SLOVENSKÁ TECHNICKÁ UNIVERZITA

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

FEI-5382-50877

Tvorba workflow manažment systému

2010

Tomáš Zuber

SLOVENSKÁ TECHNICKÁ UNIVERZITA

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Tvorba workflow manažment systému

Bakalárska práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	9.2.9 Aplikovaná informatika
Školiace pracovisko:	Fakulta elektrotechniky a informatiky
Konzultant:	doc. RNDr. Gabriel Juhás, PhD.

Bratislava 2010

Tomáš Zuber

Abstrakt

Slovenská technická univerzita v Bratislave
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Meno autora:	Tomáš Zuber
Názov bakalárskej práce:	Tvorba workflow manažment systému
Vedúci bakalárskej práce:	doc. RNDr. Gabriel Juhás, PhD.
Rok odovzdania:	Máj 2010

Cieľom práce bolo vytvoriť workflow manažment systém vo forme webovej aplikácie, ktorá umožňuje efektívne prideľovanie práce užívateľom pre rôzne pracovné aktivity namodelované pomocou Petriho sietí. Prideľovanie prác je riešené priradením užívateľa do určitých pracovných skupín, rolí a tým umožňujúc vykonávanie jednotlivých aktivít, ako prezeranie a vypĺňanie formulárov podľa práv nastavených pre rolu.
Právo na pridelovanie rolí, nastavenie práv s prípadmi a mazanie workflowu má len vlastník workflowu a administrátor. Ostatní užívatelia majú právo vidieť všetky workflowy, prípady a aktivity, ktoré môžu vykonať. Podľa priradených rolí majú právo vytvárať a mazať prípady. Aplikácia je implementovaná v Java EE s využitím Java Server Faces 2.0 a využíva objektovo relačné mapovanie Hibernate-u medzi dabázou MySQL a aplikáciou bežiacou na aplikačnom serveri Tomcat.

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 :	Tomáš Zuber
Title of the bachelor theses:	Creation of a workflow management system
Supervisor:	doc. RNDr. Gabriel Juhás, PhD.
Year of the submission:	May 2010

The main objective was to create a workflow management sytem in the form of a web application which allows effective assigning of tasks for users to various work activities modeled using Petri nets. The work allocation is solved by assigning users to working groups, roles and enabling them the execution of activities like viewing and filling out forms according to the rights set for the role. The right to assign roles, set the rights for cases and delete workflow has only the owner of the workflow and the administrator. Other users have the right to view all workflows, cases and activities which they can execute. According to the assigned roles they have permission to create or delete cases. The application is implemented in Java EE using Java Server Faces 2.0 and uses object relational mapping with Hibernate between the MySQL database and application running on a Tomcat application server.

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

Obsah

Úvod

Zámerom práce je využitie a aplikovanie poznatkov o databázach, aplikačných serveroch, klient-server aplikáciách, objektovo relačnom mapovaní, tvorbe webstránok a programovení v Jave získaných z doterajšieho štúdia. Ďalej naštudovanie a pochopenie princípu Petriho sietí, workflow manažment systémov a ich kombinácie. Hlavným cieľom bolo navrhnúť, implementovať a otestovať webovú aplikáciu workflow manažment systému používajúceho sémantiku Petriho sietí. Hlavnými časťami projektu je registrácia užívateľov, prihlásenie sa do systému, zobrazenie a práca s workflowmi, prípadmi, aktivitami, formulármi, importovanie súborov a nastavenia workflowov. Systém mal umožňiť registráciu uživateľov zadaním užívateľského mena a hesla. Následne sa užívateľ môže úspešne prihlásiť a pracovať v systéme na základe práv priradených k jeho účtu.

Importovanie súborov zahŕňa XML súbory Petriho siete a dátového modelu formulára a samotného formulára vo formáte XHTML, ktoré sa uložia do databázy. Pomocou dostupných knižníc z programov, v ktorých boli vytvorené XML súbory sa potrebné dáta ukladajú zvlášť do tabuliek databázy kvôli rýchlejšiemu prístupu k dátam.

Prvým krokom projektu bolo navrhnúť a vytvoriť tabuľky v databáze. Následne vytvoriť modely, prístupové triedy a konfiguračné súbory pre perzistenčnú vrstvu. Po tomto kroku sa môže implementovať celá biznis logika systému a vytvoriť potrebné stránky pre užívateľské rozhranie.

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

Vývojové prostredie – Eclipse – predchádzajúce skúsenosti z iných projektov

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

Databázový server – MySQL – jeden z najznámejších open source databáz

Framework pre užívateľské rozhranie – JSF 2.0 – predchádzajúce skúsenosti s verziou 1.2

Perzistenčná vrstva – Hibernate – predchádzajúce skúsenosti z iných projektov

1 Analýza problému

Na začiatkoch vývoja informačných technológií (1960) sa vytvárali samostatné programi s vlastnými užívateľskými rozhraniami a vlastným spôsobom ukladania dát, čím sa obmedzovala použiteľnosť dát. O dekádu neskôr prišlo riešenie oddelením aplikácie od dát. Vznikli databázi (database) a systémy riadenia databáz (database management system). Týmto sa odstránila záťaž manažovania dát, zjednodušili aplikácie a dáta boli jednoduchšie použiteľné aj inými aplikáciami pomocou štandardizovaných príkazov a postupov. V 80-tych rokoch sa udialo niečo podobné s užívateľským rozhraním. Vývoj systémov riadenia používateľského rozhrania (user interface management system) umožnilo programátorom oddeliť užívateľské interakcie od samotnej aplikácie. Ďalším stupňom vývoja aplikácií je workflow management, ktorého cieľom je oddeliť všeobecnú funkcionalitu od aplikácie. Predstavuje riešenie pre riadenie, kontrolu, optimalizáciu a podporu biznis procesov.

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.[1]

Jednou z takýchto štruktúr sú Petriho siete, ktoré vytvoril v 60-tych rokoch Carl Adam Petri. Tieto siete sa neskôr doplnili farbou, časom a hierarchiou. Tieto doplnky uľahčujú modelovanie zložitých procesov, kde dáta a čas sú doležitými faktormi.

Výhody Petriho sietí:

- formálna sémantika – proces špecifikovaný Petriho sieťou má precíznu definíciu, lebo sémantika klasických Petriho sietí a niektorých vylepšení (farba, čas, hierarchia) bola formálne definovaná.
- grafické zobrazenie – Petriho sieť je grafický jazyk. Dôsledkom tohto Petriho siete sú intuitívne a ľahko pochopiteľné, preto sú vhodné aj pri komunikácii s koncovými užívateľmi.
- expresivita – Petriho siet podporuje všetky primitíva potrebné pre modelovanie workflow procesov. Keďže stavy sú reprezentované explicitne, umožňujú modelovanie závislostí a implicitné voľby.
- analýza – Umožňujú overiť vlastnosti (bezpečnosť, invariantnosť, deadlocky, atď.) a vyčísliť výkonové merania (čas odozvy, čas čakania, podiel obsadenosti, atď.). [4]

1.1 Workflow manažment systém

Termín workflow manažment odkazuje na doménu, ktorá sa zameriava na logistiku biznis procesov alebo kancelársku logistiku. Jej hlavným cieľom je ubezpečiť sa, že správne aktivity sú vykonané správnym človekom v správnom čase.

Workflow Management Coalition (WfMC) definuje workflow manažment systém nasledovne: „*Systém, ktorý kompletne definuje, manažuje a vykonáva workflowy cez spúšťanie programov, ktorých poradie vykonávania je riadené počítačovou reprezentáciou workflow logiky“*. [3, str. 6]

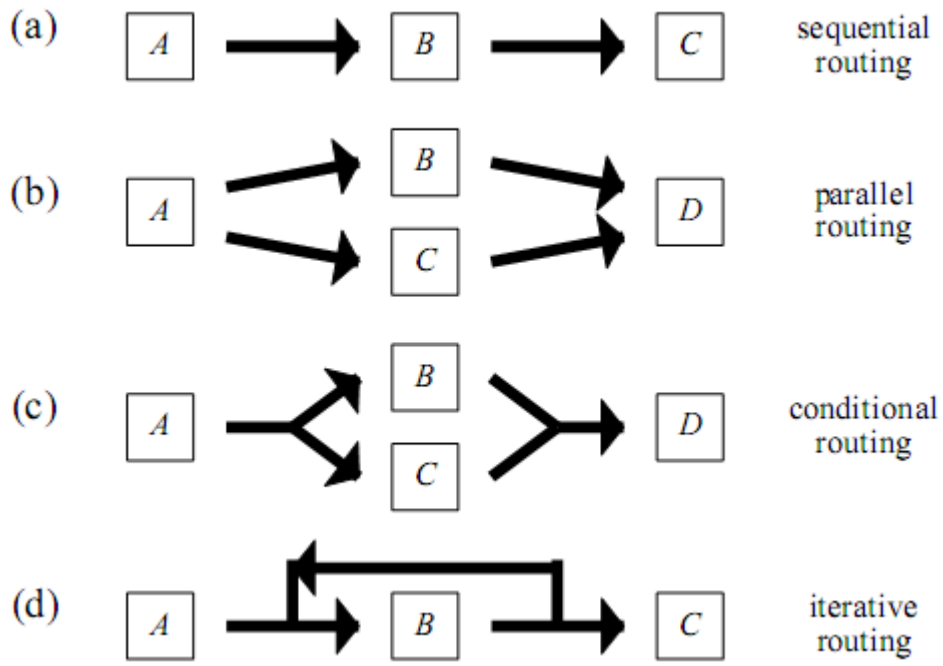
Ďalšie termíny charakterizujúce workflow manažment systém sú: biznis operačný systém, manažér workflowu, manažér prípadu a riadiaci systém logistiky.

Workflowy sú založené na prípadoch, t.j. každá časť práce je vykonaná pre špecifický prípad. Napríklad prípadmi sú pôžičky, poistné udalosti, daňové priznania, objednávky alebo žiadosť o informácie. Prípady sú často vytvárané externými zákzníkmi, ale môže sa stať, že sú vytvorené na inom oddelení v rámci jednej spoločnosti. Cieľom workflow manažmentu je zaobchádzať s prípadmi čo najefektívnejšie. Workflow procesy sú navrhnuté tak, aby dokázali spracovať podobné prípady. Prípady sú spracované vykonávaním úloh v špecifikovanom poradí. Požadované úlohy a ich poradie je dané definíciou procesov workflowu, taktiež nazývaný pracovný postup (procedúra), vývojový diagram alebo smerovacia definícia. Keďže úlohy sa vykonávajú v určitom poradí, je vhodné zistiť podmienky, ktoré zodpovedajú závislostiam medzi úlohami. Podmienka môže byť pravdivá alebo nepravdivá.[3]

Mnoho prípadov môže byť spracovaných podľa pokynov definície procesov workflowu. Dôsledkom tohto jedna úloha sa musí vykonať pre mnoho prípadov. Úloha, ktorá potrebuje byť vykonaná pre daný prípad sa nazýva pracovná jednotka (work item). Príkladom môže byť spracovanie úlohy *poslania formulára pre náhradu zákazníkovi* pre prípad *sťažnosť prijatá od zákazníka Jána Veľkého*. Väčšina pracovných jednotiek je spracovaná nejakým prostriedkom (resource). Prostriedkom môže byť stroj (tlačiareň, fax), počítač alebo človek (zamestananec). Najčastejším prostriedkom v kanceláriách a úradoch sú ľudia. Napriek tomu workflow manažment nie je obmedzený len na tieto miesta a preto sa uprednostňuje pomenovanie prostriedok. Pre uľahčenie pridelenia pracovných jednotiek k jednotlivým prostriedkom sa prostriedky zoskupujú do tried. Trieda prostriedkov je skupina prostriedkov s podobnými charakteristikami. Jedna trieda môže obsahovať mnoho prostriedkov a jeden prostriedok môže byť zastúpený vo viacerých triedach. Ak trieda

prostriedkov je založená na schopnostiach, t.j. funkčných požiadavkach ich členov, tak ju voláme rola (role). Ak triedenie je založené na štruktúre organizácie, tak túto triedu prostriedkov nazývame organizačná jednotka (organizational unit). Patria sem tímy, pobočky, oddelenia, katedry, atď. Pracovná jednotka spracovaná špecifickým prostriedkom je aktivita (activity). Ak si zoberieme fotografiu workflowu, tak vidíme prípady, pracovné jednotky a aktivity. Pracovné jednotky spájajú prípady a úlohy. Aktivity spájajú prípady, úlohy a prostriedky.

Workflow má 3 dimenzie: dimenziu prípadu, dimenziu procesu a dimenziu prostriedku. Dimenzia prípadu predstavuje, že všetky prípady sú spracované individuálne. Z hľadiska workflowu prípady sa navzájom neovplyvňujú, ale jednoznačne sa ovpyvňujú nepriamo zdielaním prostriedkov a dát. Dimenzia procesu definuje workflow procesy, t.j. úlohy a spoje medzi nimi. V dimenzii prostriedkov sú prostriedky zoskupené do rolí a do organizačných štruktúr.



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

Na obrázku sú zobrazené smerovacie konštrukcie určené medzinárodnou organizáciou Workflow Management Coalition (WfMC), ktorou cieľom je podporiť a vytvoriť štandardy pre workflow management systémy. WfMC bolo založené v roku 1993 a v januáry roku 1995 vydali slovník poskytujúci všeobecný súbor termínov pre zákazníkov, vývojárov a výskumníkov.

V slovníku sú definované 4 typy smerovania *(routing)*:

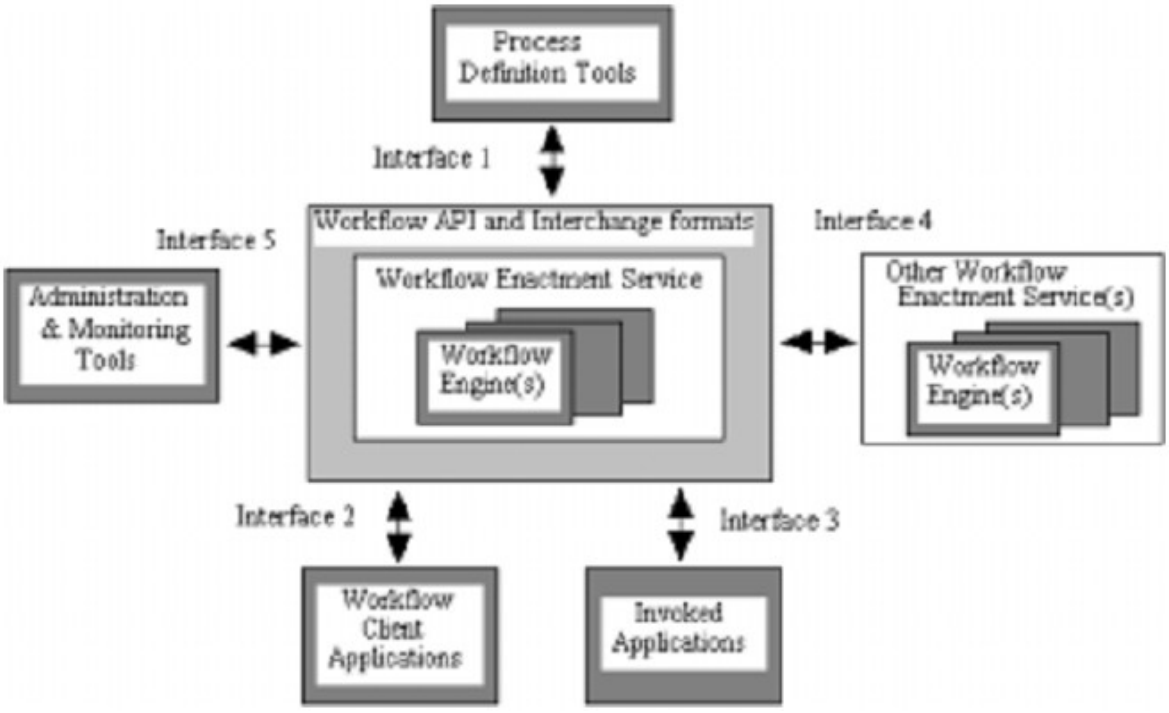
(a) sekvenčné *(sequential)*– úlohy sú vykonávané postupne za sebou

(b) paralelné *(parallel)*– viacero úloh sa vykonáva naraz v paralelných vetvách, rozdelenie do vetiev je realizované AND-splitom a následne spojenie tokov AND-joinom

(c) podmienený *(conditional)* – tok úloh závisí od definovaných podmienok, takáto štruktúra sa realizuje OR-splitom a OR-joinom

(d) iteračný *(iterative)* – niektoré úlohy alebo skupiny úloh je potrebné vykonať viackrát

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.



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

Na obrázku je prehľad tohto referenčného modelu. Zobrazuje hlavné časti a rozhrania v architektúre workflowov. Základom všetkých workflow systémov je nariaďovacia služba (enactment service). Jej úlohou je poskytnúť run-time prostredie, ktoré sa postará o kontrolu a vykonanie workflowov.

Z technického alebo manažérskeho hľadiska nariaďovacia služba môže používať viacero prostriedkov spracovania workflowu (engines). Nástroje na definíciu procesov (process definition tools) sa používajú na špecifikáciu a analýzu definícií procesov alebo na klasifikáciu prostriedkov. Tieto nástroje sa používajú v čase návrhu.
Väčšina workflow manažment systémov ponúka 3 nástroje na definíciu procesov:

- nástroj s grafickým rozhraním
- nástroj na špecifikáciu tried pre prostriedky
- simulačný nástroj na analýzu

Koncový užívateľ komunikuje so systémom cez klientskú aplikáciu (client application). Príkladom takejto aplikácie je známa metóda došlá pošta. Užívateľ môže vykonať úlohu pre daný prípad výberom pracovnej jednotky.
Administračné a monitorovacie nástroje (administration and monitoring tools) sa používajú na kontrolu a riadenie workflowu. Pomocou nich sa registrujú priebehy prípadov, odhaľujú chyby, nastavujú parametre, vymedzujú ľudia a riešia nezvyčajné stavy.
Snahou WfMC je štandardizácia a vytvorenie spoločného programovacieho rozhrania – Workflow Application Programming Interface (WAPI). Predpokladá sa vytvorenie množiny volaní pre WAPI a formáty pre vzájomnú výmenu údajov.

1.2 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. Zobrazenie miest je pomocou krúžkov a prechodov štvorčekmi.
„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 0 a viac tokenov (znakov) zobrazené ako čierne bodky. Stav M alebo taktiež nazývané značkovanie (marking) je aktuálne rozmiestnenie tokenov v miestach. Stavy sa reprezentujú nasledovne: $3p_1 + 2p_2 + 0p_3 + 1p_4$ je stav s tromi tokenmi v mieste p_1, dvomi tokenmi v p_2, nula tokenmi v p_3 a jedným tokenom v p_4. Tento stav môžeme reprezentovať aj ako $3p_1 + 2p_2 + p_4$.[1]

Počet tokenov sa môže zmeniť počas spracúvania siete. Prechody sú aktívnymi komponentmi v petriho sietiach. Menia stav siete podľa nasledovného spúšťacieho pravidla:

- Prechod t je povolený, ak každé vstupné miesto p pre prechod t obsahuje aspoň toľko tokenov, aká je násobnosť hrany medzi nimi.
- Povolený prechod môže byť spustený. Ak sa prechod t spustí, tak t spotrebuje z každého vstupného miesta p tokeny v závislosti od násobnosti hrany a vytvorí tokeny v každom výstupnom mieste v závislosti od násobnosti hrany.

„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 maximálny počet tokenov 1.

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.“

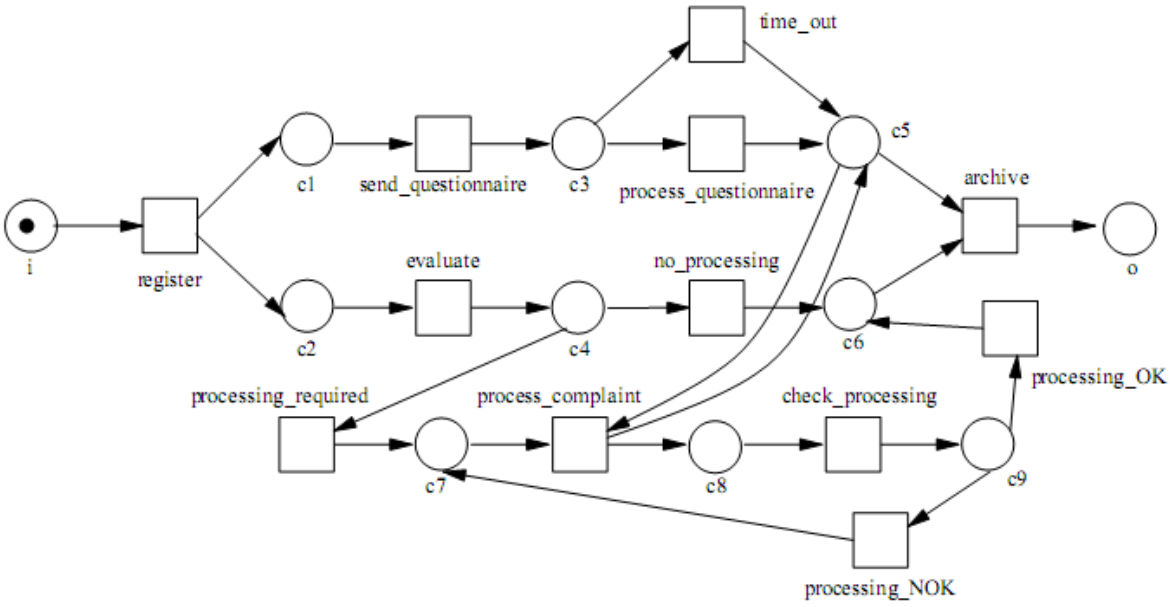
[1, str.13]

1.3 Mapovanie konceptov workflow manažmentu na Petriho siete

Väčšina workflow manažment systémov a s ním spojité metodológie oddeľujú modelovanie workflow procesov od modelovania štruktúry organizácie a prostriedkov v organizácii. Dôvodom je zníženie komplexity, zvýšené znovuvyužitie a možnosť zmeny procesu bez zmeny organizačného modelu.

V dimenzii procesov je definované, ktoré úlohy potrebujú byť vykonané a v akom poradí. Modelovanie definície workflow procesu použitím Petriho sietí je veľmi jednoduché: úlohy sú modelované prechodmi, podmienky miestami a prípady tokenmi.

Na ilustráciu mapovania konceptov workflow manažmentu na Petriho siete si môžeme predstaviť proces podávania sťažností. Ako prvé sa musí registrovať sťažnosť *(register)*. Potom sa paralelne pošle dotazník *(send_questionnaire)* sťažujúcej osobe a vyhodnocuje sa sťažnosť *(evaluate)*.Ak osoba pošle späť dotazník do 2 týždňov, tak sa spustí úloha p*rocess_questionnaire (spracovať_dotazník)*. Ak dotazník nie je poslaný späť, tak sa výsledky dotazníka vymažú *(time_out)*. Na základe výsledkov z vyhodnotenia sa sťažnosť spracuje alebo nespracuje. Aktuálne spracúvanie sťažnosti *(process_complaint)* je pozastavené, kým sa dotazník nespracuje alebo nevyprší čas čakania. Spracovanie sťažnosti je kontrolované cez úlohu *check_processing (kontrola_spracovania)*. Na koniec sa vykoná úloha *archive (uloženie)*.



Obr.3 Petriho sieť pre spracovanie sťažností [1,str.16]

Na obrázku č.6 vidno definíciu workflow procesov pre spracovanie sťažností použitím Petriho siete. Úlohy ako *register, send_questionnaire, evaluate, process_questionnaire, time_out, process_complaint, check_processing* a *archive* sú modelované prechodmi. Prechody *processing_OK* a *processing_NOK* boli pridané pre modelovanie 2 výsledkov spracovaním úlohy *check_processing*. Prechody *no_processing* a *processing_required* boli pridané kvôli podobným dôvodom. Na modelovanie stavov medzi úlohami boli pridané podmienky modelované miestami. Napríklad miestu c2 prináleží podmienka *pripravený na hodnotenie sťažnosti*. Podmienka c5 je pravdivá (obsahuje token), ak dotazník bol spracovaný alebo vypršal čas. Je potrebné si všimnúť, že c5 je prerekvizitou pre úlohy *archive a process_complaint*. Podmienka *i* je štartovacia podmienka a podmienka *o* je ukončovacia.

Definícia workflow procesu zobrazená na obr. 6 modeluje životný cyklus jedného prípadu. Vo všeobecnosti sa pracuje s viacerými prípadmi, ktoré majú rovnakú definíciu workflow procesov. Každý z týchto prípadov je reprezentovaný jedným alebo viacerými tokenmi. Ak sa v rovnakej Petriho sieti nachádza viac prípadov, tak tieto tokeny sa môžu pomiešať. Napríklad prechod *archive* môže spotrebovať 2 tokeny, ktoré patria rôznym prípadom. Takáto situácia je nežiaduca. Tento problém sa dá riešiť dvoma spôsobmi:

- Je možné použiť vysoko úrovňové Petriho siete, kde každý token má hodnotu (farbu), ktorá obsahuje informáciu o identite prislúchajúceho prípadu. Prechody nie sú spustiteľné, ak identifikátory prípadov nie sú rovnaké.
- Iným riešením je použiť samostatnú inštanciu Petriho siete pre každý prípad. Ak existuje *n* prípadov, tak sa vytvorí *n* inštancií Petriho siete.

Petriho sieť, ktorá modeluje definíciu workflow procesu (t.j. životný cyklus jedného izolovaného prípadu) nazývame Workflow sieť (Workflow net). Takáto sieť spĺňa 2 požiadavky:

- Obsahuje jedno vstupné miesto *i* (zdroj) a jedno výstupné miesto *o* (spotrebič). Token v mieste *i* reprezentuje prípad, ktorý je potrebné spracovať a token v mieste *o* zodpovedá prípadom, ktoré už boli spracované.
- Neobsahuje visiace úlohy a podmienky, t.j. všetky úlohy a podmienky prispievajú do aktuálne vykonávaného prípadu. Vzhľadom na túto skutočnosť je potrebné, aby každý prechod a miesto patrilo do cesty od miesta *i* po *o*.

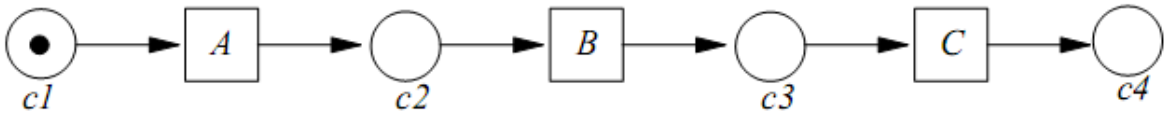
Rozmiestnenie tokenov (značkovanie) vo Workflow sieti reprezentuje stav workflowu pre jeden prípad. Stav workflowu obsahuje čiastočné informácie o stave prípadu. Navyše

prípad obsahuje workflow atribúty a aplikačné dáta. Pod pojmom workflow atribút si môžeme predstaviť riadiace premenné alebo logistický parameter ako napríklad vek sťažujúcej sa osoby, oddelenie zodpovedné za sťažnosť, dátum registrácie, atď. Ďalším zaužívaným termínom je atribút prípadu, ktorý sa pripisuje workflow atribútu špecifického prípadu. Na modelovanie týchto atribútov je možné použiť farebné Petriho siete. Aplikačné dáta sa nepoužívajú na manažovanie workflowu, ale na vykonávanie úloh. Príkladom takýchto dát je adresa sťažujúcej sa osoby alebo hodnotiace správy.

1.4 Modelovanie smerovaní pomocou Workflow sietí

Smerovacie konštrukcie definované organizáciou WfMC je možné modelovať pomocou stavebných blokov ako sú AND-split, AND-join, OR-split a OR-join. Nepochybne Workflow siete sú použiteľné na smerovanie prípadov.

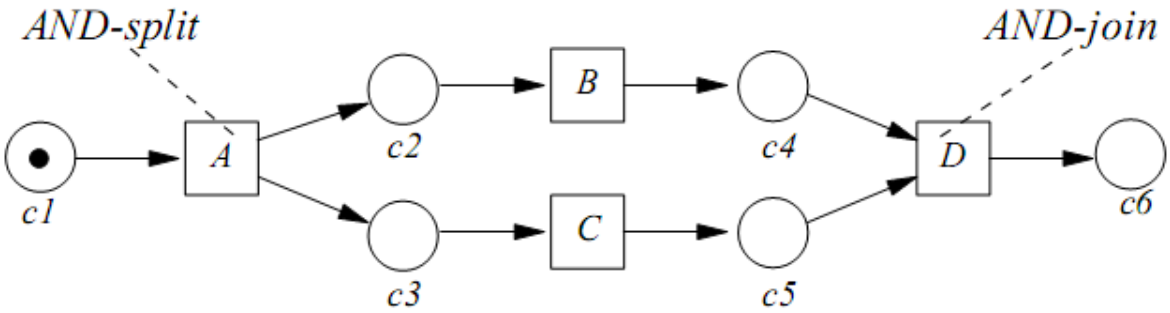
Sekvenčné smerovanie



Obr. 4 Sekvenčné smerovanie [1,str.18]

Sekvenčné smerovanie modeluje úlohy s príčinnými vzťahmi, t.j. úlohy nasledujúce za sebou. Uvažujme 2 úlohy A a B. Ak úloha B je spracovaná po ukončení úlohy A, tak A a B sú spracované sekvenčne. Na obrázku vidno, že smerovanie je modelované pridávaním miest. Miesto c2 modeluje príčinný vzťah medzi úlohami A a B, t.j. miesto c2 reprezentuje následnú podmienku pre úlohu A a predbežnú podmienku pre úlohu B. Miesto c3 modeluje príčinný vzťah medzi úlohami B a C.

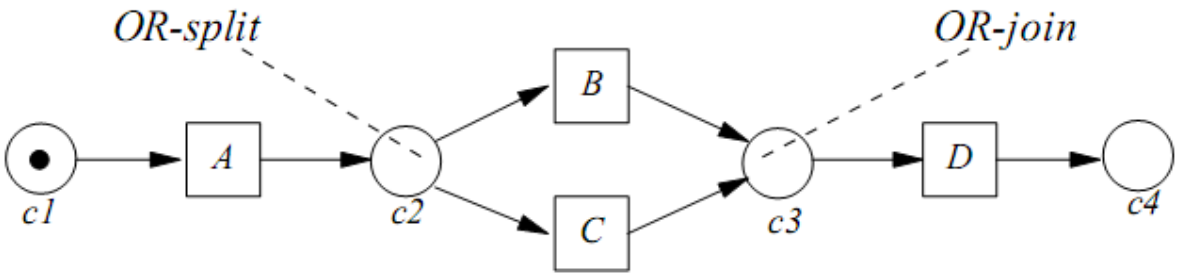
Paralelné smerovanie



Obr. 5 Paralelné smerovanie [1,str.19]

Paralelné smerovanie sa používa v situáciách, keď poradie vykonávania je menej prísne. Napríklad úlohy B a C musia byť spracované, ale ich poradie môže byť ľubovoľné. Na modelovanie takéhoto smerovania sa využijú stavebné bloky AND-join a AND-split. Z obrázku vidno, že tieto bloky modelujeme jednoduchými prechodmi. Tieto prechody možno pokladať za riadiace úlohy, ktoré boli pridané za účelom smerovania. Napriek tomu obyčajná úloha môže byť použitá ako AND-join a AND-split. Spracovanie AND-splitu A umožní vykonanie úloh B a C. AND-join D je povolené iba po vykonaní úloh B aj C, t.j. D je použité pre synchronizáciu 2 tokov. V niektorých workflow manažment systémoch nie je možné spustiť 2 úlohy naraz, ak patria k rovnakému prípadu. V takomto prípade sa nazývajú semi-parallel. Takéto smerovanie sa niekedy využíva pri predchádzaní vážnych technických problémov ako je napríklad zdielanie dát v rámci prípadu. Dáta nemožno meniť na dvoch miestach naraz (mechanizmus zámkov) kvôli zachovaniu konzistencii dát.

Podmienkové smerovanie

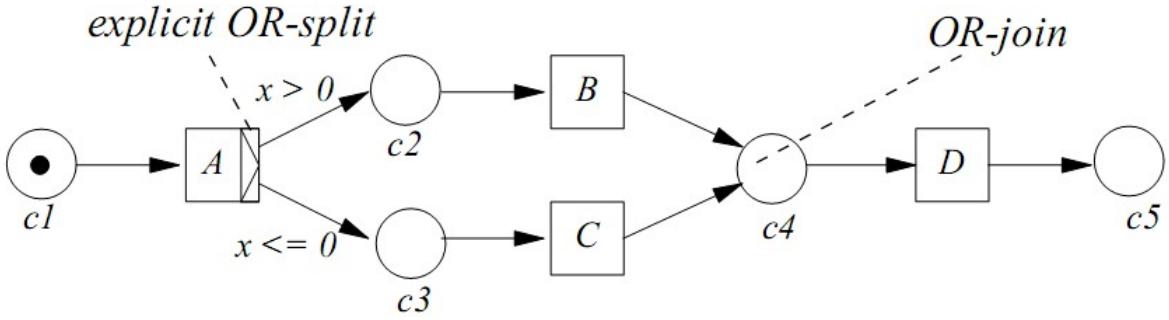


Obr.6 Podmienkové smerovanie OR [1,str.19]

Podmienkové smerovanie umožňuje smerovania, ktoré sa môžu líšiť medzi prípadmi. V tomto prípade smerovanie môže závisieť od workflow atribútov prípadu, od správania prostredia alebo od pracovného zaťaženia organizácie. Na modelovanie výberu medzi dvomi alternatívami sa používajú stavebné bloky OR-split a OR-join. OR-split sa dá modelovať miestom, z ktorého vychádza mnoho hrán a OR-join miestom, do ktorého vchádza mnoho hrán. Obrázok zobrazuje situáciu, kde za úlohou A nasleduje úloha B alebo C, t.j. vykoná sa výber medzi B a C. Po vykonaní jednej z týchto úloh sa vykoná úloha D. Miesto c2 predbežnou podmienkou pre B aj C. Napriek tomu len jedna z dvoch úloh bude vykonaná pre prípad v mieste c1.

Ak c2 obsahuje token, tak sa vykoná nedeterministický výber z B alebo C. Tento výber je deterministický len v prípade, ak sa použijú workflow atribúty. Napríklad smerovanie poistnej udalosti môže závisieť na výške odškodného. Kvôli tomuto sa musí brať do úvahy

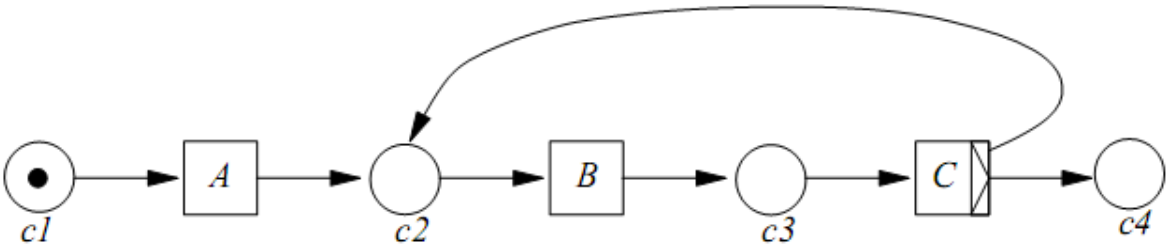
aj tento parameter ako workflow atribút. Ak odškodné prevýši určitú hodnotu, tak je potrebné vykonať doplňujúce kontroly. Existujú dva spôsoby na modelovanie výberu založeného na workflow atribútoch. Môžme použiť predchádzajúcu Workflow sieť a pridať predbežné podmienky pre každú úlohu tak, že iba B alebo C je spustiteľné, ak c2 obsahuje token. Druhým spôsobom je modelovanie deterministického výberu medzi B a C explicitným výberom.



Obr. 7 Explicitný výber medzi B a C na základe workflow atribútu x [1,str.20]

Prechod A má dve výstupné miesta c2 a c3, ale vytvorí len jeden token v c2 alebo c3. Tento výber sa uskutoční na základe hodnoty workflow atribútu x. Ak x je kladné, tak sa vykoná úloha B. V opačnom prípade, ak je x záporné alebo 0, tak sa vykoná úloha C. Na zobrazenie OR-splitu (exklusive) pre prechod A sa používa špeciálny symbol. Treba si všimnúť rozdiel medzi nedeterministickým a deterministickým výberom vzhľadom na moment výberu. Pri explicitnom OR-splite výber nastane pri vykonaní prechodu A. Pri obyčajnom OR-splite je výber realizovaný až pri vykonaní úloh B alebo C. Termínom explicit OR-split sa označujú výbery na základe workflow atribútov a implicit OR-splitom situácie, keď moment výberu nastáva čo najneskôr. Pri modelevaní workflowov je dôležité použiť správny OR-split.

Iteračné smerovanie



Obr. 8 Iteračné smerovanie [1,str.22]

Iteračné smerovanie je modelované OR-splitom. Jedna hrana z OR-splitu smeruje ďalej cestou workflowu a druhá hrana sa vracia k niektorému z predchádzajúcich miest. Na obrázku vidno, že riadiacou úlohou je úloha C, ktorá skontroluje výsledky po vykonaní úlohy B. Na základe tejto kontroly sa môže token vrátiť a úloha B sa vykoná ešte raz. Týmto spôsobom sa úloha B môže vykonať raz alebo viackrát. Iterácia je často považovaná za nežiaducu pre smerovanie, lebo sa vykonávajú rovnaké úlohy bez napredovania celkového toku. Napriek tomuto existuje mnoho situácií, kde iterácia sa nedá vynechať. Napríklad informácie poskytnuté zákazníkom sú neúplné alebo odmietnuté a je potrebné sa vrátiť na začiatok procesu.

2 Opis riešenia

2.1 Návrh projektu

Cieľom je vytvoriť workflow manažment systém vo forme webovej aplikácie použitím technológií Java Enterprise Edition, SQL a objektovo relačné mapovanie medzi databázou a aplikáciou. Hlavnou úlohou je vytvoriť biznis logiku pre riadenie workflowu namodelovaného Petriho sieťou, pre efektívne prideľovanie rolí pre užívateľov systému a vykonávanie jednotlivých úloh. Modelovanie Petriho sietí je realizované programom PNeditor. Namodelovaná sieť sa ukladá do súboru XML a používa koncovku *.pflow* (PNeditor 1) alebo *.do_not_expect_this_to_be_backwards_or_forwards_compatible* (PNeditor 2). PNeditor 2 je vo vývojovej fáze a sieť ukladá v inom formáte ako prvá verzia. Príčinou je modularizácia celého programu pre jednoduchšie vytváranie pluginov k nemu. Dôsledkom tejto reštrukturalizácie XML súboru a zmenou určitých komponentov by bolo nutné zmeniť celý dátový model workflow management systému, keďže v čase návrhu a implementácie ešte neboli známe tieto zásadné zmeny. Celý projekt bol navrhovaný pre modely z PNeditora 1. Tento problém sa vyriešil použitím zdrojových súborov a knižníc PNeditora 2, pomocou ktorých sa transformoval nový model na starý. Týmto spôsobom sa nemusel prerábať ani model ani biznis logika implementovaného systému. Prechod na novú verziu bol nutný, lebo dátový model bol implementovaný v tejto verzii. Tieto dátové modely sa pripájajú k jednotlivým prechodom a roliam. Týmto spôsobom vznikne z obyčajnej Petriho siete farebná Petriho sieť. Tokeny sú farebné, t.j. obsahujú určitú informáciu potrebnú na spracovanie.

Webová aplikácia sa štandardne vytvára pomocou databázy a aplikácie, ktorá vhodným spôsobom umožňuje užívateľovi pracovať s údajmi. Existujú rôzne spôsoby návrhu a realizácie takejto aplikácie, ale mojou úlohou bolo navrhnúť a implementovať aplikáciu pomocou 3-vrstvovej architektúry MVC:

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

Takáto architektúra zabezpečí vzájomné oddelenie dát, grafického rozhrania a biznis logiky, čo umožňuje nezávislý vývoj jednotlivých vrstiev a zjednoduší údržbu systému.

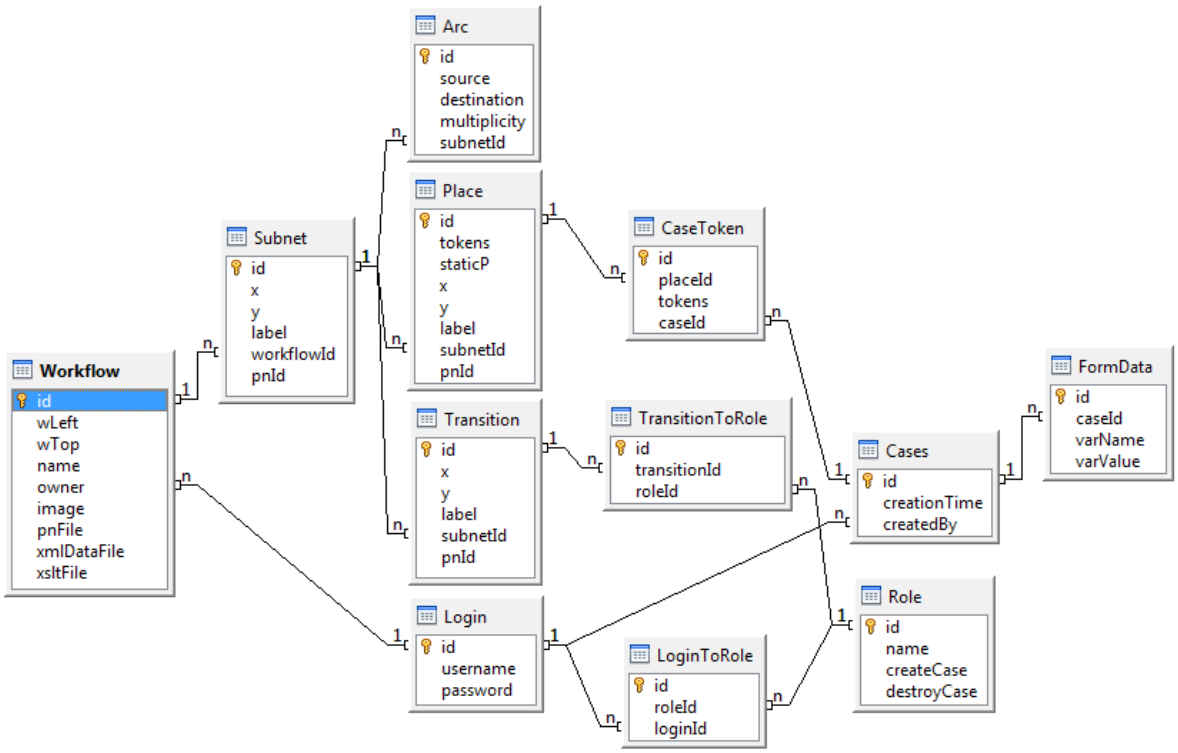
2.2 Implementácia projektu

2.2.1 Dátová vrstva

Dátovú vrstvu tvorí databáza MySQL 5.1 a perzistenčná vrstva Hibernate 3.31. Prvým krokom je návrh a tvorba tabuliek pre dáta:

- základné údaje o Petriho sieti, ktoré sa importujú z XML súboru: Workflow, Subnet, Arc, Place, Transition, Role, TransitionToRole
- prihlasovanie užívateľov a ich role: Login, LoginToRole
- prípady a ich aktuálny stav: CaseToken, XmlToken, Cases

Tabuľky so základnými údajmi o Petriho sieti obsahujú aj doplnkové informácie pre správnu funkčnosť s prípadmi a loginmi. Obsahujú id, ktoré je automaticky generované a je primárnym kľúčom a pnId, ktoré je id z XML súboru. Loginy priraďuje k roliam užívateľ, ktorý vlastní workflow. Prípady môžu pridávať a mazať užívatelia, ktorým je priradená rola s právom vytvárania/mazania prípadu.



Obr.9 Štruktúra a vzťahy tabuliek

2.2.2 Perzistenčná vrstva

Druhým krokom je tvorba perzistenčnej vrstvy medzi databázou a aplikáciou, ktorej úlohou je prevod údajov z databázy na objekty a opačne, zachovanie integrity dát a celkové zjednodušenie práce s dátami. Pre funkčnosť perzistenčnej vrstvy sú potrebné tieto kroky:

- Vytvoriť konfiguračný súbor hibernate.cfg.xml, v ktorom sa nastavia údaje potrebné na prepojenie databázy a aplikácie – adresa databázového servera, prihlasovacie údaje, driver k databáze (JDBC), meno databázy, mapované triedy a iné atribúty pre optimalizáciu výkonu.
- Vytvoriť java beany pre všetky tabuľky, ktoré budú namapované a samotné mapovanie medzi nimi s jedným z nasledujúcich spôsobov:
  - pomocou XML súborov – meno_triedy.hbm.xml
  - pomocou anotácií – Hibernate Annotations 3.4.0

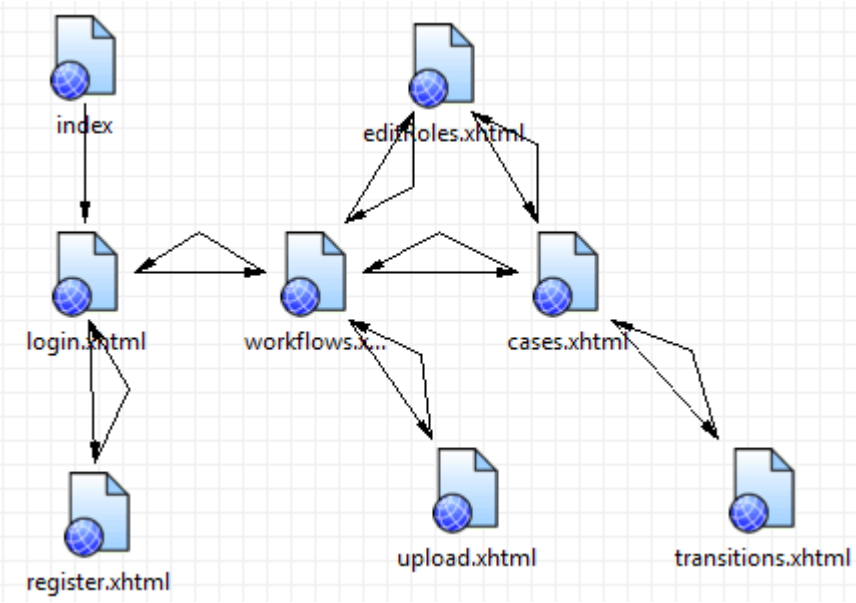
Ďalšími zaužívanými postupmi sú:

- Vytvorenie triedy HibernateUtil, ktorá implementuje hlavné metódy pre prácu s perzistenčnou vrstvou – vytvorenie/zrušenie session, commit/rollback transaction.
- Vytvorenie tried DAO pre prácu s jednotlivými tabuľkami

2.2.3 Prezentačná vrstva

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/WorkflowManagementSystem2/ sa načíta stránka index.jsp, ktorej úlohou je presmerovanie na stránku login.jsf. Tento krok je potrebný na inicializáciu a spustenie JSF.



Obr.13 Mapa stránok

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.

2.2.4 Aplikačná vrstva

Aplikačná vrstva je tvorená biznis logikou implementovanou v Jave pomocou manažovaných beanov a JSF. Manžované beany získavajú a ukladajú údaje z databázy pomocou metód z tried DAO a následne sú zobrazené prezentačnou vrstvou.

Na používanie systému je potrebné sa prihlásiť pomocou užívateľského mena a hesla. Ak nevlastníme žiadne konto, tak je potrebné sa registrovať pomocou registračného formulára. Po úspešnom príhlasení môže užívateľ pracovať s existujúcimi workflowmi alebo uploadnuť vlastný. Editáciu rolí a vymazanie workflowu môže vykonať len vlastník workflowu a administrátor. Na základe pridelených rolí systém povoluje tvorbu/mazanie prípadov a prácu s jednotlivými prechodmi.

Pre funkčnosť systému je najprv potrebné pridať potrebné knižnice a vytvoriť súbor web.xml. Až potom je možné vytvárať samotnú biznis logiku.
Nastavenia web.xml:

- <servlet> - nastavenie Faces servletu a cestu k jeho triede
- <servlet-mapping> - nastavenie vzoru (*.jsf – všetky súbory s koncovkou jsf) URL adresy, pre ktoré sa použije Faces servlet
- <welcome-file-list> - meno stránky, ktorá sa zobrazí ako prvá (index.jsp)
- <filter> - slúži na konfiguráciu Tomahawk komponentu použitého pri uploadovaní súborov. Definuje sa tu maximálna veľkosť uploadovaného súboru a veľkosť, pri ktorej sa súbor zapíše z pamäte na pevný disk.
- <filter-mapping> - nastavenie vzoru ako pri servlet-mappingu (*.jsf)

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

- hibernate – annotations, commons-annotations, core, hibernate3, ...
- commons – collections, el, fileupload, io, logging, ...
- jsf – api,impl
- jstl – api, impl
- mysql-connector-java-5.1.10-bin
- pnapi2
- datamodelCreator
- simple-xml-2.1.8
- tomahawk-1.1.9

3 Zhodnotenie projektu

Výsledný projekt spĺňa všetky požiadavky uvedené v zadaní projektu a v úvode. Boli implementované všetky potrebné časti podľa definovaného rozsahu: registrácia užívateľov, prihlásenie do systému, zobrazenie a práca s workflowmi, prípadmi, aktivitami, formulármi, importovanie súborov a nastavenia workflowov.

Obmedzenie systému je v modeloch dátových formulároch, keďže XML súbor môže obsahovať viac dátových modelov, ale systém vie pracovať zatiaľ len s jedným modelom. Pri použití viacerých modelov nastane výnimka v systéme kvôli pretečeniu v poli. Táto chyba je odstrániteľná prepracovaním niektorých častí vytvorenia polí pre jednotlivé modely. Toto obmedzenie nemá žiadny vplyv na použiteľnosť systému, ale použitím viacerých dátových modelov je vyššia prehľadnosť.

Ďalším problémom je navigácia medzi stránkami, ktorá nezabezpečuje navigáciu podľa práv užívateľa. Pri zadaní správneho URL užívateľ má možnosť vidieť, v niektorých prípadoch aj meniť údaje, ku ktorým by nemal mať prístup. Túto bezpečnostnú chybu je možné vyriešiť použitím filterov v Java Server Faces. Táto chyba patrí medzi závažné, ale vzhľadom na nedostatok času nebola odstránená. Napriek tomu systém pri správnom použití funguje podľa požiadaviek a tým pádom je implementácia biznis logiky správna a vytvorená podľa zadania.

Ďalej je potrebné zabezpečiť rovnakú funkčnosť a vzhľad v rôznych prehliadačoch, keďže zatiaľ systém funguje správne len v zabudovanom prehliadači Eclipsu. V ostatných prehliadačoch je problém so zobrazením obrázkov.

Problém môže nastať, keď užívatelia si prezerajú prípady, workflowy alebo vypĺňajú formuláre a medzitým ďalší užívateľ vymaže tú časť, ktorú si prezerajú ostatní alebo zmení práva uživateľa. V týchto prípadoch môže dojsť k určitej strate údajov. Ako napríklad strata skonzumovaného tokenu pri vypĺňaní formulára a vymazanie prípadu iným uživateľom. Tento problém treba vyriešiť obmedzením mazania len na nepoužívané časti alebo po vymazaní presmerovať uživateľov na inú stránku a vrátiť skonzumované tokeny do statických miest.

Vylepšenia do budúcnosti:

- Bezpečnosť
  - odstrániť bezpečnostné chyby URL - napr. použítím filtra pre stránky
  - heslo užívateľov ukladať v šifrovanej podobe do databázy – MD5, SHA
  - vytvoriť ochranu registrácie nových užívateľov voči botom - captcha, poslať potrvrdzujúci e-mail užívateľovi a následne aktivovať konto
- Rozšírenie funkčnosti a zlepšenie použiteľnosti
  - rozšíriť funkčnosť pre viac dátových modelov
  - pridať použiteľnosť checkboxov, radio buttonov pre formuláre
  - pridať osobné údaje k užívateľom – meno, priezvisko, e-mail
  - pridať možnosť zmeny hesla a osobných údajov
  - pridať možnosť udelovania práv pre prácu s workflowmi (zmena rolí,mazanie workflowu) – zatiaľ obmedzené na administrátora a vlastníka workflowu
  - vytvoriť systém logovania práce s workflowmi
  - presunúť tvorbu dátových modelov a formulárov priamo do systému
  - vylepšiť vzhľad stránok (hlavne formulárov)
  - zmeniť niektoré ikonky tlačidiel na zrozumiteľnejšie, prípadne k nim priradiť text
- Zníženie nákladov na systém
  - priebežne vymazať nepotrebné obrázky workflow sietí
  - znížiť počet vytvárania pomocných súborov na pevný disk
  - optimalizácia nastavení Hibernate-u, MySQL, Tomcat
  - prerobiť triedy na viac generické a eliminovať opakujúce sa implementácie rovnakých funkcií
  - použiť statické alebo finálne metódy, kde je to možné

4 Technická dokumentácia

Potrebné programy a servery:

- Eclipse JEE Galileo, Eclipse WTP 3.1.1
- MySQL 5.1.40
- Apache Tomcat 6.0.20

Všetky potrebné inštalačné súbory sú na priloženom CD.

Postup pre importovanie a konfiguráciu projektu do Eclipsu:

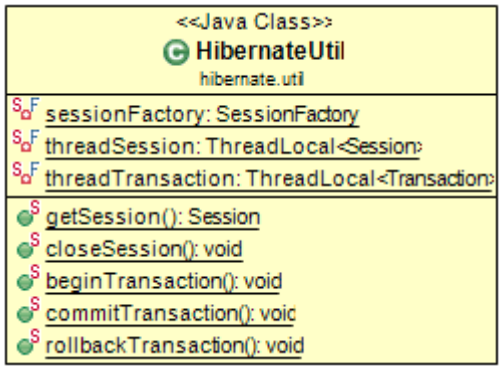
1. Rozbalíme súbor eclipse-jee-galileo-SR1-win32.zip na miesto, kde chceme mať uložené vývojové prostredie Eclipse.
2. Rozbalíme súbor wtp-R-3.1.1-...zip do adresára, kde sme rozbalili Eclipse.
3. Nainštalujeme server MySQL.
4. Rozbalíme súbor apache-tomcat-6.0.20.zip do adresára, kde chceme mať uložený aplikačný server Tomcat.
5. Spustíme Eclipse a pridáme Tomcat medzi servery → File → New → Other → Rozbalíme položku Server a vyberieme Server→ Next → Vyberieme Apache Tomcat v6.0 → Pri Server runtime environment klikneme na Add → Zadáme meno → Pri Tomcat installation directory klikneme na Browse a nastavíme cestu, kde sme rozbalili Tomcat → Finish → Finish.
6. Importneme projekt → File → Import → Web → WAR File → Next → Browse a vyberieme súbor WorkflowManagementSystem2.war (z CD) → Finish.
7. V Eclipse v Project Exploreri v Java Resources: src otvoríme súbor hibernate.cfg.xml a nastavíme url, username, password k MySQL serveru.
8. V MySQL spustíme SQL skript tableDefinitions.sql, ktorý nájdeme v Project Exploreri.
9. V Project Exploreri klikneme pravým tlačidlom na meno projektu WorkflowManagementSystem2 → Run As → Run On Server.
10. Zobrazí sa uvítacia stránka a systém je pripravený na použitie.

Postup pre importovanie a konfiguráciu projektu do Eclipsu:

1. Nainštalujeme server MySQL a nastavíme užívateľské meno root, heslo na adminadmin a ostatné nastavenia necháme defaultné.
2. Spustíme server MySQL a vytvoríme potrebnú databázu a tabuľky spustením skriptu tableDefinitions.sql.
3. Rozbalíme súbor apache-tomcat-6.0.20.zip do adresára, kde chceme mať uložený aplikačný server Tomcat. (napr. C:\Tomcat)
4. V adresári, kde sme nainštalovali Tomcat nájdeme adresár webapps a nakopírujeme sem súbor WorkflowManagementSystem2.war. (C:\Tomcat\webapps)
5. Spustíme Tomcat. (C:\Tomcat\bin\startup.bat)
6. Spustíme webový prehliadač a zadáme URL adresu http://localhost:8080/WorkflowManagementSystem2

4.1 Implementované triedy a ich metódy

4.1.1 HibernateUtil

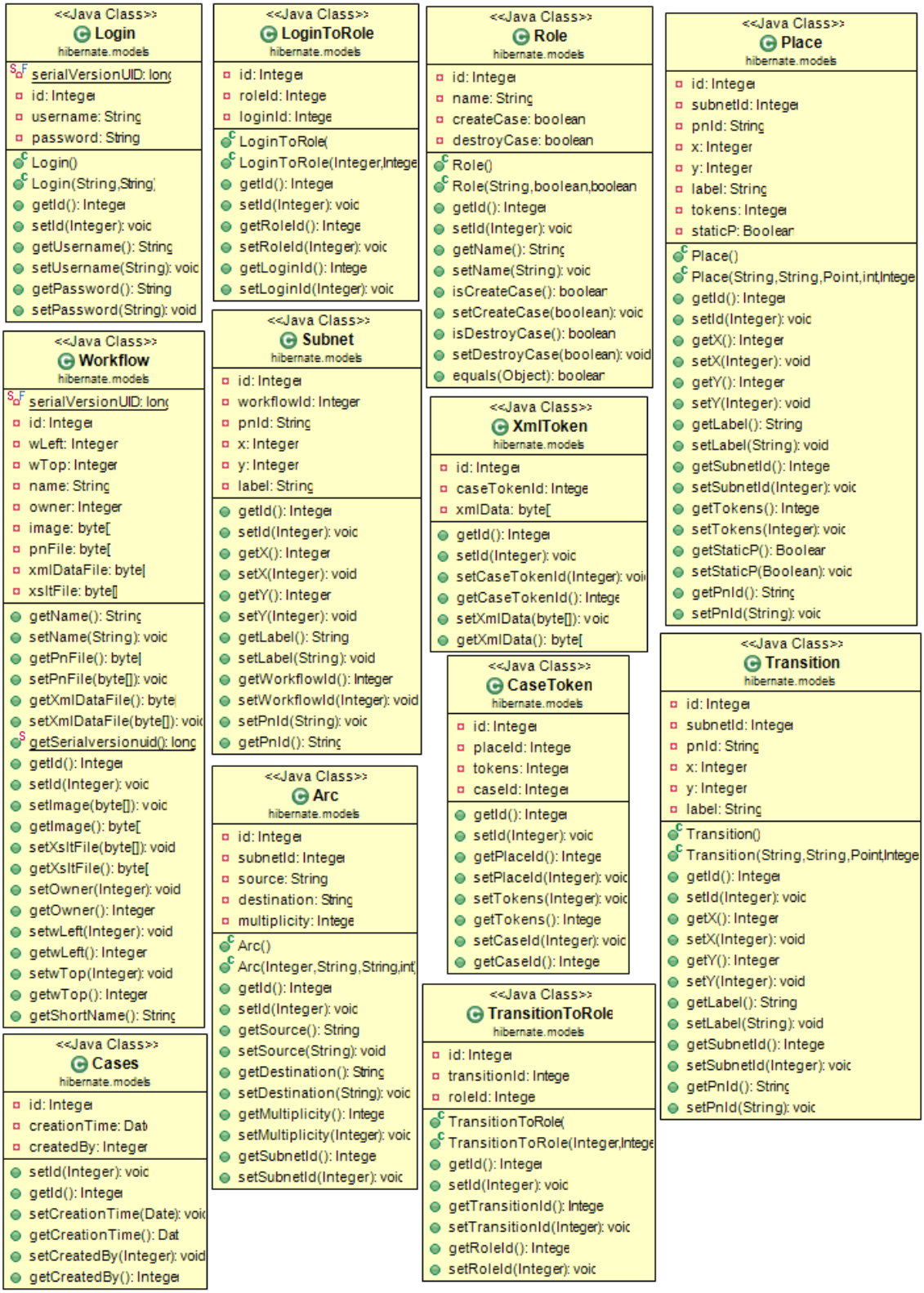


Obr.10 HibernateUtil Class diagram

Atribút sessionFactory je hneď na začiatku inicializovaný nepriamym vytvorením inštancie pomocou metódy buildSessionFactory() a nie pomocou konštruktora. Toto dovoľuje aplikácii špecifikovať vlastnosti a mapovania pre triedu SessionFactory.

Atribúty threadSession a threadTransaction sú inicializované na začiatku a pomocou ostatných metód sa v nich nastavujú potrebné atribúty.

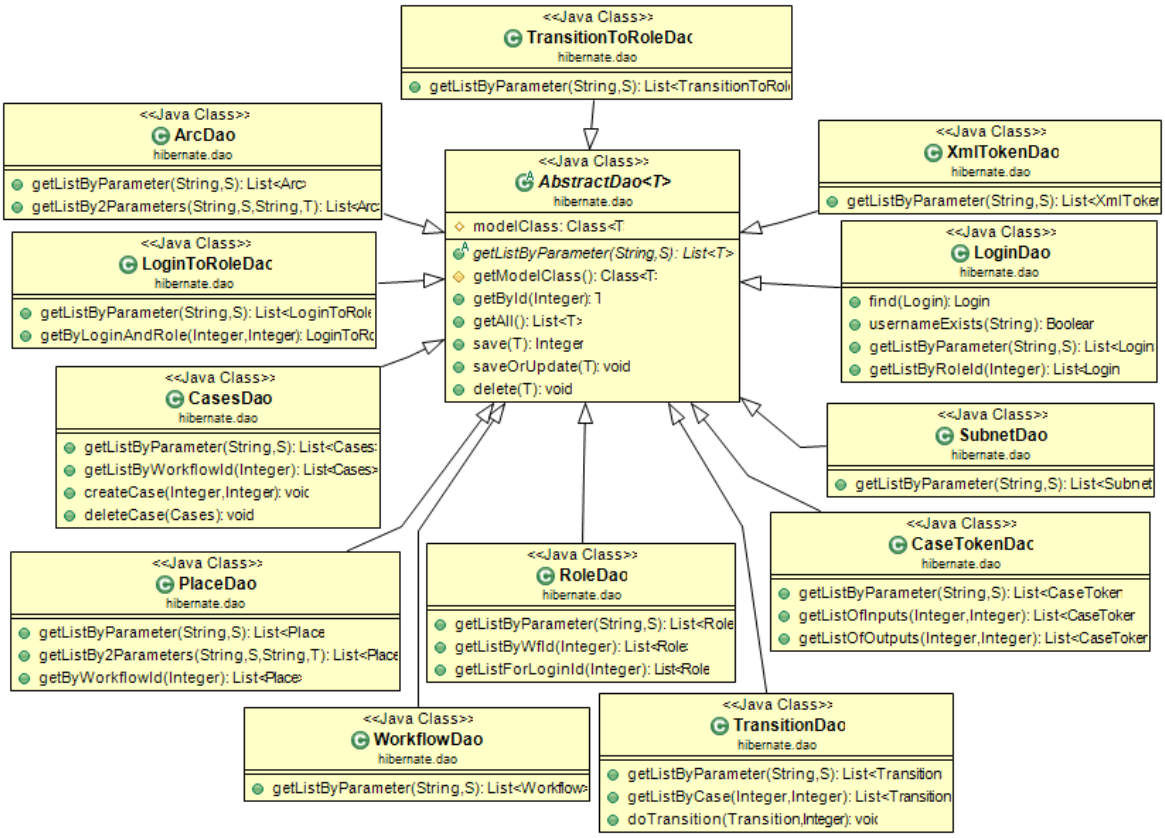
4.1.1 Hibernate modely



Obr.11 Class diagramy Hibernate modelov

Každý model je vytvorený ako Java Bean, t.j. obsahuje prázdny konštruktor, atribúty s privátnou viditeľnosťou, metódy get a set pre všetky atribúty (getters/setters) s verejnou viditeľnosťou. Modely sú doplnené anotáciami z javax.persistence. Trieda je označená anotáciou @Entity, atribút id anotáciou @Id (pre jednoznačné identifikovaníe) a @GeneratedValue (pre automatické generovanie id), atribúty nepoužívané v databáze anotáciou @Transient, atribúty s nejednoznačným dátovým typom anotáciou @Column(columnDefinition="dátovýTyp").

4.1.2 Hibernate DAO



Obr.12 Class diagramy Hibernate DAO tried

Všetky DAO triedy sú podtriedou abstraktnej triedy AbstractDao, ktorá implementuje základné metódy používané ostatnými triedami:

- Class<T> getModelClass() - slúži na získanie typu aktuálnej triedy potrebný pre použitie metód hibernatu (save, update,...)
- T getById(Integer id) – vráti objekt triedy T na základe zadaného id
- List<T> getAll() - získanie zoznamu všetkých objektov triedy T z databázy
- Integer save(T t) – slúži na uloženie objektu t do databázy a vráti id objektu

v databáze

- void saveOrUpdate(T t) – uloženie alebo aktualizáciu objektu t v databáze
- void delete(T t) – vymazanie objektu t z databázy
- abstract <S> List<T> getListByParameter(String column, S s) – abstraktná metóda, ktorá vráti zoznam objektov triedy T podľa parametra s v stĺpci column

Všetky DAO triedy implementujú metódu getListByParameter(String column, S s) a ďalšie metódy podľa potreby.

ArcDao:

- List<Arc> getListBy2Parameters(String column1, S s, String column2, T t) – vráti zoznam hrán na základe 2 parametrov

CasesDao:

- List<Cases> getListByWorkflowId(Integer workflowId) – vráti zoznam prípadov na základe workflowId spojením potrebných tabuliek (CaseToken, Place, Subnet)
- void createCase(Integer wfId, Integer createdBy) – vytvorí nový prípad pre daný workflow a uloží id používateľa, ktorým bol vytvorený. Prípad sa vytvorí klonovaním počiatočného stavu siete, t.j. získaním zoznamu miest v danom workflowe a ku každému z nich sa vytvorí CaseToken s potrebnými tokenmi.
- void deleteCase(Cases c) – vymaže prípad c z databázy a kaskádovo všetky CaseTokeny, ktoré k nemu patria

CaseTokenDao:

- List<CaseToken> getListOfInputs(Integer transitionId, Integer caseId) – vráti zoznam CaseTokenov, ktoré sú vstupom pre prechod transitionId v prípade caseId
- List<CaseToken> getListOfOutputs(Integer transitionId, Integer caseId) – vráti zoznam CaseTokenov, ktoré sú výstupom pre prechod transitionId v prípade caseId

LoginDao:

- Login find(Login login) – vyhľadanie užívateľa podľa username a password. Potrebné pre úspešné prihlásenie do systému.
- Boolean usernameExists(String username) – kontrola existencie užívateľského mena v databáze
- List<Login> getListByRoleId(Integer id) – získanie zoznamu užívateľov, ktorým je priradená rola s príslušným id

LoginToRoleDao:

- LoginToRole getByLoginAndRole(Integer login, Integer role) – vyhľadanie

objektu podľa identifikátorov užívateľa a role

PlaceDao:

- List<Place> getListBy2Parameters(String column1, S s, String column2, T t) - vráti zoznam miest na základe 2 parametrov

RoleDao:

- List<Role> getListByWfId(Integer wfId) – vyhľadanie zoznamu rolí pre daný workflow
- List<Role> getListForLoginId(Integer id) – vráti zoznam rolí, ktoré sú priradené užívateľovi s príslušným id

TransitionDao:

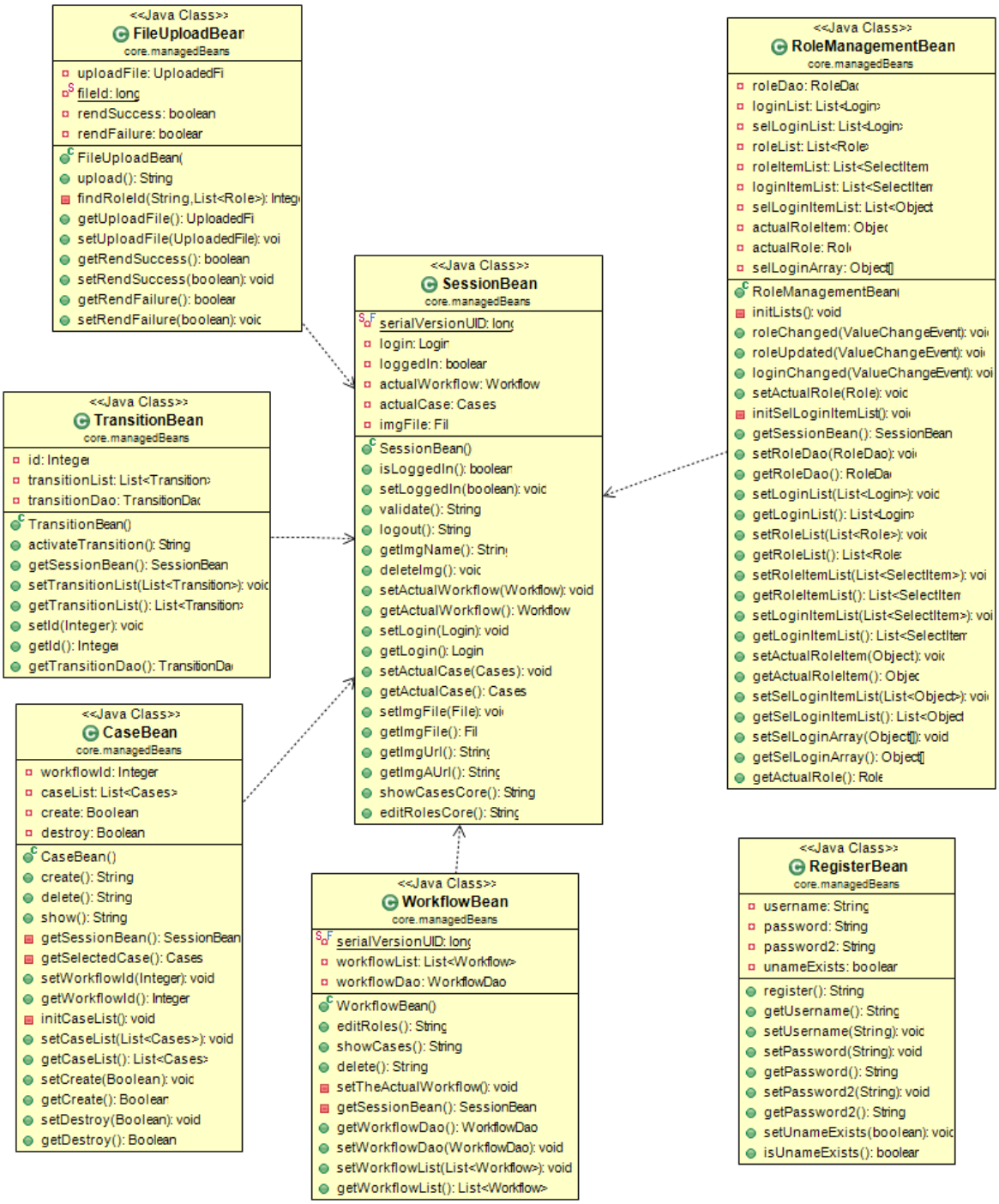
- List<Transition> getListByCase(Integer caseId, Integer loginId) – získanie zoznamu prechodov pre daný prípad, ktoré môže vykonať príslušný užívateľ
- void doTransition(Transition t, Integer caseId) – spracovanie prechodu t pre daný prípad

4.1.3 Jadro aplikačnej vrstvy

Jadro (core) je rozdelené na 2 balíky:

- managedBeans – obsahuje všetky manažované Java Beany projektu
- util – zahŕňa 2 triedy, ktoré obsahujú metódy využívané Java Bean-mi

4.1.4 ManagedBeans



Obr.14 Class diagramy manažovaných Java beanov

CaseBean:

- String create() - vytvorenie prípadu a inicializácia zoznamu prípadov
- String delete() - vymazanie prípadu a inicializácia zoznamu prípadov
- String show() - nastaví aktuálny prípad a prejde na transitions.jsf stránku

- SessionBean getSessionBean() - vráti sessionBean pomocou FacesUtil
- Cases getSelectedCase() - vráti vybratý prípad z tabuľky
- void initCaseList() - inicializuje prípady z databázy

FileUploadBean:

- String upload() - uloží uploadnutý súbor a celý workflow v ňom do databázy. Využitím metód z knižnice pnapi2.jar sa transformuje Petriho sieť do potrebnej štruktúry, ktorú používa systém, lebo celková štruktúra Petriho siete je odlišná od starej verzie a tým sa zmenila aj štruktúra XML súboru.
- Integer findRoleId(String rp, List<Role> roleListHib) – nájde a vráti rolu s menom rp v zozname roleListHib

RegisterBean:

- String register() - otestuje, či zadané údaje nie sú prázdne a kontroluje jedinečnosť užívateľského mena. Ak sú splnené tieto podmienky, tak sa zadané údaje uložia do databázy a užívateľ sa môže prihlásiť

RoleManagementBean:

- void initLists() - inicializuje zoznam uživateľov, rolí a zoznam vybratých uživateľov na základe aktuálnej role
- void roleChanged(ValueChangeEvent event) – nastaví aktuálnu rolu podľa vybranej položky zo zobrazeného zoznamu rolí
- void roleUpdated(ValueChangeEvent event) – pri zmene stavu checkboxov Create alebo Destroy case sa uložia zmeny do databázy
- void loginChanged(ValueChangeEvent event) – pri zaznačení alebo odznačení uživateľa sa zmeny uložia do databázy a obnovia sa zoznamy
- void setActualRole(Role actualRole) – nastaví aktuálnu rolu a obnoví zoznam vybraných užívateľov
- void initSelLoginItemList() - inicializuje zoznam vybraných užívateľov
- SessionBean getSessionBean() - vráti sessionBean pomocou FacesUtil

SessionBean:

- String validate() - validuje zadané prihlasovacie údaje a presmeruje ho na stránku workflows.jsf, ak sú údaje správne
- String logout() - odhlási uživateľa nastavením potrebných údajov na null, zruší session a vymaže dočasne vytvorený obrázok workflowu
- String getImgName() - vráti absolútnu cestu obrázka workflowu

- void deleteImg() - ak existuje obrázok workflowu, tak sa vymaže
- String getImgUrl() - vráti URL adresu obrázka (pre tag <img>)
- String getImgAUrl() - vráti URL adresu obrázka (pre tag <a>)
- String showCasesCore() - vytvorí náhodné meno súboru pre obrázok, uloži ho do adresára /images/temp/ a zapíše dáta obrázku z databázy do súboru
- String editRolesCore() - presmeruje na stránku editRoles.jsf

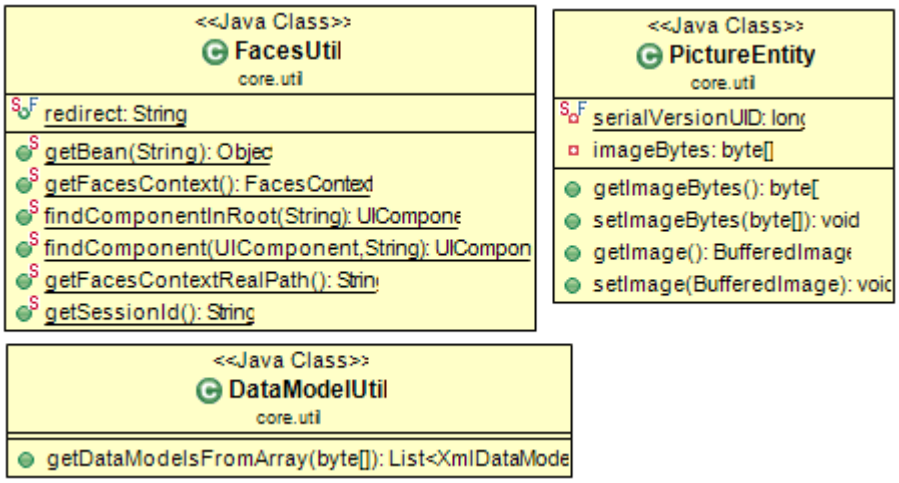
TransitionBean:

- String activateTransition() - aktivuje prechod, na ktorý bolo kliknuté v tabuľke a obnoví zoznam prechodov
- SessionBean getSessionBean() - vráti sessionBean pomocou FacesUtil

WorkflowBean:

- String editRoles() - nastaví aktuálny workflow kliknutý v tabuľke a zavolá metódu editRolesCore() triedy SessionBean
- String showCases() - nastaví aktuálny workflow kliknutý v tabuľke a zavolá metódu showCasesCore() triedy SessionBean
- String delete() - vymaže role, asociácie týchto rolí k užívateľom, prípady workflowu a samotný workflow
- void setTheActualWorkflow() - uloží aktuálny workflow vybraný v tabuľke do SessionBean

4.1.4 Util



Obr.15 Class diagramy tried v balíku Util

FacesUtil:

- Object getBean(String name) – vráti Java Bean-u na základe mena
- FacesContext getFacesContext() - vráti aktuálnu inštanciu FacesContextu
- UIComponent findComponentInRoot(String id) – nájde a vráti component na základe zadaného id
- UIComponent findComponent(UIComponent base, String id) – rekurzívne prehľadá komponenty a vráti hľadaný podľa id

PictureEntity:

- BufferedImage getImage() – vráti objekt BufferedImage vytvorený z reťazca bajtov získaných z databázy
- void setImage(BufferedImage image) – uloží objekt BufferedImage ako reťazec bajtov vhodný na uloženie do databázy

DataModelUtil:

- List<XmlDataModel> getDataModelsFromArray(byte[] buffer) – vráti zoznam XmlDataModelov z reťazca bajtov získaného z databázy

5 Záver

Navrhnutý a implementovaný workflow manažment systém spĺňa vytýčené ciele, ktoré boli definované na začiatku. Tento systém má širokú využiteľnosť v rôznych sférach. Môže slúžiť na riadenie komplexných pracovných aktivít, kontrolu a optimalizáciu pracovných postupov, na zistenie chýb alebo nedostatkov vo workflowe. Po vykonaní komplexných testov a následnom odstránení závažných chýb je možné použiť tento systém pre malé podniky. Jeho výhodou je jednoduché modelovanie workflowov pomocou Petriho sietí, vytváranie formulárov, jednoduché používanie systému, zabezpečuje delenie aktivít medzi užívateľov a povoluje im vykonávať iba akcie, na ktoré majú právo. Workflow manažment systémy sú komplexné aplikácie, ktoré potrebujú dostatok času na testovanie. Dôležité je otestovať nasledujúce činnosti:

- používanie viacerými uživateľmi naraz
- doba odozvy systému pri zaťažení
- pochopiteľnosť celkového systému pre obyčajného užívateľa

Vypracovaním tejto práce som si prehĺbil znalosti v programovaní v Jave, v konfigurácii aplikačného servera a perzistenčnej vrstvy, tvorbe tabuliek v databáze, SQL príkazoch a osvojil som si nové znalosti z oblasti Petriho sietí, workflow manažment systémov, Java Server Faces 2.0, využitie Hibernate anotácií a použitie knižnice Tomahawk na nahrávanie súborov na server. Po dokončení projektu som zistil, že čas vyhradený na analýzu a návrh projektu bol príliš dlhý, čo spôsobilo zníženie času na testovanie, keďže z času na implementáciu som nemohol ubrať. Mojím cieľom bolo navrhnúť správne aplikáciu, aby sa následne nemusela prerábať. Moje zmýšlanie bolo trocha optimistické, keď som si myslel, že môžem navrhnúť projekt podľa zadania bez akýchkoľvek následných zmien. Ďalším dôvodom predĺženia implementácie a návrhu bola zmena verzii programu, z ktorého sa importuje Petriho sieť a tým sa zmenila celá štruktúra XML súboru a aplikácia sa musela prerobiť. Z týchto skutočností vyplýva, že hlavnou príčinou chýb a nedostatkov v projekte je nedostatok času na testovanie a následnú opravu.

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://balusc.blogspot.com/2008/02/uploading-files-with-jsf.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/