Sem vložte zadanie Vašej práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalárska práca

Informační systém pro plánování služeb na oddělení

Tomáš Sergejev

Vedúci práce: Ing. Zdeněk Rybola

12. mája 2015

Poďakovanie

Chcel by som poďakovať vedúcemu práce, Ing. Zdeňkovi Rybolovi, za trpezlivosť, cenné rady a čas, ktorý mi venoval. Ďalej by som chcel poďakovať pani Lenke Cimlerovej za poskytnutie kľúčových informácii, rodine a priateľom za podporu.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 12. mája 2015 .....

Odkaz na túto prácu

Sergejev, Tomáš. *Informační systém pro plánování služeb na oddělení.* Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Táto bakalárska práca obsahuje dôkladnú analýzu problematiky plánovania služieb na pracoviskách s nepretržitou prevádzkou. Na základe tejto analýzy bolo navrhnuté riešenie aplikácie, ktoré bolo implementované formou desktopovej aplikácie, a následne dôkladne otestované. V samom závere práce popisujem nedokonalosti vytvoreného systému a spôsoby, ako ich riešiť. Tiež ponúkam iné pohľady na riešenie tejto problematiky a ďalšie možnosti rozšírenia programu pre jeho plné využitie v praxi.

Klíčová slova informačný systém, metóda, pravidlo, trieda, zamestnanec, zmena

Abstract

This bachelor thesis analyzes the problem of shifts scheduling on workplaces with non-stop operation. Based on this analysis comes the design of the solution implemented in form of a desktop application, which was carefully tested. In the thesis conclusion I describe the weak parts of the created system and their possible improvements. Also other alternatives to proposed solution and further extensions of the program are offered to increase utilization in practice.

Keywords class, employee, information system, method, rule, shift

Obsah

Zoznam obrázkov

Úvod

Na pracoviskách s nepretržitou prevádzkou je potrebné každý deň zabezpečiť dostatok zamestnancov na ich bezproblémový chod. Patria medzi ne napr. nemocnice, na ktorých je 24 hodinová pohotovosť samozrejmosťou, medicínske laboratoria, či priemyslená výroba s nepretržitou trojsmennou prevádzkou. Zamestnanci sa striedajú na denných a nočných (prípadne na troch) zmenách. K tomu je potrebný dostatočný počet zamestancov a dodržanie určitých pravidiel tak, aby nedochádzalo k porušovaniu zákonníku práce. Plánovanie zmien má na starosti vedúci pracovník. Ten najskôr zozbiera požiadavky svojich zamestnancov a následne obsadí pracovné zmeny. Jednou z možností je ručné plánovanie zmien na papier, dobrým pomocníkom môže byť tabulkový procesor Excel. Nevýhodou takéhoto plánovania je vysoká časová náročnosť, neprehľadnosť a tiež nutnosť opakovanej kontroly dodržiavania pavidiel. Ďalšou možnosťou je využitie informačného systému, ktorý umožní automatické naplánovanie zmien, kontrolu dodržania pravidiel a zvýši prehľad. Preto som bol oslovený staničnou sestrou gynekologicko pôrodníckeho oddelenia v nemocnici na Mělníku s požiadavkou na vytvorenie takého systému, ktorý by jej uľahčil prácu pri plánovaní zmien na jej pracovisku.

Kapitola 1

Cieľ práce

Cieľom tejto bakalárskej práce je analyzovať problematiku plánovania zmien, navrhnúť a implementovať informačný systém, ktorý by toto plánovanie uľahčil a zjednodušil. Systém bude plniť v prvom rade evidenčnú funkciu. Užívateľ bude môcť pomocou neho upravovať údaje o zamestnancoch, manuálne plánovať pracovné zmeny a navrhovať pravidlá. Tým sa urýchli a zjedoší práca vedúcich pracovníkov. V druhom rade bude slúžiť na automatické plánovanie zmien. Tento plán si bude môcť užívateľ (vedúci pracovník) kontrolovať, upravovať a následne vyexportovať do excelovských tabuliek. Jedná sa o pomerne rozsiahly systém, preto sa v tejto práci pokúsim zamerať na jeho podstatné a zaujímavé časti, ktoré v nej podrobne popíšem.

1.1 Štruktúra práce

V nasledujúcich odstavcoch stručne popíšem jednotlivé časti tejto práce tak, aby čitateľ získal predstavu o tom, čím sa v nich budem zaoberať.

1.1.1 Informácie

V úvode popíšem informácie, z ktorých som vychádzal a spôsob, akým som ich získal.

1.1.2 Analýza

Tieto informácie dôkladne analyzujem, vymedzím funkčné, nefunkčné požiadavky na systém a prípady použitia. Pozriem sa na existujúce riešenia.

1.1.3 Návrh

Popíšem návrh kľúčových častí systému tak, aby čitateľ pochopil základný princíp ich funkčnosti.

1.1.4 Realizácia

Popíšem spôsob, akým som realizoval zaujímavé a dôležité časti systému.

1.1.5 Testovanie

Popíšem spôsoby, akými som testoval výslednú aplikáciu.

1.1.6 Prínos

Stručne zhrniem prínos implementovaného systému.

1.1.7 Budúcnosť

Na záver zhrniem, akými spôsobmi by sa mohlo vyvíjať riešenie danej problematiky v budúcnosti.

Kapitola 2

Analýza

Táto kapitola obsahuje súhrn kľúčových informácii potrebných pre návrh systému, ich dôkladnú analýzu, z ktorej boli odvodené funkčné a nefunkčné požiadavky, a súhrn prípadov použitia, ktoré tieto požiadavky pokrývajú.

2.1 Konkrétne požiadavky užívateľa

Nasledujúca kapitola obsahuje konkrétne požiadavky na systém, z ktorých boli odvodené všeobecné funkčné požiadavky. Tie som získal od staničnej sestry gynekologicko-pôrodníckeho oddelenia po osobnom stretnutí, ktorá ma zoznámila s chodom oddelenia. Na oddelení pracuje 13 pôrodných asistentiek (PA) s rôznou výškou pracovného úvazku. Každý deň v týždni je potrebné zabezpečiť pokrytie chodu pôrodného sálu a oddelenie šestonedlia a to v počte 3 PA na dennej zmene a 2 PA na nočnej zmene. Naviac je potrebné v pracovné dni obsadiť dve ranné zmeny – zmena staničnej sestry na oddelení a PA v tehotenskej poradni. Pri plánovaní zmien musí byť zohľadnené dodržiavanie zákonníku práce. Týždenná pracovná doba v nepretržitej prevádzke je 37,5 hod., pri rannej zmene 40 hod., podľa výšky pracovného úvazku. Prípustná dĺžka zmeny je maximálne 12 hodín, vrátane prestávky. Rovnomerným rozdelením zmien je potrebné docieliť aby všetky PA odpracovali potrebný počet hodín za mesiac a zároveň sa minimalizovalo množstvo nadčasov. Staničná sestra musí dbať aj na osobné požadavky sestier a optimalizovať čerpanie dovolenky. V kolektíve pracujú PA s rôznym stupňom vzdelania a pracovných skúseností. Všetky zmeny musia byť obsadené tak, aby bol zabezpečený bezpečný chod oboch odelení. Z uvedeného som formuloval pravidlá, ktoré musia, prípadne by mali byť pri plánovaní zmien dodržané.

2.1.1 Pravidlá

Užívateľom boli požadované nasledujúce pravidlá:

1. Zamestnanec nesmie mať priradenú dennú zmenu (prípadne dovolenku) deň po nočnej zmene, respektíve dennú zmenu priamo pred nočnou zmenou v ten istý pracovný deň.

2. Deň po dennej zmene by mal zamestnanec absolvovať nočnú zmenu (deň pred nočnou zmenou dennú).

3. Na nočnej zmene potrebujeme aspoň jedného zamestnanca, ktorý má skúsenosti s prácou na danom pracovisku.

4. V prípade doktorov potrebujeme na nočnej aspoň jedného zamestnanca so skúsenosťami na danom pracovisku a aspoň jedného so skúsenosťami v obore (atestácia). Môže sa jednať o dvoch rôznych zamestnancov (každý splní jeden požiadavok) alebo o jedného zamestnanca, ktorý bude skúsený a zároveň atestovaný.

5. Zamestnanec by nemal mať 2 nočné zmeny po sebe.

6. Žiadny zamestnanec nesmie mať 3 nočné zmeny po sebe.

7. Zamestnanec by nemal pracovať 3 dni po sebe.

8. Zamestnanec nesmie pracovať 4 dni po sebe.

9. Zmena bude určená pre konkrétneho zamestnanca. Nebudeme na ňu dosadzovať iných zamestnancov.

10. Na zmenu, ktorá je vykonávaná na oddelení, potrebujeme zamestnanca, ktorý je schopný pracovať na tomto oddelení.

11. Zmenu na operačnej sále je potrebné obsadiť zamestnancom, ktorý je na túto prácu školený.

12. V prípade, že sa časť zmeny odvíja na oddelení a časť na sále, musí byť zamestnanec schopný pracovať na oboch miestach.

13. Dvaja konkrétny zamestnanci nesmú pracovať spolu, nech už je dôvod akýkoľvek.

2.1.2 Zmeny

Užívateľ potrebuje obsadiť nasledujúce zmeny:

- **Denná zmena na sále** - značená **DO**, je určená na pracovné dni aj víkendy, trvá 11,5 hodiny, na jej obsadenie je potrebný 1 zamestnanec.

- **Denná zmena na oddelení** - značená ***DO***, je určená na pracovné dni aj víkendy, trvá 11,5 hodiny, na jej obsadenie je potrebný 1 zamestnanec.

- **Denná zmena na sále alebo oddelení** - značená **DO.**, je určená iba na pracovné dni, trvá 11,5 hodiny, na jej obsadenie je potrebný 1 zamestnanec.
- **Ranná zmena** - značená **D**, je určená na pracovné dni, trvá 7,5 hodiny, na jej obsadenie je potrebný 1 zamestnanec.
- **Zmena pre vedúcu** - značená **D**, je určená na pracovné dni, trvá 7,5 hodiny.
- **Nočná zmena na sále** - značená **N**, je určená na pracovné dni aj víkendy, trvá 11,5 hodiny, na jej obsadenie je potrebný 1 zamestnanec.
- **Nočná zmena na oddelení** - značená ***N***, je určená na pracovné dni aj víkendy, trvá 11,5 hodiny, na jej obsadenie je potrebný 1 zamestnanec.
- **Dovonenka** - značená **ŘD**, zamestnanci ju môžu čerpať iba cez pracovné dni, do odpracovaných hodín sa im zaráta 11,5 hodiny.

2.2 Funkčné požiadavky

V nasledujúcej kapitole popisujem funkčné požiadavky na systém odvodené z konkrétnych požiadavkov popísaných vyššie.

2.2.1 Evidencia zamestnancov

Užívateľmi bola pri zamestnancoch požadovaná evidencia nasledujúcich informácii:

- Meno a priezvisko zamestnanca.
- Pracovné skúsenosti v obore (atestácia).
- Pracovné skúsenosti na danom pracovisku.
- Možnosť vykonávať prácu na určitom mieste (oddelenie / sála).
- Veľkosť úväzku, na ktorý zamestnanec pracuje.
- Počet odpracovaných hodín za určitý mesiac
- Počet hodín, ktoré zamestnanec odrobil nad rámec zákonom stanovenej normy, respektíve počet hodín, ktoré mu na splnenie tejto normy chýbajú.

2.2.2 Evidencia zmien

Užívateľ požadovali u zmien evidovať nasledovné:

- Identifikácia zmeny (názov zmeny + označenie).
- Typ zmeny (denná, nočná alebo špeciálna).
- Dĺžka trvania zmeny.
- Počet zamestnancov potrebných na obsadenie zmeny.
- Zmena určená na víkend alebo pracovný deň.

2.2.3 Evidencia pravidiel

Pravidlá je potrebné definovať z dvoch dôvodov:

1. Zamedzenie nežiaduceho stavu.
2. Dosiahnutie požadovaného stavu.

Z konkrétnych pravidiel som odvodil nasledovné:

- Pravidlo, ktoré bude overovať, či má zamestnanec dostatok skúseností s prácou vo svojom obore alebo na danom pracovisku, či môže pracovať na sále alebo oddelení. Pravidlom môžeme tiež určiť, pre ktorého zamestnanca je zmena určená. Zoskupuje požiadavky 9, 10, 11, 12.
- Pravidlo, ktoré určuje aké zmeny má / nemá mať deň pred / po dni, v ktorom chceme obsadiť zmenu zamestnancom. Zovšeobecňuje požiadavky 1 a 2.
- Pravidlo, ktoré overí, či zamestnanec nemá určitý počet dní po sebe priradené vybrané zmeny. Napr. požiadavky 5, 6, 7 a 8.
- Pravidlo, pomocou ktorého zaistíme, že na určitých zmenách bude vopred stanovený počet zamestnancov s požadovanými hodnotami atribútov. Konkrétne požiadavky 3 a 4.
- Pravidlo, ktoré overí, či sa určité zmeny nepokúšame obsadiť zamestnancami, ktorých na týchto zmenách nechceme spolu. Jedná sa o pravidlo č. 13.

2.2.4 Evidencia sviatkov

- Práca vo sviatok je nadštandardne finančne ohodnotená, preto je potrebné rozdeľovať služby vo sviatok medzi zamestnancov rovnomerne.
- Možnosť pridať alebo odobrať sviatok.
- Obsadzovať sviatok víkendovými zmenami.

2.2.5 Automatické generovanie rozpisu

- Generátor by mal obsadiť všetky zmeny potrebným počtom zamestnancov.
- Program bude rešpektovať všetky pravidlá, povinné neporuší, nepovinné iba v nevyhnutných prípadoch.
- Zmeny budú medzi zamestnancov rozdeľované rovnomerne.
- Pri rozdeľovaní bude rešpektovaný úväzok zamestnanca.

2.2.6 Ručné priraďovanie zmien zamestnancom

- Možnosť priradiť ľubovoľnú zmenu ktorémukoľvek zamestnancovi.
- Užívateľ bude upozornený v prípade, že priradením zmeny zamestnancovi poruší nejaké pravidlo, prípadne je zmena už plne obsadená.

2.2.7 Odstraňovanie zmien

- Možnosť odstrániť ľubovoľnú zmenu priradenú konktrétnemu zamestnancovi.
- Odstránenie všetkých zmien v určitý deň.
- Uvoľnenie všetkých zmien v mesiaci.

2.2.8 Import a export rozpisu

- Možnosť načítať plán rozdelenia zmien z excelovskej tabuľky v konkrétnom formáte.
- Možnosť exportovať plán rozdelenia zmien zo systému do tejto tabuľky.
- Rozlišovať zmeny označené kurzívou.

2.3 Nefunkčné požiadavky

Všeobecné požiadavky na systém, ktoré určujú jeho obmedzenia, dodržiavanie štandardov a majú zásadný dopad na návrh architektúry.[6]

2.3.1 Český jazyk

Program bude napísaný v českom jazyku.

2.3.2 MS Windows

Program bude spustiteľný na operačnom systéme Windows verzie XP a vyššej.

2.3.3 Grafické prostredie

Program bude obsahovať grafické prostredie realizované formou desktopovej aplikácie.

2.4 Prípady použitia

Prípady použitia (anglicky „use cases“) popisujú funkcionalitu systému z pohľadu užívateľa, „čo“ musí systém robiť.[6] Pozostávajú najmä zo slovného popisu, sú doplnené UML diagramom.[9]

2.4.1 Zoznam účastníkov

V tomto prípade obsahuje zoznam účastníkov jedinú položku. Tou je užívateľ, ktorý bude systém využívať - pridávať a upravovať zamestnancov, zmeny, sviatky či pravidlá. Jedná sa o vedúceho daného ddelenia.

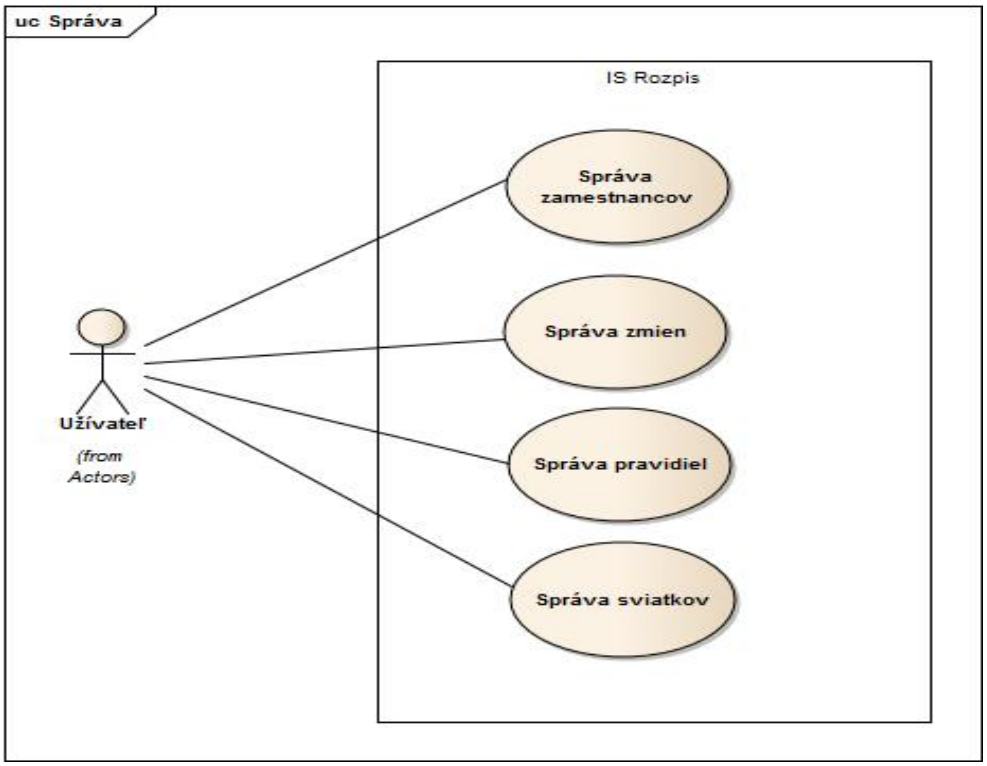
2.4.2 Podrobný popis prípadov použitia

V nasledujúcej kapitole zachytávam najdôležitejšie prípady použitia, ktoré pokrývajú funkčné požiadavky z predchádzajúcej kapitoly.

2.4.3 Správa zamestnancov

Skupina prípadov použitia spojená s pridávaním, editovaním, odstraňovaním a prehľadom zamestnancov.

- **Zobraziť zoznam zamestnancov**: Po kliknutí na položku „Seznam zaměstnanců“ v menu sa užívateľlovi zobrazí tabuľka aktívnych zamestnancov.
- **Editovať zamestnanca**: Po kliknutí na tlačidlo „Editovat“ v zobrazenom zozname zamestnancov, umiestnenom pri príslušnom zamestnancovi, sa užívateľovi zobrazí formulár s predvyplnenými údajmi o zamestnancovi, ktoré bude mať možnosť pomocou tohto formulára upraviť.
- **Pridať nového zamestnanca**: Po výbere položky „+ Přidat zaměstnance“ z hlavného menu sa zobrazí prázdny formulár, do ktorého si užívateľ vyplní informácie o novom zamestnancovi, a pridá zamestnanca do zoznamu.
- **Odstrániť zamestnanca zo zoznamu**: Po kliknutí na tlačidlo „Odstranit“ umiestnenom v riadku tabuľky, ktorý obsahuje informácie o určitom zamestnancovi a po odsúhlasení tohto úkonu užívateľom, dôjde k odstráneniu tohto zamestnanca zo zoznamu.



Obr. 2.1: Zoznam prípadov použitia spojených so správou entít

2.4.4 Správa zmien

Prípady použitia, ktoré sa týkajú evidencie zmien.

- **Zobraziť zoznam zmien**: Po kliknutí na príslušnú položku v menu sa užívateľovi zobrazí tabuľka denných, nočných alebo špeciálnych zmien, ktoré sú momentálne aktívne.

- **Editovať zmenu**: Po kliknutí na tlačidlo „Editovat“ v zobrazenom zozname zmien, umiestnenom v každom riadku tabuľky pri informáciach o konkrétnej zmene, sa zobrazí formulár s predvyplnenými údajmi o zmene, ktoré je možné pozmeniť a uložiť.

- **Pridať novú zmenu**: Po výbere jednej z položiek v hlavnom menu sa zobrazí prázdny formulár, do ktorého si užívateľ vyplní informácie o novo zavedenej zmene. Po kliknutí na tlačidlo „Přidat“ je zmena pridaná do správneho zoznamu, podľa toho, či sa jedná o dennú, nočnú, alebo špeciálnu zmenu. Tento zoznam sa nádsledne zobrazí aj s pridanou zmenou.

- **Odstrániť zmenu zo zoznamu**: Po kliknutí na tlačidlo „Odstranit“

umiestnenom v každom riadku tabuľky pri konkrétnej zmene a po odsúhlasení tohto úkonu užívateľom, dôjde k odstráneniu zmeny zo zoznamu.

- **Priradiť pravidlá**: Formulár, pomocou ktorého je možné konkrétnej zmene priradiť určité pravidlá, prípadne túto väzbu zrušiť, sa zobrazí po kliknutí na tlačidlo označené ako „Zobrazit“.

2.4.5 Správa pravidiel

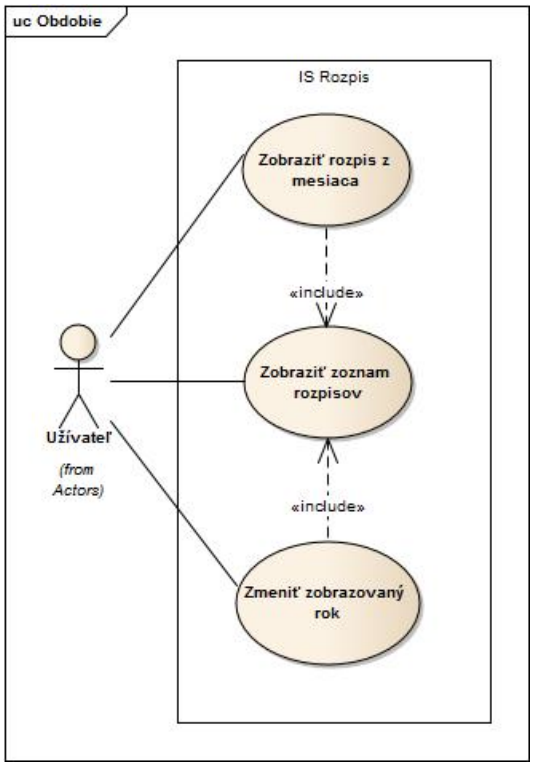
Súhrn prípadov použitia spojených s evidenciou pravidiel.

- **Zobraziť zoznam pravidiel**: Po kliknutí na príslušnú položku v menu sa užívateľovi zobrazí tabuľka, ktorá obsahuje jednotlivé druhy pravidiel.
- **Editovať pravidlo**: Po kliknutí na tlačidlo „Editovat“ pri pravidle v zobrazenom zozname sa užívateľovi zobrazí formulár s predvyplnenými informáciami o pravidle, ktoré je možné pozmeniť a uložiť.
- **Pridať nové pravidlo**: Po výbere jednej z položiek v hlavnom menu a kliknutí na tlačidlo „+ Přidat pravidlo“ sa zobrazí prázdny formulár, do ktorého užívateľ vyplní informácie o novo definovanom pravidle. Po pridaní sa užívateľovi zobrazí zoznam pravidiel s práve definovaným pravidlom.
- **Odstrániť pravidlo zo zoznamu**: Po kliknutí na tlačidlo „Odstranit“ umiestnenom v každom riadku tabuľky a po odsúhlasení tohto úkonu užívateľom, bude pravidlo odstránené z tohto zoznamu.
- **Priradiť zmeny**: Zoznamy niektorých pravidiel obsahujú tlačidlo „Zobrazit“. Po kliknutí na toto tlačidlo sa zobrazí formulár, v ktorom sú zobrazené väzby medzi pravidlom a zmenami, v rámci ktorých chceme toto pravidlo uplatniť. Pomocou tohto formulára môže užívateľ tieto väzby meniť.

2.4.6 Správa sviatkov

V tejto podkapitole sa venujem prípadom použitia spojených s pridávaním, editáciou, mazaním a prehľadom sviatkov.

- **Zobraziť zoznam sviatkov**: Po kliknutí na príslušnú položku v menu sa užívateľovi zobrazí tabuľka obsahujúca dátum a názov sviatku, podľa požadovaného roku.
- **Zmeniť rok**: Pri aktuálne zobrazovanom roku sa nachádzajú dve tlačítka, ktorými je možné posunúť sa na predchádzajúci / nasledujúci rok.
- **Editovať sviatok**: Formulár pre editovanie sviatku s predvyplnenými údajmi sa zobrazí po kliknutí na tlačidlo „Editovat“.



Obr. 2.2: Obrázok diagramu prípadov použitia spojených s nastavením zobrazovaného obdobia

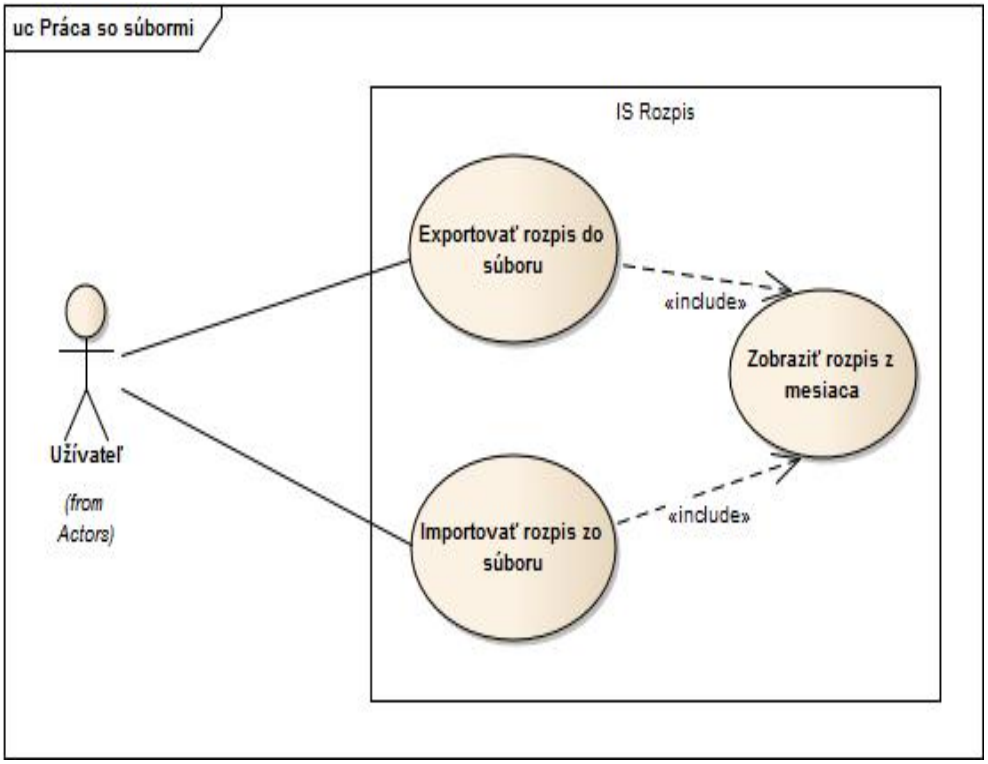
- **Pridať nový sviatok**: Po výbere položky „+ Přidat svátek“ v hlavnom menu, sa zobrazí prázdny formulár, do ktorého užívateľ vyplní informácie o novo pridanom sviatku.

2.4.7 Nastavenie obdobia

V nasledujúcich riadkoch popisujem prípady použitia spojené so zmenou zobrazovaného obdobia.

- **Zobraziť zoznam rozpisov**: Po rozkliknutí položky „Rozpis“ v hlavnom menu, a kliknutí na rovnomenné tlačidlo, sa zobrazí tabuľka rozpisov za aktuálny rok.

- **Zobraziť rozpis z mesiaca**: Po kliknutí na tab s označením príslušného mesiaca sa zobrazí konkrétny rozpis zmien za tento mesiac, log s výpisom porušených pravidiel a neobsadených / preplnených zmenách a prehľad odpracovaných hodín jednotlivých zamestnancov v zobrazenom mesiaci.



Obr. 2.3: Obrázok diagramu prípadov použitia spojených s importom a exportom

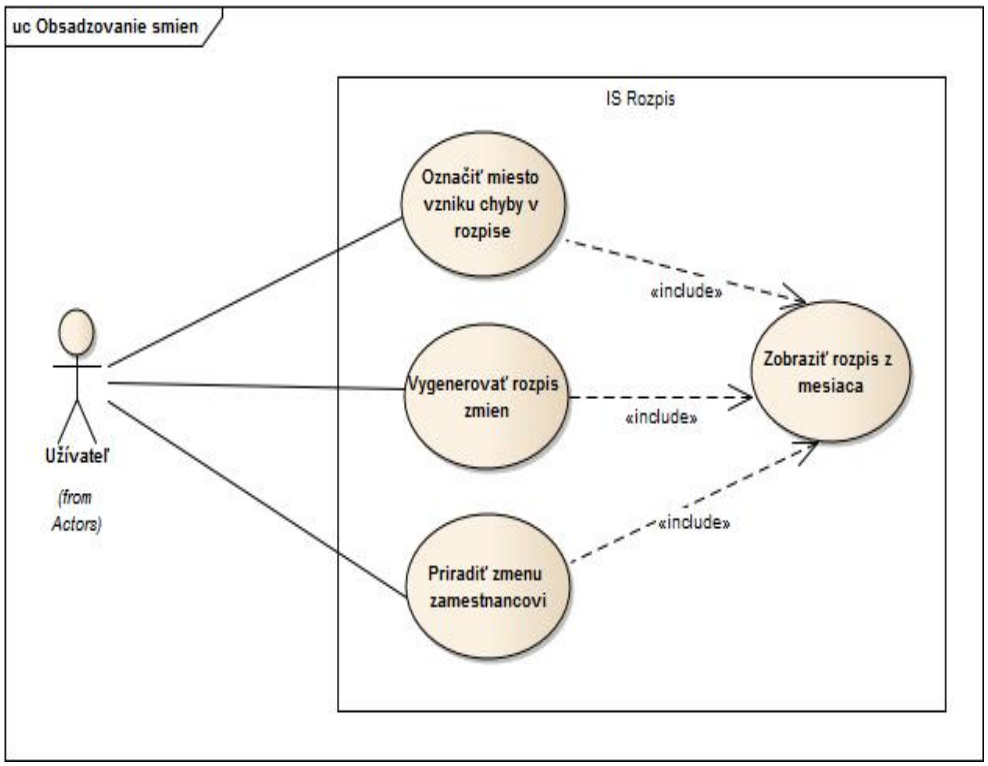
- **Zmeniť zobrazovaný rok**: Po kliknutí na jedno z tlačidiel, umiestnených na panely nástrojov, sa zobrazí rozpis zmien z minulého / budúceho roku.

2.4.8 Práca so súbormi

Táto podkapitola rozoberá prípady použitia spojené s importom a exportom rozpisu do súboru.

- **Importovať rozpis zo súboru**: Po tom, ako užívateľ klikne na tlačidlo na panely nástrojov, a vyberie požadovaný súbor, budú z tohto súboru importované všetky rozpoznané väzby medzi zamestnancami a zmenami.

- **Exportovať rozpis do súboru**: Po tom, ako užívateľ klikne na tlačidlo na panely nástrojov, a vyberie požadovaný súbor, budú do tohto súboru exportované všetky väzby medzi zamestnancami a zmenami vytvorené v aktuálne zobrazenom rozpise.



Obr. 2.4: Obrázok diagramu prípadov použitia spojených s obsadzovaním zmien

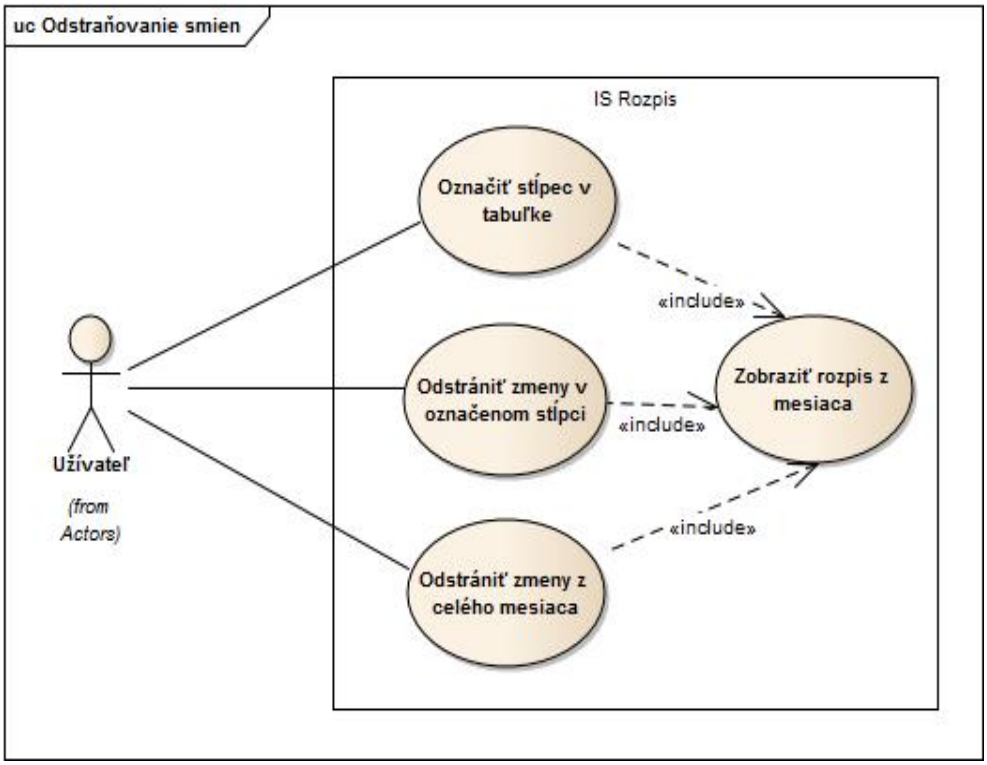
2.4.9 Obsadzovanie zmien

Prípady použitia, ktoré súvisia s obsadzovaním zmien sú popísané nižšie.

- **Vygenerovať rozpis zmien**: Na panely nástrojov sa nachádza ikonka, ktorá zahájí automatické obsadzovanie chýbajúcich väzieb medzi zamestnancami a zmenami, a to tak, aby nedošlo k žiadnemu porušeniu povinných pravidiel definovaných k daným zmenám.

- **Priradiť zmenu zamestnancovi**: Vo chvíli, keď užívateľ klikne na konkrétnu bunku tabuľky, mu bude zobrazený zoznam zmien, ktoré je možné v daný deň (reprezentovaný stĺpcom) priradiť konkrétnemu zamestnancovi (riadok tabuľky). Prípadne je možné existujúcu väzbu odstrániť či nahradiť inou.

- **Označiť miesto vzniku chyby v rozpise**: Po kliknutí na ikonku umiestnenú pri každej chybovej správe v logu, sa zvýraznia bunky v tabuľke, ktorých sa táto správa týka.

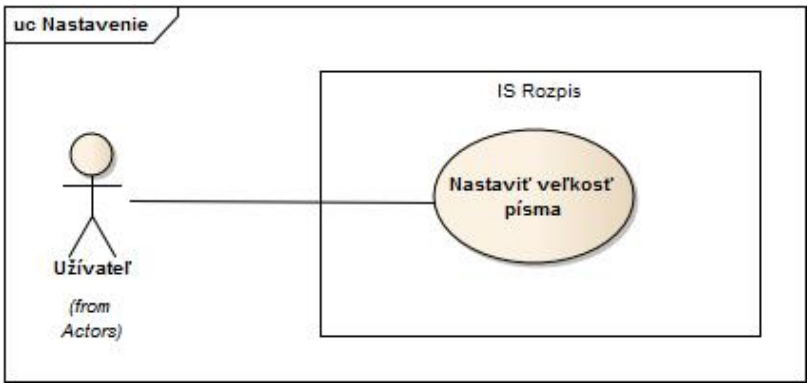


Obr. 2.5: Obrázok diagramu prípadov použitia spojených s odstraňovaním zmien

2.4.10 Odstraňovanie zmien

Prípady použitia, ktoré sú spojené s odstraňovaním zmien.

- **Posun na vedľajší stĺpec**: Na panely nástrojov sa nachádzajú dve tlačítka, ktorými sa užívateľ môže posunúť o jeden stĺpec vpravo alebo vľavo.
- **Označenie konkrétneho stĺpca**: Po kliknutí na bunku v hlavičke tabuľky sa zvýrazní stĺpec pod touto bunkou.
- **Odstrániť zmeny v označenom stĺpci**: Po kliknutí na príslušné tlačidlo na paneli nástrojov sa v označenom stĺpci odstránia väzby medzi zamestnancami a zmenami.
- **Odstrániť zmeny z celého mesiaca**: Po kliknutí na príslušné tlačidlo na paneli nástrojov, a odsúhlasení tohoto úkonu užívateľom, sa odstránia všetky väzby medzi zamestnancami a zmenami v aktuálne zobrazenom mesiaci.



Obr. 2.6: Prípad použitia spojený s nastavením veľkosti písma

2.4.11 Nastavenia

V systéme je možné nastaviť veľkosť písma.

- **Nastaviť veľkosť písma**: Po výbere príslušnej možnosti v hlavnom menu sa užívateľovi sprístupní možnosť zmeniť veľkosť písma hlavného menu, zoznamov entít a formulárov určených na editáciu a pridávanie týchto entít, či zoznam rozpisov zmien.

2.5 Rešerše stávajúcich systémov

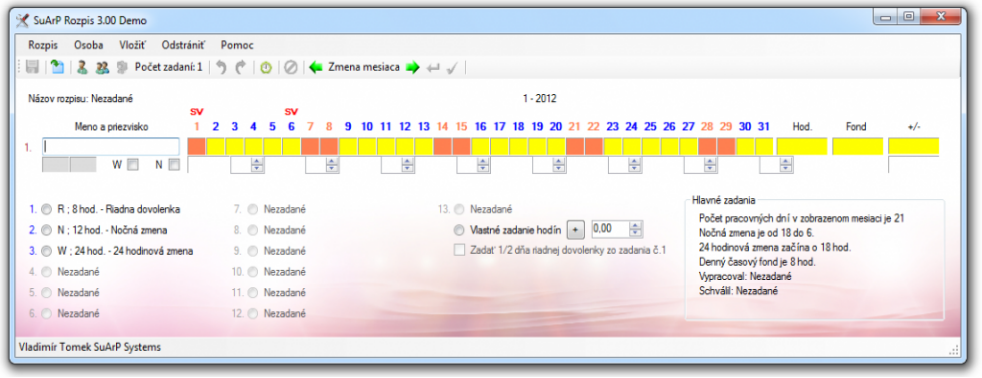
V nasledujúcej kapitole stručne popíšem dve vybrané riešenia problematiky plánovania zmien.

2.5.1 SuArP Rozpis

SuArP Rozpis je program pre rozpis a plánovanie zmien pre viacsmenné prevádzky. Umožňuje napríklad tlač rozpisu pomocou MS Office, ukladanie a načítanie rozpisu, prenosy hodín a dovoleniek z predchádzajúcich mesiacov, automatické výpočty odpracovaných hodín. [8]Plní teda najmä evidenčnú funkciu. Nie je však pomocou neho možné automatické generovanie plánu, ani import / export tohoto plánu do užívateľom požadovaného dokumentu. Je implementovaný v Slovenskom jazyku.

2.5.2 Rozpis Profi

Program Rozpis slúži na generovanie rozpisov zmien pracovníkov na pracoviskách s nepretržitou alebo smennou prevádzkou. Program je určený pre malé kolektívy. Umožňuje správu zamestnancov, pridávanie vlastných smien, definovanie 6 druhov pravidiel s rozdielnou váhou, ručné priraďovanie zmien, automatické generovanie a tlač rozpisu.[5] Bezplatná verzia programu sa dá v



Obr. 2.7: Ukážka programu SuArP Rozpis

súčastnosti využiť pre 15 zamestnancov (v dobe rešerše to bolo iba 10 zamestnancov), neumožňuje však import ani export do excelu.

Aj keď Existuje väčšie množstvo systémov zaoberajúcich sa danou problematikou, väčšina z nich je spoplatnená, alebo nepokrýva všetky požiadavky užívateľa. Preto som sa rozhodol pre návrh a implementáciu vlastného systému.

2.6 Analytický doménový model

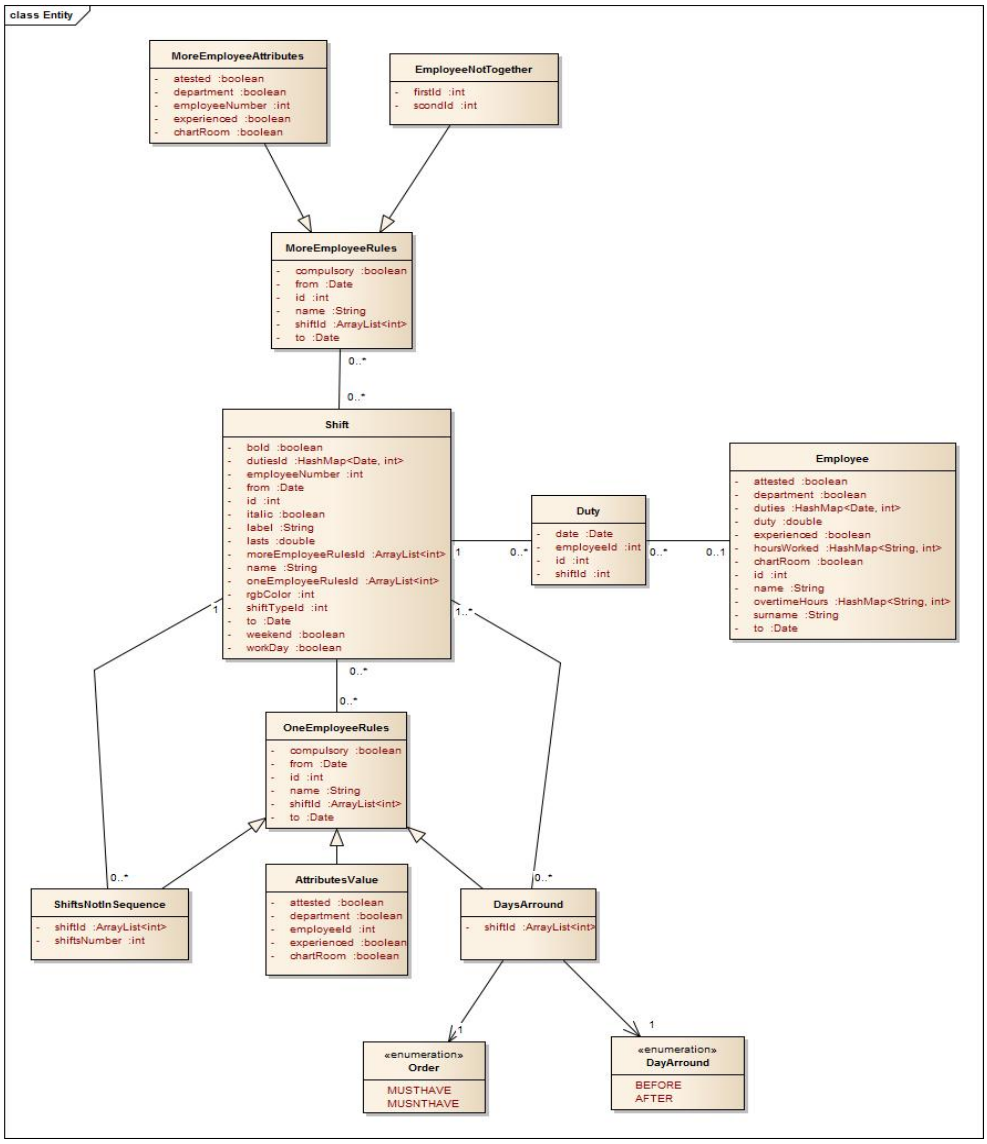
V nasledujúcej kapitole popíšem návrh entít z analytického hľadiska.

- **Zamestnanec (Employee)** Trieda obsahuje informácie o zamestnancovi – meno a priezvisko zamestnanca, možnosť pracovať na sále, možnosť pracovať na oddelení, úväzok, dostatok skúseností na danom pracovnom mieste a dostatok skúseností v obore (atestácia). Obsahuje tiež deň nástupu do práce, deň ukončenia pracovného pomeru, počet odpracovaných hodín za konkrétny mesiac a počet hodín nadčasu za tento mesiac.

- **Zmena (Shift)** Trieda obsahuje informácie o zmene. Názov zmeny, označenie (text, farba, tučné písmo, kurzíva), dĺžku trvania zmeny, počet zamestnancov potrebných na jej obsadenie, zoznam pravidiel aplikovaných na danú zmenu.

- **Výkon zmeny (Duty)** Trieda realizuje väzbu medzi zamestnancom a zmenou. Obsahuje dátum výkonu zmeny, identifikátor zamestnanca a identifikátor zmeny.

- **Pravidlo pre jedného zamestnanca (OneEmployeeRule)** Rodičovská trieda, ktorá združuje pravidlá na vyhodnotenie ktorých postačí



Obr. 2.8: Doménový model aplikácie

jeden zamestnanec, hodnoty jeho atribútov, prípadne väzby na iné entity.

Nasledujúce triedy sú potomkami triedy Pravidlo pre jedného zamestnanca, z ktorej dedia názov a príznak, či sa jedná o povinné pravidlo alebo nie:

- **Deň pred / po zmene (DaysAround)** Trieda reprezentuje pravidlo, ktorým vymedzíme aké zmeny zamestnanec musí či nesmie mať deň pred alebo deň po aktuálnom dni.
- **Hodnoty atribútov (AttributesValue)** Trieda umožňuje určiť, aké hodnoty atribútov má mať zamestnanec, ktorého chceme na zmenu dosadiť.
- **Počet zmien idúcich po sebe (ShiftsNotInSequence)** Trieda obsahuje informácie o počte zmien, ktoré nesmie mať zamestnanec po sebe, a zoznam zmien, ktorých sa to týka.
- **Pravidlo pre viacero zamestnancov (MoreEmployeeRule)** Rodičovská trieda, ktorá združuje pravidlá na vyhodnotenie ktorých potrebujeme viac zamestnancov.

  Nasledujúce triedy sú potomkami triedy Pravidlo pre viacero zamestnancov, z ktorej dedia názov a príznak, či sa jedná o povinné pravidlo alebo nie.
- **Atribúty viacerých zamestnancov (MoreEmployeeAttributes)** Trieda reprezentuje hodnoty atribútov, minimálny počet zamestnancov, ktorí musia mať dané hodnoty atribútov a zmeny, vrámci ktorých má toto pravidlo platiť.
- **Zamestnanci nepracujúci spolu (EmployeeNotTogether)** Trieda obsahuje identifikátor dvoch zamestnancov, ktorí nesmú pracovať spolu, a zmeny, ktorých sa toto obmedzenie týka.
- **Sviatok (FeriaeDay)** Trieda je určená na uchovanie dňa a mesiaca, na ktorý pripadá určitý sviatok.

Kapitola 3

Návrh

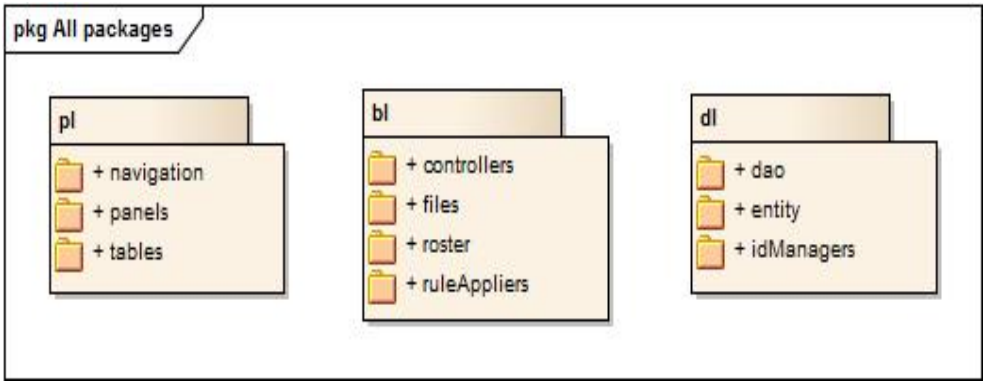
V nasludujúcej kapitole popíšem štruktúru balíčkov, do ktorých som rozdelil triedy, princíp spravovania entít a spôsob ich predávania naprieč všetkými vrstvami aplíkácie, a návrh tried zodpovedných za overovanie pravidiel.

3.1 Balíčky

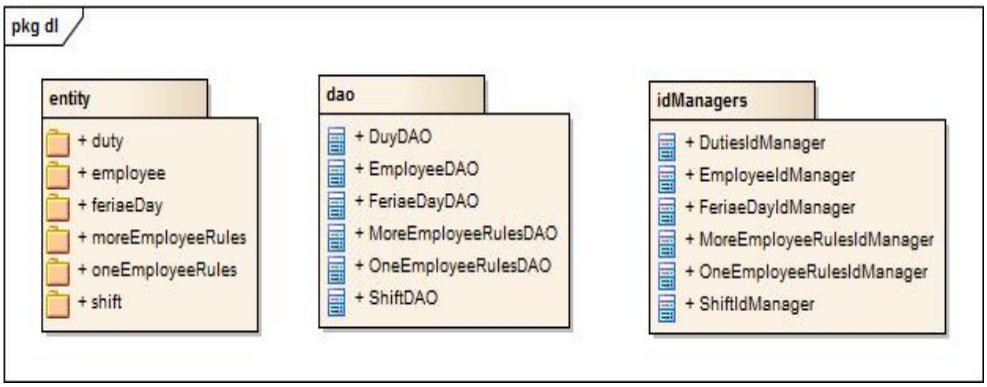
V nasledujúcej kapitole sa budem venovať návrhu tried. Triedy som rozdelil do nasledujúcich balíčkov:

3.1.1 Balíček dl

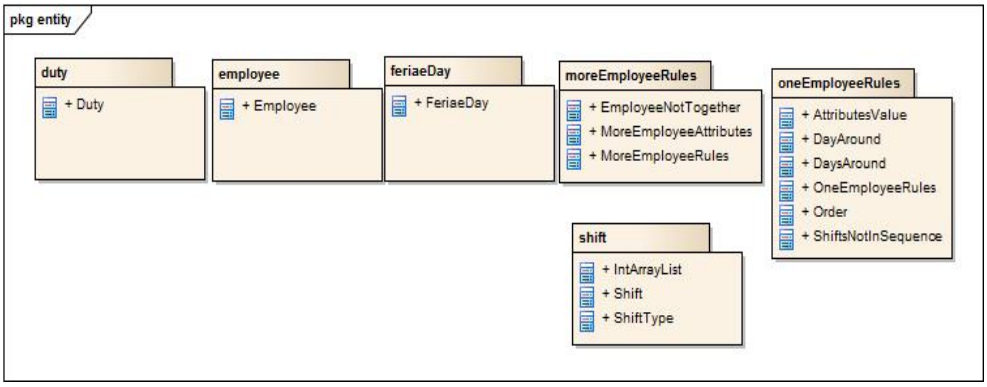
Nasledujúci balíček združuje balíčky, ktoré obsahujú triedy zodpovedné za datovú logiku aplikácie.



Obr. 3.1: Zoznam balíčkov, do ktorých sú rozdelené triedy



Obr. 3.2: Balíček obsahujúci triedy datovej vrstvy



Obr. 3.3: Balíček združujúci entity

3.1.1.1 Balíček entity

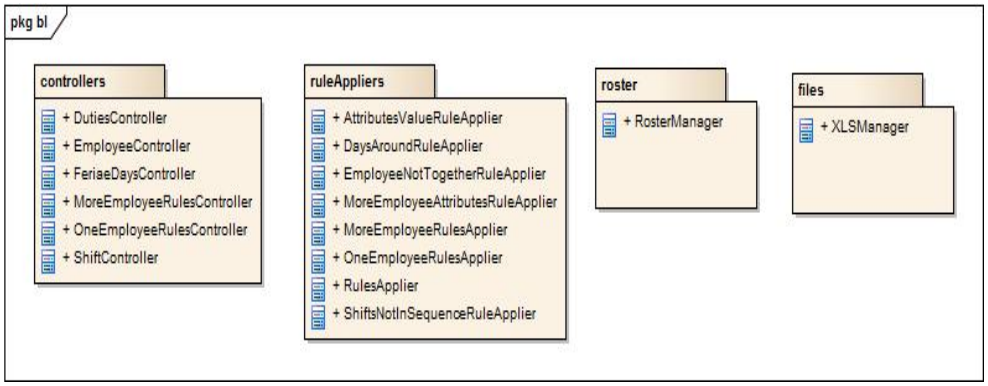
Nasledujúci balíček združuje balíčky, ktoré obsahujú triedy zodpovedné za datovú logiku aplikácie.

3.1.2 Balíček bl

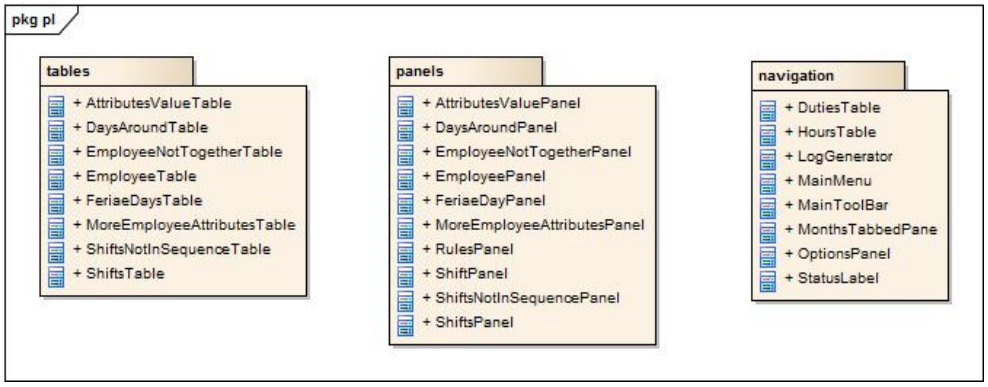
Nasledujúci balíček združuje balíčky, ktoré obsahujú triedy zodpovedné za business logiku aplikácie.

3.1.3 Balíček pl

Nasledujúci balíček združuje balíčky, ktoré obsahujú triedy zodpovedné za grafické užívateľské rozhranie systému.



Obr. 3.4: Balíček tried business vrstvy

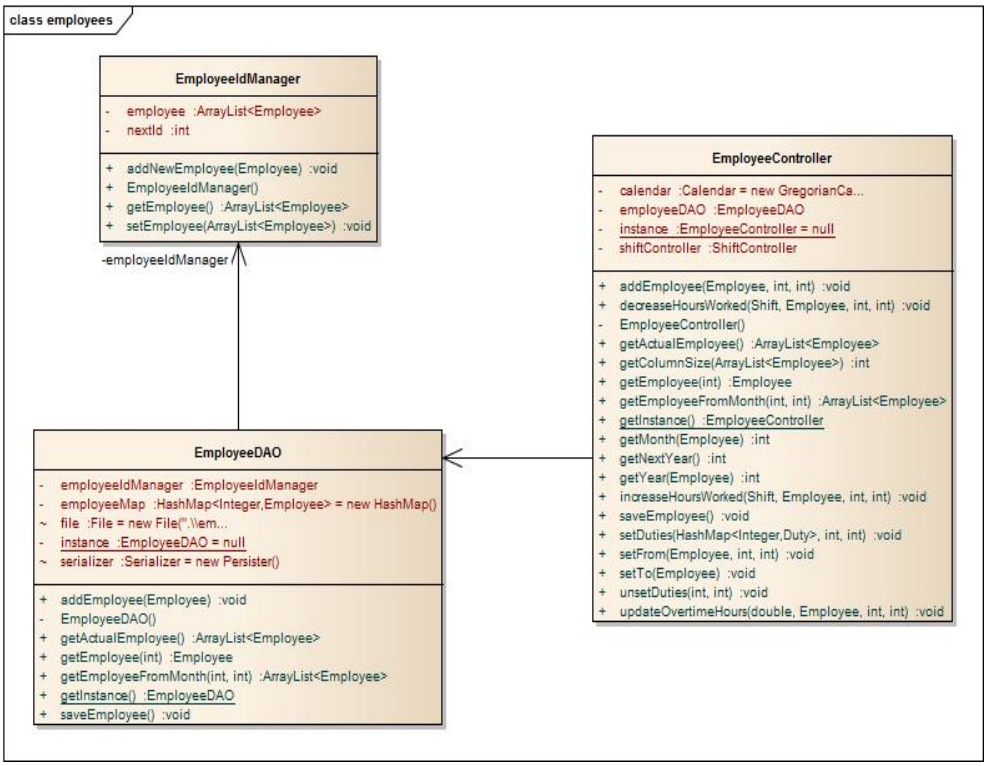


Obr. 3.5: Balíček tried prezentačnej vrstvy

3.2 Návrh tried zodpovedných za správu entít

V tejto kapitole podrobne popíšem triedy, ktoré sú zodpovedné za správu entít. Na úvod popíšem datovú reprezentáciu entít. Na ich serializovanie do formátu XML som použil knižnicu simpleXML.[7] Aby bolo toto možné, je potrebné do triedy pridať sériu anotácii:

- **@Root**: Označenie koreňového elementu, túto anotáciu pridávam nad definíciu triedy.
- **@Attribute**: Označenie atribútu koreňového elementu, túto anotáciu som použil na uchovanie identifikátoru entity.
- **@ElementList**: Umožňuje serializovať kolekcie, napr. ArrayList.
- **@ElementMap**: Hashovacie mapy je možné serializovať pomocou tejto anotácie.



Obr. 3.6: Diagram tried zodpovedných za správu zamestnancov

- **@Element**: Anotáciu som pridával nad všetky ostatné atribúty entity.

3.2.1 Entita Employee

Princíp spravovania entít je podobný, na ukážku som vybral entitu employee.

3.2.1.1 EmployeeIdManager

Trieda, ktorá je zodpovedná za prideľovanie jedinečného identifikátoru zamestnancom. Zároveň slúži ako „obalovacia trieda“ pre účely uloženia kolekcie zamestnancov do súboru s príponou xml. Obsahuje ArrayList zamestnancov employee a hodnotu ďalšieho identifikátoru uloženú v premennej nextId. Metóda AddNewEmployee uloží novo vytvoreného zamestnanca do kolekcie, a zároveň mu pridelí jedinečný identifikátor.

3.2.1.2 EmployeeDAO

Trieda je zodpovedná za načítanie zamestnancov zo súboru. Skladá sa z triedy IdManager, hashovacej mapy zamestnancov, kde kľúč ku konkrétnemu za-

mestnancovi tvorí identifikátor. Je navrhnutá ako singletone, takže obsahuje metódu getInstance, ktorá volá prívátny konštruktor tejto triedy. S touto instanciou pracuje trieda EmployeeController. V konštruktore sa pokúsim načítať IdManager zo súboru, ak súbor neexistuje, vytvorím nový IdManager. Následne uloží zamestnancov do hashovacej mapy. Pozostáva z týchto metód:

- **getActualEmployee**: Metóda vráti kolekciu zamestnancov, ktorých dátum platnosti od (from) je pred prvým dňom aktuálneho mesiaca (prípadne tento deň) a dátum platnosti do (to) je buď rovný poslednému dňu v aktuálnom mesiaci, väčší, prípadne nie je nastavený.

- **getEmployeeFromMonth**: Metóda vráti ArrayList zamestnancov, pre ktorých platí, že požadovaný mesiac sa nachádza medzi dátumom platnosti od (from) a dátumom platnosti do (to).

- **getEmployee**: Metóda, ktorej parametrom je identifikátor entity, vráti túto entitu z hashovacej mapy.

- **addEmployee**: Metóda pridá novú entitu najskôr do triedy IdManager, pomocou netódy add, ktorú som popisoval vyššie, čím jej bude pridelený jedinečný identifikátor. Následne uložíme túto entitu aj do hashovacej mapy.

- **saveEmployee**: Metóda uloží triedu EmployeeIdManager do súboru. Nový súbor nahradí pôvodný.

3.2.1.3 EmployeeController

Trieda zodpovedná za business logiku spojenú so správou zamestnancov. Jedná sa o singletone, takže referenciu na jedinú instanciu tejto triedy si „vypýtam“ vo chvíli, keď ju potrebujem (v triedach prezentačnej logiky a business logiky spojených so správou zamestnancov). Využíva instanciu triedy EmployeeDAO, z ktorej získava požadovaných zamestnancov pomocou metód popísaných vyššie.

- **unsetDuties**: Metóda, ktorá pre požadovaný mesiac odstráni všetky väzby medzi entitami employee a duty. Metóda je volaná vo chvíli, keď chceme odstrániť rozpis z určitého mesiaca, alebo pred importovaním dát zo súboru.

- **setDuties**: Metóda prijíma parametrom kolekciu entít duties, rok a mesiac. Následne vytvorí väzbu medzi každou entitou duty z tejto kolekcie a zamestnancom, ktorého identifikátor je v nej uložený. Táto metóda je volaná vo chvíli, keď načítam data zo súboru.

- **increaseHoursWorked**: Metóda zvýši počet odpracovaných hodín zamestnanca za požadovaný mesiac o dĺžku trvania zmeny na ktorú sme zamestnanca dosadili. Je volaná vo chvíli, keď zamestnancovi zmenu priradíme (ručné priradenie, pri automatickom generovaní alebo pri importe dát zo súboru).

- **decreaseHoursWorked**: Metóda naopak zníži počet odpracovaných hodín zamestnanca za požadovaný mesiac o dĺžku trvania zmeny. Volaná je vo chvíli, keď chceme odstrániť väzbu medzi zamestnancom a touto zmenou.

- **updateOvertimeHours**: Metóda upraví počet hodín, ktoré zamestnanec odpracoval nad / pod požadovanú normu. Volaná je vždy po zvyšovaní/znižovaní počtu odpracovaných hodín.

3.2.2 Entita Duty

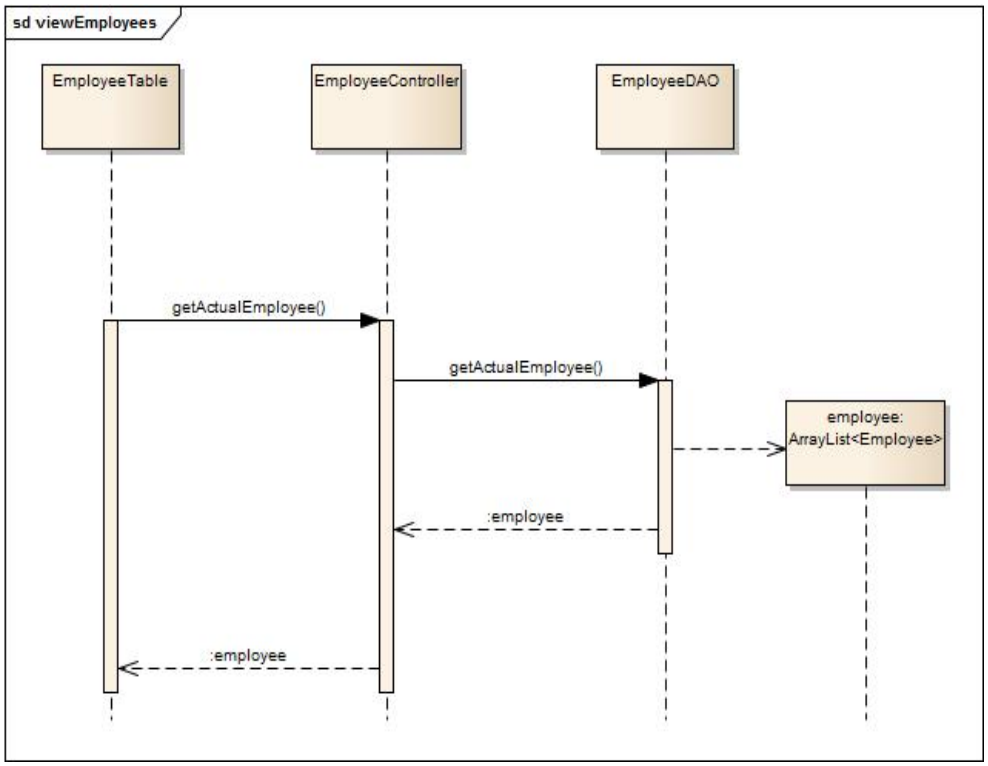
V tejto podkapitole popisujem odlišnosti pri spáve entít Duty oproti iným entitám.

3.2.2.1 DutyDAO

Špecifickým prípadom je trieda DutyDAO. Pri entite duty je použitá dvojitá identifikácia. Každý výkon zmeny je určený dátumom a identifikátorom, ktorý je jednoznačný vrámci určitého mesiaca. Jednotlivé entity sú uložené do rôznych súborov, podľa mesiaca. Metóda getDutiesFromMonth, ktorej parametrami sú mesiac a rok, najskôr zistí, či je v hashovacej mape uložený požadovaný IdManager. Ak nie, načíta IdManager zo súboru, ktorého názov je tvorený rokom a mesiacom `(rok_mesiac)`, ktorý je uložený v zložke pomenovanej podľa roku. Triedu dutiesIdManager vráti triede DutiecController, ktorá s ňou bude pracovať ďalej. Pomocou metódy saveDuties uloží idManager predaný parametrom do príslušnej zložky.

3.2.2.2 DutiesController

Na rozdiel od iných Controllerov trieda DutiesController nie je singletone. Tiež zodpovedá za triedu DutiesIdManager, z ktorej si uloží entity Duties do hashovacej mapy. Koštruktor je buď bezparametrický, alebo je v parametre zadaný rok, mesiac a DutiesIdManager. Za zmienku stojí metóda SetMonth, ktorá si od triedy DutyDAO vyžiada idManager obsahujúci väzby medzi zmenami a zamestnancami za mesiac, ktorý jej predá parametrom. Následne si tieto uloží do hashovacej mapy, s ktorou ďalej pracuje.



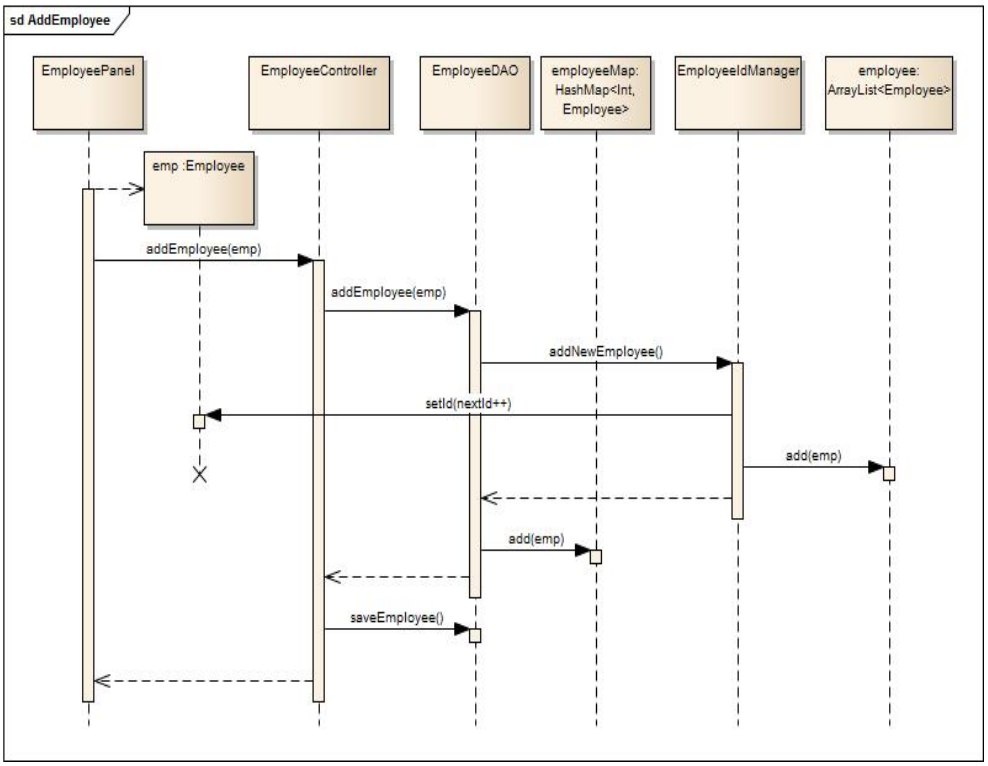
Obr. 3.7: Sekvenčný diagram znázorňujúci proces načítania zoznamu aktívnych zamestnancov

3.3 Zobrazovanie a pridávanie entít do systému

Zoznam entít som navrhol ako tabuľku, ktorá má v hlavičke atribúty danej entity. V každom riadku je zobrazená jedna konkrétna entita. Na konci tabuľky sa nachádza tlačidlo „Editovat“, po kliknutí naň sa zobrazí formulár s predvyplnenými údajmi o tejto entite. Tlačidlo „Odstranit“, ktoré odstráni entitu zo zoznamu. V prípade zmien obsahuje tlačidlo, ktoré zobrazí formulár na ktorom je pomocou checkboxov možné vytvoriť väzbu medzi zmenou a pravidlami. Podobným spôsobom je možné vytvoriť väzbu medzi pravidlom a zmenami vrámci ktorých chceme toto pravidlo aplikovať.

3.3.0.3 Zobrazenie zoznamu zamestnancov

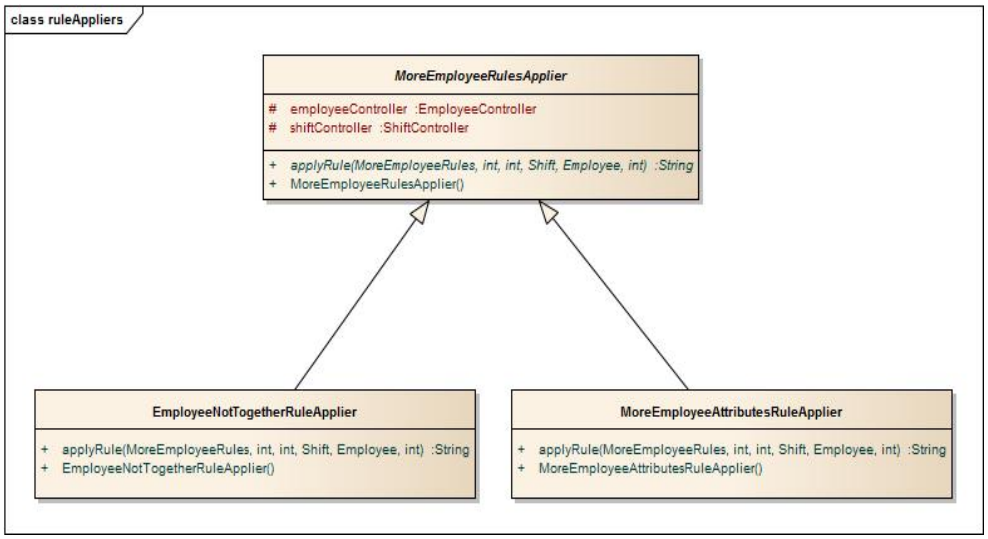
Po kliknutí na tlačidlo „Seznam zaměstnanců“ v hlavnom menu zavolá trieda EmployeeTable metódu getActualEmployee nad triedou EmployeeController, ktorá si pomocou triedy EmployeeDAO a rovnomennej metódy načíta zoznam aktívnych zamestnancov. Tento zoznam vráti tabuľke, ktorá ho zobrazí užívateľovy spôsobom, ktorý som popísal vyššie.



Obr. 3.8: Sekvenčný diagram zobrazujúci proces pridania nového zamestnanca do systému

3.3.0.4 Proces pridania zamestnanca do systému

Celý proces začína vo chvíli, keď užívateľ klikne na tlačidlo „Uložit“ na formulári pre pridanie zamestnanca so správne vyplnenými údajmi. Trieda EmployeePanel vytvorí novú instanciu triedy Employee. Túto predá triede EmployeeController pomocou metódy AddEmployee. Controller ju rovnomennou metódou predá triede EmployeeDAO, ktorá zavolá metódu addNewEmployee nad triedou EmployeeIdManager, ktorá si uloží entitu do kolekcie, a nastaví jej jedinečný identifikátor, uložený v premennej nextId, ktorú následne inkrementuje o 1. Premennú employee s prideleným identifikátorom následne trieda EmployeeDAO uloží do vlastnej hashovacej mapy employeeMap. Trieda EmployeeController potom zavolá metódu saveEmployee nad Triedou EmployeeDAO, ktorá uloží modifikovanú instanciu triedy EmployeeIdManager do súboru.



Obr. 3.9: Diagram znázorňujúci triedy obsahujúce business logiku overovania pravidiel pre viacero zamestnancov

3.4 Návrh tried zodpovedných za overovanie pravidiel

Táto kapitola popisuje princíp overovania pravidiel a generovania srpáv o ich porušení v logu.

3.4.1 Princíp overovania pravidiel

Na oddelenie business logiky od dátovej som použil triedy OneEmployeeRulesApplier a MoreEmployeeRulesEmplayer, ktorých potomkovia preťažujú metódu applyRule. Tej predáme parametrom entitu, obsahujúcu datovú reprezentáciu pravidla, a ostatné informácie potrebné na jeho vyhodnotenie. Za vyhodnotenie pravidiel, ktoré sú priradené nejakej zmene zodpovedá trieda RulesApplier. Tá overuje dodržiavanie pravidiel vo chvíli, keď sa zmenu pokúšame obsadiť nejakým zamestnancom. Z tohoto dôvodu je v určitých prípadoch nutné definovať dvojicu pravidiel. Ak napr. nechceme, aby po nočnej zmene nasledovala denná, tak je nutné definovať tiež pravidlo, ktoré overí, či deň pred dennou zmenou nebol zamestnanec dosadený na nočnú zmenu.

3.4.2 Návrh logu chybových správ

Správy v logu sa delia podľa toho, či upozorňujú na neobsadené / preplnené zmeny, alebo na porušenie pravidiel. Tak isto sa delia podľa závažnosti (podľa toho, či bolo porušené povinné alebo nepovinné pravidlo). Trieda LogGenerator najskôr skontroluje obsadenie zmien a následne si nechá od triedy Ru-

lesApplier vygenerovať správy o porušených pravidlách v konkrétny deň. Pri načítaní tabuľky naplánovaných zmien sa tieto správy vygenerujú pre každý deň. Každá správa obsahuje tlačidlo, ktoré v tabuľke zvýrazní stĺpec, prípadne konkrétne bunky v ktorých k danému problému došlo. Nasleduje dátum, a správa, ktorá obsahuje meno zamestnanca, názov zmeny, na ktorú bol dosadený, a názov pravidla, ktoré tým bolo porušené. Červenou farbou sú zvýraznené porušenia povinných pravidiel, oranžovou nepovinných. Po dosadení konkrétneho zamestnanca na zmenu stačí skontrolovať obsadenie smien v tomto dni, porušenie pravidiel typu OneEmployeeRules v riadku (pre tohto zamestnanca) a pravidlá MoreEmployeeRules v stĺpci (pre deň, v ktorý sme zamestnanca na zmenu dosadili).

Kapitola 4

Realizácia

Kapitola obsahuje detailný popis overovania pravidiel, spôsob, akým som postupoval pri automatickom rozdeľovaní zmien, v závere rozoberám spôsob, akým som plán zmien importoval a exportoval.

4.1 Overovanie pravidiel

V tejto kaptole popisujem princíp overovania jednotlivých druhov pravidiel.

- **AttributesValueRuleApplier**: Trieda je zodpovedná za overenie pravidla AttributesValue. Metóda applyRule využije pravidlo a zamestnanca. Porovná požiadavky na atribúty definované pravidlom a hodnoty atribútov zamestnanca, a v prípade rozdielu vráti chybovú hlášku. V opačnom prípade vráti hodnotu null.

- **DaysAroundRulesApplier**: Trieda je určená na overenie pravidla DaysAround. Metóda applyRule v tomto prípade využije pravidlo, zamestnanca, a deň, ktorý je určený rokom, mesiacom a dňom v týždni. Metóda sa prvý krát vetví podľa hodnoty výčtu Order: switch(DaysAround.getOrder()) case:MUSTHAVE case:MUSNTHAVE. Následne sa vetví podľa hodnoty výčtu DayAround. Ak je hodnota AFTER, nastaví kalendár na nasledujúci deň, ak BEFORE, nastaví kalendár na deň predchádzajúci. V každom prípade zistí, či mal zamestnanec v požadovaný deň nejakú zmenu (employee.getDutyIds().containsKey(date.getTime())). Ak áno, pomocou realizovaných väzieb medzi entitami zistí o akú zmenu sa jedná – triedu DutiesController nastaví na príslušný mesiac metódou setMonth (pre prípad pretečenia do nasledujúceho / predchádzajúceho mesiaca).

- **ShiftsNotInSequenceRuleApplier**: Trieda je zodpovedná za overenie pravidla ShiftsNotInSequence. Metóda applyRule využije pravidlo, zamestnanca, a deň, v ktorý chceme zamestnancovi zmenu priradiť. Kalendár nastaví na tento deň. V cykle najskôr posúva kalendár do minulosti.

Pre každý deň nastaví triedu DutiesController na daný mesiac pomocou metódy setMonth – pre prípad pretečenia do predchádzajúceho mesiaca. Pomocou väzieb zistí, akú zmenu mal zamestnanec priradenú. Ak je zmena definovaná pravidlom, zvýši čítač zmien a pokračuje v kontrole až kým čítač nepovolených zmien nedosiahne hodnotu určenú pravidlom, alebo nenarazí na zmenu, ktorá pravidlom definovaná nie je, či prípad, že zamestnanec nemal žiadnu zmenu. Podobným spôsobom overí dni v budúcnosti. V prípade, že čítač dosiahol hodnotu uvedenú v pravidle, vráti metóda chybovú hlášku. V opačnom prípade vráti hodnotu null.

- **EmployeeNotTogetherRuleApplier**: Trieda realizuje overenie pravidla EmployeeNotTogether. Metóda applyRule potrebuje pravidlo, zamestnanca a deň. V prvom rade zistí, či zamestnanec, ktorého sa pokúšame dosadiť na zmenu, je jedným zo zamestnancov, ktorého nechceme na danej zmene s iným zamestnancom. Ak áno, pomocou triedy EmpoyeeController, id druhého zamestnanca uvedeného v pravidle a realizovaných väzieb zistí, či mal druhý zamestnanec v daný deň zmenu, a či je táto zmena definovaná pravidlom. V tom prípade vráti metóda chybovú hlášku, inak null hodnotu.

- **MoreEmployeeAttributesRuleApplier**: Trieda je zodpovedná za overenie pravidla MoreEmployeeAttributes. Metóda applyRule využije pravidlo, zamestnanca a deň. Najskôr si načíta zamestnancov z požadovaného mesiaca. Z nich postupne odstráni zamestnancov, ktorí nemajú v daný deň priradenú niektorú zo zmien definovanú pravidlom. Pridá k nim zamestnanca, ktorého chceme dosadiť na jednu z týchto zmien. Do premennej haveFulfilled si uloží počet zamestnancov, ktorí majú hodnoty atribútov zhodné s hodnotami atribútov definovaných pravidlom. Premenná employeeNumber vyjadruje počet zamestnancov potrebných na obsadenie všetkých zmien uvedených v pravidle. Porovná počet zamestnancov, ktorých je ešte potrebné na zmeny dosadiť (employeeNumber – employees.size()) s počtom zamestnancov, ktorí musia spĺňať požiadavky na hodnoty atribútov (moreEmployeeAttributes.getMustFulfil() - haveFulfilled). Ak je prvá hodnota menšia (v budúcnosti sa môže objaviť dostatok zamestnancov, ktorý budú mať požadované hodnoty atribútov), pravidlo je vyhodnotené ako splnené. V opačnom prípade metóda vráti chybovú hlášku. Ak na denných zmenách potrebujeme 2 skúsených zamestnancov, na obsadenie zmien sú potrební 4 zamestnanci, môžeme bez obáv dosadiť 2 neskúsených, ale vo chvíli, keď dosadíme 3. zamestnanca, ktorý bude mať hodnotu atribútu experienced nastavenú na false, dôjde k porušeniu pravidla.

4.2 Rozradzovací algoritmus

Nasledujúca kapitola obsahuje popis a implementáciu algoritmu použitého na obsadenie zmien zamestnancami.

4.2.1 Vlastnosti algoritmu

Navrhovaný algoritmus by mal zabezpečiť optimálne rozdelenie zamestnancov na zmeny. Túto optimalitu posudzujeme z 3 hľadísk:

- **Dodržiavanie pravidiel**: Optimálne rozdelenie je to, pri ktorom bude porušených čo najmenej nepovinných pravidiel (žiadne povinné pravidlo nemôže byť porušené).
- **Rovnomerné rozdelenie zmien**: Optimálne je to rozdelenie, pri ktorom sú zmeny medzi zamestnancov rozdelené rovnomerne (zamestnanec nebude mať všetky zmeny na začiatku mesiaca a na konci voľno).
- **Počet odpracovaných hodín**: Zamestnancom budú zmeny prideľované tak, aby neboli v počte ich odpracovaných hodín veľké rozdiely. Taktiež je nutné rešpektovať úväzok, na ktorý zamestnanec pracuje.

Nie vždy je možné nájsť riešenie, ktoré by bolo optimálne zo všetkých 3 hľadísk, preto je potrebné nájsť nejaký kompromis. Ako základ som zvolil heuristiku, ktorá sa snaží prideliť na zmenu toho zamestnanca, ktorý ju absolvoval najdávnejšie.

4.2.2 Implementácia algoritmu

Algoritmus realizuje trieda RosterManager pomocou metód, ktoré detailne popíšem v nasledujúcich riadkoch.

- **ShiftEmployeePriority**: Privátna trieda, ktorá slúži na uchovanie informácii o tom, s akou vysokou prioritou chceme určitého zamestnanca priradiť na konkrétnu zmenu. Pre zjednodušenie práce tiež obsahuje hodnotu úväzku na ktorý zamestnanec pracuje.
- **makeRoster**: Metóda, ktorá je zodpovedná za obsadenie zmien zamestnancami. Na začiatku rozdelí zmeny na tie, ktoré sú vykonávané cez víkend a ktoré v pracovný deň. Následne sa v cykle posúva po dni do minulosti. Pre každý deň volá metódu scanOneDayPriorities, v závislosti od toho, či sa jedná o pracovný deň alebo víkend. Cyklus pokračuje ďalej, kým nie sú nastavené všetky priority – metóda scanOneDayPriorities vráti hotnotu true, alebo sme narazili na ukončovaciu podmienku, ktorou je „preskúmanie“ 90 dní do minulosti. Počet dní som zvolil odhadom – 3 mesiace do minulosti som pokladal za dostatočne dlhú dobu. Ak sme na určitú kombináciu zmena – zamestnanec nenarazili, pomocou metódy

bigPriorities nastavíme prioritu (90 * úväzok zamestnanca) premennej shiftEmployeePriority s touto kombináciou. Nasleduje samotné rozdelenie zmien, ktoré realizuje metóda occupateDuties, ktorá je volaná zvlášť pre víkendové a zvlášť pre týždenné zmeny. Pole priorít následne zoradí metóda sortPrioritiesArray. Túto kombináciu metód volá pre každý deň v mesiaci.

- **SetOneDayPriorities**: Metóda pre zadaný deň skontroluje všetky zmeny. Ak narazí na kombináciu zmena - zamestnanec, ktorá sa ešte v zozname nenachádza, uloží ju. Do novo vytvorenej premennej shiftEmployeePriority uloží identifikátor tejto zmeny a zamestnanca, prioritu (hodnotu, ktorá začína na 1 a pri každom posune do minulosti iterujeme o 1) vynásobenú hodnotou úväzku, na ktorý zamestnanec pracuje. Novú premennú následne umiestni do poľa na správne miesto tak, aby bolo zoradené vzostupne.

- **occupateDuties**: Do ArrayListu occupatedShifts si uloží tie zmeny, ktoré sú už obsadené (je nežiaduce, aby algoritmus menil to, čo užívateľ ručne nastavil, prípadne načítal zo súboru). Následne prechádza pole shiftsEmployeesPriorities od najvyššieho indexu, na ktorom sú uložené informácie o zmene, ktorú chcem obsadiť určitým zamestnancom s najvyššou prioritou. Ak zmena nie je obsadená, a zamestnanec nemá v tento deň priradenú inú zmenu, overí, či priradením zmeny zamestnancovi nedôjde k porušeniu nejakého užívateľom definovaného pravidla. Ak nie, priradí zamestnanca na zmenu, upraví počet jeho odpracovaných hodín a uloží identifikátor zmeny do kolekcie obsadených zmien. Takto pokračuje, kým neobsadí všetky zmeny, alebo neprejde celé pole. Ak neboli obsadené všetky zmeny, pri druhom prechode pola postupuje rovnakým spôsobom s tým rozdielom, že kontroluje iba dodržanie povinných pravidiel. Pri každom obsadení zmeny vynuluje prioritu a presunie prvok na koniec poľa. Na záver zvýši každú prioritu o hodnotu úväzku zamestnanca.

- **SortPrioritiesArray**: Jedným z parametrov metódy je pole ShiftsEmployeesPriorities, ktoré je potrebné zoradiť podľa priorít vzostupne. Pri prechode na ďalší deň zvýši každému prvku jeho prioritu o hodnotu úväzku zamestnanca. Môže sa stať, že zamestnanec s vyšším úväzkom predbehne zamestnanca s úväzkom nižším. Algoritmus prechádza pole od najvyššieho indexu. V prípade, že narazí na prvok s najvyšším úväzkom (v našom prípade 1) preskočí ho. Ak je úväzok nižší, pokračuje v poli nižšie, až kým nenarazí na prvok s úväzkom 1, ktorého priorita je nižšia (prípadne rovná) alebo neprejde celé pole. Ak narazí na prvok s vyššou prioritou, zaznamená si jeho index, a na záver tieto dva prvky prehodí. Podobným spôsobom pokračuje, kým neprejde celé pole.

4.3 Import a export

V tejto kapitole popisujem, akým spôsobom som realizoval import a export plánu zmien do súboru. Na úvod poviem pár slov o formáte súboru. V dokumente s príponou XLS sú excelovské sheety radené zostupne, tzn. prvý sheet obsahuje rozpis z decembra. Ďalej je nutné, aby mená zamestnancov boli v prvom stĺpci, plán rozpisu začínal v 5. stĺpci, skutočnosť v 4. stĺpci riadok pod plánom. Ak chce užívateľ správne importovať rozpis napr. z roku 2013, musí byť tento rok aktuálne zobrazený v systéme. V opačnom prípade import a export nebudú fungovať. Pre lepšie pochopenie formátu prikladám v prílohe na cd reálny rozpis služieb z roku 2013. Sprvu som uvažoval o importovaní a exportovaní do CSV formátu. Ten však neumožňuje jednoduchým spôsobom rozlišovať kurzívu. Preto som zvolil knižnicu Apache POI, ktorá umožňuje načítať data priamo z excelovského súboru, napr. vo formáte XLS.[1] To nám umožňuje trieda HSSFWorkbook, pomocou ktorej si súbor v tomto formáte načítame.[4] Trieda HSSFSheet slúži na načítanie konkrétneho excelovského sheetu, s ktorým budeme ďalej pracovať. [3]

- **Import dát zo súboru**: V prvom rade načítame sheet z požadovaného mesiaca. Následne iterujeme jedným cyklom riadky a druhým cyklom bunky v tomto riadku. Reťazec z bunky porovnáme s menami zamestnancov (biele znaky sú ignorované, nezáleží na poradí mena a priezviska). Ak bolo rozpoznané meno nejakého zamestnanca, posunieme sa o 4 stĺpce v prípade, že chcel užívateľ načítať plán rozpisu. V prípade, že chcel načítať skutočnosť, posunieme sa o riadok nižie a následne o 3 stĺpce doprava. Preskúmam počet buniek zhodných s počtom dní v danom mesiaci. Ak sa text v bunke zhoduje s označením nejakej zmeny, a štýl bunky je zhodný so štýlom označenia tejto zmeny (rozlišujeme iba kurzívu, farbu písma a tučný štýl ignorujeme) definujeme premennú duty, ktorej nastavíme dátum, identifikátor rozpoznanej zmeny a zamestnanca. Tú následne uložíme do idManagera, ktorý metóda po preskúmaní celého súboru vráti. Ak narazíme na označenie, ktoré sa nezhoduje s označením žiadnej zmeny, ignorujeme ho.
- **Export dát do súboru**: Opäť prechádzame tabuľku bunku po bunke, keď rozpoznáme meno nejakého zamestnanca, nastavíme cellIterator na požadovanú bunku. Riadok prechádzame bunku po bunke (deň po dni) a kontrolujeme, či zamestnanec mal v daný deň priradenú nejakú zmenu. Ak áno, do premennej typu HSSFCellStyle uložíme štýl bunky a z tejto premennej získame jej font.[2] Následne pre bunku vytvoríme vlastný štýl a font, ktorý nastavíme na kurzívu, podľa toho, či je zmena rozlíšená kurzívou alebo nie. Ak nemal zamestnanec pridelenú žiadnu zmenu, uložíme do bunky prázdny reťazec (v tomto prípade je žiaduce prepísanie pôvodného súboru).

Kapitola 5

Testovanie

V nasledujúcich riadkoch zmienim, akým spôsobom som testoval funkčnosť systému.

5.1 Manuálne testovanie

Pre účely testovania som v systéme vytvoril konkrétne zmeny a pravidlá tak, ako ich popisujem v podkapitole 2.1. Zoznam zamestnancov zodpovedal reálnemu zloženiu zamestnancov. Pre otestovanie importu a exportu som použil reálny rozpis služieb gynekologicko pôrodníckeho oddelenia nemocnice na Mělníku, uložený vo formáte XLS.

5.1.1 Import zo súboru

Všetky zmeny boli načítané, pôvodný rozpis uložený v systéme bol nahradený. Zamestnancom bol upravený počet odpracovaných hodín, log obsahoval správne vygenerované chybové hlášky o porušených pravidlách a neobsadených či preplnených zmenách.

5.1.2 Export do súboru

Upravený rozpis som exportoval do rovnakého súboru. Pôvodné označenia zmien v ňom boli nahradené označeniami zo systému. Štýl písma zostal nezmenený – s výnimkou kurzívy, ktorá bola nastavená tam, kde to bolo potrebné.

5.1.3 Automatické generovanie rozpisu

Automatické generovanie rozpisu som testoval v prípade, že neboli v systéme žiadne rozpisy z minulosti, boli importované reálne rozpisy z minulosti (3 mesiace) a v prípade, že časť zmien bolo obsadených zamestnancami. Výsledky som zhodnotil nasledovne:

- **Obsadenie zmien**: Všetky zmeny boli obsadené, s výnimkou prípadu, keď to nebolo možné (nebol k dispozícii dostatočný počet zamestnancov, ak som napr. každému pridelil na konkrétny deň dovolenku).

- **Dodržiavanie pravidiel**: Nebolo porušené žiadne pravidlo, nepovinné pravidlá (zmena 3 dni po sebe, nočná zmena po dennej) boli porušené len výnimočne.

- **Počet odpracovaných hodín**: Zamestnancom, ktorí pracujú na polovičný úväzok bolo pridelených menej zmien ako zamestnancom pracujúcim na plný úväzok.

- **Prepisovanie vopred pridelených zmien**: Väzby medzi zamestnancami a zmenami, ktoré boli vytvorené pred spustením algoritmu neboli nijako porušené.

5.1.4 Pravidlá

V nasledujúcej podkapitole popisujem testovanie každého druhu pravidla.

- **AttributesValue**: V systéme som definoval pravidlo, ktoré overuje, či zamestnanec, ktorého sa pokúšam dosadiť na zmenu, je kvalifikovaný na prácu na oddelení. Následne som toto pravidlo priradil dennej zmene na oddelení. Keď som sa pokúsil na zmenu dosadiť zamestnanca, ktorý mal hodnotu atribútu department nastavenú na false, bol som upozornený na to, že bude porušené nejaké pravidlo. Po odsúhlasení bol zamestnanec dosadený na zmenu a v logu sa objavila správa o porušení tohto pravidla. Ak bola hodnota atribútu u zamestnanca nastavená na true, zmena mu bola okamžite pridelená.

- **ShiftsNotInSequence**: Definoval som si toto pravidlo vrámci denných zmien, ako počet dní som zvolil 3. Pravidlo bolo týmto denným zmenám automaticky pridelené. Bol som upozornený na jeho porušenie v prípade, ak som priradil 3. zmenu v rade pred, za aj medzi 2 už pridelené zmeny. Tak isto som bol upozornený na prelome mesiacov či rokov.

- **DaysAround**: Skúsil som definovať toto pravidlo na deň pred aj po zmene, s hodnotou výčtu Order nastavenou na MUSTHAVE i MUSNTHAVE. Dodržiavanie pravidla bolo správne kontrolované, a to i na prelome mesiacov a rokov.

- **MoreEmployeeAttributes**: V rámci nočných zmien som definoval pravidlo, ktoré zaistí, že na týchto zmenách bude aspoň 1 skúsený zamestnanec. Pri 1. neskúsenom zamestnancovi sa nedialo nič, pri 2. som bol upozornený na porušenie tohto pravidla.

- **EmployeeNotTogether**: V rámci denných zmien som definoval toto pravidlo pre 2 zamestnancov, ktorým nič iné nebránilo v spoločnej práci na týchto zmenách. Na jeho porušenie som bol upozornený po pridelení druhého zamestnanca.

Kapitola 6

Využitie a budúcnosť práce

V nasledujúcej kapitole zhrniem prínos systému pre vedúcich pracovníkov, ďalej sa budem zaoberať vylepšeniami a úpravami súčasnej verzie systému, a tiež tým, kam by sa jeho vývoj mohol posunúť v budúcnosti.

6.1 Manažérske a ekonomické prínosy

Implementovaný systém prinesie vedúcemu pracovníkovi najmä úsporu času a sprehľadnenie jeho práce. Prvým momentom, ktorý prinesie úsporu času, je získavanie požiadavkov na zmeny a dovolenky od zamestnancov. V súčastnosti si tieto evidujú do zošita, z ktorých si ich musí sám triediť a prepisovať do elektronickej podoby. Vedúcemu aj zamestnancom by ušetrilo čas, ak by si svoje požiadavky na požadované zmeny, voľné dni a čerpanie dovolenky zapisovali do vopred pripraveného google dokumentu, z ktorého by si tieto mohol skopírovať do svojej šablóny, následne importovať do systému. Pomocou logu má možnosť skontrolovať, či tieto požiadavky navzájom nekolidujú / neporušujú pravidlá. V druhom rade ušetrí čas pri automatickom generovaní. Výsledný plán síce vyžaduje kontrolu, táto však zaberie menej času ako jeho ručné zostavovanie. Ďalšiu úsporu času predstavuje zmena naplánovaného rozpisu. Ak napríklad zrušíme zamestnancovi naplánované zmeny kvôli chorobe, automatický generátor obsadí uvoľnené zmeny inými zamestnancami. Prepis požiadavkov zaberie vrchnej sestre, pre ktorú som tento systém implementoval, asi 1 hodinu. Samotné plánovanie zmien trvá minimálne 2 hodiny, úprava plánu v priebehu mesiaca trvá približne rovnako. Ak odrátame čas potrebný na úpravy plánu v systéme, úspora času predstavuje minimálne 4 hodiny mesačne.

6.2 Rozšírenia súčasnej podoby aplikácie

Súčasná verzia systému nie je dokonalá. V tejto kapitole sa zaoberám niektorými z týchto nedokonalostí, ktorých odstránenie by zaistilo jeho lepšie využitie.

- **ShiftsSet**: V systéme by bola možnosť priradiť zamestnancovi množinu zmien, ktoré môže vykonávať, na ostatné by ho bolo možné priradiť po predchádzajúcom upozornení. Automatický generátor by zamestnancov na zmeny, ktoré nemajú v tejto množine, nepriraďoval.

- **DayShiftsSet**:V súčasnej verzii systému je možné zadať, akú zmenu zamestnanec chce, prípadne mu priradiť špeciálnu zmenu „voľno“, takže mu automatický generátor žiadnu zmenu nepriradí. Pomocou tohto nástroja by bolo možné definovať množinu zmien, na ktoré zamestnanec v určitý deň nechce / nemôže nastúpiť. Ak napr. nebude môcť absolvovať dennú zmenu (lekárske vyšetrenie) ale nočnú zmenu áno, bude v tejto množine zoznam denných zmien. Automatický generátor potom narazí na túto množinu, ak bude zamestnanec najvhodnejším kandidátom na absolvovanie nejakej dennej zmeny z tejto množiny, zmenu mu nepriradí. Ak však bude vhodným kandidátom na nočnú zmenu, pridelí mu ju.

- **Úprava rozradzovacieho algoritmu**:

  - **Počet odpracovaných hodín**: Rozšíriť súčasnú heuristiku o ďalšiu, ktorá bude strážiť počet odpracovaných hodín u zamestnancov, a prideľovať zmeny prednostne tým, ktorí odpracovali menej hodín ako by mali.
  - **Počet dní voľna**: Heuristika, ktorá bude strážiť počet dní, počas ktorých nebola zamestnancovi priradená žiadna zmena. Pri určitom počte takýchto dní bude zamestnancovi priradená nejaká zmena (pri ktorej nebude porušené žiadne pravidlo a ktorá bude inou heuristikou vybraná ako najvhodnejšia pre tohto zamestnanca).

- **Pravidlo MoreEmployeeAttributes**: Súčasné pravidlo funguje tak, že zabezpečí na zmene určitý počet zamestnancov, ktorý spĺňajú určité požiadavky na hodnotu atribútov, prípadne viac. Bolo by vhodné mať možnosť špecifikovať, či vyžadujeme „aspoň“, „práve“ alebo „maximálne“ toľko zamestnancov s požadovanými vlastnosťami.

- **Role zamestnancov**: V súčasnej verzii systému je nutné robiť rozpis zmien pre rôzne role zamestnancov zvlášť – zvlášť pre zdravotné sestry, zvlášť pre doktorov či sanitárov. Ak by chcel užívateľ na nočnej zmene dve sestričky, jednu sanitárku a jedného doktora, mohol by každému zamestnancovi definovať rolu (napr. pomocou atribútu) a následne použiť

na zmenu pravidlo MoreEmployeeAttributes s úpravou, ktorú popisujem vyššie.

6.3 Verzia pre pokročilých užívateľov

Systém by umožňoval väčšiu variabilitu pri navrhovaní pravidiel a atribútov, ktoré majú tieto pravidlá pri prideľovaní zmien zamestnancom overovať.

- **Voliteľné atribúty**:
  - **BoolLike**: Atribút, ktorý bude môcť nadobúdať 2 hodnoty – true / false. V pôvodnej verzii systému napr. atribút experienced.
  - **Element**: Užívateľ si v systéme definuje referenčnú množinu, atribút by následne nadobúdal jednu konkrétnu hodnotu z tejto množiny. Definujeme napr. množinu obcí trvalého pobytu zamestnancov, z ktorých každému zamestnancovi priradíme práve jednu.
  - **DataSet**: Na rozdiel od predchádzajúceho prípadu, atribút bude môcť nadobúdať viac hodnôt z referenčnej množiny. Vytvoríme si napr. množinu odborných schopností, ktorými môžu zamestnanci disponovať. Atribút môže v tomto prípade nadobúdať viac hodnôt z referenčnej množiny.
  - **Number**: Atribút nadobúdajúci číselnú hodnotu, napr. vek, počet rokov odpracovaných na pracovisku či vzdialenosť bydliska od pracoviska v km.
- **Overovanie pravidiel**:
  - Pravidlá AttributesValue a MoreEmployeeAttributes by obsahovali jeden z atribútov BoolLike, Element alebo DataSet nastavený na požadovanú hodnotu a príslušnou podmienkou.
  - V prípade atribútu Number by bolo definované, či má byť hodnota tohto atribútu u zamestnanca rovná, väčšia, menšia, prípadne z určitého intervalu.
- **Výhody**: Takto navrhnutý systém by bol univerzálnejší, takže by sa dal použiť na viacerých pracoviskách. Tiež by bolo možné pružnejšie reagovať na zmeny požiadavkov na systém.
- **Nevýhody**: Podobný systém by bol z užívateľského pohľadu náročnejší na používanie.

6.4 Automatizovaná verzia systému

Základnou myšlienkou pre takýto návrh systému bol predpoklad, že každý užívateľ má nejaký zoznam rozpisov služieb z minulosti. Z týchto rozpisov by potom bolo možné zistiť nasledujúce informácie:

- **Zoznam zamestnancov**: Meno a priezvisko každého zamestnanca.
- **Zoznam zmien**: Z tabuľky by bolo možné načítať zmeny v prípade, ak by každé 2 rôzne zmeny boli označené iným spôsobom. Zároveň by systém musel rozlišovať denné a nočné zmeny, napr. každá denná zmena by musela v názve obsahovať D a každá nočná N. V opačnom prípade by bola zaradená medzi špeciálne zmeny. Ďalej by sa dalo vypozorovať, či je zmena určená na pracovný deň alebo na víkend. Z toho, koľkým zamestnancom bola zmena priradená by bolo zrejmé, koľko zamestnancov na obsadenie zmeny potrebujeme. Dĺžka trvania zmeny by buď bola zadaná v rozpise, alebo by bola do systému zadaná ručne. Prípadne by sa dala dopočítať z počtu odpracovaných hodín zamestnancov a zmien, ktoré im boli priradené (tu si však nie som istý, či by bol výsledok vždy jednoznačný).
- **ShiftsNotInSequence**: Z minulých rozdelení by bolo zrejmé, aký je maximálny počet zmien, ktoré mal nejaký zamestnanec priradené po sebe. Vyšší počet zmien by bol nežiadúci.
- **DayAround**: Porovnaním všetkých kombinácii zmien by sa dalo zistiť, ktoré sa v rozpise nevyskytovali, vyskytovali len občas, alebo naopak veľmi často.
- **EmployeeNotTogether**: Pre zamestnancov, ktorí nikdy nemali zmenu spolu, by sme definovali toto pravidlo, nech už bol dôvod akýkoľvek. Ak napríklad dvaja zamestnanci niky spolu neslúžili, pretože boli obaja neskúsený a na zmene sme chceli aspoň jedného skúseného, bude to zaistené pomocou tohto pravidla.
- **ShiftSet**: Za predpokladu, že nechceme aby zamestnanec vykonával určité zmeny mu priradíme do zoznamu tie, ktoré vykonával aspoň raz. Zmeny nevykonával napr. pre to, že sa vykonávajú na oddelení, a tento zamestnanec na to nie je dostatočne kvalifikovaný.

Jednotlivé informácie spolu nesúvisia, takže by bolo možné skúmať ich paralelne. Po načítaní informácii by boli zamestnanci pridelení na zmeny, pričom by boli rešpektované zistené pravidlá. Ak by užívateľ narazil na nejakú nezrovnalosť, upravil by rozpis a nechal ho programom znovu naskenovať, prípadne by informácie o zamestnancoch, zmenách alebo pravidlách upravil ručne.

Záver

Pre vytvorenie mojej práce bolo kľúčové získať informácie o chode pracoviska s nepřetržitou prevádzkou a analyzovať ich. Za modelové pracovisko som si vybral Gynekologicko-pôrodnícke oddelenie nemocnice Mělník. Tento cieľ sa mi podarilo splniť. Na základe tejto analýzy som navrhol riešenie informačného systému, podľa návrhu som tento systém implementoval. Práca prehľadne vymedzuje funkčné a nefunkčné požiadavky a ich pokrytie formou prípadov použitia, ktoré som doplnil prehľadnými diagramami. Implementovaný informačný systém nie je dokonalý a je na ňom stále čo vylepšovať, no napriek tomu splnil očakávania. Užívateľovi umožňuje získať lepší prehľad o zamestnancoch, ktorých má na starosti, zmenách, ktoré potrebuje každý mesiac obsadiť a pravidlách, ktoré pri tom musí dodržiavať. Nespornou výhodou je automatická kontrola dodržiavania pravidiel, ktorých porušenie je prehľadne vypísané v logu. Nezanedbateľná je úspora času, korú prinesie automatické plánovanie zmien.

Literatúra

[1] Apache: *Apache Poi*. [cit. 10.5. 2015]. Dostupné z WWW: <https://poi.apache.org/spreadsheet/>

[2] Apache: *HSSFCellStyle*. [cit. 10.5. 2015]. Dostupné z WWW: <https://poi.apache.org/apidocs/org/apache/poi/hssf/usermodel/HSSFCellStyle.html>

[3] Apache: *HSSFSheet*. [cit. 10.5. 2015]. Dostupné z WWW: <https://poi.apache.org/apidocs/org/apache/poi/hssf/usermodel/HSSFSheet.html>

[4] Apache: *HSSFWorkbook*. [cit. 10.5. 2015]. Dostupné z WWW: <https://poi.apache.org/apidocs/org/apache/poi/hssf/usermodel/HSSFWorkbook.html>

[5] Cstlabs: *Rozpis Profi*. [cit. 10.5. 2015]. Dostupné z WWW: <http://www.ctlabs.cz/rozpis/profi.html>

[6] Mlejnek, J.: Analýza a sběr požadavků. https://edux.fit.cvut.cz/courses/BI-SI1/_media/lectures/03/03.prednaska.pdf, 2015, [cit. 7.5. 2015].

[7] SimpleXML: *SimpleXML*. [cit. 10.5. 2015]. Dostupné z WWW: <http://simple.sourceforge.net/download/stream/doc/tutorial/tutorial.php>

[8] SlunečniceCZ: *SuArP Rozpis*. [cit. 10.5. 2015]. Dostupné z WWW: <http://www.slunecnice.cz/sw/suarp-rozpis/>

[9] Vranić, V.: Prípady použitia. http://www2.fiit.stuba.sk/~vranic/mps/p/p02.pdf, 2013, [cit. 7.5. 2015].

Dodatok **A**

Zoznam použitých skratiek

CSV Comma-Separated Values

GUI Graphical user interface

UML Unified Modeling Language

XML Extensible markup language

Dodatok **B**

Obsah priloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresár so spustiteľnou formou implementácie
- `TestData` ..... adresár obsahujúci testovacie dáta
- `src`
  - `impl` ..... celý projekt
    - `src` ..... zdrojové kódy implementácie
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `sergetom2015.pdf` ..... text práce vo formáte PDF