParameter("id") == null)
   {
      this.getSeznamLicenci();
   }
   else
   {
      this.getLicence();
   }
}
```

Zde se podle uložených atributů požadavku určí akce pro obsloužení požadavku. Je to jeden z možných postupů, jak rozpoznat typ požadavku a jeho obsluhu. Poslední fragment ukazuje konkrétní metodu která získá data z databáze a připraví je pro vrácení na JSP stránku. Data se zde konkrétně ukládají jako atributy požadavku stránky.

```
private void getSeznamLicenci() {
   List<Licence> seznamLicenci = produktDao.getLicence(true);
   List<Licence> aktivniLicence = new ArrayList<Licence>();
   List<Licence> vyrazeneLicence = new ArrayList<Licence>();

   if (seznamLicenci != null) {
      for (Licence licence : seznamLicenci) {
         if (licence.isZaznamAktivni() == true) {
            aktivniLicence.add(licence);
         }
         else {
            vyrazeneLicence.add(licence);
         }
      }

      this.request.setAttribute("aktivniLicence",aktivniLicence);
```

```
        this.request.setAttribute("vyrazeneLicence",vyrazeneLicence);
    }
}
```

Tato řádka kódu odpovídá v obrázku 4 bodu *2* a *3*, tedy volání příslušného objektu pro přístup do databáze a získání odpovídajících dat.

```
List<Licence> seznamLicenci = produktDao.getLicence(true);
```

Po ukončení metody *handle* se vrací řízení na JSP stránku se získanými daty. Toto odpovídá bodu *4* z obrázku 4.