

Prehlásenie autora

Čestne prehlasujem, že táto bakalárska práca je mojim pôvodným autorským dielom, na ktorom som pracovala samostatne na základe vlastných teoretických a praktických poznatkov získaných počas štúdia a informácií z dostupnej literatúry. Všetky zdroje, literatúru a pramene, ktoré som pri vypracovaní použila, riadne citujem s uvedením plného odkazu na príslušný zdroj. Uvedenú prácu som vypracovala pod vedením prof. RNDr. Gabriela Juhása, PhD..

Bratislava, dňa 20. 05. 2016

.....

Anna Demeterová

Pod'akovanie

Ďakujem rodine, priateľovi Matúšovi, vedúcemu práce prof. RNDr. Gabrielovi Juhásovi, PhD. a ďalším študentom, ktorí sa podieľali na tvorbe komplexného Workflow manažment systému, ktorého je táto práca súčasťou.

Abstrakt

Univerzita: Slovenská technická univerzita v Bratislave

Fakulta: Fakulta elektrotechniky a informatiky

Študijný program: Aplikovaná informatika

Školiace pracovisko: Ústav informatiky a matematiky

Autor: Anna Demeterová

Názov práce: Workflow manažment systém - procesný server

Vedúci práce: prof. RNDr. Gabriel Juhás, PhD.

Bratislava 2016

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Kľúčové slová: Petriho siete, workflow manažment, prípady použitia

Abstract

University: Slovak Technical University in Bratislava

Faculty: Faculty of Electrical Engineering and Information Technology

Study programme: Applied Informatics

Training workplace: Institute of Computer Science and Mathematics

Author: Anna Demeterová

Title: Workflow management system - process server

Thesis supervisor: prof. RNDr. Gabriel Juhás, PhD.

Bratislava 2016

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Keywords: Petri nets, workflow management, case

Obsah

0 Úvod	8
1 Analýza problému	9
1.1 Workflow manažment systémy	9
1.1.1 Prípad použitia - case	9
1.1.2 Úloha - task	10
1.1.3 Workflow rozmery	10
1.2 Petriho siete	10
1.2.1 História	10
1.2.2 Klasické Petriho siete	11
1.2.3 Spustiteľnosť a spustenie prechodu	11
1.2.4 Grafické znázornenie	12
1.2.5 Inhibítorové a reset hrany	12
1.3 WFMS založené na Petriho sieťach	13
1.3.1 Modelovanie smerovania	14
1.3.2 Analýza modelu procesu	16
1.3.3 ??Spôľahlivosť?? - Soundness	16
1.4 Použité technológie	17
1.4.1 HTML, HTML5	17
1.4.2 PHP	18
1.4.3 SQL, MySQL	19
1.4.4 AJAX	19
1.4.5 XML, SVG	20
1.4.6 JSON	20
1.5 Použité návrhové vzory	20
1.5.1 Model-View-Controller (MVC)	20
1.5.2 Registry - Register	21
1.5.3 Singleton	22
2 Opis riešenia - WFMS	23
2.1 Procesný portál	23
2.2 Procesný editor	23
2.3 Editor manažmentu rolí a organizačnej štruktúry	24
2.4 Editor formulárov a dátového modelu	24

2.5	Procesný server	24
2.6	Server manažmentu rolí a používateľov	25
2.7	Server formulárov a dátového modelu	25
3	Opis riešenia - Procesný server	27
3.1	Výber technológií	28
3.2	Výber návrhových vzorov, návrh systému	28
3.3	Jadro systému	29
3.3.1	Register	30
3.3.2	Smerovač	30
3.3.3	Objekt pre prístup k databáze	31
3.3.4	Objekt pre autentifikáciu	31
3.3.5	Manažér upozornení	31
3.3.6	Autoloader	31
3.4	Modely	31
3.5	Ovládače	32
3.6	Náhľady	32
3.7	Databáza	32
3.8	Implementácia biznis logiky	32
3.8.1	Uloženie Petriho siete do databázy	32
3.8.2	Vytvorenie prípadu	33
3.8.3	Správa workflow prípadu	34
3.8.4	Funkcia vrátenia úlohy	38
3.8.5	Uchovávanie histórie	38
3.8.6	Vizualizácia prípadu	39
3.9	Testovanie riešenia - možné úpravy a rozšírenia	40
3.10	Use case diagramy	40
4	Záver	41

Zoznam obrázkov

1	Grafické značenie objektov Petriho sietí	12
2	Grafické značenie inhibítor a reset hrán	12
3	Príklad workflow procesu pomocou Petriho siete - Žiadosť o pôžičku . .	14
4	Stavebné bloky modelovania workflow	15
5	Sekvenčné smerovanie[1]	15
6	Paralelné smerovanie[1]	15
7	Podmienkové smerovanie[1]	15
8	Využitie programovacích jazykov na serverovej strane pre webové stránky ku 25.4.2016[24]	19
9	Model-View-Controller	21
10	Register	21
11	Server manažmentu rolí a používateľov	25
12	Server formulárov a dátového modelu	27
13	Štruktúra Procesného servera	29
14	Formulár na vytvorenie prípadu	34

0 Úvod

V praxi sa často stretávame s potrebou riadenia interných biznis procesov firiem alebo pracovných tímov. Väčšinou sa riadenie realizuje pomocou rôznych softvérových systémov. V týchto prípadoch nastáva problém nejednotnosti systémov, alebo potreby znalostí komplikovaného modelovacieho, prípadne programovacieho jazyka. Často je potrebné funkcionality nových procesov implementovať do zabehnutých systémov, čo je spojené so zvýšenými finančnými výdavkami a časovou investíciou.

Riešením tohto problému by bol jednotný systém, ktorý by umožňoval pomocou jednoduchého a ľahko pochopiteľného modelovacieho jazyka pridávať funkcionality rôznych procesov bez nutnosti odborného programovania.

Našou snahou bolo navrhnúť a vytvoriť takýto Workflow manažment systém, v ktorom sa procesy budú modelovať pomocou Petriho sietí a v jednoduchých editoroch na front-ende bude možné pridávať dodatočnú funkcionality, a to pridávanie formulárov a zdefinovanie používateľských rolí pre jednotlivé úlohy. Navyše, v systéme sa procesy budú môcť zdieľať medzi rôznymi firmami, čiže proces jednej firmy si budú môcť zakúpiť iné.

Cieľom tejto bakalárskej práce je namodelovaný proces pomocou Petriho siete uložiť do databázy, umožniť vytvorenie tzv. case-u prípadu a na základe jeho logiky ho reprezentovať vo forme úloh pre používateľov. Taktiež umožňuje vizualizáciu prípadov, aby bolo zistiteľné, v akom stave sa ktorý prípad nachádza, ktoré úlohy sú splnené, ktoré sa vykonávajú a ktoré ešte vykonané neboli.

1 Analýza problému

1.1 Workflow manažment systémy

Cieľom mnohých informačných systémov v dnešnej dobe je podpora a monitorovanie biznis procesov. Je žiadané, aby tieto systémy kontrolovali správny tok práce v organizáciách. Na základe tejto potreby a špecifikácie sa podľa[1] zrodil termín **Workflow manažment systém** (ďalej WFMS). W.M.P. van der Aalst v [1] definuje WFMS ako všeobecný softvérový nástroj, ktorý umožňuje definovanie, realizáciu, registráciu a kontrolu pracovných postupov. Termín workflow sa často používa ako synonymum pre biznis proces [9] a môžeme ho chápať ako rozdelenie komplikovanejšieho procesu na postupnosť jednoduchších krokov. Túto postupnosť je potrebné mať správne zadefinovanú, ak chceme, aby WFMS pracoval správne, ako je žiadané, pretože kvalita WFMS stojí na tomto zadefinovaní. WFMS ponúkajú riešenie pre podporu a optimalizáciu komplexnejších činností - procesov, riadi, aby aktivity boli plnené správnym človekom v riadnom čase v správnom poradí. Podľa Workflow Management Coalition[1] sa tieto systémy vzťahujú k doménam sústredeným na logistiku biznis procesov. *“Workflow manažment systémy úplne definujú, riadia a vykonávajú workflow pomocou spúšťania softvéru, ktorého poradie jednotlivých krokov je riadené počítačovou reprezentáciou workflow logiky.”* Tieto systémy nie sú limitované na špecifickú biznis aplikáciu, sú široko použiteľné.[9]

Ako príklad uveďme proces Žiadosť o pôžičku. Tento proces má pevne zadefinované kroky, ich postupnosť, pravidlá pre ich splnenie. Banka opakovane používa tento proces pre správny postup pri posudzovaní žiadostí.

1.1.1 Prípad použitia - case

Podľa [1] WFMS sú založené na *case-och/ prípadoch*, čo znamená, že z procesov sú používateľmi vytvárané prípady použitia, takzvané case -y (ďalej prípad). Prípad je unikátna reprezentácia workflow procesu a má svoju identitu.[4] Cieľom WFMS je riadiť prípady čo najefektívnejšie. Opäť uvažujme proces Žiadosť o pôžičku. Zákazník požiada o pôžičku vo firme používajúcej tento proces, považujeme to za jeden prípad, alebo prípad použitia. Ak o pôžičku zažiada iný zákazník, považujeme to za iný prípad. Taktiež ak by ten istý zákazník opätovne podal ďalšiu žiadosť, opäť je to nový prípad.

1.1.2 Úloha - task

Podľa [9] prípady procesov sú reprezentované ako tasky - úlohy. Sú to nedeliteľné - atomické logické jednotky procesu.[3] Tieto úlohy plnia používatelia, zamestnanci vo firme, prípadne tvorcovia prípadu, čo bývajú často ľudia mimo firmu, ako napríklad zákazníci. S týmito úlohami môžu byť späté rôzne podúlohy, ako napríklad vyplnenie formulárov, dodanie dokumentov, a iné. V [1] je spustiteľná úloha špecifického prípadu definovaná ako **work item - pracovná položka**. Podľa tej istej literatúry nazývame **activity - aktivitou** vykonávanie pracovnej položky. Jednotlivé úlohy sú vykonávané zdrojmi, ako napríklad, počítače, alebo častejšie ľudia. Ak opäť ako príklad použijeme proces Žiadosť o pôžičku, jednotlivé úlohy by mohli byť napríklad Posúdenie žiadosti, Dohodnutie splátok, Podpísanie zmluvy a iné. Na úlohu Dohodnutie splátok môže byť naviazaná podmienka vyplnenia splátkového kalendára a jeho podpísanie žiadateľom.

1.1.3 Workflow rozmery

Na základe 1.1.1 , 1.1.2 , [1] a [9] má workflow 3 rozmery, a to:

- *Rozmer prípadu* - znamená, že každý prípad je riadený individuálne. Prípady sa navzájom neovplyvňujú.
- *Rozmer procesu* - smerovanie procesu je špecifikované.
- *Rozmer zdrojov* - zdroje spracúvajú čas-y, vykonávajú úlohy.

Ak má skupina zdrojov nejakú podmienku schopností a predpokladov pre jej členov, tak túto skupinu voláme **rola - role**. [1] Môžeme tak pre úlohy definovať, akú rolu má mať používateľ, ktorý ju má vykonať. Napríklad vo vyššie uvedenom príklade môžeme definovať, že Posúdenie žiadosti môže vykonať iba zamestnanec s rolou finančný poradca. Taktiež je možné uviesť, ktorá rola môže nový prípad vytvoriť.

1.2 Petriho siete

1.2.1 História

Pôvod Petriho sietí sa v [5] datuje v šesťdesiatych rokoch v dizertačnej práci "Communication with Automata" Carla Adama Petriho (*12.júl 1926 – † 2.júl 2010)[11]. Niektoré zdroje [6] uvádzajú, že Petri tieto siete vymyslel už v roku 1939 na modelovanie chemických procesov. Podľa [5], jeho skorá práca neobsahuje Petriho siete a ich definície tak, ako ich poznáme v dnešnej dobe. Doteraz existuje viacero definícií Petriho sietí.

1.2.2 Klasické Petriho siete

Podľa [2] klasické Petriho siete sú zvláštny druh orientovaného grafu. Tento graf pozostáva z 2 typov elementov, a to z **uzlov** a **hrán**.

Petriho sieť považujeme za graf **bipartitný** [1] [3], čo znamená, že uzly v týchto grafoch sú 2 typov, a to **miesto** - *place* a **prechod** - *transition*. Tieto uzly sú spojené pomocou **hrán** - *arcs*. Platí pravidlo, že 2 uzly toho istého typu nemôžu byť spojené hranou. [3] Ako sme spomínali v 1.2.1, definíci Petriho siete je viac, uvádzame definíciu z [7].

Definícia 1. *Petriho sieť je orientovaný bipartitný graf reprezentovaný päticou (P, T, F, W, m_0) , kde:*

- $P = \{p_1, \dots, p_n\}$ - konečná neprázdna množina miest;
- $T = \{t_1, \dots, t_n\}$ - konečná neprázdna množina prechodov ;
- $P \cap T = \emptyset, P \cup T \neq \emptyset$;
- $F \subseteq (P \times T) \cup (T \times P)$ - množina hrán (toková relácia);
- $W: F \rightarrow N \cup \{0\}$ - váhová funkcia;
- $m_0: P \rightarrow N \cup \{0\}$ - počiatkové značkovanie;

Váhová funkcia predstavuje množinu funkčných hodnôt váh všetkých hrán v Petriho sieti. Počiatkové značkovanie vyjadruje počiatkový stav Petriho siete, priradzuje každému miestu počet **značiek** - *tokenov*. [7] Ich počet v miestach sa počas spúšťania prechodov siete mení.

Miesta môžeme označovať ako vstupné a výstupné vzhľadom na jeden prechod. Ako vstupné ho označíme, ak hrana z neho vychádza a smeruje do daného prechodu. Naopak, z tohto prechodu hrana smeruje do miesta, ktoré označujeme výstupné.

1.2.3 Spustiteľnosť a spustenie prechodu

Prechod značíme **spustiteľný**, ak platí, že značkovanie každého vstupného miesta vzhľadom na daný prechod, je väčšie alebo rovné, ako váhová funkcia hrany, ktorá do skúmaného prechodu vchádza. Teda:

$$\forall p \in P : m(p) \geq I(p, t) \quad (1)$$

kde I je vstupná - *input* matica danej Petriho siete. [7]

Ak prechod spĺňa tieto podmienky, jeho spustenie môžeme rozdeliť do 2 krokov: skonzumovanie značiek a produkcia značiek. Prechod spustíme a ten z každého

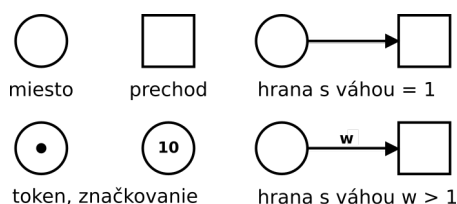
vstupného miesta *skonzumuje* taký počet značiek, aká je váha príslušných spojovacích hrán. Následne sa vyprodukovujú značky do výstupných miest, pre každé miesto toľko, akú váhu má spojovacia hrana. Nastala zmena značkovania:

$$\forall p \in P : m'(p) = m(p) + O(p, t) - I(p, t) \quad (2)$$

kde O je výstupná - *output* matica danej Petriho siete. [7]

1.2.4 Grafické znázornenie

Petriho siete sú grafickým jazykom. [1] Miesta sú reprezentované kružnicou, prechody štvorcom a hrany oblúkmi, pričom šípka hrany indikuje smer toku. Váhy hrán sa značia číslom na hrany, ak je váha hrany rovná jednej, váha sa neznačí. Značky sa v grafoch značia ako čierne body v mieste, prípadne pri vyšších hodnotách ich značíme číselne.[7]



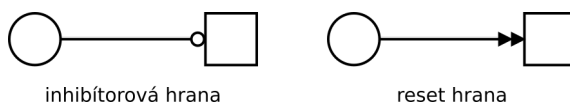
Obr. 1: Grafické značenie objektov Petriho sietí

1.2.5 Inhibítorové a reset hrany

V nasledujúcej podkapitole využívame informácie z [8]. V našom systéme využívame okrem normálnych hrán aj hrany inhibítorové a reset hrany. Jedná sa o jednoduché rozšírenie. Tieto hrany môžu byť použité na namodelovanie blokovania a vymazania. Obidva typy majú smer z miesta do prechodu, čiže sú to vstupné hrany.

Reset hrany umožňujú prechodu zkonsumovať všetky značky z miesta, z ktorého vychádzajú, čiže spustením prechodu v tomto mieste nezostane žiadna značka. V sieti ich zaznačujeme dvojitou šípkou pri prechode.

Inhibítorové hrany povoľujú spustenie prechodu len v tom prípade, ak miesto z ktorého vychádza inhibítorová hrana obsahuje menší počet značiek, ako je váha tejto hrany. Graficky sa tieto hrany neukončujú na prechode šípkou, ale malou kružnicou.



Obr. 2: Grafické značenie inhibítor a reset hrán

1.3 WFMS založené na Petriho sieťach

V minulosti, softvérové systémy boli navrhnuté tak, aby kontrolovali jeden proces, prípadne dokonca iba jednu úlohu. Neboli generické, každá zmena procesu vyžadovala implementáciu. Petriho siete sú silný nástroj na špecifikovanie smerovania prípadov procesu. Sú grafické, intuitívne, ľahko naučiteľné. Mapovanie procesu do Petriho siete je zrejmé - úlohy sú namodelované pomocou prechodov a závislosti sa modelujú pomocou miest, hrán. Prípady sú reprezentované značkami. Určuje dynamické správanie pre každý prípad z tohto procesu.[9] Petriho sieť, ktorá modeluje workflow proces sa nazýva WorkFlow sieť(d ďalej WF-sieť). [1]. WF siete majú 3 podmienky:

1. Jedno zdrojové miesto (i) a jedno konečné miesto (o).Značka v i zodpovedá začiatku prípadu a značka v o prípadu, ktorý už nemá úlohy na splnenie, jeho prechody už nie sú spustiteľné a prípad je ukončený.
2. Každá úloha - prechod a podmienka - miesto prispievajú k spracovaniu prípadov, čiže všetky prechody a miesta by mali byť umiestnené na ceste z miesta i do miesta o [1]
3. Ku konečnému miestu nie je pripojená žiadna reset hrana.

Definícia podľa [10] znie:

Definícia 1. Petriho sieť môžeme nazývať **Workflow-sieťou** vtedy a len vtedy, ak:

- V sieti je len jediné **zdrojové** miesto

$$\{ p \in P \mid \bullet p = \emptyset \} = \{i\}$$

a jediné **konečné** miesto

$$\{ p \in P \mid p\bullet = \emptyset \} = \{o\}$$

- Každý uzol siete je dostupný na ceste od i ku o , teda

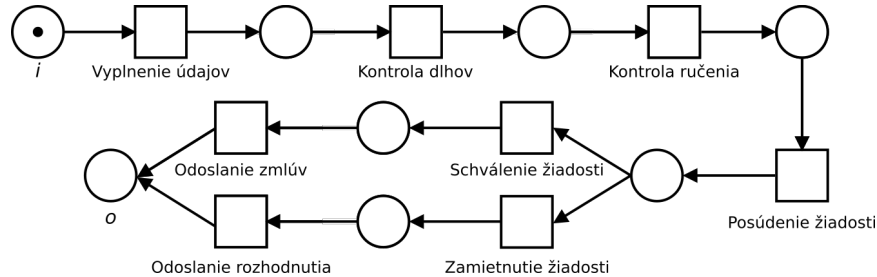
$$\forall n \in P \cup T: (i, n) \in F^* \text{ a } (n, o) \in F^*$$

- V sieti nie je žiadna reset hrana pripojená ku konečnému miestu

$$\forall_{t \in T} o \notin R(t)$$

Ako $\bullet p$ označujeme množinu prechodov zdieľajúcich miesto p ako výstupné miesto a $p \bullet$ množinu prechodov, ktoré zdieľajú p ako vstupné miesto.

Príklad spomínaný v kapitole 1.1 Žiadosť o pôžičku zjednodušene mohol vyzeráť nasledovne:

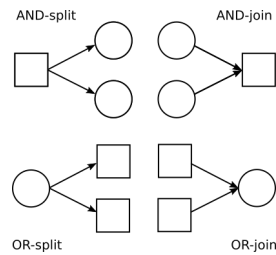


Obr. 3: Príklad workflow procesu pomocou Petriho siete - Žiadosť o pôžičku

V tejto sieti považujeme za zdrojové miesto (i) to, ktoré jediné má značku a predchádza prechodu *Vyplnenie údajov*. Je jasne definovaná postupnosť úloh, ktoré sa majú plniť. Úlohy *Vyplnenie údajov*, *Kontrola dlhov*, *Kontrola ručenia*, *Posúdenie žiadosti*, *Schválenie žiadosti*, *Zamietnutie žiadosti*, *Odoslanie zmlúv*, *Odoslanie rozhodnutia* boli namodelované pomocou prechodov. Vďaka zadefinovanej postupnosti a počiatočnému značkovaniu sa tieto úlohy nemôžu vykonávať v inom poradí, ako je definované. Po *Posúdení žiadosti* nemôžu nastať obidva prípady - aj *Schválenie pôžičky* aj *Zamietnutie pôžičky*. Predpokladajme, že v tejto sieti je pri týchto oboch prechodoch referencia na prechod *Posúdenie žiadosti*, čo znamená, že tieto úlohy môže vykonávať len používateľ, ktorý žiadosť posúdil a vďaka značkovaniu sa môže rozhodnúť len pre jednu z týchto úloh. Aj keby sme so spomínanou referenciou nepočítali, nemôže sa stať, aby jeden používateľ žiadosť schválil a iný používateľ ju zamietol, pretože v mieste nasledujúcom po prechode *Posúdenie žiadosti* nemôže nastať prípad, aby tam boli dva značky. Za konečné miesto (o) považujeme miesto, do ktorého je vyprodukovaná značka po *Odoslaní rozhodnutia* pri neschválení pôžičky, prípadne po *Odoslaní zmlúv* po jej schválení. V momente, keď je v tomto mieste značka, v sieti už nie je žiaden prechod spustiteľný, prípad považujeme za ukončený.

1.3.1 Modelovanie smerovania

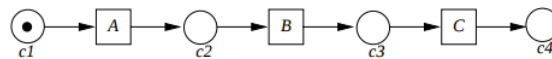
V tejto podkapitole využívame vedomosti z [1] a [9]. Na nasledovnom obrázku vidíme stavebné bloky, pomocou ktorých modelujeme logiku a podmienky WF procesov.



Obr. 4: Stavebné bloky modelovania workflow

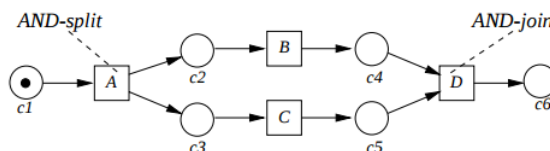
Poznáme 4 typy smerovania

- *Sekvenčné* - rieši kauzálne vzťahy medzi úlohami, úlohy sa vykonávajú sekvenčne za sebou.



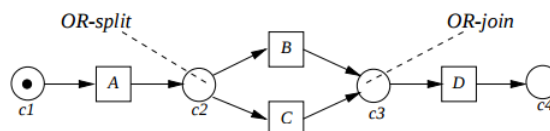
Obr. 5: Sekvenčné smerovanie[1]

- *Paralelné* - využíva sa, keď sa niektoré úlohy môžu riešiť paralelne, prípadne na poradí ich riešenia nezáleží. Na modelovanie paralelity sa používajú bloky AND-split a AND-join.



Obr. 6: Paralelné smerovanie[1]

- *Podmienkové* - umožňuje rozhodovanie, smerovanie, v ktorom sa prípady môžu líšiť. Je závislé na atribútoch prípadu. Toto smerovanie sa realizuje pomocou blokov OR-split a OR-join



Obr. 7: Podmienkové smerovanie[1]

- *Iteračné* - pomáha riešiť situácie, ak prechod má byť spustený viackrát

1.3.2 Analýza modelu procesu

Aj napriek intuitívnosti a nenáročnosti Petriho sietí môžu nastať pri modelácii procesu viac typov chýb, ktoré by negatívne ovplyvňovali WFMS, a to napríklad, nekonečné cykly, deadlocky a iné. Preto je potrebné analyzovať navrhnutú Petriho sieť aby sa týmto chybám predchádzalo. V [1] W.M.P. van der Aalst definuje 3 typy analýzy sietí, a to:

- *Validácia* - testovanie, či sa proces správa tak, ako je očakávané. Toto sa skúma interaktívnou simuláciou, vytvorí sa nejaký počet testovacích prípadov a sleduje sa, ako ich systém zvláda
- *Verifikácia* - zisťovanie korektnosti siete
- *Analýza výkonu* - určenie, či model spĺňa požiadavky (napr. čo sa týka zdrojov)

1.3.3 ??Spôľahlivosť?? - Soundness

Okrem 3 podmienok Workflow-siete spomenutých v predchádzajúcej kapitole by podľa [10] a [1] mali spĺňať ešte nasledujúce 2 podmienky:

1. Každý prípad by mal skončiť hneď, ako je značka v konečnom mieste o a všetky ostatné miesta sú prázdne - neobsahujú žiadna značka.
2. V procese by nemali byť žiadne “mŕtve” úlohy, teda by malo byť možné vykonať ľubovoľnú úlohu nasledovaním vhodnej cesty vo WF-sieti.

Tieto 2 vlastnosti sú zhrnuté vo vlastnosti Spôľahlivosť, ktorej definícia podľa [1]

Definícia 1. *Postup namodelovaný pomocou WF-siete je **spôľahlivý** vtedy a len vtedy ak:*

- *Pre každý stav M dosiahnuteľný zo stavu i existuje taká postupnosť spúšťania prechodov, ktorá vedie zo stavu M do stavu o :*

$$\forall_M (i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} o)$$

- *Stav o je jediný stav dosiahnuteľný zo stavu i s minimálne 1 značkou v mieste o :*

$$\forall_M (i \xrightarrow{*} M \wedge M \geq o) \Rightarrow (M = o)$$

- *Nie sú tam žiadne mŕtve prechody:*

$$\forall_{t \in T} \exists_{M, M'} i \xrightarrow{*} M \xrightarrow{t} M'$$

1.4 Použité technológie

1.4.1 HTML, HTML5

Nasledujúca kapitola je písaná na základe informácii z [12], [13] a [14]. **HTML** je značkovací jazyk pre písanie webových dokumentov, ktorého zmyslom je umožniť zobrazenie textového a obrazového obsahu stránok. Táto skratka pochádza zo spojenia Hyper Text Markup Language. Tieto dokumenty sa píšú do definovaných značiek - tagov (napr. `<p>Toto je odstavec</p>`) a majú koncovku `.html`. HTML nevytvára vzhľad, čiže nehovorí prehliadaču, nič o veľkosti, farbe a iných vlastnostiach, iba pomocou značiek definuje, čo je nadpis, odkaz, odstavec, obrázok, formulár a iné. Značky sú väčšinou párové, no existujú aj nepárové, ako napríklad značka obrázku ``, vstupného poľa `<input />` alebo koniec riadku `
`.

HTML5 je momentálne najnovší štandard HTML predstavený v roku 2014. Okrem iného, priniesla sadu nových značiek, a to značky pre *header*, *footer*, *article* - článok, *section* - sekciu pre grafické a multimediálne elementy ako *audio*, *video*, *svg* a *canvas*. Taktiež predstavila nové typy formulárových vstupov a to číslo, dátum, čas, kalendár a rozsah. Niektoré zastaralé značky nahradila pomocou CSS vlastností, napr. značky `` a `<center>`.

HTML, rovnako ako ďalej spomenuté CSS a JavaScript sa vykonávajú na tzv. frontende, čiže v prehliadači na strane klienta, nie na strane servera.

HTML dokument sa vizuálne upravuje pomocou **CSS**, čo je skratka pre *Cascading Style Sheets*, čiže kaskádové štýly. Tento jazyk nie je značkovací, ale štýlovací. Podľa [15] existujú 3 spôsoby zápisu CSS, a to:

1. *Inline* - zápis priamo do HTML elementov, napr.
`<p style="font-weight: bold;">Odstavec</p>`
2. Použitím `<style>` elementu vnoreného v `<head>` značke
3. Vytvorenie samostatného `.css` súboru

Vďaka CSS môžeme HTML elementom nastavovať veľkosť, pozíciu, farbu, zarovnanie, ohraničenie a mnoho iných vlastností, ktoré robia HTML dokumenty atraktívnejšími pre používateľa. Aj responzivita pre rôzne veľkosti zariadení sa rieši pomocou využitia kaskádových štýlov. Najnovšia verzia je momentálne CSS3.

V [16] sa uvádza, že **JavaScript** je programovací jazyk vytvorený pre potrebu písania skriptov ovplyvňujúcich HTML dokumenty, čiže je to skriptovací jazyk. Javasc-

ript robí HTML stránky interaktívnejšími a dynamickejšími. Môžeme ho zapisovať 2 spôsobmi:

1. Do *párovej značky* `<script>` priamo v HTML dokumente
2. Do *samostatného .js súboru*

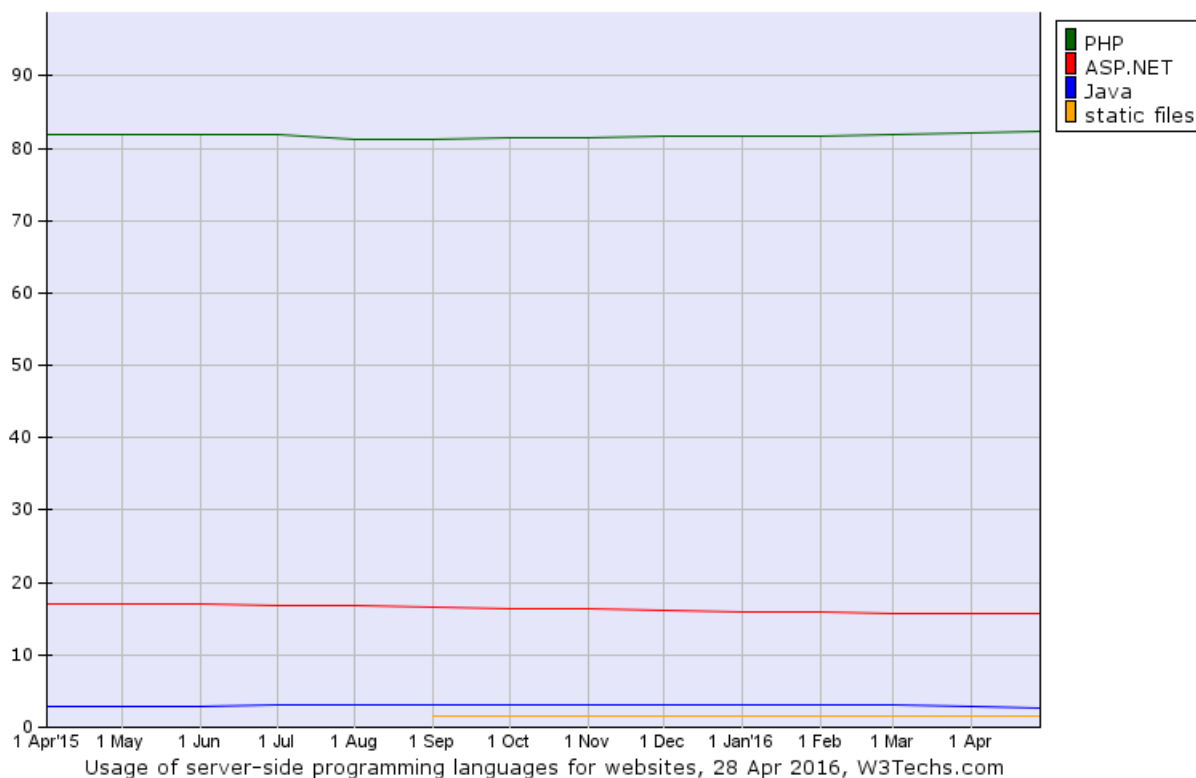
Pomocou javascriptu sa môže vykonávať validácia vstupných hodnôt od používateľa, dynamicky meniť CSS štýly, pridávať a odoberať HTML elementy, hýbať s elementami a viac iných akcií. V prehliadačoch je možné JavaScript zakázať. V tomto prípade sa dá použiť značka `<noscript>`, ktorá definuje alternatívny obsah, ak strana klienta nepodporuje skriptovanie pomocou tohto jazyka.

1.4.2 PHP

Informácie v tejto kapitole čerpáme z [17] a [26]. **PHP** je skriptovací jazyk vykonávaný na strane servera - backende využívaný pre tvorbu klient-server aplikácií. PHP je rekurzívna skratka z *PHP: Hypertext Preprocessor*. Do PHP súborov môžeme písať okrem HTML kódu aj kódy HTML, CSS a JS a majú koncovku `.php`. v `php` súbore píšeme HTML priamo ako v `.html` súbore, CSS píšeme v rovnakom formáte, ako bolo spomínané v predchádzajúcej podkapitole a PHP kód vkladame do súboru do značiek `<?php /*Tu sa píše kód*/ ?>`. Môžu sa použiť aj značky `<? ... ?>` ale je potrebné to nastaviť na serveri.

Pomocou PHP sa môžu vykonávať náročnejšie výpočty na strane servera, ktoré sa neodporúčajú robiť v Javascripte v prehliadači klienta. PHP dynamicky generuje HTML obsah, ktorý sa zobrazí v prehliadači. Okrem toho, PHP umožňuje prácu so súbormi, čiže ich otváranie, čítanie, upravovanie, a mazanie. PHP vďaka komunikácii s HTML dokumentmi a akciám POST a GET dokáže získavať dáta od používateľov z formulárov a tie môže spracovávať. Okrem iného, umožňuje pripojenie k databáze a odosiela na ňu požiadavky, ako vyberanie dát, upravovanie, vymazávanie a tiež zmeny tabuliek. Využíva sa aj šifrovanie dát, teda zabezpečuje bezpečnosť stránky.

PHP sa v dnešnej dobe teší veľkej obľube[24] (Vid' obrázok 8. Na obrázku sú zobrazené programovacie jazyky s percentom využitia viac, ako 1%). Jedným z dôvodov môže byť, že je zadarmo stiahnuteľný aj použiteľný. Taktiež je podporovaný najvyužívanejšími platformami - Windows, Linux, Unix, Mac OS X[23]. Podporuje viac typov databáz ako napríklad MySQL, PostgreSQL, Oracle alebo MS SQL[26]. Je kompatibilný s väčšinou serverov, ktoré sa dnes používajú[17]. Umožňuje objektovo orientované programovanie.



Obr. 8: Využitie programovacích jazykov na serverovej strane pre webové stránky ku 25.4.2016[24]

1.4.3 SQL, MySQL

Na základe informácií z [18], **SQL** je definovaný ako štandardný jazyk pre prístup a manipuláciu s databázou. SQL znamená skratene *Structured Query Language*. Je to ANSI(*American National Standards Institute*) štandard. Vďaka nemu vykonávame požiadavky na databázu pre získavanie, vkladanie, mazanie a upravovanie dát. Prostredníctvom SQL môžeme vytvárať aj tabuľky, meniť ich, vymazávať, a. i.. Tento jazyk umožňuje nastavovať práva prístupu používateľov k dátam.

MySQL je voľne dostupný relačný databázový softvér, ktorý podporuje používanie SQL na prístup k dátam a ich spracovanie. [25]

1.4.4 AJAX

Technológia **AJAX** - *Asynchronous JavaScript and XML* umožňuje asynchrónnu komunikáciu a výmenu dát so serverom bez obnovenia stránky. Bez použitia AJAX-u, sk chce webová aplikácia komunikovať so serverom, musí zobrazíť novú stránku, prípadne opätovne načítať aktuálnu. [26] V [19] uvádza internetové štandardy,

na ktorých je AJAX založený:

- XMLHttpRequest - asynchrónna výmena dát so serverom
- JavaScript/DOM - zobrazenie alebo manipulácia s informáciou
- CSS - naštýľovanie dát
- XML - často využívané na výmenu dát

1.4.5 XML, SVG

Podľa informácií z [21] a [26] **XML** je skratka pre *Extensible Markup Language* a bolo navrhnuté tak, aby sa ľahko čítalo aj človekom, aj počítačom. Slúži na ukladanie a prenos dát. Ako HTML, tak aj XML považujeme za značkovací jazyk, pretože informácie, dáta sa vpisujú do značiek, avšak XML nepoužíva vopred definované značky, tie definuje autor dokumentu.

SVG - *Scalable Vector Graphics* definuje vektorovú grafiku prostredníctvom XML formátu. Jeho hlavnými výhodami sú tie, že nestráca kvalitu približovaním obrázka a je editovateľný v textových editoroch.[22]

1.4.6 JSON

JSON je skratka výrazu *JavaScript Object Notation*. Je to formát ukladania a výmeny dát. Nie je závislý na programovacom jazyku, pretože je to txt a je podobného formátu, ako XML. Funkcie JavaScriptu aj PHP umožňujú konverziu JSON dát na objekty a naopak. Dáta v ňom môžu byť ukladané hierarchicky. Často sa využíva namiesto XML, pretože nepoužíva tagy, je kratší, rýchlejší na čítanie a zápis. Všetky informácie v tomto odseku pochádzajú z [20].

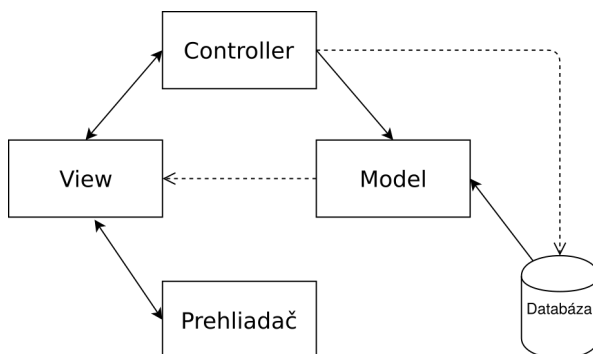
1.5 Použité návrhové vzory

Táto kapitola je písaná na základe vedomostí z [27] a [28].

1.5.1 Model-View-Controller (MVC)

Model-View-Controller je štruktúrálny návrhový vzor, ktorý oddeľuje logiku aplikácie od používateľského rozhrania. Používateľské rozhranie - *view* komunikuje s dátami - *model* prostredníctvom *controllera*. *Controller* - *ovládač* obsahuje biznis logiku aplikácie, manipuluje dáta - prijíma ich a odosiela do modelu. Teda *model* reprezentuje

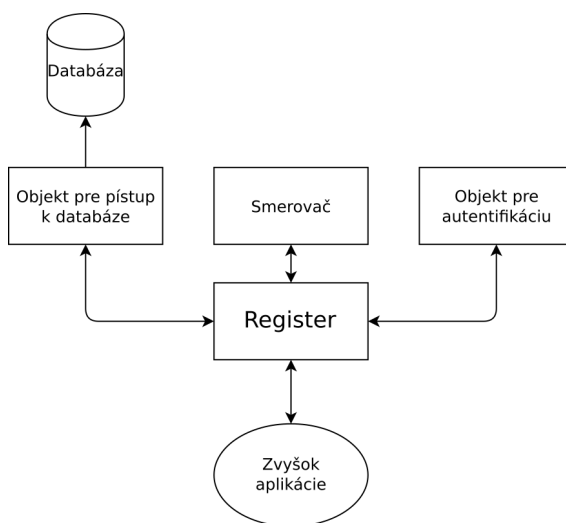
dáta, ktoré sú primárne uložené v databáze. Na obrázku 9 vidíme komunikáciu medzi modelom a databázou. *View* - *náhľad* zobrazuje obsah stránky v internetovom prehliadači a odosiela vstupné dáta do *ovládača*, ktorý ich na základe naimplementovanej logiky spracováva. Pri špecifickejších zložitejších požiadavkách na databázu, môže tieto požiadavky spracovať *ovládač*.



Obr. 9: Model-View-Controller

1.5.2 Registry - Register

Návrhový vzor **register** je taktiež štrukturálny vzor. Jeho hlavnou ideou je uchovávanie objektov aplikácie. Potreba *registra* rastie pri architektúre MVC, kde je potrebné zdieľať viacero objektov medzi rôznymi *modelmi* a *controllermi*, ako napríklad objekt na prístup k databáze, objekt autentifikácie, a.i.. *Register* nám umožňuje uchovávať tieto objekty na jednom mieste, je prístupným bodom ku potrebným objektom.



Obr. 10: Register

1.5.3 Singleton

Singleton radíme medzi vytvárajúce návrhové vzory. Rieši potrebu toho, aby v celom behu sa vytvorila len jediná inštancia triedy a poskytuje k nej globálny prístupový bod. Konštruktor tejto triedy je privátny, verejná je prístupová metóda, ktorá zabezpečuje to, aby sa z nej nevytvorilo viac, ako jeden objekt.

2 Opis riešenia - WFMS

Naším cieľom bolo naimplementovať komplexný Workflow manažment systém vo forme portálu, ktorý by umožňoval namodelovanie procesov pomocou Petriho sietí, priradenie rolí, referencií a formulárov úlohám. Umožní vytvárať prípad z namodelovaného procesu a na základe Workflow-siete riadi workflow prípadov. Taktiež je možná organizácia používateľov vo firmách a v rolách. Na prechod sa môže viazať formulár, ktorý treba vyplniť pre ukončenie úlohy. Náš systém sme rozdelili na 7 častí, ktoré implementovali 7 členovia tímu v rámci bakalárskeho projektu.

2.1 Procesný portál

Procesný portál vypracoval Martin Kováčik. Portál umožňuje registráciu a prihlasovanie používateľov do nášho systému. Prihlásený si môže manažovať svoj účet, meniť heslo a v prípade straty hesla požiadať o jeho reset a zaslanie nového hesla na registrovaný email. Používateľ má umožnené vytvoriť si vlastnú organizáciu - firmu a spravovať ju, ako aj používateľov v nej - pridávať, odstraňovať, priradovať im práva pre manažment firmy. Ak firma nemá nastavené heslo, používatelia sa do nej môžu voľne pridávať, do iných pre pridanie je potrebné heslo. Keďže náš systém by mal fungovať ako verejný portál, namodelované Workflow-siete sú verejné, avšak nie voľne použiteľné. Workflow-siete si používateľ bude môcť pre svoju firmu nakopírovať (keďže systém je robený v rámci bakalárskej práce, používatelia si zatiaľ procesy nekupujú, ale kopírujú) aj s jej namodelovanými vlastnosťami, čiže s rolami a formulármi. Úlohou Martina Kováčika bolo teda aj vyriešiť toto kopírovanie procesov do firiem. Tým pádom, ak firma používateľa by chcela používať rovnaký proces, ako iná firma s podobným zameraním a už funkčne namodelovanou Workflow-sieťou želaného workflow procesu, tak by si túto firmu mohol nakopírovať do svojej firmy a začať ju využívať. Táto časť funguje podobne, ako e-shop, teda WF-sieť si používateľ objednáva.

2.2 Procesný editor

Procesný editor riešil Tomáš Priboj. Cieľom bolo naimplementovať editor na strane klienta, pomocou ktorého by sa modelovali Petriho siete. V editore je možné modelovať vo forme svg. Všetkými namodelovanými objektami je možné pohybovať. Ak je k miestu alebo prechodu pripojená hrana, tak pri pohybovaní ostáva pripojená a hýbe sa spolu s miestom alebo prechodom. Editor umožňuje simuláciu behu siete spúšťaním

prechodov, tým pádom sa dá skontrolovať, či Workflow-sieť je správne namodelovaná, ako bolo špecifikované. Prechodom sa dávajú označenia, ktoré reprezentujú názov úlohy v prípade. Model je možné uložiť na lokálny disk ako SVG alebo XML súbor. Sieť je taktiež možné z XML načítať. Pre potreby workflow, používateľ sieti priradí popis a výmenou dát vo formáte JSON, SVG a XML službou naimplementovanou na strane Procesného servera(vid' kapitolu 3) žiada o uloženie do databázy. Po úspešnom uložení prijme id, ktoré jej bolo v databáze priradené, a v prípade integrácie presmeruje na Editor formulárov a dátového modelu. Používateľ si z databázy môže načítať siete, ktoré tam už uložil.

2.3 Editor manažmentu rolí a organizačnej štruktúry

Časť Editor manažmentu rolí a organizačnej štruktúry sa venoval Kristián Stroka. Tento editor umožňuje ... TODO

2.4 Editor formulárov a dátového modelu

Editor formulárov a dátového modelu vytvoril Ján Poláček. Vďaka tomuto editor, aplikácia poskytuje možnosť navrhovať dátový model pre formuláre, ktoré sa budú vyplňať počas plnenia úloh prípadov. Editor načíta svg súbor navrhutej Workflow-siete a navrhnuté formuláre sa mapujú na prechody reprezentujúce úlohy. Formulár je možné vytvoriť, odstrániť, upraviť a importnúť ich do XML a SVG súborov. Naopak, formuláre je možné aj načítať zo súborov. Návrh formulárov spočíva v tvorbe, ako aj editácii, prípadne mazaní vstupných polí. Vstupné polia môžu byť typu krátka alebo dlhá odpoveď, *select* - rozklikávací zoznam, alebo *checkboxy* - zaklikávacie políčka. Pri každom vstupe sa dá určiť, či je povinný, viditeľný alebo upraviteľný. Dá sa im pridať popis, ktorý špecifikuje, čo dané vstupné pole reprezentuje, prípadne vyžaduje. Rozklikávací zoznam a tiež zaklikávacie políčka sa môžu zoradiť v abecednom poradí. Vytvorí dataset, čo sú premenné, do ktorých sa ukladajú vstupné hodnoty z polí. Tento editor je integrovaný s časťou Server formulárov a dátového modelu a to spočíva v odosielaní a prijímaní dát vo forme JSON.

2.5 Procesný server

Časť Procesný server je témou tejto bakalárskej práce, čiže jej riešeniu a implementácii venujeme celú kapitolu 3.

2.6 Server manažmentu rolí a používateľov

Server manažmentu rolí a používateľov mal na starosti Pavol Martiš. Rieši organizačnú štruktúru používateľov, čo sa rolí týka. Umožňuje zobrazíť všetkých používateľov firmy a zoraďovať ich podľa rôznych atribútov, ako napríklad meno, priezvisko, e-mailu, dokonca podľa id. Dôležitou funkciou tejto aplikácie je priradenie používateľov k roliam v rámci firmy, čo je možné individuálne, alebo skupinovo. V rámci firmy je umožnené vytvárať nové role, alebo je možné importnúť ich z XML súboru. Role, na ktoré nie sú naviazané žiadne procesy firmy, je možné z tejto firmy odstrániť, aby bolo možné udržiavať prehľadnosť medzi rolami a neuchovávať všetky staršie, nepoužívané role. V prípade vyžiadania z Editoru manažmentu rolí a organizačnej štruktúry, pripraví JSON so všetkými rolami žiadanej firmy. Komunikácia s týmto editorom taktiež spočíva v tom, že pri novovytvorenej Workflow-sieti, ukladá namapované role na prechody a referencie prechodov do databázy s tým, že aj v databáze to namapuje tak, aby to zodpovedalo predlohe.

The screenshot displays the 'workflowmarket' role management interface. On the left, a sidebar lists actions: 'Create process', 'Edit processes', 'Delete processes', 'Role management' (highlighted), 'User Management', and 'Delete firm'. The main area shows a table of users with the following data:

Check all	Id	First name	Last name	E-mail
<input type="checkbox"/>	1	Pavol	Martiš	pvl.martis@gmail.com
<input type="checkbox"/>	2	Ábel	Hraško	hrasko@gmail.com
<input type="checkbox"/>	3	Vierofub	Vlk	vlk@is.stuba.sk
<input type="checkbox"/>	4	Vojeslava	kollárova	kollarova@gmail.com
<input type="checkbox"/>	5	Trofin	Jeník	jenik@yahoo.com
<input type="checkbox"/>	6	Slavoľub	Slávik	slavik@gmail.com
<input type="checkbox"/>	7	Sergius	Kocka	kocka@stvorec.sk
<input type="checkbox"/>	8	Sabrina	Lacková	lackova@centrum.sk
<input type="checkbox"/>	9	Rozália	Zúbková	zubkova@gmail.com

Below the table, there are buttons for 'add to role', 'Role1', and 'Do it!'. The interface also includes a search bar and filters for 'General filters: All | Without role' and 'Roles filters: Role1 | Role2'.

Obr. 11: Server manažmentu rolí a používateľov

2.7 Server formulárov a dátového modelu

Server formulárov a dátového modelu je časť naprogramovaná Jakubom Kováčikom. Táto aplikácia reaguje, keď používateľ cez Editor formulárov chce uložiť namapované formuláre. Tieto formuláre a XML súbor sa uložia do databázy, prípadne

opačným smerom, Editor si môže vyžiadať proces s formulármi a prislúchajúce XML. V prípade vytvorenia procesu, vytvorí sa klon formulárov, ktorých zobrazenie a vyplňanie má je jednou z funkcií tejto aplikácie. Takýto formulár už nie je možné upravovať v editore. Ak pomocou akcie GET prijme id prípadu a prechodu, zobrazí prislúchajúci formulár. Používateľ má možnosť prácu s formulárom uložiť, aj napriek tomu, aké nie je ešte úplne a validne vyplnený. Pri uložení sa totiž validácia nevykonáva, a používateľ nestratí svoju doterajšiu prácu. Ak používateľ formulár *odošle*, validácia sa vykoná a uloží sa do databázy. V prípade kompletnej integrácie formulár musí byť pre ukončenie úlohy zvalidovaný.

The image shows a web form titled "Údaje zákazníka" (Customer Data) with a close button (X) in the top right corner. The form contains the following fields and controls:

- Meno *** (Name): A text input field containing the value "Ladislav".
- Priezvisko *** (Surname): An empty text input field.
- Pohlavie *** (Gender): A dropdown menu with the selected option "Muž" (Male).
- Domáce zvieratká *** (Pets): A section titled "Zvoľte Vaše domáce zvieratká" (Choose your pets) with three radio button options: "Mačka" (Cat), "Pes" (Dog), and "Iné" (Other).
- Buttons:** Two buttons at the bottom: "Ulož formulár" (Save form) in a pink box and "Odošli formulár" (Submit form) in a blue box.

Obr. 12: Server formulárov a dátového modelu

3 Opis riešenia - Procesný server

Témou tejto bakalárskej práce je časť WFMS - procesný server. Aplikácia spracuje požiadavku Procesného editora pre uloženie Workflow-siete do databázy. Jeho hlavnou úlohou je umožniť vytváranie prípadov a ich riadenie podľa modelu Workflow-siete. Bolo potrebné zabrániť kolíziám. Systém taktiež umožňuje uchovávanie histórie a zobrazuje vizualizáciu prípadov. Rieši vrátenie úlohy pomocou *cancel* tlačidla. Všetky

funkcionality aplikácie podrobne opíšeme v tejto kapitole.

3.1 Výber technológií

Pre implementáciu aplikácie sme sa rozhodli použiť programovací jazyk PHP, kvôli jeho voľnej dostupnosti. Jedným z dôvodov výberu tohoto jazyka bola aj jeho široká podpora na serveroch. Kvôli týmto dôvodom sme sa s ostatnými členmi tímu dohodli na použití PHP aj napriek tomu, že nikto z nás nemal ohľadom neho nemal profesionálnejšie vedomosti.

Keďže sme chceli zjednotiť vstupy, výstupy, dáta jednotlivých aplikácií, museli sme sa dohodnúť aj na type databázového servera a vybrali sme MySQL. Tento server podporuje PHP jazyk a je to relačný server. Tým pádom nám umožňuje mapovať vzťahy medzi databázovými tabuľkami a taktiež je voľne dostupný. Pracuje na rôznych platformách, čo je pre náš tím dôležité, pretože nepracujeme na jednotnej platforme. Teda, MySQL spĺňalo všetky naše podmienky.

Komunikáciu medzi *frontendom* a *backendom* zväčša vykonávame asynchrónne, pomocou technológie AJAX, aby nebolo potrebné presmerovanie, prípadne opätovné načítanie stránky.

Pre výmenu dát medzi editorovými a serverovými časťami sme si vybrali JSON, pretože ako bolo spomenuté v kapitole 1.4.6, JSON je rýchlo čitateľný, je asociatívny a dá sa funkciami PHP a JavaScript prekonvertovať na objekt. Dáta ukladáme aj vo forme SVG, aby boli graficky čitateľné a vo forme XML, aby sa dáta dali exportovať na lokálny disk a boli jednoducho čitateľné človekom.

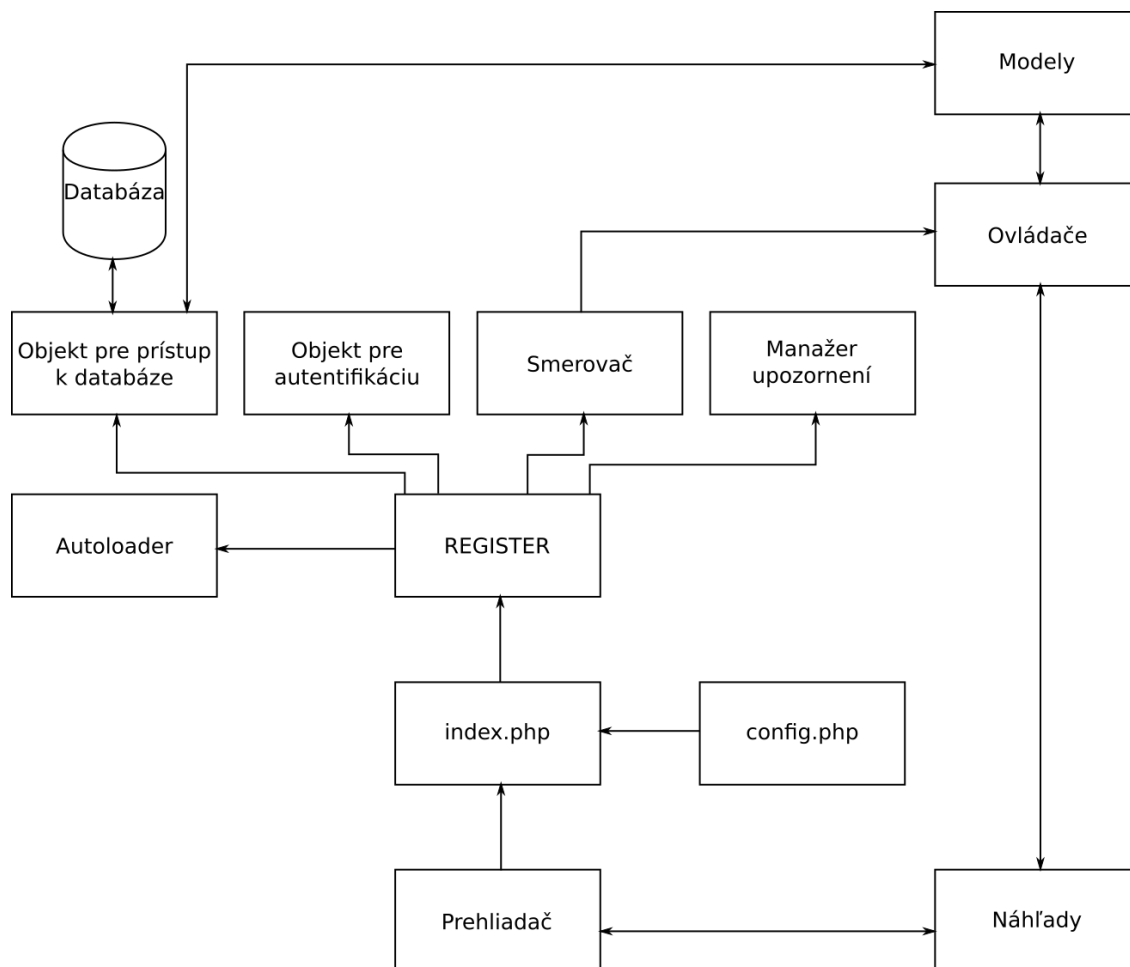
Používateľské rozhranie je riešené pomocou HTML5. Pri výbere technológií, HTML5 poznali všetci členovia tímu a mali s ňou programátorské skúsenosti. Je podporovaný všetkými prehliadačmi a v dnešnej dobe je to aktuálny HTML štandard.

3.2 Výber návrhových vzorov, návrh systému

Pri tomto výbere sme sa inšpirovali literatúrou [27]. Po naštudovaní návrhového vzoru *Model-View-Controller*, ktorému sme venovali kapitolu 1.5.1 sme usúdili, že pomocou neho môžeme oddeliť biznis logiku aplikácie od používateľského rozhrania. Vďaka vzoru *Register* budeme uchovávať dôležité objekty, ako prístup k databáze, autentifikácia a smerovač na jednom mieste a budú prístupné zo všetkých miest systému, kde to bude potrebné. V našom *Registri* sme využili aj vzor *Singleton*, vďaka ktorému

je zabezpečené to, že budeme mať len jednu inštanciu triedy *Register*.

Graficky, štruktúra aplikácie s tokom výmeny dát je zobrazená na obrázku 14



Obr. 13: Štruktúra Procesného servera

3.3 Jadro systému

Cieľom bolo vytvoriť samostatne bežiacu aplikáciu Procesný server. Keďže sme sa rozhodli implementovať aplikáciu bez použitia *frameworku*, aby bolo čo najmenšie zaťaženie serverovej strany, bolo potrebné naprogramovať jadro, na ktorom je aplikácia postavená. Ako spomíname v predchádzajúcej kapitole, držali sme sa 3 návrhových vzorov, na základe ktorých sme rozvrhli štruktúru systému. Naše jadro pozostáva z nasledujúcich častí:

- Register
- Smerovač
- Objekt pre prístup k databáze

- Objekt pre autentifikáciu
- Manažér upozornení
- Autoloader

Každú z týchto častí si priblížime v nasledujúcich kapitolách.

3.3.1 Register

Register je v našej aplikácii reprezentovaný triedou *Flow* v súbore *Flow.php*. Keďže cieľom bolo, aby počas behu aplikácie sa nevytvorilo viac ako jedna inštancia tejto triedy, tak konštruktor tejto triedy je privátny, rovnako ako metóda *clone*. *Index.php* zavolá *Singleton* metódu *Registra*:

```
public static function app(){
    if(empty(self::$instance)){
        /*Vytvorí novú inštanciu registra a vráti ju*/
    }
    else{
        /*Vráti už skôr vytvorený register, nevytvára nový*/
    }
}
```

Na základe vrátenej inštancie naštartuje aplikáciu, pričom sa funkcia *start()* načíta konfiguračné nastavenia a uložia sa nasledujúce objekty, a to *Autoloader*, *Objekt pre autentifikáciu*, *Objekt pre prácu s databázou* a *Smerovač*, ktoré budú dostupné prostredníctvom *Registra* počas behu aplikácie. Taktiež sa tu skontroluje, či je v *SESSION* používateľ prihlásený pomocou Objektu pre autentifikáciu. V tejto triede je okrem iného naimplementovaná metóda *getConfig(\$key)*, ktorá vracia konfiguračné nastavenie podľa názvu ako vstupného parametra.

3.3.2 Smerovač

Smerovaču odpovedá trieda *Router*. Jeho hlavnou funkciou je spracovať vstupnú URL adresu, ktorá je definovaná v tvare:

www.nazovdomeny.sk/index.php/nazovovladaca/nazovfunkcie/parametre

Napríklad URL:

www.nazovdomeny.sk/index.php/case/create/vstupnyparamater

zavolá *CaseController.php* a jeho funkciu *create(\$str)*, kde *\$str* = 'vstupnyparameter'. Viacero parametrov sa oddeľuje lomkou /.

Objekt smerovača sa vytvára a uchováva v Registri.

3.3.3 Objekt pre prístup k databáze

Tento objekt je inštancia triedy *PdoConnection* a táto trieda dedí od PHP triedy PDO. Zabezpečuje pripojenie k databáze na základe informácií z konfiguračného súboru získaných prostredníctvom Registra. Má funkcie pre požiadavky *INSERT*, *UPDATE* a *DELETE*, ktoré na základe vstupných parametrov žiadanú požiadavku poskladajú do reťazca znakov a vykonajú ju v databáze.

3.3.4 Objekt pre autentifikáciu

Kvôli testovaniu aplikácie bola potreba naprogramovať funkcionality pre autentifikáciu, aj napriek tomu, že táto časť je súčasťou Procesného portálu. Má naimplementované metódy, ktoré overujú, zaisťujú prihlásenie do systému. Na testovanie boli potrební minimálne 2 používatelia. Do systému sa dá prihlásiť 3 účtami. Pri prípadnej integrácii bude táto časť nahradená metódami, ktoré implementuje Procesný portál.

3.3.5 Manažér upozornení

Objekt triedy *Alertmanager* uľahčuje správu a zobrazovanie upozornení pre používateľa. Upozornenia nie sú zobrazované priamo, ale sú ukladané do *SESSION*, aby ich bolo možné zobraziť v prípade presmerovania, respektíve AJAX volania.

3.3.6 Autoloader

Autoloader je trieda PSR-4 štandardu [29]. Slúži na pripájanie súborov s PHP triedami na základe menného priestoru a jemu prislúchajúcim súborovým adresárom. Vďaka jeho použitiu odpadá nutnosť používať funkcie PHP *include*, respektíve *require*.

3.4 Modely

Modely predstavujú v našej aplikácii dáta. Prostredníctvom *PdoConnection* objektu vkladajú data do databázy, vyberajú a upravujú ich, prípadne mažu. Väčšina vytvorených modelov reprezentujú jednu tabuľku v databáze.

AbstractModel je abstraktná nadtrieda, ktorú dedia všetky modely. Spája atribúty a funkcie, ktoré sú rovnaké pre každú triedu modelov. Táto trieda umožňuje generické vytvorenie parametrov pre špecifický model, vďaka konštruktoru, ktorý *DESCRIBE* požiadavkou na databázu získa stĺpce požadovanej tabuľky. Podobne, ako

v triede *PdoConnection*, tak tu sú zadané metódy, ktoré na základe vstupných parametrov vyskladajú *SELECT* požiadavku pre konkrétnu tabuľku.

3.5 Ovládače

Ovládače využívame na spracovanie dát buď z modelov, alebo od používateľa. Výpočty a biznis logika sa vykonáva v ovládačoch.

Podobne, ako bol *AbstractModel* vytvorený pre modely, pre ovládače sme vytvorili **AbstractController**. Jeho pointou je to isté, čiže spojiť spoločnú funkcionality do jednej abstraktnej nadtriedy, aby sme sa vyhli opakovaniu kódu vo viacerých triedach. Okrem iného, trieda implementuje funkcie pre obmedzenie prístupu k funkciám ovládačov, na základe práv, čiže neprihlásený používateľ nemá právo volať niektoré funkcie. Aj z tohto dôvodu bolo nutné naprogramovanie autentifikácie, ako bolo spomenuté v kapitole 3.3.4. Taktiež sme tu napísali funkciu, ktorá zavolá správny náhľad a odovzdá mu dáta, ktoré má zobrazit.

3.6 Náhľady

PHP súbory nahľadov špecifikujú a generujú HTML kód pre prehliadač. Ovládače im odovzdávajú dáta, a náhľady zapracujú tieto dáta do štruktúry stránky. Cez náhľady sa rieši aj používateľský vstup. Vďaka formulárom môže používateľ odoslať dáta alebo požiadavky na serverovú stranu, ktorá požiadavku vyhodnotí a spracuje.

3.7 Databáza

3.8 Implementácia biznis logiky

3.8.1 Uloženie Petriho siete do databázy

Po požiadavke používateľa o uloženie Workflow-siete Z Procesného editora do databázy, editor pomocou AJAX-u zavolá funkciu, ktorá sa nachádza v *ApiController-i* našej aplikácie. Uskutoční sa komunikácia a spracovanie prijatých dát. Prijaté dáta akciou POST sú nasledovné:

- JSON
- XML
- SVG

JSON sa prekonveruje do objektu, ktorý obsahuje všetky potrebné informácie, pre beh Workflow-siete

1. Do tabuľky databázy *petri_net* sa uloží Petriho sieť s atribútmi ako meno a popis, ktoré je v JSON-e, XML a SVG a z aplikácie sa získa ID prihláseného používateľa, ako aj aktuálny dátum a čas. Pri úspechu sa vráti priradené ID, pri neúspechu vypíše chybu a sieť sa neuloží.
2. V situácii, že uloženie siete bolo úspešné, tak pre každé miesto z JSON-a sa vytvorí záznam v databáze, kde sa pre každý uchováva jeho meno(ak je zadané), počiatočné značkovanie a id v XML súbore. Uchovanie tohto ID je potrebné pre spracovanie rolí a formulárov na serverovej strane pri vytvorení novej siete. Taktiež sa používa pri vizualizácii prípadu, kde sa na jeho základe zaznačuje aktuálne značkovanie. Zaznačuje sa aj cudzí kľúč ID Petriho siete.
3. Podobným spôsobom sa uložia prechody, ktoré nemajú atribút pre počiatočné značkovanie.
4. Spracujú sa dáta ohľadom hrán. Pri hranách typu *regular* sa zisťuje, či je ich zdrojové miesto prechod, alebo miesto a na základe typu sa všetky hrany uložia do prislúchajúcej tabuľky.
5. Odošle sa ID siete späť do Procesného editora.

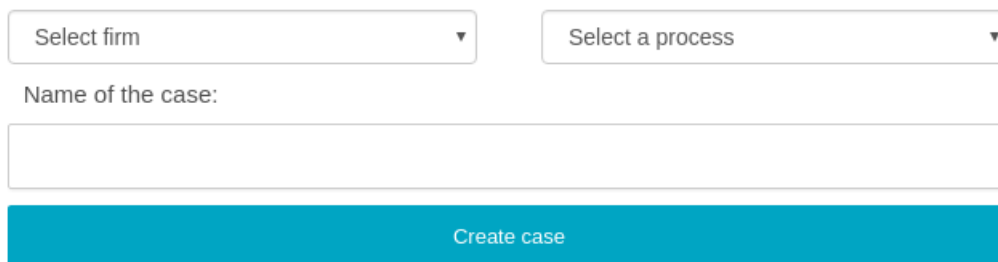
Pomocou takto spracovaných dát a uložených môže bežať *workflow* procesu.

3.8.2 Vytvorenie prípadu

Ak používateľ chce vytvoriť prípad, zadá požiadavku na server. Server, na základe používateľového ID musí vykonať požiadavku do databázy, ktorá vyfiltruje procesy podľa podmienok:

- Proces musí byť priradený jednej, alebo viacerým firmám, v ktorých je používateľ zaradený.
- Používateľ musí mať vo firme, ktorá má právo využívať proces, rolu, ktorá je definovaná ako tá, ktorá môže prípady tohto procesu vytvárať.

Všetky vrátené dáta sa spracujú do požadovaného formátu, čo v tomto prípade je asociatívne pole, kde kľúč je id firmy a hodnota je pole procesov. Zobrazí sa nasledujúci formulár:



Obr. 14: Formulár na vytvorenie prípadu

Po vybratí firmy z ľavého rozbaľovacieho zoznamu sa pomocou JavaScriptu vygenerujú možnosti pravého rozbaľovacieho zoznamu podľa ID vybratej firmy. Ak používateľ vyplní všetky polia a odošle formulár, server vytvorí záznam prípadu v databáze, kde je zaznačené, kto a kedy ho vytvoril a pod akou firmou bol vytvorený. Do tabuľky *case_marking* sa prekopíruje počiatočné značkovanie, ktoré sa počas behu prípadu bude tu meniť. Ak sa používateľ pokúša odoslať formulár bez toho, aby vyplnil všetky polia, je upozornený na pole, ktoré nevyplnil a systém mu oznámi, že prípad sa nevytvoril.

3.8.3 Správa workflow prípadu

Táto časť práce bola najzložitejšia na implementáciu. Okrem hľadania a vypočítavania spustiteľnosti prechodov, ich spúšťania spojeného s konzumáciou a produkciou značiek, sa spája aj dlhými náročnými požiadavkami na databázu. Môžeme ju rozdeliť na 3 časti:

1. Výber prechodov z databázy a zistenie ich spustiteľnosti
2. Prevzanie úlohy
3. Ukončenie úlohy

Každú z týchto častí si popíšeme podrobnejšie.

Výber prechodov z databázy a zistenie ich spustiteľnosti. O túto časť sa stará trieda *TaskController*. Je potrebné požiadať databázu o prechody. Samozrejme, zisťovať spustiteľnosť pre všetky prechody firiem, v ktorých je používateľ, by bolo zdĺhavé a výpočtovo náročné, rozhodli sme sa vypracovať požiadavku na databázu, ktorá má dané podmienky:

- Prechody patria prípadom vytvorených vo firmách, do ktorých používateľ patrí.

- Ak je na úlohu naviazaná referencia, úloha sa pridá do zoznamu.
- Ak je na tento prechod naviazaná rola, používateľ musí mať v danej firme túto rolu.

Okrem týchto podmienok, požiadavka ešte definuje, aby sa k dátam naviazali aj informačné údaje, ako napríklad názov úlohy alebo názov caseu. Všetky tieto podmienky sú zadefinované v jednej komplikovanejšej požiadavke, aby sme počet zisťovaných úloh obmedzili na minimum. V tomto bode sa zisťuje spustiteľnosť prechodov nasledujúcim spôsobom. Opäť si to vyžaduje požiadavky do databázy, ktoré vrátia hrany vstupujúce do daného prechodu a značkovanie v miestach pre zodpovedajúci prípad.

Kontroluje sa, či každá hrana vstupujúca do miesta spĺňa podmienky podľa jej typu, ktoré sme zadefinovali na základe kapitoly 1.2:

- Normálna hrana: značkovanie m miesta p je väčšie alebo rovné, ako váhová funkcia w tejto hrany h .

$$m_p \geq w(h)$$

- Inhibítorová hrana: značkovanie m miesta p je menšie, ako váhová funkcia w tejto hrany h .

$$m_p < w(h)$$

- Reset hrana: pre tento typ hrany nie je zadefinovaná podmienka, ktorá by ovplyvňovala spustiteľnosť prechodu, ako bolo spomínané v kapitole 1.2.5.

Automaticky, ak aspoň pre jednu hranu a značkovanie miesta nie je táto podmienka splnená, funkcia vráti *bool* hodnotu *FALSE* a prechod sa z poľa úloh vymaže. Ak sa splnia všetky hodnoty, funkcia vráti hodnotu *TRUE*. Po odkontrolovaní každej úlohy nasleduje riešenie referencií. Okrem zistenia, či referencované prechody vykonal používateľ, bolo našou snahou čo možno najviac obmedziť *deadlocky*, ktoré mohli vzniknúť zlým nastavením referencií. Riešime to týmito spôsobmi:

Bola referencovaná úloha spustená?:

1. ÁNO: Spustil prihlásený používateľ referencovanú úlohu?
 - (a) ÁNO: Nechaj úlohu v zozname úloh a zaznač k nej dôraz na referenciu.
 - (b) NIE: Je používateľ, ktorý spustil túto úlohu stále vo firme?

- i. ÁNO: Vymaž úlohu zo zoznamu úloh pre prihláseného používateľa.
 - ii. NIE: Má prechod naviazanú rolu?
 - A. ÁNO: Má prihlásený používateľ v danej firme túto rolu?
 - ÁNO: Nechaj úlohu v zozname úloh.
 - NIE: Vymaž úlohu zo zoznamu úloh pre prihláseného používateľa.
 - B. NIE: Nechaj úlohu v zozname úloh.
2. NIE: Má prechod naviazanú rolu?
- (a) ÁNO: Má prihlásený používateľ v danej firme túto rolu?
 - i. ÁNO: Nechaj úlohu v zozname úloh
 - ii. NIE: Vymaž úlohu zo zoznamu úloh pre prihláseného používateľa.
 - (b) NIE: Nechaj úlohu v zozname úloh

Tu sa ukončí overovanie spustiteľnosti a kontrola referencií a všetky vyfiltrované úlohy sa zobrazia používateľovi vo forme tabuľky. Každý riadok, čiže úloha, má tlačidlo, pomocou ktorého si používateľ môže úlohu vziať. Na obrázku môžeme vidieť, že pri úlohe, ktorý mal referenciu na používateľa sa zobrazí výkričník. Túto úlohu môže totiž splniť len on. Zobrazujeme aj, na ktorú z jeho rolí sa úloha viaže. Tieto úlohy si môže používateľ vďaka rozbaľovaciemu zoznamu vytriediť na základe firmy.

TODO: SCREEN.

Prevzatie úlohy. Používateľ sa rozhodne vziať úlohu a klikne na odosielajúce tlačidlo. Aj napriek tomu, že tento zoznam je pomocou AJAXu každých 30 sekúnd aktualizovaný (časový údaj 30 sekúnd sme zvolili kvôli tomu, že obnovenie údajov je spojené s náročnejšími výpočtami a požiadavkami do databázy), môže nastať situácia, že používateľ sa pokúsi zobrať úlohu, ktorú už niekto pár sekúnd pred ním vzal. Kvôli tejto situácii, po odoslaní žiadosti pre prevzatie úlohy, je prvá vec, ktorú kontrolujeme, či je úloha ešte stále k dispozícii. Ak nie, používateľovi sa zobrazí upozornenie, že úloha už nie je k dispozícii. V prípade, že úloha je stále voľná, priradí sa používateľovi. Táto fáza zodpovedá v mapovaní na Petriho sieť spusteniu prechodu - skonzumovaniu značiek. Dochádza k týmto akciám:

- Z databázy sa vyberú hrany, ktoré do tohoto miesta vchádzajú
- Z databázy sa vyberie značkovanie prípadu, z ktorých tieto hrany vychádzajú

- Upraví sa značkovanie na základe pravidiel hrany:
 - Normálna hrana h umožní prechodu t skonzumovať toľko značiek z miesta p , aká je váhová funkcia w tejto hrany. Značkovanie sa zmení

$$m_p = m_{po} - w(h)$$

kde m_{po} je značkovanie miesta p pred konzumáciou značiek.

- Reset hrana skonzumuje všetky značky z miesta, odkiaľ vychádza, $m_p = 0$.
- Inhibítorová hrana nemení značkovanie v zdrojovom mieste.

V databáze sa zaznačí, kto úlohu vzal a časový údaj, kedy ju vzal. V tejto fáze sa ešte neprodukujú žiadne značky do výstupných miest.

V prípade integrácie je naplánované prepojenie úlohy so Serverom formulárov a dátového modelu, ktoré bude fungovať nasledovne. Na zobrazenie formuláru používateľ stlačí tlačidlo *Zobraziť* pod hlavičkou formulár. Naša aplikácia zavolá Server formulárov a odošle mu ID prípadu a úlohy. Tak, ako bolo spomenuté v opise tejto časti aplikácie v 2.7, používateľ si bude môcť formulár predvyplniť a uložiť bez validácie, avšak pri pokuse o ukončenie úlohy sa bude kontrolovať validita vyplnenia formuláru a ak správne vyplnený nebude, používateľ bude na to upozornený a úloha sa neukončí.

Ukončenie úlohy. V Petriho sieti môžeme túto fázu chápať ako vyprodukovanie značiek. Používateľ úlohu ukončí a vykonajú sa tieto akcie:

- Do databázy sa zaznačí dátum a čas, kedy používateľ túto úlohu ukončil.
- Z databázy sa vyberú všetky hrany. Pre každú z nich sa nájde vstupné miesto prípadu a zaznačí sa nové značkovanie:

$$m_p = m_{po} + w(h)$$

- Overia sa 2 skutočnosti:
 - Je nejaká úloha v prípade práve vykonávaná?
 - Je nejaká úloha v prípade spustiteľná? Ak nastala situácia, kde obidve tieto podmienky sú negatívne, do databázy sa zaznačí koniec tohto prípadu s aktuálnym dátumom a časom. V opačnom prípade, ak je aspoň jedna podmienka pravdivá, prípad je stále aktívny a riadi sa jeho *workflow* naďalej tak, ako je to popísané v tejto kapitole.

3.8.4 Funkcia vrátenia úlohy

Naším cieľom v tomto bode bolo umožniť používateľov vrátiť úlohu, bez toho, aby to zmenilo stav Workflow-siete pred tým, než ju vzal. Pri požiadavke zrušenia, databázu požiadame o všetky hrany vstupujúce do prechodu zodpovedajúcemu úlohe. Pri obyčajných hranách a inhibítor hrane sme nenarazili na žiaden problém. Keďže inhibítorová hrana nekonzumuje žiadne značky z miesta, tak pre ňu sme nemuseli hľadať žiadne riešenie. Každú obyčajnú hranu riešime tak, že ku aktuálnemu značkovaniu miesta, z ktorého vychádza, pripočítame hodnotu váhovej funkcie tejto hrany.

Pri reset hranách to bolo zložitejšie. Tu sme nevedeli pomocou žiadnej hodnoty z databázy zistiť, koľko značiek táto hrana umožnila precodu skonzumovať. Ako prvé riešenie sme navrhli to, že sme do databázy do tabuľky úloh pridali stĺpec s *defaultnou* hodnotou *null*. Ak sa spustil niektorý prechod, do ktorého vstupovala reset hrana, tak sa do tohoto atribútu sa zaznačil počet značiek, aký táto hrana skonzumovala. Na základe tejto informácie sme vedeli pri vrátení úlohy zdrojovému miestu pripočítať presný počet značiek, aký bol odtiaľ skonzumovaný.

Problém nastal, ak do prechodu vchádzalo viac, ako jedna hrana. Pri tomto probléme sme sa rozhodli odstrániť z tabuľky úloh pridaný atribút pre uchovávanie skonzumovaných značiek, a vytvorili sme tabuľku, do ktorej ukladáme údaje reprezentujúce ID úlohy, ID reset hrany a počet značiek. Tieto údaje nám umožňujú plnohodnotne vrátiť značky do pôvodných miest pri zrušení úlohy aj v situácii, kedy má prechod viac, ako 1 vstupné miesto spojené s reset hranou.

3.8.5 Uchovávanie histórie

Ako bolo spomenuté v predošlých kapitolách, pri viacerých údajoch uchovávame dátum a čas. Pre túto funkciu sme sa rozhodli, aby bolo možné odsledovať históriu uložených Petriho sietí, vytvorených prípadov a úloh. Preto sme sa rozhodli uchovávať tieto údaje:

- Petriho sieť
 - dátum a čas vytvorenia
 - ID používateľa, ktorý sieť vytvoril
- Prípad
 - dátum a čas vytvorenia
 - ID používateľa, ktorý prípad vytvoril

- dátum a čas ukončenia
- Úloha
 - ID používateľa, ktorý úlohu vykonáva
 - dátum a čas vytvorenia
 - dátum a čas ukončenia

Vďaka týmto údajom zisťujeme, vyššie spomenuté funkcionality, a to či je prípad ukončený, alebo ktoré úlohy ešte neboli skončené. Na základe toho umožňujeme zobrazenie *Histórie úloh používateľa*, *Histórie prípadov* a zobrazenie *Aktuálnych prípadov*.

TODO: SCREEN

3.8.6 Vizualizácia prípadu

Aby používateľ mal predstavu, v akom stave sa prípad nachádza, rozhodli sme sa naimplementovať vizualizáciu prípadov, ktorá je možná vďaka spolupráci s Tomášom Pribojom - Procesný editor. Prebieha to týmto spôsobom:

1. Používateľ si z tabuľky aktuálnych prípadov vyberie prípad pre vizualizáciu a stlačí formulárové tlačidlo pre zobrazenie.
2. Zadá sa požiadavka na databázu, ktorá vráti XML zodpovedajúcej Petriho siete.
3. Z databázy sa vyberie aktuálne značkovanie vybraného prípadu.
4. Z databázy sa vyberú úlohy, ktoré sú aktuálne pre tento prípad vykonávané.
5. Počiatočné značkovanie v XML sa nahradí týmto aktuálnym značkováním
6. Zavolá sa funkcia procesného editora, ktorá sieť zaznačenú v upravenom XML reťazci vykreslí pomocou SVG, umiestni značky a vyfarbí spustiteľné prechody na zeleno a nespustiteľné na červeno.
7. Naša aplikácia SVG vykreslí.
8. Pre každú úlohu, ktorá je vykonávaná zaznačíme do prechodu ikonu a pridáme jej atribút *title* s menom používateľa, ktorý úlohu vykonáva.

TODO SCREEN

3.9 Testovanie riešenia - možné úpravy a rozšírenia

3.10 Use case diagramy

- Popis: tu príde popis
- Vstupné podmienky pre používateľa:
- Chybový tok:
- Zmena stavu systému:

4 Závěr

Literatúra

- [1] AALST, W.: *The Application of Petri Nets to Workflow Management*. Dostupné na internete: <http://wwwis.win.tue.nl/~wvdaalst/publications/p53.pdf>
- [2] DESEL, J. - REISIG, W.: *Lectures on Petri nets I: Basic models*. Germany: Universität Karlsruhe: Springer, 1998, ISBN 3-540-65306-6
- [3] LEWIS, A.H.: *An introduction to Petri nets*[cit.2016.04.25]. Dostupné na internete: <<http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=B8F8CDAA7484532525753>>
- [4] EHRIG, H. a kol. *Petri Net Technology for Communication-Based Systems, Advances in Petri Nets*. Germany: Springer, 2003, ISBN 3-540-20538-1
- [5] EHRIG, H a kol. *Unifying Petri Nets*. Germany: Springer, 2001, ISBN 3-540-43067-9
- [6] PETRI, C.A. - REISIG, W. 2008 *Petri net*. [online] [cit. 27.04.2016]. Dostupné na internete: <http://www.scholarpedia.org/article/Petri_net>
- [7] JUHÁS, G.: *Modelovacie formalizmy udalostných systémov* Bratislava: RT systems s.r.o., 2011, ISBN 978-80-970519-1-4, p. 29-45
- [8] VERBEEK, H.M.W. a kol.: *Reduction Rules for Reset/Inhibitor Nets* Netherlands: Elsevier, 2009, Dostupné na internete: <<http://wwwis.win.tue.nl/~wvdaalst/publications/p572.pdf>>
- [9] AALST, W. - HEE, K.: *Workflow Management Models, methods and systems*Cambridge, Massachusetts: The MIT Press, 2004, ISBN 0-262-01189-1, p. 267-268
- [10] AALST, W.M.P a kol.: *Soundness of Workflow Nets: Classification,Decidability, and Analysis*, Netherlands, [online] [cit. 27.04.2016]. Dostupné na internete: <<http://alexandria.tue.nl/repository/books/636105.pdf>>
- [11] Prof. Dr. Carl Adam Petri [online] [cit. 27.04.2016]. Dostupné na internete: <http://www.informatik.uni-hamburg.de/TGI/mitarbeiter/profs/petri_eng.html>
- [12] DOMES,M.: *Tvorba WWW stránek pro úplné začátečníky* Brno: Computer Press, 2012, ISBN 978-80-251-2160-3

- [13] *Introduction to HTML*. [online] [cit. 27.04.2016]. Dostupné na internete: <http://www.w3schools.com/html/html_intro.asp>
- [14] *HTML5 Introduction*. [online] [cit. 27.04.2016]. Dostupné na internete: <http://www.w3schools.com/html/html5_intro.asp>
- [15] *CSS Introduction*. [online] [cit. 27.04.2016]. Dostupné na internete: <http://www.w3schools.com/css/css_intro.asp>
- [16] *JavaScript Introduction*. [online] [cit. 27.04.2016]. Dostupné na internete: <http://www.w3schools.com/js/js_intro.asp>
- [17] *PHP 5 Introduction*. [online] [cit. 27.04.2016]. Dostupné na internete: <http://www.w3schools.com/php/php_intro.asp>
- [18] *Introduction to SQL*. [online] [cit. 27.04.2016]. Dostupné na internete: <http://www.w3schools.com/sql/sql_intro.asp>
- [19] *AJAX Introduction*. [online] [cit. 27.04.2016]. Dostupné na internete: <http://www.w3schools.com/php/php_ajax_intro.asp>
- [20] *JSON Tutorial*. [online] [cit. 27.04.2016]. Dostupné na internete: <<http://www.w3schools.com/json/>>
- [21] *XML Tutorial*. [online] [cit. 27.04.2016]. Dostupné na internete: <<http://www.w3schools.com/xml/default.asp>>
- [22] *SVG Tutorial*. [online] [cit. 27.04.2016]. Dostupné na internete: <<http://www.w3schools.com/svg/default.asp>>
- [23] *PHP Manual*. [online] [cit. 27.04.2016]. Dostupné na internete: <<https://secure.php.net/manual/en/index.php>>
- [24] *Historical trends in the usage of server-side programming languages for websites*. [online] [cit. 27.04.2016]. Dostupné na internete: <http://w3techs.com/technologies/history_overview/programming_language>
- [25] *What is MySQL? What is a Database? What is SQL?*. [online] [cit. 27.04.2016]. Dostupné na internete: <<http://www.thesitewizard.com/faqs/what-is-mysql-database.shtml>>

- [26] DOMES,M.: *1001 tipů a triků pro PHP* Brno: Computer Press, 2012, ISBN 978-80-251-2940-1
- [27] PEACOCK,M.: *PHP 5 e-commerce Development* Birmingham: Packt Publishing Ltd., 2010, ISBN 978-1-847199-64-5, p. 19-60
- [28] *Design Patterns*. [online] [cit. 27.04.2016]. Dostupné na internete: <https://sourcemaking.com/design_patterns>
- [29] *PSR-4: Autoloader*. [online] [cit. 29.04.2016]. Dostupné na internete: <<http://www.php-fig.org/psr/psr-4/>>

Obsah CD nosiča

Obsah priloženého CD nosiča je nasledovný:

- v priečinku *dokument* sa nachádza text diplomovej práce vo formáte PDF
- v priečinku *aplikácia* sú zdrojové súbory výslednej funkčnej aplikácie, a tiež skript pre vytvorenie tabuliek v databáze
- v priečinku *dokumentácia* je dokumentácia zdrojového kódu vo formáte HTML obsahujúca popis jednotlivých tried a ich metód