**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE**
**FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-5384-50877

# IMPLEMENTÁCIA WORKFLOW MANAŽMENT SYSTÉMU

## DIPLOMOVÁ PRÁCA

**2012** **Bc. Tomáš Zuber**

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE**
**FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-5384-50877

# IMPLEMENTÁCIA WORKFLOW MANAŽMENT SYSTÉMU

## DIPLOMOVÁ PRÁCA

| | |
|---|---|
| Študijný program: | Aplikovaná informatika |
| Číslo študijného odboru: | 2511 |
| Názov študijného odboru: | Aplikovaná informatika |
| Školiace pracovisko: | Fakulta elektrotechniky a informatiky |
| Vedúci záverečnej práce/školiteľ: | doc. RNDr. Gabriel Juhás, PhD. |

**Bratislava 2012** **Bc. Tomáš Zuber**

# Abstrakt

Slovenská technická univerzita v Bratislave
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

| | |
|---|---|
| Študijný program: | Aplikovaná informatika |
| Meno autora: | Bc. Tomáš Zuber |
| Názov diplomovej práce: | Implementácia workflow manažment systému |
| Vedúci diplomovej práce: | doc. RNDr. Gabriel Juhás, PhD. |
| Rok odovzdania: | Máj 2012 |

Workflow manažment systém vo forme webaplikácie bol implementovaný so správcovskou a užívateľskou časťou. Hlavnými cieľmi bolo zabezpečiť použiteľnosť v praxi, konzistentnosť dát pri práci viacerých užívateľov s rovnakým workflowom, registráciu nových užívateľov a ich manažovanie, použiť bezpečnostné prvky, ktoré ochránia aplikáciu pred bežnými útokmi. Užívatelia v rámci systému majú jednu z rolí administrátora, manažéra alebo užívatela a v rámci všetkých workflowov majú priradené role, ktoré povolia aktivácie úloh pre danú rolu.
Workflowy sú vytvárané pomocou Petriho sietí a dátových formulárov skladajúcich sa z dátového modelu, webstránok a validácií polí. Systém používa k funkčnosti viacero výstupov z hotových aplikácií a integruje ich do jedného komplexného prostredia.
Pri implementácii boli použité najnovšie technológie vývoja Java Enterprise aplikácii ako sú Java Server Faces 2.1, Enterprise Java Beany 3, Java Persistence API 2, Hibernate 3.6, aplikačný server GlassFish 3.1, databáza MySQL 5.1, vývojové prostredie Eclipse 3.6 a systém pre správu zdrojových kódov Subversion.

Kľúčové slová: workflow manažment systém, Petriho sieť, web aplikácia, Java

# Abstract

Slovak University of Technology in Bratislava
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

| | |
|---|---|
| Degree course: | Applied Informatics |
| Author : | Bc. Tomáš Zuber |
| Title of the diploma theses: | Implementation of a workflow management system |
| Supervisor: | doc. RNDr. Gabriel Juhás, PhD. |
| Year of the submission: | May 2012 |

The workflow management system in the form of a webapplication was implemented with an adminstrator and user part. The main goals were to provide usability in real life, data consistency with multiple users using the same workflow, registration of new users and their management, use security features which defend the application from common attacks. In the scope of the system the users may have the role of an administrator, manager or user and in the scope of the workflows they are assigned to roles which allow the activation of the tasks for the given role.

The workflows are created using Petri nets and data forms consisting of a data model, websites and field validations. The functionality is provided from the output of multiple applications and integrates them into one complex environment.

At development the newest technologies for Enterprise Java were used like Java Server Faces 2.1, Enterprise Java Beans 3, Java Persistence API 2, Hibernate 3.6, application server GlassFish 3.1, database MySQL 5.1, integrated development environment Eclipse 3.6 and source code management Subversion.

Key words: workflow management system, Petri net, web application, Java

# Obsah

# 1 Úvod

Cieľom práce je využiť, aplikovať a integrovať poznatky z doterajšieho štúdia o databázach, aplikačných serveroch, klient-server aplikáciách, objektovo relačnom mapovaní, tvorbe webstránok, programovaní v Jave, Petriho sietiach do uceleného projektu Workflow manažment systém.

Hlavnou úlohou je navrhnúť, implementovať a otestovať webovú aplikáciu, ktorá je schopná manažovať vytváranie, priradenie a spracovanie úloh registrovanými užívateľmi. Celkový workflow bude zabezpečovať sémantika Petriho sietí, ktorá umožňuje jednoduché vytvorenie rôznych sietí. Aplikácia v sebe zahrnie komponenty na registráciu užívateľov, prihlásenie sa do systému, zobrazenie a práca s workflowmi, prípadmi, úlohami, formulármi, importovanie súborov z iných projektov, nastavenia workflowov a právomocí užívateľov.

Importovanie súborov bude možné nahrávaním cez webstránku, ktorá tieto súbory spracuje a uloží do databázy alebo do adresárovej štruktúry. Pri spracovaní sa budú používať knižnice, ktoré používajú príslušné programi pri vytváraní týchto súborov alebo budú vytvorené vlastné nástroje. Importovaná bude Petriho sieť z PN editora, dátové modely, dátové formuláre a ďalšie potrebné súbory.

Systém bude zložený z 3 hlavných komponentov na základe návrhového vzoru MVC – Model, View, Controller. Model zahŕňa tabuľky v databáze, ich reprezentácia Java triedami a funkcie pre správu údajov medzi nimi. Controller bude spravovať tieto údaje, spracovávať ich, navigáciu medzi stránkami a zabezpečí väčšinu biznis logiku systému. View komponent rieši zobrazenie, zbieranie a validáciu údajov.

Pri návrhu som sa rozhodol použiť len voľne dostupné programy a servery:

Vývojové prostredie – Eclipse – najpoužívanejšie IDE pre vývoj v Jave

Aplikačný server – Glassfish – nízke hardvérové nároky, jednoduchá integrácia v Eclipse

Databázový server – MySQL – jeden z najznámejších open source databáz

Užívateľské rozhranie – JSF 2.0 – jednoduchý framework spĺňajúci najnovšie štandardy

Perzistenčná vrstva – Hibernate – najpoužívanejší perzistenčný framework

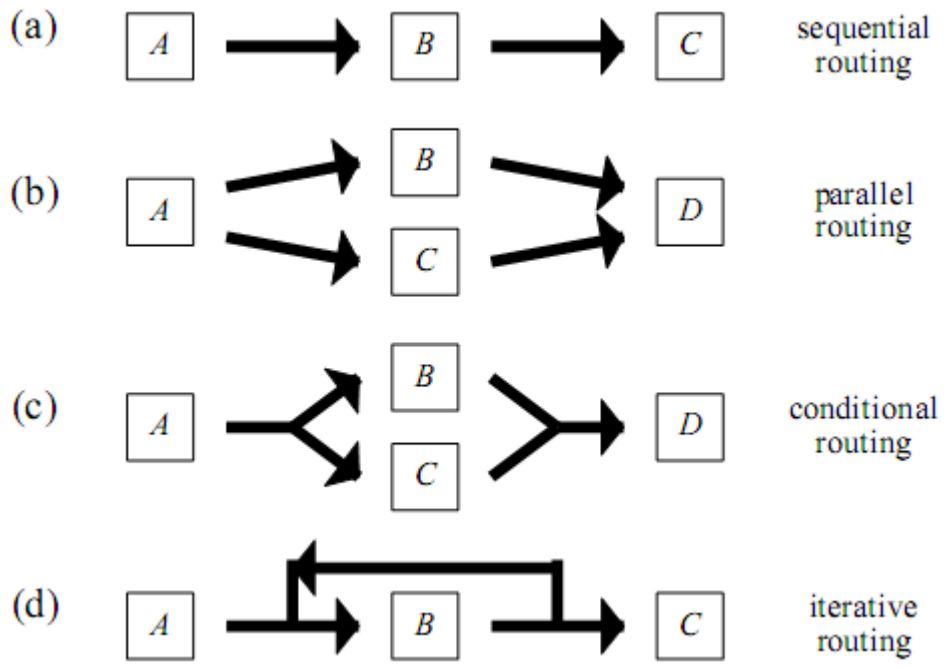
# 2 Analýza problému

V rokoch 1960 bolo trendom vytvárať samostatné programi, ktoré používali nejednotné užívateľské rozhrania a nejednotné spôsoby ukladania dát, čo spôsobilo neprenositeľnosť údajov medzi aplikáciami. O 10 rokov neskôr programátori prišli s riešením. Oddelili aplikáčnú časť od dátovej pomocou databáz a systémov riadenia databáz. Tento krok zabezpečil rovnaký prístup k dátam cez SQL dotazy a tým zjednodušil prácu aplikácie.
V roku 1970 podobný krok aplikovali na užívateľské rozhranie. Vyvynuli sa systémy riadenia používateľského rozhrania, ktoré oddelili užívateľskú časť od programu.
Ďalším krokom vývoja systémov je workflow manažment, ktorý má zabezpečiť separáciu funkcionality od systému. Jeho hlavnými prínosmi sú jednoduchšie riadenie, optimalizácia, kontrola a prehľadnejšie biznis procesy.

*„Workflow management systém (WFMS) je všeobecný programový nástroj, ktorý poskytuje definíciu, vykonanie, registráciu a kontrolu procesov. Procesy sú jedným z najdôležitejších častí workflow managementu , preto je dôležité použiť vhodnú existujúcu štruktúru pre modelovanie a analýzu workflow procesov.“* [18]
Takúto štruktúru vytvoril Carl Adam Petri v roku 1939, ktorá sa nazvala podľa jeho mena Petriho sieť – Petri net. Jej nedostatky a využitie v špeciálnych prípadoch postupom času boli odstránené doplnkami ako farba, čas a hierarchia. Výhodou Petriho sietí je formálna sémantika (majú presnú definíciu), grafické zobrazenie (prehľadné a ľahko pochopiteľné aj pre netechnických ľudí), expresivita (možnosť implicitného modelovania) a analýza (invarianty, deadlocky, bezpečnosť, čas čakania, odozvy a celková priechodnosť siete). [4]

## 2.1 Workflow manažment systém

Workflow manažment systém je doména zameriavajúca sa na logistiku procesov. Kritickým faktorom pri vykonávani týchto procesov je zabezpečiť správne poradie vykonávania úloh správnym človekom.
Tento systém je definovaný podľa Workflow Management Coalition (WfMC) ako:
„*Systém, ktorý kompletne definuje, manažuje a vykonáva workflowy cez spúšťanie programov, ktorých poradie vykonávania je riadené počítačovou reprezentáciou workflow logiky“*. [3, str. 6]




Obr.1 Smerovacie konštrukcie [1,str.7]

Smerovacie konštrukcie zobrazené na obrázku boli vytvorené medzinárodnou organizáciou Workflow Management Coalition (WfMC), ktorá podporuje a vytvára štandardy pre workflow manažment systémy. WfMC bola vytvorená v roku 1993 a o dva roky neskôr vydali slovník všeobecných termínov pre vývojárov, zákazníkov a výskumníkov.

Slovník definuje štyri typy smerovania:

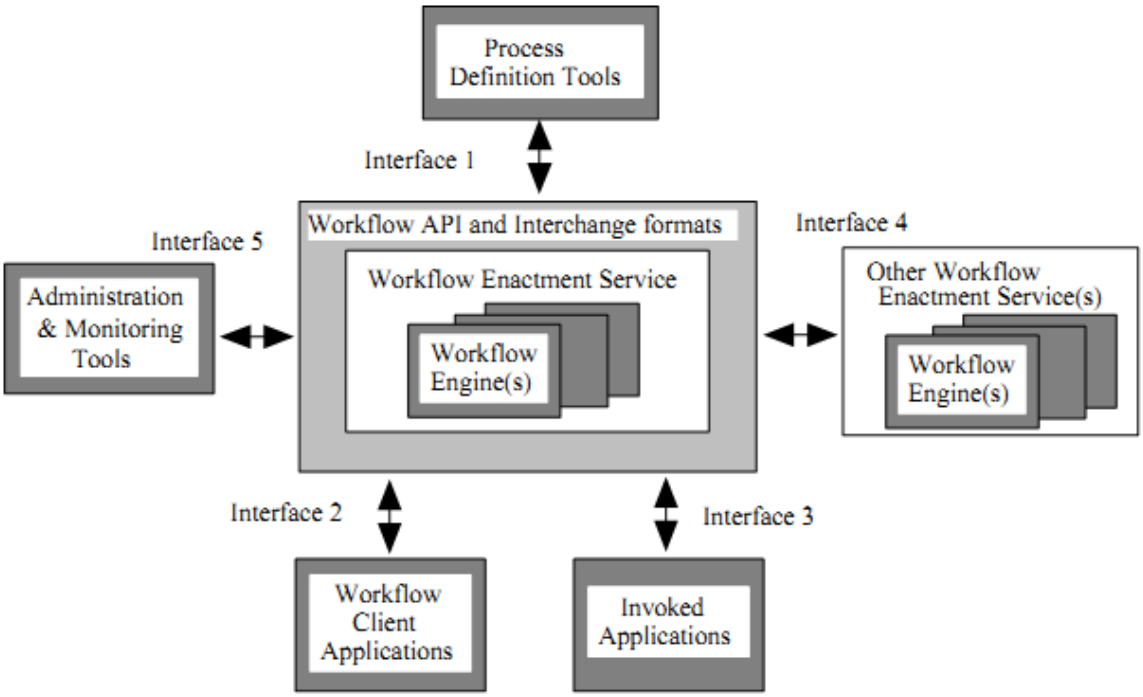
(a) sekvenčné *(sequential)* – postupné vykonávanie úloh za sebou

(b) paralelné *(parallel)* – paralelné vykonávanie úloh naraz pomocou AND-splitov a AND-joinov

(c) podmienený *(conditional)* – vykonanie úloh záleží od určitých podmienok realizované OR-splitmi a OR-joinami

(d) iteračný *(iterative)* – vykonanie úlohy alebo skupiny úloh viackrát za sebou

*„Workflow manažment systém musí byť nainštalovaný, správne nakonfigurovaný a naplnený dátami o procesoch. Vzhľadom na túto skutočnosť sa termín workflow systém používa pre celkové nasadenie workflow manažment systému zahrňujúc definíciu procesov, organizačné dáta, využitie, aplikačné dáta, systém riadenia databáz, konfiguračné súbory a ostatné programové komponenty použité pre aktuálny systém.“* [18]




Obr.2 Referenčný model podľa WFMC [1,str.10]

Referenčný model zobrazuje najdôležitejšie rozhrania a časti architektúry workflow manažment systému. Základom všetkých workflow systémov je nariaďovacia služba (enactment service). Jej úlohou je poskytnúť run-time prostredie, ktoré sa postará o kontrolu a vykonanie workflowov.

## 2.2 Petriho siete

*„Klasická petriho sieť je orientovaný bipartitný graf s dvoma typmi uzlov nazývaných miesta (places) a prechody (transitions). Uzly sú spojené cez orientované hrany (arcs). Spojenia medzi uzlami rovnakého typu nie sú povolené.“* [18]

*„Definícia 1:*
Petriho sieť je trojica (P, T, F), pre ktorú platí:

- P je konečná množina miest
- T je konečná množina prechodov, $P \cap T = \emptyset$
- F je podmnožina $(P \times T) \cup (T \times P)$, F je množina hrán „ [1, str.12]

*„Miesto p je nazývané vstupné miesto pre prechod t, ak existuje orientovaná hrana z p do t. Miesto p je nazývané výstupné miesto pre prechod t, ak existuje orientovaná hrana z t*

*do p. Hrany môžu mať kladnú násobnosť. Ak nie je vyznačená násobnosť, počíta sa s násobnosťou 1. Miesta obsahujú nula a viac tokenov.“* [18]

Prechody sú aktívne prvky siete, ktoré menia značkovanie (stavy) podľa nasledovných pravidiel:

- Prechod je spustiteľný, ak každé jeho vstupné miesto obsahuje aspoň toľko tokenov, koľko je násobnosť hrany miestom a prechodom.
- Ak sa prechod spustí, tak sa z každého vstupného miesta spotrebujú tokeny podľa násobnosti hrany a vytvorí sa toľko tokenov v každom výstupnom mieste, koľko má násobnosť hrana.

*„Definícia 2 (živosť):*
Petriho sieť (PN, M) je živá, ak pre každý dosiahnuteľný stav M' a pre každý prechod t existuje stav M" dosiahnuteľný z M', ktorý povolí t.

*Definícia 3 (ohraničenosť, spoľahlivosť):*
Petriho sieť (PN, M) je ohraničená, ak každé miesto p obsahuje také prirodzené číslo n, že pre každý dosiahnuteľný stav je počet tokenov v p menej ako n. Sieť je spoľahlivá, ak pre každé miesto je maximálne jeden token.
Cesty spájajú uzly v súvislosti s hranami. Cesta je jednoduchá, ak každý uzol je jedinečný.

*Definícia 4 (cesta,abeceda):*
Nech PN je Petriho sieť. Cesta C z uzla $n_1$ do uzla $n_k$ je taká postupnosť $(n_1,n_2,...,n_k)$, že $(n_i,n_{i+1}) \in F$ pre $1 \leq i \leq k - 1$. $\alpha(C) = \{n_1,n_2,...,n_k\}$ je abeceda C. C je jednoduché, ak pre ľubovoľnú dvojicu uzlov $n_i$ a $n_j$ na C, $i \neq j \Rightarrow n_i \neq n_j$.

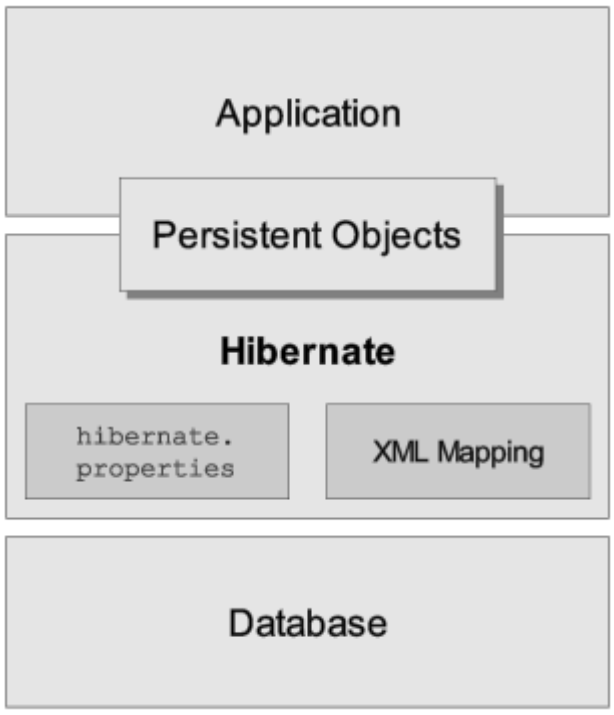
*Definícia 5 (silno súvislá):*
Petriho sieť je silno spojená, ak pre každú dvojicu uzlov x a y, existuje cesta z x do y.“
[1, str.13]

## 2.3 Perzistenčný framework Hibernate

Hibernate je skupina projektov, ktoré slúžia na objektovo-relačné mapovanie dát:

- Hibernate core
- Hibernate search
- Hibernate tools
- Hibernate validator
- Hibernate metamodel generator

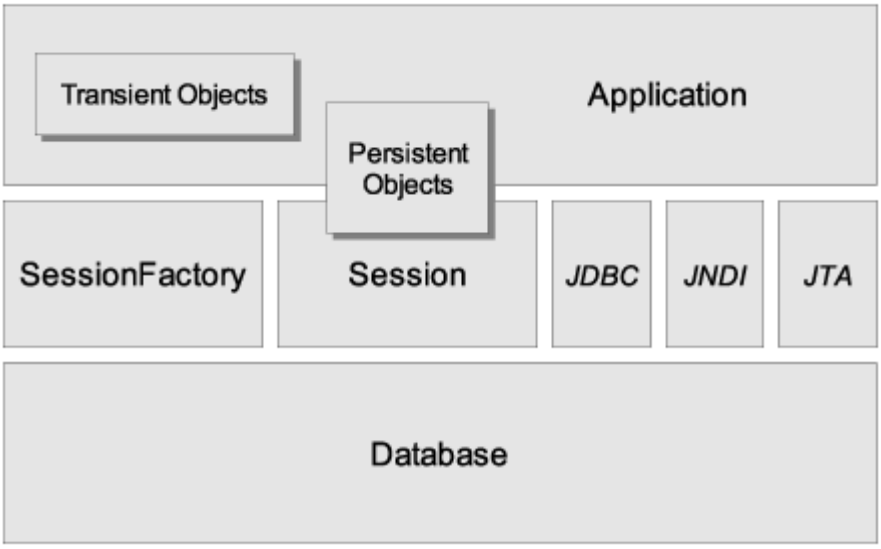
Hibernate core je hlavný projekt a ostatné projekty slúžia na zjednodušenie práce s týmto frameworkom. Zámerom projektu je zrýchliť a zjednodušiť vývoj aplikácii s relačnými databázami. Termín objektovo-relačné mapovanie je technika mapovania dát z objektov na relačné dáta v databáze a opačne. Toto riešenie odbremení prográmatora od vytvárania SQL dotazov pre získanie dát. Prográmtorovi stačia základné znalosti z databáz a môže jednoducho ukladať,meniť a mazať dáta cez funkcie Hibernate-u. Namapované dáta sa pri transakciách automaticky pretypujú a validujú.




Obr.3 Prehľad modelu 3 vrstvovej aplikácie s Hibernate-om [14]

Pre použitie je potrebné stiahnuť knižnice, pridať ich do classpath-u a vytvoriť konfiguračné súbory. Hibernate potrebuje vedieť ako má načítavať a ukladať objekty perzistentnej triedy. Tieto informácie mu vieme poskytnúť konfiguračným súborom, ktorý mu povie na akú databázu sa má pripojiť, akým prihlasovacím menom a heslom, na akú URL adresu a ktoré triedy má namapovať. V novších verziách je možné niektoré nastavenia urobiť aj pri inštanciovaní Hibernate objektu a pomocou Java anotácií.

Hibernate pracuje s objektami v rámci session, v ktorých sú dáta objektov synchronizované s dátami v databáze, t.j. pri každej zmene hodnoty sa uložia nové dáta. V prípade, že je potrebné zmeniť viac dát atomicky, tak je možné začať transakciu, zmeniť údaje a poslať transakciu – commit. V prípade chyby sa všetky zmeny v rámci transakcie vrátia do pôvodného stavu – rollback. Transakcie fungujú podobne ako pri obyčajnom SQL dotazovaní.




Obr.4 Minimálna architektúra aplikácie s Hibernate-om [14]

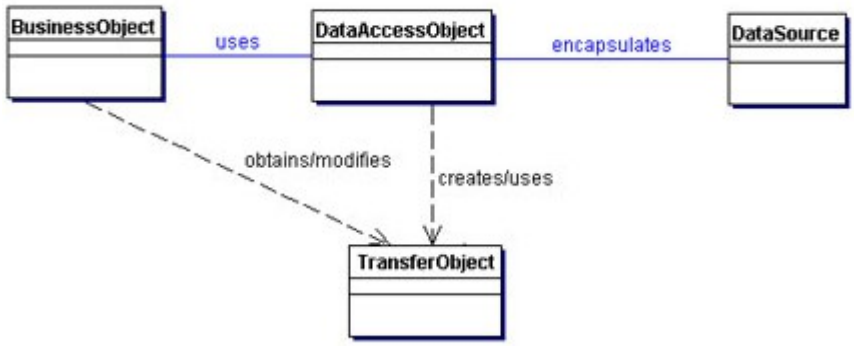
### 2.3.1 Mapovanie tried

Mapovanie tried je vytvárané cez konfiguračné súbory, ktoré sa viažu na jednu triedu. Tento súbor zahŕňa všetky atribúty triedy, ktoré chceme mapovať na databázu. Dátové typy premenných vie Hibernate väčšinou automaticky nastaviť, ale v niektorých prípadoch je lepšie explicitne definovať typ, ktorý sa má použiť v databáze. Ďalším nastavením je primárny kľúč podľa, ktorého sa dá objekt jednoznačne identifikovať. Tento kľúč môže používať generátor Hibernate-u pre vytvorenie unikátnej hodnoty pri vytváraní nových objektov. Závislosti medzi triedami je možné definovať nastaveniami one-to-one, many-to-

one, one-to-many, many-to-many a skupiny údajov bag a set. Všetky tieto nastavenia je možné urobiť aj cez anotácie priamo v kóde Java triedy. Mapovať sa dá len POJO (Plain Old Java Object) triedy – obsahujú konštruktor bez parametrov, sú serializovateľné a k atribútom sa pristupuje cez getter a setter metódy.

### 2.3.2 Všeobecné použitie a návrhové vzory

Pri perzistenčnej vrstve je zvykom použiť návrhové vzory DTO (Data Transfer Object) a DAO (Data Access Object). Objekty DTO sa používajú pre výmenu dát medzi aplikáciou a databázou. Neobsahujú žiadne metódy okrem getterov a setterov. Slúžia čisto na ukladanie údajov. Objekty DAO sa používajú pre implementovanie bežných operácii nad dátami ako sú vytváranie, načítanie, zmena, mazanie, výber závislých dát a iné. Zaobaľujú prístupové mechanizmy potrebné pre prácu s databázou, skrývajú detaily pred klientom a zabezpečujú jednoduchú zmenu zdroja dát.




Obr.5 Data Access Objects [15]

## 2.4 Enterprise Java Beans

Enterprise bean-y sú Java EE komponenty, ktoré implementujú Enterprise JavaBeans (EJB) technológiu. Enterprise bean-y bežia v EJB kontainery v rámci aplikačného servera. Je transparentný pre programátora, ale poskytuje servisy na systémovej úrovni ako napríklad transakcie a bezpečnosť. Vďaka týmto servisom je možné rýchle vytvorenie a nasadenie enterprise bean, ktoré spoločne vytvárajú základ jadra transakčných Java EE aplikácií.

EJB je komponent na strane servera (server side), ktorý zapuzdruje biznis logiku aplikácie. Biznis logika je kód spĺňajúci požiadavky aplikácie. EJB zjednodušujú tvorbu veľkých,

distribuovaných aplikácií, lebo poskytujú servisy na úrovni systému a tým umožňujú programátorovi sústrediť sa na biznis problémy. Za manažment transakcií, zachovanie integrity dát a bezpečnú autorizáciu je zodpovedný kontainer. Ďaľšou výhodou je prenositeľnosť komponentov medzi rôznymi aplikáciami alebo medzi rôznymi Java EE servermi, ktoré spĺňajú požadované štandardy.
Existujú 2 typy Enterprise Bean:

- Session – vykoná úlohu pre klienta
- Message driven – funguje ako poslucháč pre určité typy správ

### 2.4.1 Session bean

Session bean-a reprezentuje jedného klienta v rámci aplikačného servera. Klient volá jej metódy, aby sa dostala k aplikácii nasadenej na servery a vykonáva prácu za klienta ochraňujúc ho od komplexity spúšťaním biznis úloh vo vnútri servera.
Session bean-a nie je zdielateľná, môže mať len jedného klienta rovnakým spôsobom ako jedna interaktívna session môže mať iba jeden užívateľ. Session bean nie je perzistentná, jej dáta nie sú ukladané v databáze, čiže pri ukončení komunikácie klientom, session bean už ďalej nepatrí klientovi. Z hľadiska manažovania ich stavu ich delíme na Stateful a Stateless.
Všeobecné okolnosti, kedy využiť session bean-y:

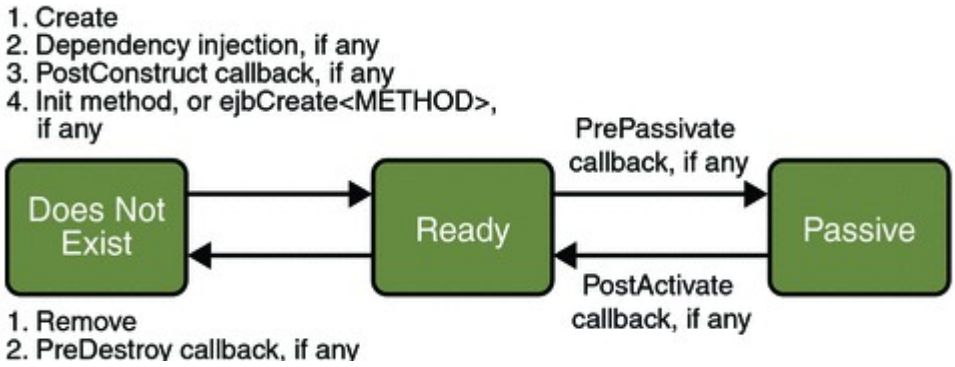
- v každom časovom okamžiku iba jeden klient môže mať prístup k inštancii bean-y
- stav bean-y nie je perzistentný, existuje iba po krátku dobu (niekoľko hodín)
- bean-a implementuje web servis

**Stateful session bean**

Pri stateful session bean-ach je stav objektu zložený z hodnôt jeho inštančných premenných, ktoré reprezentujú unikátny stav klient-bean session. Keďže klient komunikuje s bean-ov, tento stav sa často nazýva komunikačný stav. Tento stav je udržiavaný, kým trvá klientova session. Ak klient vymaže bean-u alebo končí, tak sa ukončí session a stav zmizne. Táto nestála vlastnosť neprináša so sebou žiadny problém, lebo po ukončení komunikácie medzi klientom a bean-ov už nie je potrebné udržiavať stav.
Okolnosti, kedy využiť stateful session bean-y:

- Stav bean-y reprezentuje interakciu medzi bean-ov a špecifickým klientom

- Bean-y potrebujú udržiavať informácie o klientovi medzi volaniami metód
- Bean-a pracuje medzi klientom a inými komponentmi v aplikácii a vytvára tak zjednodušený pohľad pre klienta
- Bean-a manažuje tok práce medzi viacerými enterprise beanami



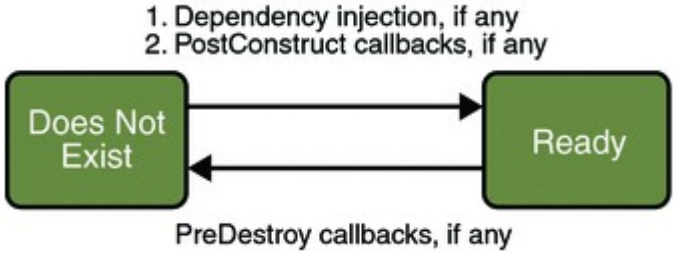

Obr.6 Stateful session bean [16]

**Stateless session bean**

Stateless session bean-y neudržiavajú komunikačný stav s klientom. Ak klient zavolá metódy bean-y, tak inštančné premenné bean-y môžu obsahovať stav špecifický pre klienta, ale len po dobu kým trvá toto volanie. Po skončení metódy by klientsky stav nemal byť zachovaný, ale klienti môžu zmeniť stav inštančných premenných v pool-ovaných stateless bean-ach a tento stav je udržiavaný až po ďalšie volanie bean-y. Okrem fázy volania metód sú všetky inštancie stateless bean rovnaké, čo umožňuje EJB kontaineru priradiť inštanciu každému klientovi. Keďže stateless session bean-y môžu podporovať viacero klientov, tak sú ľahšie škálovateľné pre aplikácie s vysokým počtom klientov. Typicky aplikácie potrebujú menej stateless session bean ako stateful session bean pre udržanie rovnakého počtu klientov. Stateless session bean-y sú jediné enterprise bean-y, ktoré môžu implementovať web servis.

Pre zvýšenie rýchlosti je možné použiť stateless session bean-u ak:

- Stav bean-y neobsahuje dáta špecifické pre klienta
- Pri volaní metódy, ktorá vykonáva generické úlohy pre všetkých klientov, ako napríklad posielanie e-mailov




Obr.7 Stateless session bean [16]

### 2.4.2 Message driven bean

Message driven bean-a je enterprise bean-a, ktorá umožňuje Java EE aplikáciám spracovať správy asynchrónne. Bežne sa správa ako JMS (Java Message Service) listener, ktorý je podobný k udalostnému listner-u, ale prijíma JMS správy namiesto udalostí. Správy môžu byť posielané každým Java EE komponentom, JMS aplikáciou alebo aj systémom bez Java EE technológií.

Najviac viditeľným rozdielom medzi message driven bean-ami a session bean-ami je v prístupe k nim. Message driven bean-y nie sú prístupné cez interface-y, ale obsahujú len triedy. Z niekoľkých pohľadov sa podobajú stateless session bean-am:

- Neudržiavajú žiadne dáta ani komunikačné stavy pre špecifických klientov
- Všetky inštancie message driven bean sú rovnaké, čo umožňuje EJB kontaineru priraďovať správy ľubovoľnej inštancii bean-y. Kontainer môže pool-ovať tieto inštancie umožňuje konkurentné spracovanie prúdu správ.
- Jedna message driven bean-a môže spracovať správy od mnoho klientov

Inštančné premenné message driven bean-y môžu obsahovať stav pri spracovávaní klientskych správ, napr. JMS API spojenie, otvorené databázové spojenie alebo objektová referencia na enterprise bean objekt. Klientske komponenty nealokujú message driven bean-y a nespúšťajú metódy priamo na nich. Namiesto toho klient pristupuje k bean-e napr. JMS posiela správy na MessageListener – správový cieľ message driven bean-y. Tento cieľ sa nastavuje počas nasadzovania použitím zdrojov aplikačného servera.

Charakteristika message driven beany:

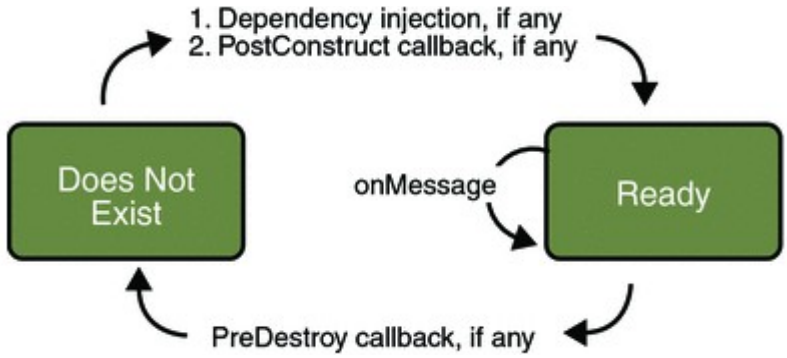
- Spúšťané po prijatí klientskej správy
- Spúšťané asynchrónne

- Majú relatívne krátky život
- Nereprezentujú priamo zdielané dáta v databáze, ale môžu k nim pristupovať a meniť tieto dáta
- Môžu používať transakcie
- Sú bezstavové

Po príchode správy kontajner zavolá metódu onMessage na spracovanie správy. Metóda onMessage pretypuje správu na jednu z piatich JMS typov a spracuje ju na základe biznis logiky aplikácie. Táto metóda môže volať pomocné metódy alebo session bean-u na spracovanie informácie v správe alebo uloženie do databázy.

Správa môže byť doručená message driven bean-e v rámci transakčného kontextu, čiže operácie v rámci onMessage metódy sú súčasťou jednej transakcie. Ak spracovanie správy zlyhá, správa bude znovu doručená.

Message driven bean-y je vhodné použiť, keď je potrebné posielať a prijímať asynchrónne. Týmto sa odstráni viazanie zdrojov servera na blokujúce synchrónne prijímanie správ a vo všeobecnosti JMS správy by nemali byť prijímané synchrónne.




Obr.8 Java messaging service [16]

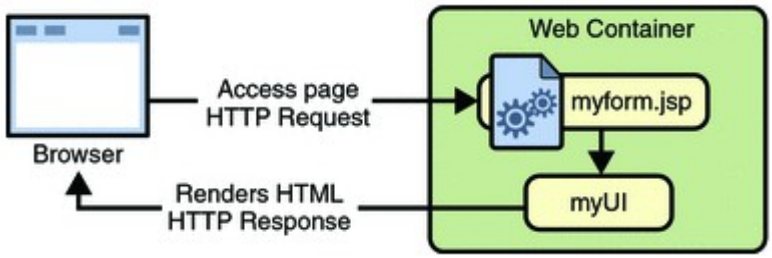
## 2.5 Java Server Faces

Java Server Faces (JSF) je server side framework užívateľského rozhrania (UI) pre Java web aplikácie. Hlavnými komponentmi tejto technológie je API pre reprezentáciu UI komponentov, manžovanie ich stavu, spracovanie udalostí, validácie na strane servera, konverzia dát, navigácie medzi stránkami, podpora internacionalizácie, rozšíriteľnosť vlastností a dve Java Server Pages (JSP) knižnice s vlastnými tag-mi pre zobrazenie UI komponentov a ich naviazanie na server side objekty. Správne definované programovacie modely a tag-ové knižnice zjednodušujú tvorbu a údržbu web aplikácií:

- Vkladanie komponentov na stránku pridaním komponentového tag-u

- Naviazanie udalostí generovaných komponentmi na server side aplikačný kód
- Navmapovanie UI komponentov na server side dáta
- Konštrukcia UI cez prepoužiteľné a rozšíriteľné komponenty
- Uložiť a obnoviť UI stav aj po živote požiadaviek servera

Vytvorené JSF UI komponenty bežia na servery a zobrazujú sa klientovi.




Obr.9 Java Server Faces technológia [17]

Stránka myform.jsp je JSP stránka, ktorá obsahuje JSF tagy. Zobrazuje UI komponenty použitím vlastných tag-ov definovaných JSF technológiou. UI web aplikácie manažuje objekty referencované JSP stránkov. Tieto objekty zahŕňajú:

- Objekty UI komponentu mapované na tagy v JSP stránke
- Všetky udalostné listenery, validátory a konvertory, ktoré su registrované na komponenty
- Java bean komponenty zapuzdrujú dáta a funkcionalitu komponentov špecifické pre aplikáciu

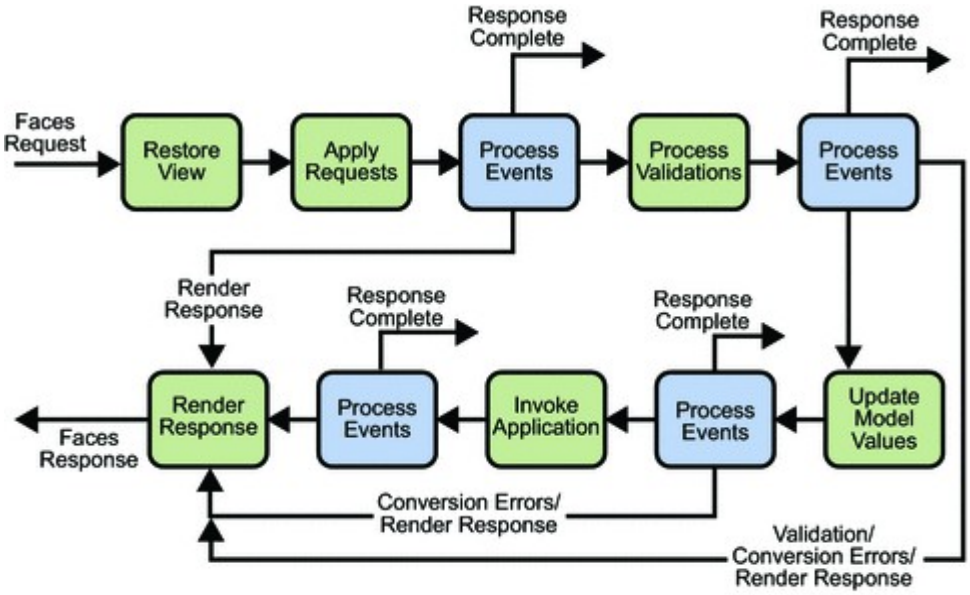
Najväčšou výhodou JSF technológie je oddelenie funkcionality od zobrazovania. Web aplikácie používajúce JSP technológiu dosahujú toto oddelenie len čiastočne. JSF aplikácia naviac poskytuje mapovanie HTTP požiadaviek na spracovanie udalostí a manažovanie UI elementov ako objektov so stavom na serveri.
Oddelenie funkcionality od zobrazovania taktiež umožňuje programátorom v tíme pracovať oddelene a sústrediť sa na vlastnú časť práce a poskytuje jednoduché premostenie týchto komponentov. JSF technológia poskytuje bohatú architektúru pre manažovanie stavov komponentov, spracovanie dát komponentov, validovať vstupy od užívateľov a spracovanie udalostí.

Všeobecne aplikácie s JSF používajú:

- Skupinu JSP alebo XHTML stránok
- Skupinu backing bean, ktoré sú Java bean-y definujúce vlastnosti a funkcionalitu UI komponentov na stránke
- Nasadzovacie deskriptory (web.xml)
- Skupina vlastných objektov vytvorených programátorom – komponenty, validátory, konvertory alebo listenery
- Skupina vlastných tag-ov na reprezentáciu vlastných objektov

Kroky používané pri vývoji:

- Mapovanie inštancie FacesServlet
- Tvorba stránok s UI komponentmi a tag-mi s jadra
- Definovanie navigácií medzi stránkami v konfiguračnom súbore
- Pridávanie manažovaných bean do konfiguračného súboru




Obr.10 Java Server Faces life cycle [17]

# 3 Opis riešenia

Výsledkom práce by mal byť workflow manažment systém vo forme webovej aplikácie, ktorý používa technológie Java EE, SQL a objektovo relačné mapovanie medzi databázou a aplikáciou JPA/Hibernate. Hlavnou úlohou je vytvoriť biznis logiku pre riadenie workflowu namodelovaného Petriho sieťou, efektívne prideľovanie rolí pre užívateľov systému a vykonávanie jednotlivých úloh.

Modelovanie Petriho sietí je realizované v programe PNeditor, ktorý ju ukladá vo formáte XML. Dátové modely sú vytvárane v Data modelery, ktorý ich takisto ukladá v XML formáte. Dátové formuláre a validácie v rámci nich budú vytvorené takisto externým programom, ktorého výstupom budú XHTML stránky, CSS a Javascript súbory zabalené v ZIP súbore. Všetky tieto súbory sa budú importovať do systému nahrávaním súborov cez webstránku na to určenú.

Takáto webaplikácia sa štandardne vytvára pomocou databázy a aplikácie, ktorá dokáže spracovávať údaje v nej. Existujú rôzne spôsoby realizácie takejto aplikácie, ja som sa rozhodol navrhnúť a implementovať ju pomocou 3-vrstvovej architektúry MVC, ktorá je najrozšírenejšia:

- Model – Dátová vrstva – predstavuje model dát uložených v databáze
- View – Prezentačná vrstva – zahŕňa všetky grafické rozhrania
- Controller – Aplikačná vrstva – obsahuje biznis logiku aplikácie

Táto architektúra zabezpečuje vzájomné oddelenie dát, grafického rozhrania a biznis logiky, čo umožňuje jednoduchšiu udržateľnosť, rozšíriteľnosť systému alebo zmenu niektorej vrstvy.

## 3.1 Architektúra

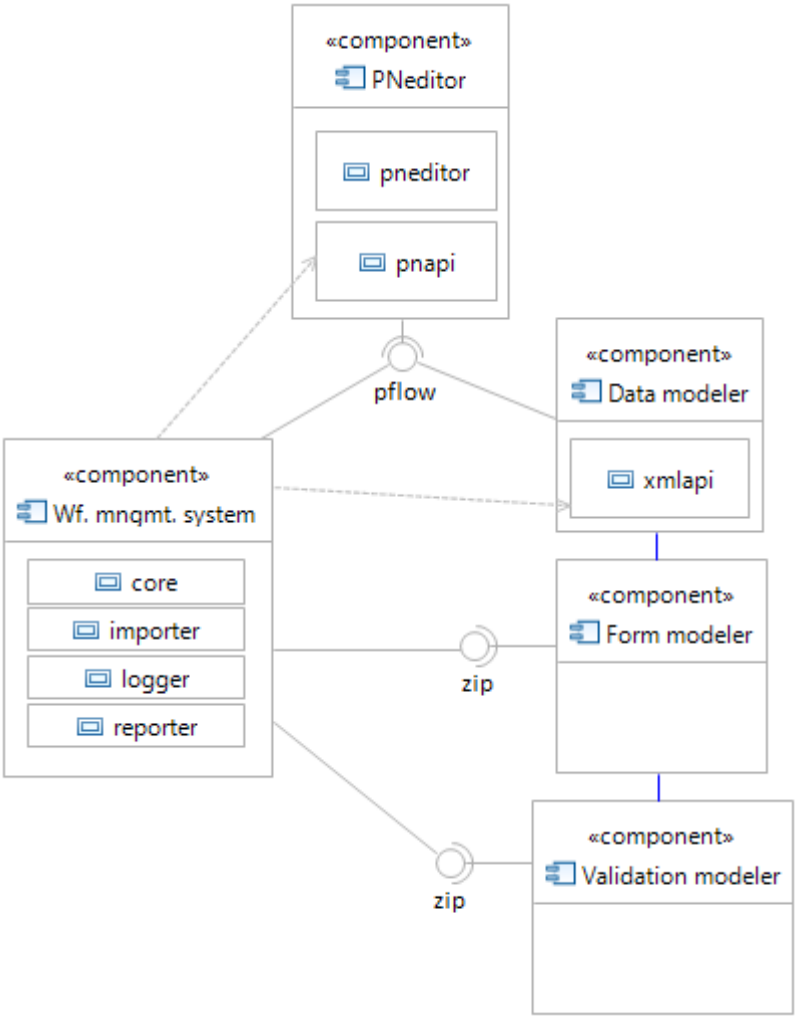
**Meta architektúra**

| Zásady | Mechanizmy | Kľúčové koncepty |
|---|---|---|
| Tvorba wf. | Petriho siete ako wf. | PNeditor |
| Manažment wf. | Integrácia cez importovanie | Dátový modeler |
| Inštancie wf. | Tvorba prípadov | Formulárový modeler |
| Aktuálny stav wf | Webaplikácia | Validačný modeler |
| Manažment užívateľov | Databáza | Wf. manažment systém |
| Manažment právomocí | HTML a CSS | |
| Dátové formuláre | Javascript validácie | |
| Validácie údajov | Zobrazenie siete cez AJAX | |
| | Práca s XML | |

*wf. - workflow

Podčiarknuté zásady, mechanizmy a komponenty patria do rozsahu tejto práce. Všetky ostatné sú alebo budú vytvorené inými vývojármi.

**Logická architektúra**




Obr.11 Logická architektúra

| Komponent | Vývojári | Stav |
|---|---|---|
| PNeditor 2 | Martin Riesz,<br>Ján Suchý a ďalší | Hotový |
| Data a Form modeler | Richard Koháry | Vo vývoji |
| Validation modeler | Peter Baranec | Vo vývoji |
| Workflow Management System | Tomáš Zuber | Vo vývoji |

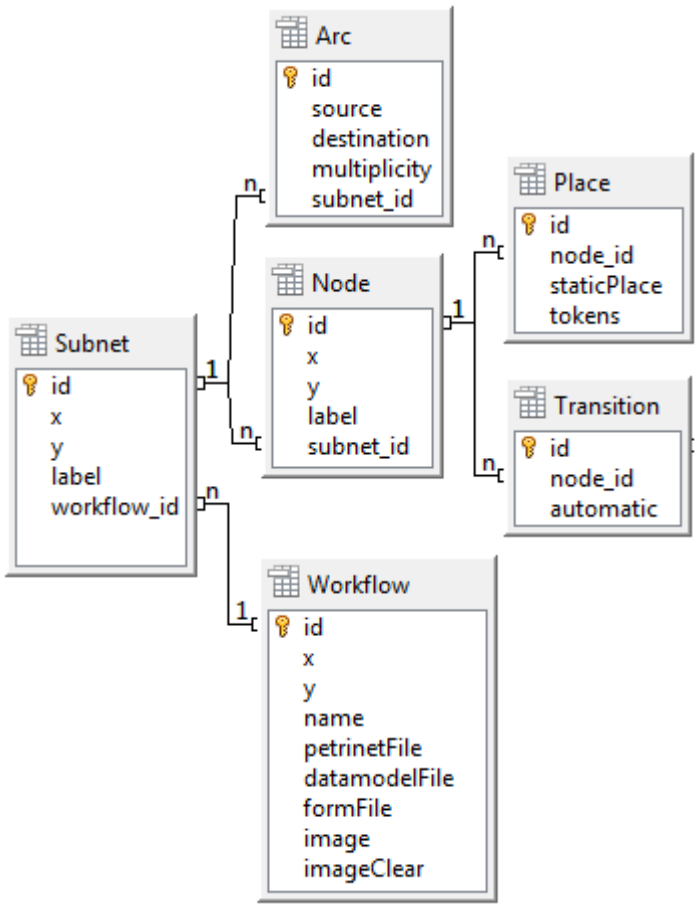
## 3.2 Návrh dátovej vrstvy

### 3.2.1 Data Transfer Objects

Dátová vrstva sa skladá z databázy MySQL a perzistenčnej vrstvy JPA/Hibernate. Tabuľky v databáze budú generované z dátových objektov perzistenčnou vrstvou. Základnou množinou objektov je reprezentácia dát Petriho siete:

- Miesta - Place
- Prechody - Transition
- Hrany - Arc

Tieto objekty zahŕňajú v sebe aj dodatočné dáta o ich rozložení v rámci siete a role priradené prechodom.

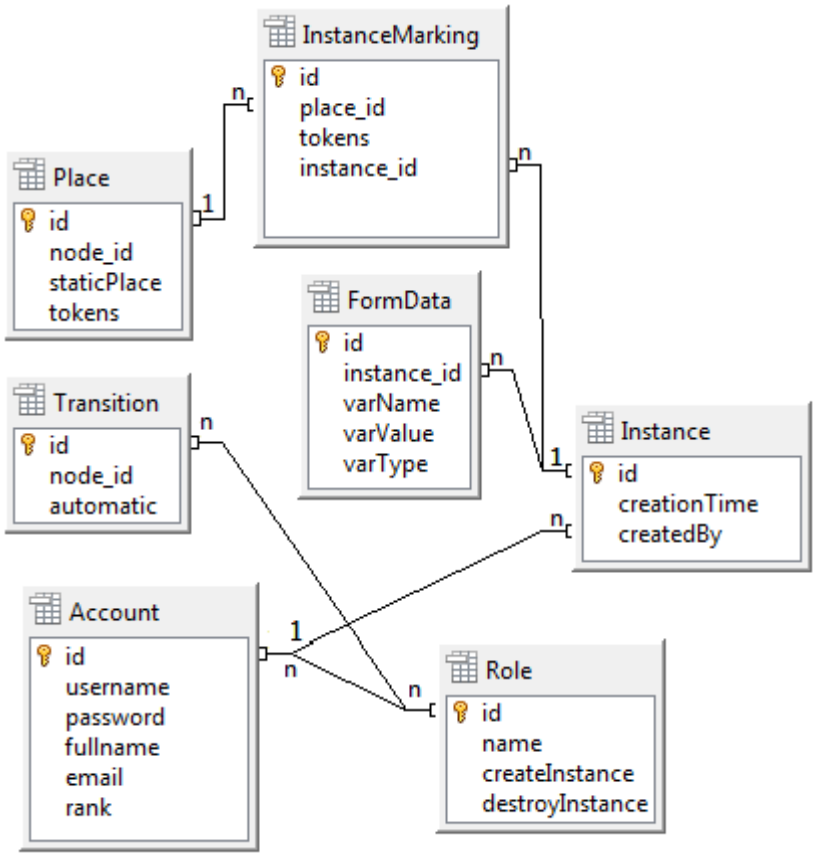



Obr.12 Štruktúra a vzťahy tabuliek 1.časť

Pre vytvorenie workflow manažment systému je potrebné priradiť Petriho sieť k workflowu, pridať tabuľky pre viaceré inštancie a inštančné značkovania pre vytváranie prípadov pre workflow. Ďalej je potrebné vytvoriť objekty pre uživateľov, ich priradenie

k jednotlivým rolám v rámci workflowov a nastavenie ich právomocí v rámci celého systému. Pri plnení úloh je ešte potrebné ukladať údaje zadané užívateľmi. Týmto dostaneme celý zoznam potrebných dátových objektov:

- Workflow
- Inštancia/Prípad – Instance
- Inštančné značkovanie – Instance Marking
- Rola – Role
- Právomoci – Rank
- Dáta z formulárov – Form Data




Obr.13 Štruktúra a vzťahy tabuliek 2.časť

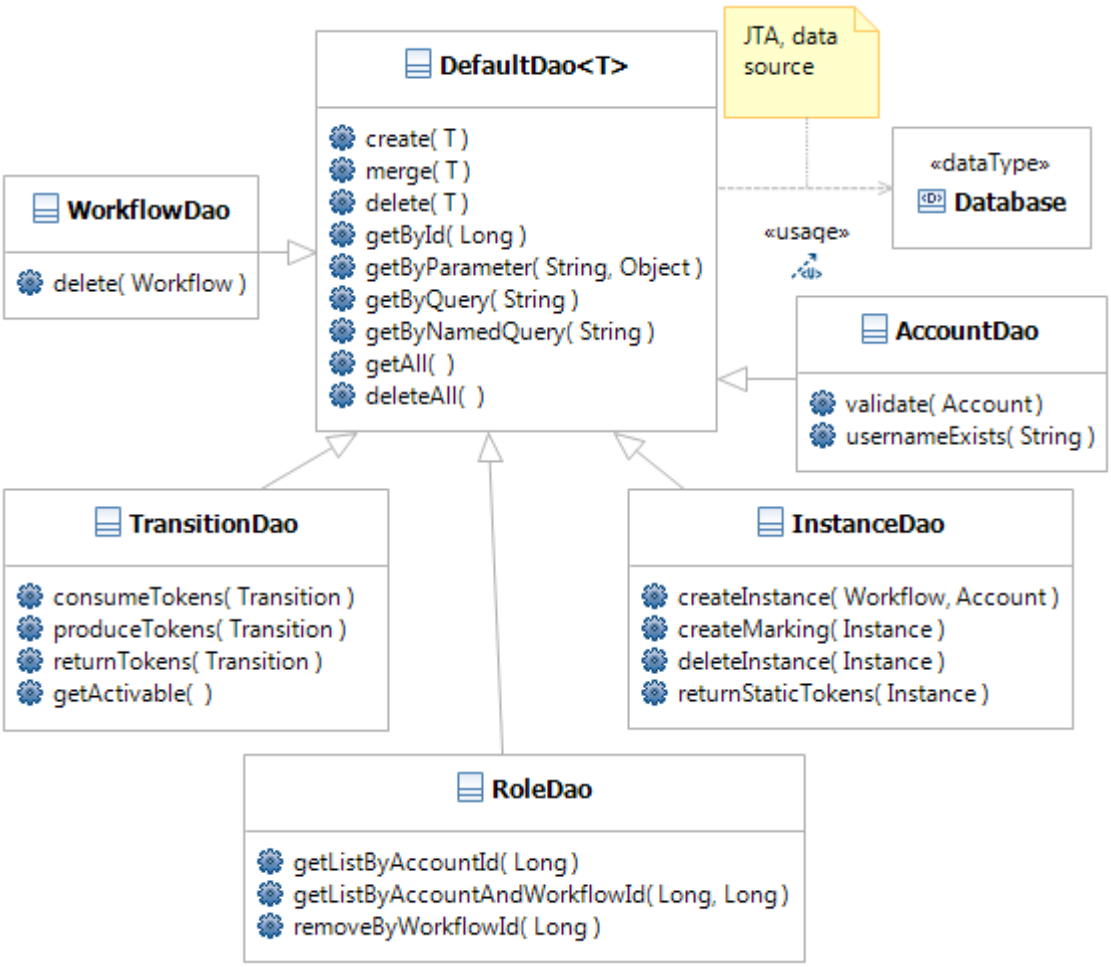
Je potrebné vytvoriť Java beany pre všetky modely a zabezpečiť mapovanie JPA anotáciami, podľa ktorých sa následne vygenerujú tabuľky. Relácie N ku N sú spracované perzistenčnou vrstvou do medzitabuľky.

### 3.2.2 Data Access Object

Pre prístup a prácu s dátami je potrebné vytvoriť DAO triedy, ktoré zvyšujú abstrakciu k prístupu dát, čo zjednodušuje prístup k údajom a skrýva operácie perzistenčnej vrstvy. Treba vytvoriť konfiguračný súbor hibernate.cfg.xml, v ktorom sa nastavia údaje potrebné na prepojenie databázy a aplikácie – adresa databázového servera, prihlasovacie údaje, ovládač k databáze (JDBC), meno databázy, mapované triedy a iné atribúty pre optimalizáciu výkonu.

Keďže dátové objekty majú vzájomné referencie a niektoré z nich majú silnú závislosť medzi sebou, tak je potrebné niektoré údaje načítavať a spracovávať spoločne. Z tohto hľadiska je potrebné vytvoriť viacero DAO tried, ktoré majú špecifické metódy na riešenie týchto problémov. Z druhého hľadiska všetky objekty zdielajú potrebu načíatania, ukladania a mazania dát. Tieto funkcionality budú implementované v jednej triede a ostatné triedy ich budú dediť.

**Diagram DAO tried:**




Obr.13 Návrh DAO tried

## 3.3 Návrh aplikačnej vrstvy

Aplikačná vrstva bude obsahovať Enterprise Java Beany (EJB), v ktorých bude implementovaná biznis logika importovania súborov, bezpečnostné prvky, manažovanie práv užívateľov, atď. Metódy EJB by mali byť prístupné len po úspešnej autorizácii užívateľa do systému. Systém umožní neregistrovaným užívateľom registrovať sa (chránené captchou proti robotom) a následne používať systém so základnými právomocami. Tieto právomoci môže meniť len užívateľ s administrátorskými právami. Pri prvom spustení systému sa automaticky vytvorí administrátorský účet s prihlasovacím menom admin a heslom admin. Toto heslo je možné kedykoľvek zmeniť v aplikácii v sekcii administrácie vlastného účtu.

Vykonávanie úloh vo workflowoch je povolené na základe priradených rolí užívateľovi. Tieto role môže priraďovať užívateľ s manažérskymi alebo administrátorskými právomocami. Po aktivácii úlohy sa užívateľovi zobrazí dátový formulár, kde môže zadávať potrebné údaje a poslať ich do systému, ktorý ich následne uloží do databázy.

Prvým krokom je vytvorenie súboru web.xml:

- <servlet> - nastavenie Faces servletu a cestu k jeho triede
- <servlet-mapping> - nastavenie mapovania (*.jsf – všetky súbory s koncovkou jsf) URL adresy, pre ktorých sa spracujú údaje Faces servlet-om
- <welcome-file-list> - mená stránok, z ktorých sa zobrazí prvá nájdená (index.jsp)
- <filter> - slúži na konfiguráciu ďalších filtrov, ako napr. Captcha
- <filter-mapping> - nastavenie mapovania ako pri servlet-mappingu (*.jsf)

Potrebné knižnice sa ukladajú do adresára WEB-INF\lib. Najdôležitejšie z nich sú:

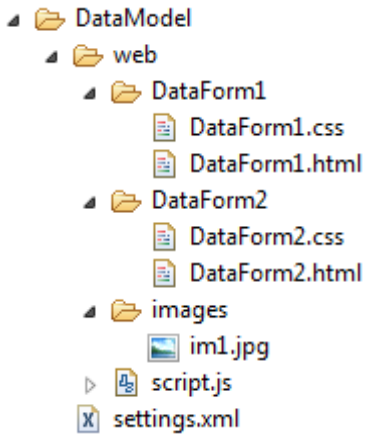
- hibernate – annotations, commons-annotations, core, hibernate3, ...
- commons – collections, lang, ...
- jsf – api,impl
- jstl – api, impl
- mysql-connector-java
- pnapi2
- pneditor2
- simple-xml-2.1.8
- simplecaptcha
- knižnice pre načíatanie dátového modelu

### 3.3.1 Importovací komponent

Na importovanie súborov slúži zvlášť stránka, kde si užívateľ vyberie súbory, ktoré chce nahrať do systému. Keďže súbory pochádzajú z rôznych aplikácií bolo potrebné pridať ku knižnicám pnapi2.jar a pneditor2.jar, ktoré slúžia na parsovanie údajov z pflow2 súboru a namapovanie na objekty použité v rámci aplikácie. Týmto spôsobom získame Petriho sieť, ktorá umožňuje pracovať s namodelovaným workflowom.
Ďalším nepovinným súborom je ZIP súbor, ktorý obsahuje dátové modely a dátové formuláre. Workflow je plne funkčý aj bez tohto súboru. Rozdiel bude len pri práci

s úlohami, kde pri aktivácii úlohy sa nezobrazia žiadne formuláre, ale len tokeny sa posunú v Petriho sieti. Pre zníženie počtu súborov je možné v budúcnosti pridať Petriho sieť do ZIP súboru a uploadovať iba jeden súbor do aplikácie namiesto dvoch.

Štrukúra ZIP súboru:




Obr.14 Všeobecná štruktúra ZIP súboru (fotka z Eclipse)

**DataModel** – je meno dátového modelu, ktorý je nastavený v dátovom modeleri, meno tohto adresára nie je nikde využité

**settings.xml** – je súbor s dátovým modelom, t.j. XML súbor, ktorý obsahuje všetky premenné s menom a typom použité v dátových formulároch a obsahuje naviazanie jednotlivých dátových formulárov na prechody v Petriho sieti

**web** – je adresár obsahujúci všetky dátové formuláre, obrázky a javascripty, môže obsahovať 1 až N dátových formulárov (DataForm1) a k nim prislúchajúce súbory (obrázky a javascripty)

**DataForm1** – je jeden dátový formulár, ktorý obsahuje HTML stránku, prípadne aj CSS súbor na nastavenie štýlov

Importovanie ZIP súboru má na starosti:

- rozbalenie súborov
- parsovanie settings.xml
  - vytvorenie objektov pomocou knižnice simplexml
  - uloženie premenných do databázy – ich meno, typ a prislúchajúci workflow
  - namapovanie formulárov na prechody
- prekopírovanie súborov z web adresára do adresára webaplikácie

/WebContent/forms/form<ID workflowu>

- v HTML súboroch sú upravené cesty v src a href tak, aby pri zobrazovaní formulárov sa pripájali k stránke všetky externé súbory, týmto spôsobom je možné ukladať obrázky a javascripty zvlášť k jednotlivým dátovým formulárom alebo vyššie o jednu úroveň, aby ich mohli prepoužiť viaceré formuláre

### Štruktúra settings.xml

```
<settings>
   <variables>
      <variable name="Premenna1" type="String"/>
      <variable name="Premenna2" type="Integer"/>
      <variable name="Premenna3" type="Date"/>
   </variables>
   <dataModels>
      <dataModel name="DatovyFormular1">
         <transition>prechod1</transition>
         <transition>prechod3</transition>
      </dataModel>
      <dataModel name="DatovyFormular2">
         <transition>prechod2</transition>
      </dataModel>
   </dataModels>
</settings>
```

- celý súbor má dve hlavné časti – zoznam premenných a zoznam dátových modelov
- element premennej (variable) má atribúty meno (name) na ich jednoznačné identifikovanie a typ (type) na nastavenie typu dát, ktoré premenná môže obsahovať
- mená premenných v rámci dátového modulu musia byť unikátne
- element dátového modelu (dataModel) má atribút meno (name), ktorý odkazuje na adresár s HTML webstránkou a CSS štýlom a obsahuje zoznam elementov s identifikátorom prechodu (transition)
- jeden formulár môže byť priradený k viacerým prechodom, ale každý prechod môže mať k sebe priradený iba jeden formulár

### Štruktúra HTML formulárov

```
<html><head>
<link rel="stylesheet" type="text/css" href="DatovyFormular.css"></link>
<meta http-equiv="content-type" content="text/html;"
      charset="utf-8"></meta>
<title>DatovyFormular</title></head>
<body><form action="${postaction}" method="post">
      <input type="hidden" name="transitionId" value="${TRANSITION_ID}"/>
      <div id="attr4">
            <div class="label"></div>
            <input type="text" name="Premenna1" value="${Premenna1}"
```

```
disabled="disabled"/>
      </div>
      <div id="attr5">
            <div class="label"></div>
            <input type="text" name="Premenna2" value="${Premenna2}"/>
      </div>
      <div id="attr8">
            <div class="label"></div>
            <input type="submit" name="submit"/>
      </div>
</form></body></html>
```

- v hlavičke HTML súboru sa definuje linka na CSS súbor, ktorý obsahuje rozmiestnenie prvkov na stránke a názov formulára
- telo HTML stránky obsahuje:
  - element *form* má v *action* nastavené *${postaction}*, ktoré sa vždy nahrádza pri zobrazení formulára URL adresou servletu, ktorý spracúva údaje poslané cez formulár
  - skrytý *input* element obsahuje identifikátor prechodu, ktorý bol spustený, *${TRANSITION_ID}* je nahradené aktuálnym prechodom
  - vstupné údaje sa zadávajú do *input*-ov s menom z dátového modelu a ako hodnota sa uvádza vo formáte ${meno premennej}, tieto hodnoty sa pred zobrazením nahradia s hodnotami uloženými v databáze
  - formulár sa posiela tlačítkom *submit*

## 3.4 Návrh prezentačnej vrstvy

Prezentačnú vrstvu tvorí Java Server Faces 2.0 (JSF), implementácia Mojarra 2.0.2. Tento framework umožňuje využitie faceletov, ktorých použitie úplne oddeluje kód aplikácie od webstránok XHTML. JSF používa vlastné tagy, ktoré sa transformujú na XHTML tagy. Toto zabezpečuje použiteľnosť webových prehliadačov a nie je potrebné vytvárať klientskú aplikáciu. JSF umožňuje vytvárať aj vlastné komponenty a priamo má implementovanú podporu pre AJAX (iba verzia 2.0).

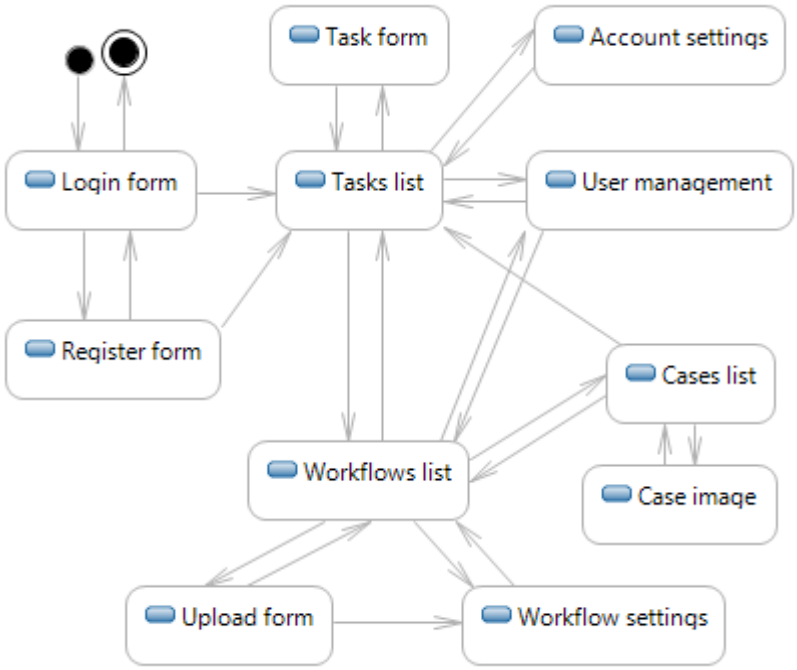
JSF 2.0 umožňuje vytvoriť manažované Java Beany pomocou anotácií (@ManagedBean) a podporuje navigáciu medzi stránkami pomocou metód Java Beanu, ktorej návratová hodnota je String. Týmto sa odstráni potreba konfigurácií v XML súbore faces-config.xml. Pri navigácii na hlavnú stránku http://host:8080/wfms/ sa načíta stránka index.jsp, ktorej úlohou je presmerovanie na stránku login.jsf. Tento krok je potrebný na inicializáciu a spustenie JSF.

Všetky stránky sú vytvorené ako obyčajné XHTML webstránky podľa štandardov W3C doplnené o Facelety, ktoré zaručujú dynamický obsah stránky. Na začiatku každej stránky sú definované namespace-y v tagu html:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
```

Tagy z JSF HTML a CORE sa transformujú na štandardné HTML tagy, čo umožňuje použitie kaskádových štýlov CSS, ktoré oddelujú vzhľad od štruktúry stránky a týmto ďalej zvyšujú celkovú prehľadnosť projektu.

**Mapa stránok:**




Obr.15 Navigácia medzi stránkami

Pre lepšiu prehľadnosť boli vynechané šípky so všetkých stránok smerujúce k Login form, ktoré by zobrazovali obyčajné odhlásenie užívateľa zo systému.

### 3.4.1 Zobrazenie prípadov

Zobrazenie aktuálneho stavu Petriho siete pre jednotlivé prípady je realizované pomocou javascriptu a AJAXových volaní. Obrázok Petriho siete je uložený v databáze bez tokenov a ten je zobrazený. Tokeny sú generované, ich súradnice sa dynamicky vypočítavajú

zo súradníc z PNeditora a tak sa zobrazujú na správnom mieste podľa aktuálneho stavu prípadu. Tieto súradnice sú periodicky posielané každé 2 sekundy a tým sa zabezpečí zobrazenie aktuálneho stavu bez obnovenia celej stránky.

# 4 Implementácia aplikácie

Zdrojové kódy v Jave sú umiestnené v adresári *src*, UML diagramy v adresári *model* a JUnit testy v adresári *test*. Jednoduché JUnit testy boli vytvorené len pre určité časti komponentov, aby bola otestovaná ich funkcionalita správne a rýchlo. Slúžia hlavne na otestovanie základných funkcionalít a prácu perzistenčnej vrstvy s databázou. UML diagramy boli vytvorené s pluginom Eclipse UML2 tools. Slúžia hlavne kvôli lepšiemu pochopeniu systému – navigácia medzi stránkami, práva používania systému na základe rolí, základné objekty pre prístup k dátam a objekty prenášajúce dáta.

Konfiguračné súbory, webstránky, CSS štýly, javascripty, obrázky, komponenty, šablóny, properties a externé knižnice sú uložené v adresári WebContent. Nastavenia webaplikácie sú uložené vo *WEB-INF/web.xml*, jazykové nastavenia pre JSF sú vo *WEB-INF/faces-config.xml*, role pre prístup k stránkam v *WEB-INF/sun-web.xml* a konfiguračný súbor vo *WEB-INF/classes/META-INF/persistence.xml*.

**Konfigurácia web.xml**

```
<welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

- welcome-file-list slúži na nastavenie začiatočnej stránky projektu, na ktorú je nasmerovaný, keď do prehliadača zadá základnú URL adresu (napr. localhost:8080/wfms)
- stránka *index.jsp* slúži len ako smerovač na prihlasovaciu stránku *login.xhtml* kvôli správnej inicializácii JSF servletu

```
<servlet>
      <servlet-name>Faces Servlet</servlet-name>
      <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
</servlet>
```

- Faces Servlet slúži ako kontajner, ktorý spracúvava, filtruje a validuje požiadavky klienta
- parsuje XHTML stránky a vytvára JSF stránky s dynamickými údajmi
- load-on-startup inicializuje Faces Servlet hneď pri spustení aplikácie a nečaká na prvé spustenie JSF stránky

```
<servlet-mapping>
      <servlet-name>Faces Servlet</servlet-name>
      <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
```

- servlet-mapping slúži na mapovanie servletov na URL adresy pomocou vzorov
- v tomto prípade sú namapované všetky stránky s URL adresou končiacou s *jsf*

```
<context-param>
      <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
      <param-value>.xhtml</param-value>
</context-param>
<context-param>
      <param-name>username</param-name>
      <param-value>admin</param-value>
</context-param>
<context-param>
      <param-name>password</param-name>
      <param-value>admin</param-value>
</context-param>
```

- kontextové parametre sa načítavajú aplikáciou pri spustení
- slúžia na konfiguráciu aplikácie a framework, ktoré aplikácia používa
- javax.faces.DEFAULT_SUFFIX nastavuje, ktoré stránky sa majú prekladať na JSF stránky
- parametre username a password sú použité pri prvom spustení aplikácie, keď ešte nie sú vytvorené tabuľky v databáze a po ich vytvorení musí existovať nejaký administrátor, ktorý môže nastavovať právomoci iných užívateľov

```
<error-page>
      <exception-type>
            javax.faces.application.ViewExpiredException
      </exception-type>
      <location>/login.jsf</location>
</error-page>
```

- nastavenie error-page slúži na presmerovanie užívateľa pri chybe na zvolenú stránku
- tu je konkrétne nastavené presmerovanie na prihlasovaciu stránku v prípade, že užívateľovi expiruje session

```
<listener>
      <listener-class>
            com.sun.faces.config.ConfigureListener
      </listener-class>
</listener>
<listener>
      <listener-class>logic.util.DatabaseInitializer</listener-class>
</listener>
```

- listenery sú spúšťané pri udalostiach

- com.sun.faces.config.ConfigureListener je potrebné nastaviť pre funkčnosť JSF
- logic.util.DatabaseInitializer uloží administrátora do databázy

```
<filter>
      <filter-name>LoginFilter</filter-name>
      <filter-class>presentation.filter.LoginFilter</filter-class>
</filter>
```

- LoginFilter kontroluje, či je užívateľ prihlásený

```
<filter-mapping>
      <filter-name>LoginFilter</filter-name>
      <url-pattern>/account-settings.jsf</url-pattern>
      <url-pattern>/case-image.jsf</url-pattern>
      <url-pattern>/cases.jsf</url-pattern>
      <url-pattern>/tasks.jsf</url-pattern>
      <url-pattern>/upload.jsf</url-pattern>
      <url-pattern>/user-management.jsf</url-pattern>
      <url-pattern>/workflows.jsf</url-pattern>
      <url-pattern>/workflow-settings.jsf</url-pattern>
      <dispatcher>REQUEST</dispatcher>
      <dispatcher>INCLUDE</dispatcher>
      <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

- mapovanie LoginFilter-a na potrebné stránky
- prihlasovacia a registračná stránka nevyžadujú prihláseného užívateľa

```
<servlet>
      <servlet-name>SimpleCaptcha</servlet-name>
      <servlet-class>
            nl.captcha.servlet.SimpleCaptchaServlet
      </servlet-class>
</servlet>
<servlet-mapping>
      <servlet-name>SimpleCaptcha</servlet-name>
      <url-pattern>/simpleCaptcha.png</url-pattern>
</servlet-mapping>
```

- SimpleCaptcha je použitá ako ochrana pred robotmi na internete
- užívateľia sú povinný zadať kód z captchy pri registrácii

```
<security-role>
      <role-name>USER</role-name>
</security-role>
<security-role>
      <role-name>MANAGER</role-name>
</security-role>
<security-role>
      <role-name>ADMIN</role-name>
</security-role>
```

- tri rôzne role pre efektívne nastavenie právomocí

```
<security-constraint>
      <display-name>UserConstraint</display-name>
      <web-resource-collection>
            <web-resource-name>protected</web-resource-name>
            <description />
            <url-pattern>/account-settings.jsf</url-pattern>
            <url-pattern>/tasks.jsf</url-pattern>
            <http-method>GET</http-method>
            ...
      </web-resource-collection>
      <auth-constraint>
            <description />
            <role-name>ADMIN</role-name>
            <role-name>MANAGER</role-name>
            <role-name>USER</role-name>
      </auth-constraint>
</security-constraint>
```

- security-constraint je ďalší bezpečnostný prvok, ktorým mapujeme role systému k jednotlivým stránkam
- rovnakým spôsobom sú namapované aj ostatné stránky pre manažérov a adminov

```
<login-config>
      <auth-method>FORM</auth-method>
      <realm-name>wfms</realm-name>
      <form-login-config>
            <form-login-page>/login.jsf</form-login-page>
            <form-error-page>/login.jsf</form-error-page>
      </form-login-config>
</login-config>
```

- login-config mapuje prihlasovací formulár na príslušnú stránku

**Konfigurácia faces-config.xml**

```
<application>
      <message-bundle>messages</message-bundle>
      <locale-config>
            <default-locale>en</default-locale>
            <supported-locale>en</supported-locale>
            <supported-locale>sk</supported-locale>
      </locale-config>
      <resource-bundle>
            <base-name>messages</base-name>
            <var>msgs</var>
      </resource-bundle>
</application>
```

- v JSF 2 už nie je potrebné definovať manažované beany v tomto súbore, keďže je možné použiť anotácie, ktoré umožňujú rovnaké možnosti konfigurácie
- message-bundle, locale-config a resource-bundle definujú mená súborov, ktoré obsahujú jazykové mutácie všetkých textov v aplikácii

**Konfigurácia sun-web.xml**

```
<context-root>/wfms</context-root>
<security-role-mapping>
      <role-name>USER</role-name>
      <group-name>USER</group-name>
</security-role-mapping>
<security-role-mapping>
      <role-name>MANAGER</role-name>
      <group-name>MANAGER</group-name>
</security-role-mapping>
<security-role-mapping>
      <role-name>ADMIN</role-name>
      <group-name>ADMIN</group-name>
</security-role-mapping>
```

- pre správnu funkčnosť bezpečnosti na základe rolí je potrebné namapovať jednotlivé role ku skupinám (v tomto prípade sú mená rolí a skupín rovnaké)

**Konfigurácia persistence.xml**

```
<persistence-unit name="wfmsPU" transaction-type="JTA">
<provider>org.hibernate.ejb.HibernatePersistence</provider>
<jta-data-source>jdbc/wfmsDS</jta-data-source>
```

- wfmsPU je meno perzistenčnej jednotky, na ktorú sa odvolávame v Java kódoch
- transaction-type môže mať hodnoty *RESOURCE_LOCAL* (bez transakcií) a *JTA* ( Java Transaction API – s transakciami)
- typ JTA je použitý na zachovanie integrity údajov uložených v databáze, lebo systém môžu používať naraz viacerí užívateľia a manipulovať s rovnakými hodnotami
- provider je poskytovateľom perzistencie – bol použitý Hibernate, ale s menšími úpravami by mal fungovať aj EclipseLink
- jta-data-source je meno dátového zdroja, ktorý je vytvorený v databáze

```
<class>persistence.model.Account</class>
<class>persistence.model.Arc</class>
<class>persistence.model.FormData</class>
<class>persistence.model.FormDataValue</class>
<class>persistence.model.Instance</class>
<class>persistence.model.InstanceMarking</class>
<class>persistence.model.Place</class>
<class>persistence.model.Role</class>
<class>persistence.model.Subnet</class>
<class>persistence.model.Transition</class>
<class>persistence.model.Workflow</class>
```

- nadefinovaním tried v elementoch *class* vie perzistenčný framework, ktoré triedy sa budú mapovať na databázové tabuľky a podľa anotácií nastaví potrebné parametre

```
<properties>
      <property name="hibernate.dialect"
            value="persistence.util.CustomMysqlDialect"/>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
      <property name="hibernate.max_fetch_depth" value="5"/>
</properties>
```

- persistence.util.CustomMysqlDialect slúži na nastavenie parametrov špecifických pre databázový server
- hibernate.hbm2ddl.auto automaticky vytvorí alebo aktualizuje zmeny v tabuľkách
- hibernate.max_fetch_depth je použitý na nastavenie maximálnej hĺbky SQL joinov

## 4.1 Perzistenčná vrstva

Perzistenčná vrstva sa skladá z dvoch hlavných častí (objekty na prístup k dátam - DAO a objekty prenášajúce dáta - DTO) a pomocných objektov (objekty s konštantami a nastaveniami). Trieda *persistence.QueryConstants* obsahuje všetky konštanty použité v rámci perzistenčnej vrstvy – mená parametrov, mená pomenovaných dotazov a samotné dotazy.

Všetky DTO triedy sa nachádzaju v Java balíčku *persistence.model,* DAO triedy *persistence.dao* a pomocné triedy v *persistence* a *persistence.util*.

### 4.1.1 Objekty prenášajúce dáta

```
@Entity
@NamedQueries ({
      @NamedQuery(name = QueryConstants.VALIDATE_ACCOUNT,
                  query = QueryConstants.VALIDATE_ACCOUNT_QUERY),
      @NamedQuery(name = QueryConstants.CHECK_USERNAME,
                  query = QueryConstants.CHECK_USERNAME_QUERY),
      @NamedQuery(name = QueryConstants.ACCOUNTS_BY_ROLE_ID ,
                  query = QueryConstants.ACCOUNTS_BY_ROLE_ID_QUERY)
})
public class Account extends Name implements Serializable {
      private static final long serialVersionUID = 1769955608L;

      @Column(nullable = false, length = Constants.USERNAME_LENGTH,
unique = true)
      private String username;
      @Column(nullable = false, columnDefinition = "CHAR(64)")
      private String password;
      @Column(length = Constants.EMAIL_LENGTH)
      private String email;
      @Column(nullable = false, length = Constants.RANK_LENGTH)
      @Enumerated(EnumType.STRING)
      private Rank rank = Rank.USER;
```

```
@ManyToMany(cascade = { CascadeType.MERGE }, mappedBy = "accounts",
            fetch = FetchType.LAZY)
private List<Role> roles = new ArrayList<Role>();...
```

- každý model musí mať anotáciu *@javax.persistence.Entity*, ktorej úlohou je povedať perzistenčnému frameworku, aby mapoval triedu na tabuľku v databáze
- anotácie *@javax.persistence.NamedQueries* a *@javax.persistence.NamedQuery* slúžia na definovanie SQL dotazov, ktoré sa ukladajú do cache pamäte a tým zabezpečujú rýchlejšie odozvy ako dynamicky vyskladané dotazy
  - *name* a *query* u obyčajné reťazce znakov, ale kvôli prehladnosti a jednoduchšej zmene bola zvlášť vytvorená trieda *persistence.QueryConstants* s konštantami
- interface *java.io.Serializable* je implentovaný len kvôli perzistencii, pri jeho implementácii nie je potrebné pridať žiadne metódy, ale stačí pridať *serialVersionUID*
- anotácia *@javax.persistence.Column* umožňuje podrobnejšie nastavenia pre jednotlivé polia
  - *nullable* – pole môže alebo nemôže byť null v tabuľke
  - *length* – definovanie dĺžky reťazca v tabuľke
  - *unique* – pole musí alebo nemusí byť unikátne v stĺpci
  - *columnDefinition* – je použitý na presné nastavenie dátového typu v databáze
- anotácia *@javax.persistence.Enumerated* ukladá v databáze údaje ako enumeráciu namiesto obyčajných stringov
- anotácia *@javax.persistence.ManyToMany* slúži na mapovanie dvoch tried so vzťahom M ku N – jedno uživateľské konto môže byť zaradené do viacerých rolí a jedna rola môže byť priradená k viacerým užívateľom
  - automaticky je vygenerovaná tabuľka, ktorá obsahuje dva stĺpce s primárnymi kľúčmi jednotlivých tried
  - *cascade* – kaskádovo ukladá, aktualizuje alebo zmaže údaje v tabuľke podľa zoznamu objektov – *CascadeType.MERGE* aktualizuje údaje pri zmene údajov v zoznem rolí
  - *mappedBy* – prepojí polia tried anotované s *@ManyToMany*
  - *fetch* – môže byť *FetchType.LAZY* (zoznam rolí sa doťahuje až keď je volaný v kóde, ale toto volanie musí byť v rámci tej istej perzistenčnej session) alebo *FetchType.EAGER* (naplní zoznam rolí okamžite) – v tomto prípade je použitý *LAZY* loading, lebo nie vždy potrebujeme role užívateľa – zmenšíme množstvo

ťahaných dát z databázy

```
@Entity
@NamedQueries(value = {
            @NamedQuery(name = QueryConstants.WORKFLOW_BY_INSTANCE_ID,
                        query =
QueryConstants.WORKFLOW_BY_INSTANCE_ID_QUERY),
            @NamedQuery(name = QueryConstants.WORKFLOW_BY_TRANSITION_ID,
                        query =
QueryConstants.WORKFLOW_BY_TRANSITION_ID_QUERY)
})
public class Workflow extends Position implements Serializable {
      private static final long serialVersionUID = 13498993L;

      @Lob
      @Basic(fetch = FetchType.LAZY)
      private byte[] image;
      @Lob
      @Basic(fetch = FetchType.LAZY)
      private byte[] emptyImage;

      @OneToMany(mappedBy = "workflow", cascade = { CascadeType.REMOVE },
            fetch = FetchType.LAZY)
      private List<Subnet> subnets = new ArrayList<Subnet>();
```

- anotácia *@javax.persistence.Lob* a anotácia *@javax.persistence.Basic* predstavujú binárne dáta v databáze – tu sú použité na uloženie obrázku Petriho siete so začiatočným značkovaním a bez značkovania
- anotácia *@javax.persistence.OneToMany* automaticky pridáva pole na prepojenie tabuľiek so vzťahom 1 ku N – jeden workflow môže mať mnoho podsietí

Všetky modely sú vytvorené podobným spôsobom ako *Workflow* a *Account*. Obsahujú údaje, ktoré sú uvedené v tabuľkách v sekcii Data Tranfer Objects. Menšie odlišnosti sú pri triedach *Name, Position* a *Identity*, ktoré sú vytvorené kvôli prepoužitiu rovnakých častí v triedach. Trieda *Identity* obsahuje len jedno pole *id*, ktoré slúži stále ako primárny kľúč v tabuľkách. *Name* rozširuje triedu *Identity* a obsahuje pole *name* na pomenovanie objektov. *Position* rozširuje triedu *Name* a obsahuje x-ovú a y-ovú súradnicu objektu v Petriho sieti.

### 4.1.2 Objekty na prístup k dátam

Triedy týchto objektov sa nachádzajú v Java balíčku *persistence.dao*. Všetky tieto triedy dedia od abstraktnej triedy *DefaultDao*, ktorá zahŕňa všeobecné metódy na prácu s údajmi v databáze.

**DefaultDao**

```
public abstract class DefaultDao<T> implements Serializable {
	private static final long serialVersionUID = 2591597143570645281L;
	private static final Logger LOGGER =
		Logger.getLogger(DefaultDao.class);

	@PersistenceContext(unitName = Constants.PERSISTENCE_UNIT)
	protected EntityManager em;
	private Class<T> clazz;
	private final String SELECT_A_FROM_TABLE;

	public DefaultDao(Class<T> clazz) {
		this.clazz = clazz;
		SELECT_A_FROM_TABLE = "SELECT a FROM " + getClassName()
			+ " AS a ";
	}

	private String getClassName() {
		return clazz.getSimpleName();
	}
...
```

- všetky triedy rozširujúce túto triedu musia volať konštruktor s parametrom, v ktorom je nastavená trieda entity, s ktorou DAO trieda pracuje
- *LOGGER* – slúži na logovanie chýb, upozornení a debugovacích správ
- anotácia *@javax.persistence.PersistenceContext* pridáva triede kontext pre použitie perzistencie a musí mať zadané meno *unitName* – toto meno je definované v konfiguračnom súbore *persistence.xml* v elemente *persistence-unit* v atribúte *name*
- *javax.persistence.EntityManager* je trieda, ktorá manažuje úkony robené v perzistenčnej vrstve, jej inštancia sa nevytvára cez operátor *new*, ale je vsunutá cez dependency injection perzistenčným frameworkom na základe anotácie *@javax.persistence.PersistenceContext*
- *Class<T> clazz* je použitá pri vyhľadávaní cez metódu *find(Class, Object)* a pre vytvorenie dotazu *SELECT_A_FROM_TABLE* pre výber všetkých hodnôt

```
public List<T> findByParameter(String column, Object value){...
public List<T> findBy2Parameters(String column1, Object value1,
		String column2, Object value2) { ...
public T findById(Long id) { …
```

- metódy findBy sa vyskladávajú z obyčajných JPQL dotazov, v ktorých sa údaje nastavujú cez parametre
- podľa poradia uvedenia metód – hľadať na základe jedného parametra, dvoch

parametrov a na základe primárneho kľúča

```
public List<T> getAll() { ...
public Query getNamedQuery(String name) { ...
public List<T> findByQuery(String jpql) { ...
public void executeQuery(String jpql) { ...
public void removeAll() { ...
```

- podľa poradia uvedenia metód – výber všetkých hodnôt v tabuľke, vrátenie pomenovaného dotazu (uvedené vždy pri modeloch), vyhľadávanie podla JPQL dotazu, vykonanie JPQL dotazu, vymazanie všetkých hodnôt v tabuľke

```
public T persist(T t) { ...
public T merge(T t) { ...
public void remove(T t) { ...
public void detach(T t) { ...
```

- tieto metódy priamo volajú metódy s rovnakým menom z *EntityManager*-a
- podľa poradia uvedenia metód – uloženie nového záznamu, aktualizácia záznamu v databáze, vymazanie záznamu, oddelenie záznamu od objektu (prerušenie automatickej aktualizácie údajov v rámci session)

Všetky DAO triedy sú bezstavové Enterprise Java Bean-y, vďaka čomu nie je potrebné explicitne manažovať transakcie, lebo sú manažované autmaticky. Konkrétne implementácie DAO tried obsahujú metódy na hľadanie údajov podľa pomenovaných dotazov a špeciálne metódy na vytváranie a mazanie:

- *WorkflowDao*
  - *remove(Workflow workflow)* – vymaže najprv inštancie, role a formuláre prepojené s týmto workflowom
  - *removeAll()* - vyberie všetky workflowy a vymaže ich pomocou predchádzajúcej metódy
- *RoleDao* – uvedené metódy slúžia pridávanie a mazanie údajov zložitejším spôsobom, lebo medzi rolami, prechodmi a užívateľmi sú vzťahy M ku N
  - *addAccount(Role role, Account account)*
  - *addAccount(Collection<Role> roles, Account account)*
  - *addAccounts(Role role, Collection<Account> accounts)*
  - *removeAccount(Role role, Account account)*
  - *removeAllAccounts(Role role)*
  - *removeAllTransitions(Role role)*

- *InstanceDao*
  - *createInstance(Long workflowId, Long userId)* – vytvorenie inštancie, značkovaní a formulárových dát
  - *returnStaticTokens(Long instanceId)* – vrátenie statických tokenov do statických miest pri mazaní
- *AccountDao*
  - *usernameExists(String username)* – kontrola, či už užívateľské meno existuje
  - *validate(Account account)* – validácia užívateľského mena a hesla
- *TransitionDao*
  - *isTaskActivable(Transition transition, Long instanceId)* – zistí, či je prechod spustiteľný v danej inštancii
  - *getActivableTasks(Long accountId, Long instanceId)* – vráti aktivovateľné prechody pre danú inštanciu a užívateľa
  - *activateTask(Long transitionId, Long instanceId)* – spustí prechod v danej inštancii

## 4.2 Aplikačná vrstva

**logic.importer.Pflow2IMporter**

Obsahuje len jednu verejnú metódu *save(byte[][] files, String path)*, ktorá sa stará o parsovanie a uloženie nahratých súborov. Jej v stupom je pole nahratých súborov a cesta, kam sa súbory budú ukladať. Trieda ďalej obsahuje niekoľko privátnych metód na ukladanie a transformáciu jednotlivých častí Petriho sietí do systému. Hrany sú premapované na pomocou primárnych kľúčov prechodov a miest, role sú priradené k prechodom a mená formulárov sú priradené k prechodom pomocou konfiguračného súbora.

**logic.importer.ZipManager**

Úlohou tejto triedy je rozbaliť všetky súbory do požadovanej adresárovej štruktúry a doplniť cesty (href, src) v súboroch tak, aby smerovali na miesto kde sa súbory uložia. Celý tento proces vykoná metóda *unzip(InputStream input, String path)*.

**logic.security.Hash**

Pomocná trieda Hash slúži na vypočítanie hashu z textu na základe zvoleného algoritmu (MD5, SHA-1, SHA-256,...). Statická metóda *getHash(String text, String algorithm)* vráti hash kód textu podľa zvoleného algoritmu.

**logic.singleton.CacheBean**

V tejto triede sú uložené údaje, ktoré sa načítajú pri spustení aplikácie, menia sa zriedkavo a často sú volané. Slúžia ako medzipamäť medzi aplikáciou a databázou. Pridávanie, mazanie a čítanie týchto údajov je synchronizované pomocou zámky *javax.ejb.Lock*. Uchováva sa v nej zoznam všetkých užívateľov, mená hodností podľa vybraného jazyka, údaje pre dropdown boxy s užívateľmi a jazykmi.

**logic.singleton.LoginManagerBean**

Úlohou tejto triedy je udržiavať zoznam aktuálne prihlásených užívateľov v dátovej štruktúre *Set.*

**logic.util**

- *DatabaseInitializer* – pridanie administrátora pri prvom spustení aplikácie
- *PictureUtil* – transformuje pole bytov na *BufferedImage* a naopak

## 4.3 Prezentačná vrstva

Prezentačná vrstva sa skladá z 2 častí:

- Java triedy – manažované beany, ktoré uchovávajú, validujú a spracúvajú dáta, servlety, validátory a filtre
- web komponenty – webstránky, štýly, javascripty a šablóny

Všetky web stránky používajú šablónu *WebContent/resources/templates/layout.xhtml*, ktorá rozďeluje stránku na hlavičku, navigačnú časť, obsah stránky a pätičku pomocou CSS štýlov uložených v súbore *WebContent/resources/css/layout.css*.

Kvôli prehľadnosti a prepoužiteľnosti boli vytvorené tri vlastné komponenty v adresári *WebContent/resources/components*:

- accountForm.xhtml – pridávanie alebo zmena užívateľského mena, hesla,

potvrdenie hesla, celé meno a email s validáciami povolených dĺžok a použiteľných znakov

- inputSecretValid.xhtml – vstup pre heslo, ktoré povoluje veľké a malé písmená, čísla a niektoré špeciálne znaky
- inputTextValid.xhtml – vstup pre užívateľské meno, povolené znaky sú veľké a malé písmená, pomlčka a podtržník

Ku každej web stránke (v adresári *WebContent*) je priradená manažovaná beana (v balíčku *presentation.bean*) alebo servlet (v balíčku *presentation.servlet*), ktorá spravuje údaje v rámci nej:

- *login.xhtml* a *LoginBean.java* – prihlásenie uživateľa do systému užívateľským menom a heslom, navigácia na registračnú stránku, zmena jazyka
- *register.xhtml* a *AccountBean.java* – registrácia nového užívateľa – unikátne užívateľské meno v rámci systému, heslo, potvrdenie hesla, meno, e-mailová adresa a captcha kód
- *account-settings.xhtml* – zmena údajov užívateľa (okrem užívateľského mena)
- *tasks.xhtml* a *TaskBean.java* – tabuľka aktivovateľných prechodov v rámci prípadu pre daného užívateľa s možnosťou aktivácie
- *workflows.xhtml* a *WorkflowBean.java* – tabuľka všetkých workflowov s možnosťou zobraziť prípady, zmeniť alebo vymazať workflow
- *upload.xhtml* a *FileUploadServlet* – nahrávanie nových workflowov s dátovými modelmi a formulármi
- *cases.xhtml* a *InstanceBean.java* – tabuľka prípadov pre daný workflowov, pridávanie nových prípadov s unikátnym menom (v prípade prázdneho políčka sa vygeneruje meno), možnosť zmazať alebo zobraziť aktuálny stav prípadu
- *case-image.xhtml* a *ImageServlet.java* – zobrazuje obrázok Petriho siete bez značkovania a dynamicky cez Javascript a AJAX doťahuje zo servera aktuálny stav siete a vypočítaním polôh miest vykresluje tokeny
- *workflow-settings.xhtml* a *WfSettingsBean.java* – zmena mena, priradenie rolí k užívateľom, zmena práv na vytváranie a mazanie prípadov
- *user-management.xhtml* a *UserManagementBean.java* – tabuľka užívateľov s možnosťou zmeny role v rámci systému alebo zmazanie užívateľa

# 5 Záver

Navrhnutý a implementovaný workflow manažment systém umožňuje efektívne a bezpečné vytváranie rôznych webaplikácií, od najjednoduchších až po rozsiahle workflowy. Využitie systému je obmedzené len vlastnosťami Petriho sietí a kreativitou uživateľov. Pomocou Petriho sietí, statických miest, rolí, právomocí,dátových modelov a formulárov je možné vytvoriť nespočetné množstvo aplikácií.

O bezpečnosť a integritu dát sa starajú nasledujúce technológie:

- captcha – ochrana voči robotom
- servletové filtre – presmerovanie užívateľa na prihlasovaciu stránku, ak nie je prihlásený a chce sa dostať na zabezpečenú stránku
- bezpečnostné obmedzenia na jednotlivé stránky – užívateľovi sa aj pri obídení filtru nezobrazí stránka, ak jeho rola nemá príslušné právomoci
- Enterprise Java Beany – zabezpečujú prístup viacerých užívateľov k rovnakým komponentom
- transakcie – prístup k dátam v databáze je atomický

a vlastnosti systému:

- zaradenie uživateľov do jednej z troch rolí (administrátor, manažér, užívateľ) a dostupnosť funkcionalít na základe nich
- priradenie uživateľov k roliam v rámci workflowov

Efektívne a jednoduché vytváranie workflowov umožňujú nasledovné aplikácie a komponenty:

- PNeditor – kreslenie, simulovanie a rôzne vyhodnotenia Petriho siete
- Dataform modeler – grafický nástroj na tvorbu a mapovanie dátových modelov a formulárov pre Petriho sieť
- Validation modeler – nástroj na vytváranie vlastných validácií pre vstupné polia a použitie Javascript komponentov (napr. výber dátumu)
- Workflow management system
  - webaplikácia integrujúca výstupy zo všetkých aplikácií
  - jednoduché použitie – intuitívne grafické rozhranie, možnosť použiť len Petriho sieť, Petriho sieť s formulármi alebo Petriho sieť s formulármi aj validáciami

- jednoduché nasadenie – stačí prekopírovať program, spustiť databázový a aplikačný server, zadať URL adresu do webového prehliadača
- rozdelenie užívateľov do rolí zvlášť v rámci systému a workflowov

Systém je možné použiť v rôznych odvetviach, kde sa firma riadi nejakým workflowom ako napríklad:

- zdravotníctvo – príchod pacienta k lekárovi, sestrička zadá údaje pacienta, systém pri zadávaní validuje údaje, lekár vyšetrí pacienta, zadá recept do systému, pacient si vyzdvihne lieky v lekárni, lekárnik zaznamená výdaj liekov do systému
- poisťovníctvo – nastane poistná udalosť, klient zavolá do pobočky a pracovník vytvorí nový prípad a zadá potrebné údaje a posunie úlohu ďalej, ďalšiemu zamestnancovi sa zobrazí nová úloha, doplní ďalšie údaje a môže zavolať späť zákazníkovi a informovať o nasledujúcich krokoch

Implementáciou systému som sa naučil používať nové technológie a prehĺbil som si znalosti v:

- objektovo orientované princípy a programovanie v Jave
- konfigurácia aplikačného servera Glassfish – bezpečnosť a JNDI
- konfigurácia perzistenčnej vrstvy JPA a Hibernate
- tvorba normalizovaných tabuliek v databáze MySQL
- JPQL dotazy na prístup k dátam
- namapovanie Petriho sietí na workflow systém
- nahrávanie súborov cez Servlet 3.0
- komponenty a šablóny v JSF 2.0
- použitie transakcií cez Enterprise Java Bean 3.0

# 6 Literatúra


1. W.M.P. van der Aalst, *The Application of Petri Nets to Workflow Management*, http://wwwis.win.tue.nl/~wvdaalst/publications/p53.pdf
2. Aalst,W. - Hee, K. *Workflow Management Models, Methods, and Systems*, The MIT Press Cambridge, Massachusetts London, 2002, ISBN 0-262-01189-1
3. Hollingsworth David, *Workflow Management Coalition The Workflow Reference Model ,* 1995

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.79.1336&rep=rep1&type=pdf

4. Juhás,G. - Lorenz, R. - Mauser, S., Complete Process Semantics for Inhibitor Nets. Application and Theory of Petri Nets and Other Models of Concurrency, Springer-Verlag, 2007, LNCS 4546
5. Doğaç, A., Workflow management systems and interoperability, Springer-Verlag Berlin, 1998, ISBN 3-540-64411-3
6. Jablonski, S. - Bussler, C., *Workflow management: modeling, concepts, architecture and implementation*, International Thomson Computer Press, 1996, ISBN 1850322228
7. Java Persistence API v príkladoch z Hibernate, Gary Mak a Róbert Novotný, 2009 http://ics.upjs.sk/~novotnyr/java/java-persistence-api-tutorial/jpa.pdf
8. Java EE 5 Tutorial, Sun Microsystems, Inc., 2007, http://java.sun.com/javaee/5/docs/tutorial/doc/JavaEETutorial.pdf
9. MySQL 5.1 Reference Manual http://dev.mysql.com/doc/refman/5.0/en/index.html
10. JSF tutorial with Eclipse and Tomcat http://balusc.blogspot.com/2008/01/jsf-tutorial-with-eclipse-and-tomcat.html
11. Uploading files with JSF http://www.servletworld.com/servlet-tutorials/servlet3/multipartconfig-file-upload-example.html
12. Using Hibernate with Tomcat http://community.jboss.org/wiki/UsingHibernatewithTomcat
13. JSF 2.0 Tutorials, JavaServer Faces 2.0 with Facelets and Ajax http://www.coreservlets.com/JSF-Tutorial/jsf2/
14. Hibernate manuál

http://docs.jboss.org/hibernate/core/4.0/manual/en-US/html_single/#tutorial-firstapp

15. Základné J2EE návrhové vzory – Data Access Objects

http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

16. Oracle Java EE 5 Tutorial – EJB

http://docs.oracle.com/javaee/5/tutorial/doc/bnbmt.html#bnbmu

17. Oracle Java EE 5 Tutorial – JSF

http://docs.oracle.com/javaee/5/tutorial/doc/gentextid-10788.html

18. Tomáš Zuber – Tvorba workflow management systému, bakalárska práca, Bratislava: STU, 2010, 39s.