MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Demonštrácia konceptov UML na modeloch zvoleného systému

Bakalárska práca

Lucia Svítková

Brno 2009

Prehlásenie

Prehlasujem, že táto práca je mojim pôvodným autorským dielom, ktoré som vypracovala samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používala alebo z nich čerpala, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

V Brne dňa 25.5.2009 Lucia Svítková

Poďakovanie

Chcela by som sa poďakovať Ing. RNDr. Barbore Zimmerovej, Ph.D. za odbornú a trpezlivú pomoc pri spracovávaní bakalárskej práce a tiež za jej cenné rady a pripomienky.

Zhrnutie

Práca zoznamuje čitateľa so základnými konceptmi modelovacieho jazyka UML. V spolupráci s druhým študentom som navrhla zadanie ilustračného systému. Na základe tohoto zadania som vypracovala sadu vybraných UML modelov zobrazujúcich koncepty tohto jazyka a to vždy niekoľko modelov pre každý zvolený diagram. K daným diagramom som uviedla aj diagram s častými chybami a zobrazila jeho korektné riešenie. Každý diagram je doprevádzaný krátkym zhrnutím demonštrovaných konceptov.

Kľúčové slová

UML, diagram prípadov užitia, textová špecifikácia prípadov užitia, diagram aktivít, sekvenčný diagram, komunikačný diagram, diagram prehľadu interakcií, diagram časovania.

Obsah

1 Úvod

Objektovo-orientovaná analýza je v súčasnosti využívaná pri vývoji mnohých rozsiahlych informačných systémov. Jazyk UML je najrozšírenejší nástroj objektovo-orientovanej analýzy a návrhu, ktorý spája všetky výrazové prostriedky.

Cieľom tejto bakalárskej práce je demonštrácia konceptov jazyka UML. Účelom nie je urobiť kompletnú a konzistentnú analýzu zvoleného systému, ale vytvoriť sadu UML modelov poukazujúcich na podstatné prvky a časté chyby pri vytváraní modelov.

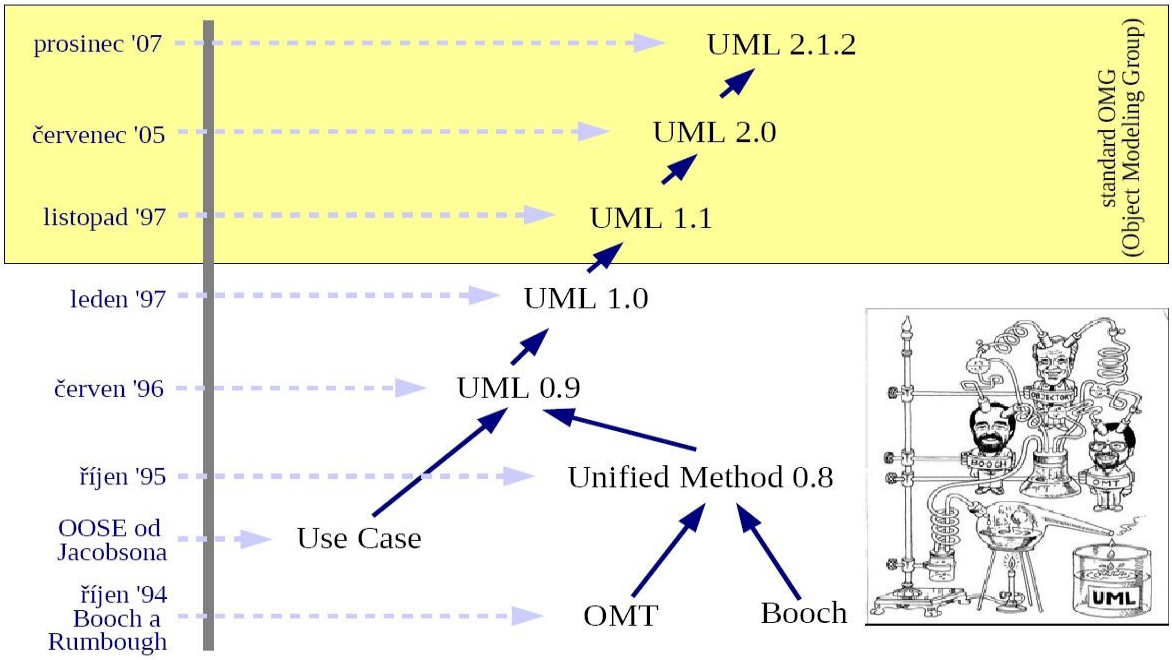
Vo svojej bakalárskej práci som spolupracovala s druhým študentom. Spoločne sme navrhli ilustračný systém. Textová špecifikácia ilustračného systému je uvedená v kapitole 3 s názvom *Špecifikácia ilustračného systému*. Jedná sa o systém letiska. Rozhodli sme sa zvoliť tento systém z dôvodu jeho rozsiahlosti, aby sme tak mohli na ňom demonštrovať všetky základné koncepty tvorby diagramov v jazyku UML. Potom sme obaja na základe uvedeného ilustračného systému prezentovali tvorbu nami vybraných modelov.

Text práce má nasledujúci štruktúru. V kapitole 2 s názvom *Modelovací jazyk UML* v krátkosti opisujem nástroj, s ktorým som pracovala. V kapitole 3 s názvom *Špecifikácia ilustračného systému* je uvedená textová špecifikácia systému, na ktorom je v ďalších kapitolách prezentovaná tvorba diagramov V kapitole 4 s názvom *Rozpracovanie určených diagramov* opisujem nasledujúce diagramy: diagram prípadu užitia, textová špecifikácia prípadov užitia, diagram aktivít, sekvenčný diagram, komunikačný diagram, diagram prehľadu interakcií a diagram časovania. Opis každého zmieneného diagramu obsahuje dve podkapitoly a to *Popis*, v ktorom je stručne opísaný daný diagram a podkapitola *Príklady diagramov*, kde sú uvedené príklady diagramov aj s komentárom opisujúcim základné koncepty tvorby. Podkapitola *Príklady diagramov* taktiež zobrazuje príklad nesprávne utvoreného diagramu s popisom chýb vyskytujúcich sa na ňom a s uvedeným korektným riešením. Podkapitola *Popis* je rozdelená do troch častí. Prvá časť *Účel diagramu* v krátkosti uvádza možnosti diagramu v procese analýzy ako aj výhody jeho použitia. V druhej časti s názvom *Komponenty* opisujem základné stavebné prvky diagramu a v poslednej časti s názvom *Časté chyby pri tvorbe* sú uvedené chyby, ktoré sa najviac vyskytujú pri používaní diagramov.

2 Modelovací jazyk UML

UML (Unified Modeling Language = Jednotný modelovací jazyk) [11] v dnešnej dobe patrí medzi najrozšírenejšie nástroje objektovo-orientovanej analýzy a návrhu informačných systémov. Jedná sa o univerzálny nástroj pre modelovanie systému, ktorý zjednodušuje proces návrhu softwaru pomocou tvorby vizuálnych modelov. UML má za cieľ zjednotiť všetky používané výrazové prostriedky. Definuje štandardné prvky používané v modeloch. Pomáha zaručiť, že všetci, ktorí sa na vývoji softwaru podieľajú, komunikujú spoločným jazykom.

Vývoj UML začal v roku 1994, keď Grady Booch a Jim Rumbaugh začali vo firme Rational Software spájať svoje metodiky – Booch a OMT. Na konci roku 1995 do firmy Rational Software vstúpil Ivar Jacobson so svojou metodológiou OOSE (Object-Oriented Software Engineering) [12]. Výsledkom ich práce bol návrh UML (verzie 0.9) a metodika RUP. Štandardizačná organizácia OMG v roku 1997 prijala ako štandard UML verzie 1.1, v ktorej boli začlenené prvky z ďalších metodík. Postupne sa upresňovala špecifikácia a vznikali ďalšie verzie 1.2 (1998), 1.3 (1999), 1.4 (2001) a 1.5 (2002). Väčšie zmeny boli začlenené do verzie 1.3. Od roku 2001 OMG pripravovala verziu 2.0, ktorá prináša podstatné rozšírenia. Najnovšia verzia je 2.2, ktorá vznikla vo februári 2009.



Obrázok č.1: História jazyka UML do roku 2007 [9]

Pre tvorbu modelov UML (od vezie 2.0) využíva 13 rôznych diagramov. Nie všetky diagramy musia byť súčasťou každého modelu. Medzi najdôležitejšie diagramy patria diagram prípadu užitia a diagram tried.

Diagramy UML možno rozdeliť na [10]:

1.Diagramy ukazujúce statickú štruktúru systému:

- *Diagram tried (Class diagram)* zobrazuje triedy a ich vzťahy.
- *Diagram objektov (Object diagram)* zobrazuje objekty a ich vzťahy.
- *Diagram balíkov (Package diagram)* zobrazuje rozdelenie systému do balíkov.
- *Diagram komponent (Component diagram)* zobrazuje štruktúru realizácie softwarového systému zachyteného pomocou softwarových komponent, rozhraní a ich vzťahov.

- *Diagram rozmiestnenia (Deployment diagram)* zobrazuje štruktúru run-time systému zachytenú pomocou softwarových prvkov a ich vzťahov.
- *Diagram vnútornej štruktúry (Composite structure diagram)* zobrazuje vnútornú štruktúru daného klasifikátoru (trieda, prípad užitia, aktivita). Jedná sa o run-time štruktúru.

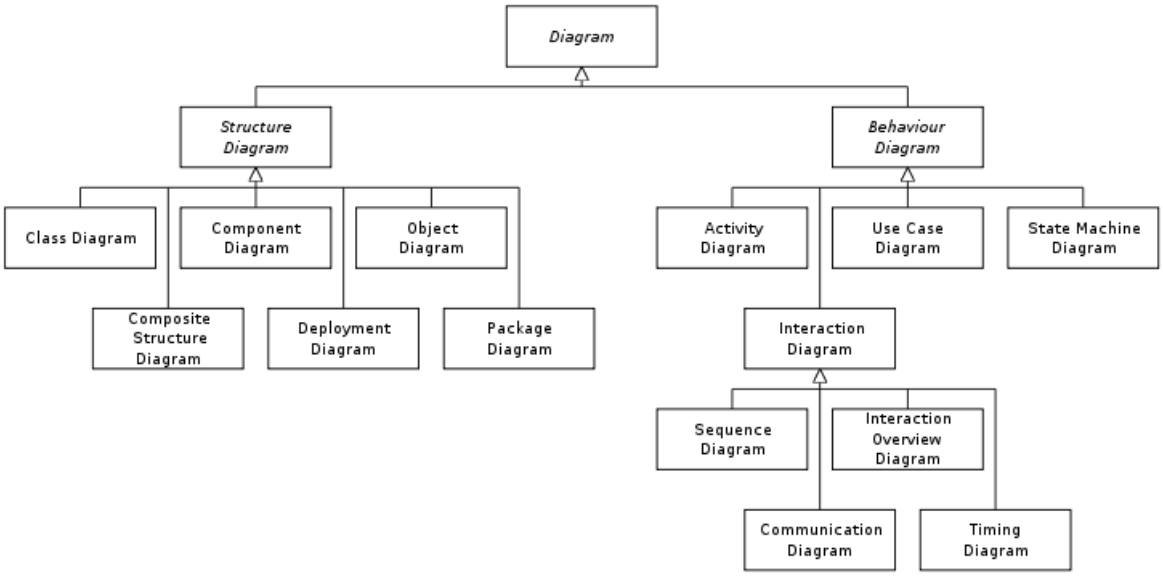
2.Diagramy ukazujúce dynamické správanie systému:

- *Diagram prípadov užitia (Use case diagram)* zobrazuje dynamickú štruktúru systému z pohľadu užívateľa.

3.Diagramy ukazujúce dynamické chovanie jednej triedy:

- *Statový diagram (State-chart diagram)* popisuje stavy objektu a prechody medzi stavmi.
- *Diagram aktivít (Activity diagram)* popisuje postupnosť akcií.
- *Diagram interakcií (Interaction diagram)* zahrňuje 4 typy diagramov:
  a) *Sekvenčný diagram (Sequence diagram)* popisuje spoluprácu objektov prostredníctvom sekvencie zasielaných správ.
  b) *Komunikačný diagram (Communication diagram)* zobrazuje interakciu organizovanú podľa interagujúcich objektov a prepojenie medzi nimi.
  c) *Diagram časovania (Timming diagram)* zaznamenáva časové obmedzenia spojené so zmenou stavu jednotlivých objektov.
  a) *Diagram prehľadu interakcií (Interaction overview diagram)* spája vlastnosti diagramu aktivít a sekvenčného diagramu.

Pomocným nástrojom pri tvorbe diagramov jazyka UML je *Textová špecifikácia prípadu užitia (Textual specification)*, ktorá popisuje akcie odohrávajúce sa v jednotlivých prípadoch užitia na diagrame prípadov užitia.



Obrázok č.2: Grafické znázornenie rozdelenia diagramov UML [6]

3 Špecifikácia ilustračného systému

Účel systému: Systém sa snaží simulovať chod letiska. Pre jednoduchosť nie sú uvedené všetky operácie reálneho systému letiska. Niektoré z uvedených operácií sú zjednodušené alebo pozmenené, lebo reálne systémy sú príliš komplikované a v niektorých veciach nejednotné. Pre účely následnej analýzy bude postačujúca špecifikácia systému opísaná v bode popis systému.

Popis systému: Súčasťou systému bude rezervácia a predaj leteniek, evidencia letov odchádzajúcich a prichádzajúcich z letiska. Ďalej bude systém viesť evidenciu zamestnancov letiska a umožňovať kontrolu pasažiera a jeho batožiny.

Online rezerváciu a predaj leteniek umožňujú sprostredkovateľské spoločnosti. Pasažier si môže zakúpiť letenku aj priamo na letisku. Systém umožňuje pasažierovi prezerať aktuálny harmonogram letov na svetelných tabuliach na letisku.

Pasažier je povinný dostaviť sa na letisko najmenej hodinu pred štartom svojho letu. V prípade, že má pasažier už letenku rezervovanú, tak na prepážke na letisku odovzdá kontrolórovi svoje doklady. Ten potom skontroluje, či je jeho letenka zaplatená a skontroluje taktiež jeho údaje na letenke. V prípade, že letenka ešte nie je zaplatená, kontrolór vyzve pasažiera, aby zaplatiť letenku. Ak pasažier nemá rezervovanú letenku, požiada kontrolóra, aby tak učinil ak je ešte voľné miesto v požadovanom lete.

Kontrolór ďalej skontroluje batožinu pasažiera. Pasažier má právo na jednu bezplatnú batožinu tzv. príručnú, ktorú si berie so sebou do lietadla, za ostatnú batožinu si musí zaplatiť. Cena za prepravu batožiny závisí od jej hmotnosti. V prípade, že hmotnosť niektorej batožiny prekračuje stanovené limity, je pasažier povinný doplatiť rozdiel podľa sadzby letiska. Limity na prepravovanú batožinu záležia od konkrétneho typu a triedy letu.

Po vybavení týchto náležitostí je pasažier a jeho batožina skontrolovaná detektorom. Kontrolór má právo zrušiť pasažierovi letenku v prípade, že kontrola neprebehla podľa požiadavok.

Manažér letiska riadi celý chod letiska. Zamestnáva nových zamestnancov, vyhodnocuje štatistiky, kontroluje svojich zamestnancov, prijíma a rieši prípadné sťažnosti pasažierov. Vytvára harmonogram letov t. j. priraďuje letom konkrétne príletové a odletové dráhy a mení poprípade čas ich odchodu alebo príchodu.

Letisko len sprostredkúva lety. Existuje však viac leteckých spoločnosti. Každá letecká spoločnosť má pridelené nejaké lety a sama si riadi réžiu letov t. j. zamestnáva svojich pilotov a letušky. Letecká spoločnosť pristupuje do systému prostredníctvom svojho manažéra, ktorému manažér letiska pridelil právo prístupu do systému. Manažér leteckej spoločnosti zadáva do systému informácie o lete ako typ letu, triedy letu a cenník prepravovanej batožiny. Priraďuje letom konkrétne lietadlá, aby si tak zákazník mohol pri rezervácii vybrať konkrétne miesta na sedenie.

4 Rozpracovanie určených diagramov

4.1 Diagram prípadov užitia (Use case diagram)

4.1.1 Popis

Účel diagramu [3]: Diagram prípadov užitia sa snaží zachytiť dynamické správanie celého analyzovaného systému. Patrí medzi jeden z prvých a najdôležitejších diagramov UML. Vytvára koncept, ktorý je dobre zrozumiteľný ako z pohľadu zadávateľa systému, tak aj z pohľadu vývojového tímu. Mal by sa vytvoriť, čo najjednoduchší a najprehľadnejší, aby bol zákazníkovi ľahko pochopiteľný.

Komponenty [1]: *Účastník (Actor)* – externá entita t. j. entita nachádzajúca sa mimo systém, ktorá komunikuje so systémom. Najčastejšie v úlohe účastníka vystupujú užívatelia komunikujúci so systémom. Okrem toho v tejto úlohe môžu vystupovať externé systémy, či hardwarové časti. Účastníkom môže byť aj čas, v prípade, keď požadujeme od systému, aby spúšťal nejakú akciu v pravidelných časových intervaloch.

Prípad užitia (Use case) – postupnosť činností vrátane premenných postupností a chybových postupností, ktoré systém, podsystém alebo trieda môžu vykonať prostredníctvom interakcií s externými účastníkmi.

Relácia (Relationship) – vzťah medzi prípadom užitia a účastníkom, po prípade medzi dvomi účastníkmi alebo dvomi prípadmi užitia. Určuje, ktoré elementy navzájom spolupracujú. Relácia medzi uvedenými komponentami môže byť buď jednosmerná alebo obojsmerná.

Typy relácií: *Zovšeobecnenie účastníka (Actor generalization)* – vzťah medzi obecnejším a konkrétnejším účastníkom. Konkrétnejší účastník dedí funkcionalitu od obecnejšieho účastníka a navyše môže svoju funkcionalitu rozširovať.

Zovšeobecnenie prípadov užitia (Use case generalization) – vzťah konkrétnejšieho a obecnejšieho prípadu užitia. Konkrétnejší prípad užitia dedí funkcionalitu od obecnejšieho prípadu užitia a navyše môže svoju funkcionalitu rozširovať.

Relácia include – vzťah medzi prípadmi užitia, kde jeden prípad užitia začleňuje správanie iného prípadu užitia.

Relácia extend – vzťah medzi prípadmi užitia, kde jeden prípad užitia rozširuje svoje správanie pomocou jedného alebo viacerých fragmentov správania sa iného prípadu užitia.

Hranice systému (System boundary) – oddeľujú systém od vonkajšieho okolia. Účastníci sa nachádzajú mimo systém a prípady užitia sú vo vnútri systému.

Časté chyby pri tvorbe [9]: Chybou pri hľadaní účastníkov je zobrazenie užívateľov, ktorí priamo nekomunikujú so systémom, ale komunikujú prostredníctvom ďalšej osoby. Vzniká tak väzba medzi účastníkom a účastníkom, ktorá nemá vplyv na vývoj softwaru a jej prítomnosť je nadbytočná. Na diagrame sa zobrazuje len ten účastník, ktorý priamo komunikuje so systémom.

Relácie *include* a *extend* je treba používať s rozvahou. Nedoporučuje sa ich používať vo veľkej miere, lebo to môže viac zneprehľadniť diagram ako ho zjednodušiť. Navyše použitie množstva relácii *include* navádza na funkčnú dekompozíciu. Reláciu *extend* sa doporučuje používať iba v prípade, že rozširujúcu funkcionalitu nie je jednoduché zabudovať do už existujúceho systému a je jednoduchšie použiť túto reláciu.

Niekedy sa zamieňa význam osôb a rolí. Účastníci plnia úlohu rolí – nehovorí sa o tom, aké konkrétne osoby vystupujú v danej roli. V jednej roli môže vystupovať viacej osôb a jedna osoba môže plniť úlohu viacerých rolí.

Nie je vhodné, aby prípady užitia popisovali iba malú časť rozsiahlej interakcie. Vedie to k tomu, že na diagrame je množstvo prípadov užitia, čo zneprehľadní diagram. Navyše sa môže stať,

že na diagrame budú ako samostatné prípady užitia vystupovať prípady užitia, ktoré sú na sebe závislé a môžu byť vykonané len v určitom poradí.

4.1.2 Príklady diagramov

Obrázok č. 3 zobrazuje prvotný pohľad na systém letiska, ktorého textová špecifikácia je uvedená v kapitole 3 s názvom *Špecifikácia ilustračného systému*.

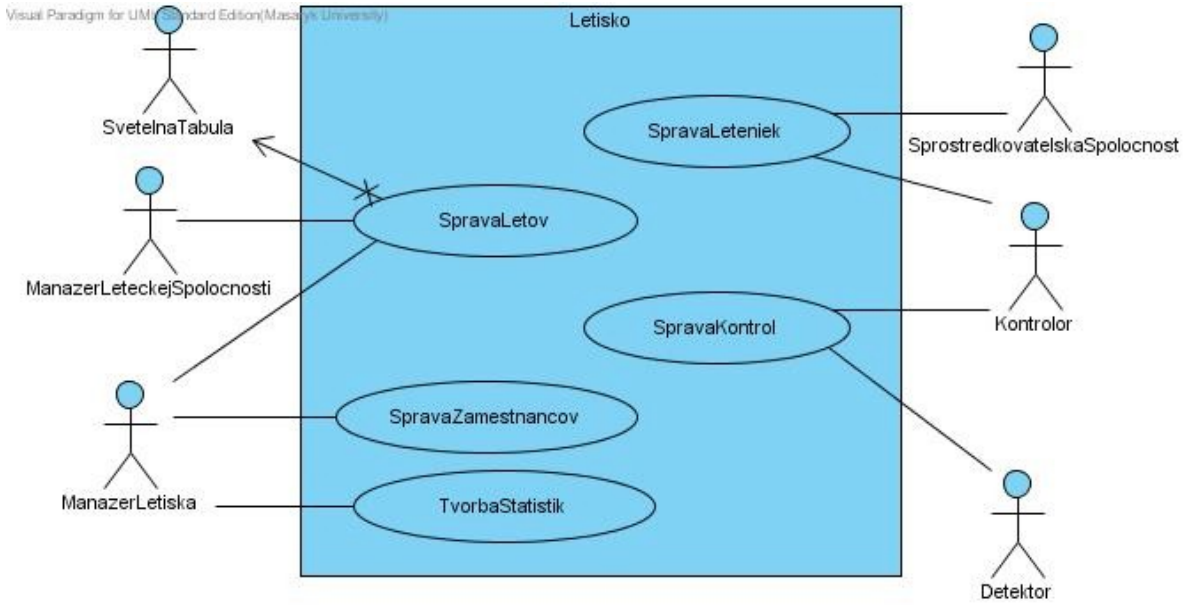
Na začiatku je dobré určiť si hranice systému. UML hranicu systému zobrazuje ako obdĺžnik s názvom systému. V tomto prípade bude názov systému *Letisko* nakoľko sa jedná o systém letiska. Účastníci sa nachádzajú mimo systém t. j. mimo obdĺžnik a prípady užitia sú súčasťou systému – sú zobrazené vnútri obdĺžnika.

Na základe analýzy textovej špecifikácie si určím účastníkov t. j. entity ovplyvňujúce systém. UML zobrazuje účastníkov ako postavičky. Každý účastník má svoj názov, ktorý by mal reflektovať s názvom entity uvedenej v textovej špecifikácií analyzovaného systému. Názvy by nemali obsahovať diakritiku a medzery medzi slovami v prípade viacslovných názvov a každé ďalšie nové slovo by malo začínať veľkým písmenom, aby bolo jasné, kde slovo začína a kde končí. Tento úzus sa používa z dôvodu, že väčšina programovacích jazykov nepodporuje tieto znaky a úlohou analýzy je vytvoriť vhodné podklady pre následné programovanie. Medzi účastníkov tohoto systému budú patriť: *SprostredkovatelskaSpolocnost*, *ManazerLetiska*, *ManazerLeteckejSpolocnosti*, *Kontrolor* ale aj *SvetelnaTabula* a *Detektor* – zariadenia, ktoré tiež ovplyvňujú stav systému resp. sú ovplyvňované systémom.

Následne hľadám v textovej špecifikácií systému činnosti, ktoré sú vykonávané so systémom. Tieto činnosti budú reprezentované prípadmi užitia. Prípad užitia sa znázorňuje ako ovál vnútri, ktorého je názov prípadu užitia. Názov každého prípadu užitia musí byť slovesnou väzbou nakoľko prípad užitia opisuje nejakú akciu. Názvy volíme tak, aby vyjadrovali podstatu činnosti daného prípadu užitia. Pre názvy prípadov užitia platia také isté pravidlá, ako je to v prípade názvoslovia účastníkov. Uvedený systém letiska bude spravovať databázu leteniek. Do systému sa budú zadávať nové letenky, prehliadať už zadané letenky, meniť údaje zadanej letenky, platiť letenky, či poprípade rušiť letenky z databáze. Všetky uvedené akcie spolu súvisia, preto ich zobrazím pomocou jedného prípadu užitia, ktorý nazvem *SpravaLeteniek*. Ďalej bude systém spravovať databázu letov. Budú sa zadávať lety, upravovať lety, rušiť lety. Táto činnosť je zobrazená prípadom užitia *SpravaLetov*. Databáza zamestnancov sa bude spravovať pomocou prípadu užitia *SpravaZamestnancov*. Tvorbu štatistík bude zahrňovať prípad užitia *TvorbaStatistik*. Posledným prípadom užitia na uvedenom diagrame bude prípad užitia *SpravaKontrol*, ktorý bude zahrňovať kontrolu letenky, kontrolu pasažiera a jeho batožiny.

Fakt, že daný účastník pracuje s daným prípadom užitia sa na diagrame prípadov užitia znázorňuje čiarou medzi prípadom užitia a účastníkom. Táto komunikácia môže buď obojsmerná t. j. účastník komunikuje so systémom a aj systém komunikuje s účastníkom (značí ako šípka bez orientácie) alebo jednosmerná, v tom prípade smer komunikácie určuje smer šípky. S prípadom užitia *SpravaLeteniek* bude pracovať *SprostredkovatelskaSpolocnost* ako aj *Kontrolor* – medzi uvedenými komponentami bude relácia. Jedná sa o obojsmernú komunikáciu, lebo uvedený účastníci zapisujú dáta do systému, ale aj systém posiela údaje uvedeným aktérom. V prípade interakcie zadanie letenky komunikáciu aktéra so systému predstavuje napr. zadanie údajov letenky a komunikácia opačným smerom je využitá napr. pri zobrazení letenky. S prípadom užitia *TvorbaStatistik* a *SpravaZamestnancov* komunikuje *ManazerLetiska*. S prípadom užitia *SpravaLetov* komunikuje *ManazerLetiska* a *ManazerLeteckejSpolocnosti*. A medzi uvedeným prípadom užitia a účastníkom *SvetelnaTabula* je uvedená jednosmerná komunikácia. Smer komunikácie ide smerom k *SvetelnejTabuli*. Systém teda komunikuje so *SvetelnouTabulou* – posiela jej údaje o letoch, ktoré má zobraziť ale *SvetelnaTabula* už so systémom nekomunikuje – neposiela už nijaké dáta do späť do systému. S prípadom užitia *Sprava-*

Kontrol bude komunikovať *Kontrolor* a *Detektor*.



Obrázok č. 3 : Diagram prípadov užitia

Na obrázku č. 4 je zobrazený ten istý systém ako na obrázku č. 3, ale už sú v ňom zobrazené niektoré pokročilé prvky use case diagramu.

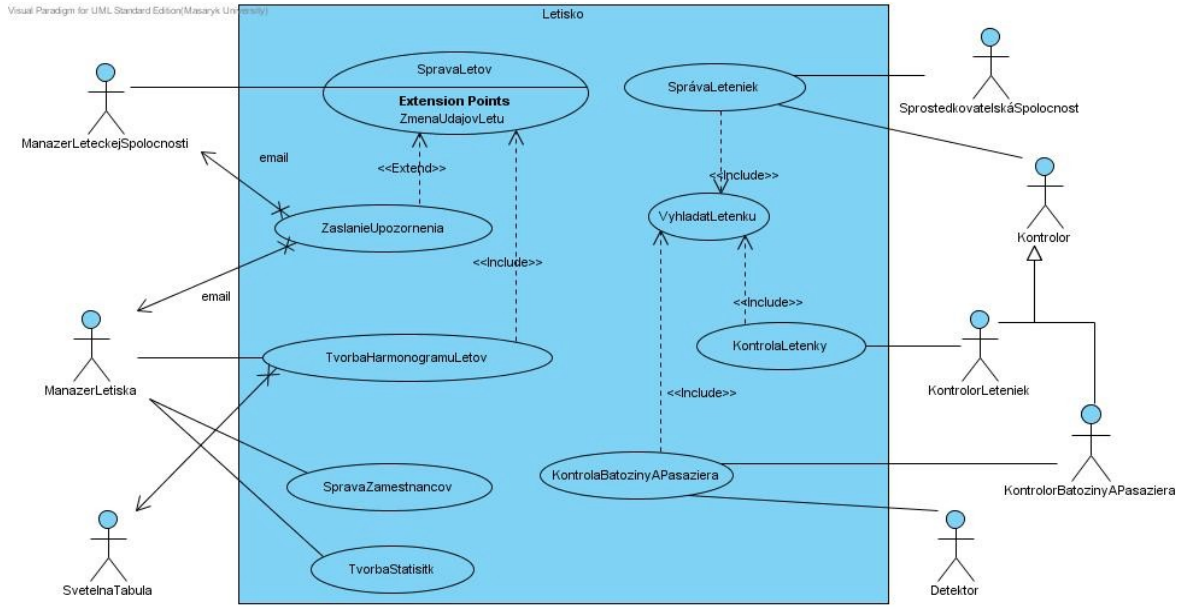
ManazerLetiska tvorí harmonogram letov – prípad užitia *TvorbaHarmonogramuLetov* (textová špecifikácia daného prípadu užitia je zobrazená v prílohe na obrázku č. 26). Potom ako ho vytvorí musí doplniť medzi informácie o letoch čas odchodu a príchodu, čiže musí pristúpiť k prípadu užitia *SpravaLetov*. To sa zobrazí vzťahom medzi zmienenými dvomi prípadmi užitia. Bude sa jednať o reláciu *include* nakoľko prípad užitia *TvorbaHarmonogramuLetov* využíva správanie prípadu užitia *SpravaLetov*.

V prípade, že nastane zmena údajov letu musí byť o tom oboznámený jednak *ManazerLetiska* a jednak *ManazerLeteckejSpolocnosti*. Na diagrame je to zobrazené reláciou *extend*. Jedná sa o rozširujúcu funkcionalitu. *Rozširujúci bod (Extension point)* je nazvaný *ZmenaUdajovLetu*, keďže táto situácia je spustená v okamihu, keď *ManazerLetiska* alebo *ManazerLeteckejSpoločnosti* zmení údaje letu. Relácia *extend* sa nachádza medzi prípadom užitia *SpravaLetov* (textová špecifikácia tohto prípadu užitia je zobrazená v prílohe na obrázku č. 24) a novovytvoreným prípadom užitia *Zaslanie-Upozornenia* (textová špecifikácia tohto prípadu užitia je zobrazená v prílohe na obrázku č. 25), ktorého úlohou bude zaslať emailom upozornenie o zmene údajov letu *ManazeroviLetiska* a *ManazeroviLeteckejSpoločnosti*, aby sa mohli podľa toho zariadiť. Oba zmienený účastníci ale nič s uvedeným prípadom užitia *ZaslanieUpozornenia* nevykonávajú, čo je zobrazené jednosmernou komunikáciou.

Kontrolora možno rozdeliť na toho, čo kontroluje letenky a toho, čo kontroluje pasažiera a jeho batožinu. Použijem generalizáciu na strane aktérov. Obecnejším aktérom bude *Kontrolor* a konkrétnejšími účastníkmi bude *KontrolorLetenky* a *KontrolorPasazieraABatoziny*. Oba účastníci pracujú s prípadom užitia *SpravaLeteniek* (špecifikácia tohoto prípadu užitia zobrazená pomocou diagramu aktivít je na obrázku č. 8 v kapitole 4.3.2), teda táto funkcionalita je spoločná pre oboch účastníkov a z toho dôvodu sa jedná o funkcionalitu, ktorú má ich nadtyp – teda uvediem na diagrame reláciu medzi účastníkom *Kontrolor* a prípadom užitia *SpravaLeteniek*. *KontrolorLetenky* pristupuje k tomuto

prípadu užitia, aby si vyhľadať v systéme údaje o letenke, ktorú má skontrolovať, či letenku zadal, ak ho pasažier o to požiada. *KontrolorPasazieraABatoziny* pristupuje, k tomuto prípadu užitia, aby mohol pasažierovi zrušiť letenku, ak pasažier, či jeho batožina neprešli kontrolou *Detektorom*. Navyše *KontrolorLetenky* pracuje s prípadom užitia *KontrolaLetenky* (špecifikácia tohoto prípadu užitia zobrazená pomocou diagramu aktivít je v prílohe na obrázku č. 29). *KontrolorPasazieraABatožiny* pracuje s prípadom užitia *KontrolaBatožinyAPasažiera* (špecifikácia tohoto prípadu užitia zobrazená pomocou diagramu aktivít je v prílohe na obrázku č. 28). *KontrolaPasazieraABatoziny* prebieha tak, že najprv prejde pasažier cez *Dekektor* a potom sa prekontroluje aj jeho batožina. Prípadné nezrovnalosti rieši daný *Kontrolor* – buď požiada o zopakovanie kontroly, alebo ak aj následná kontrola nie je v poriadku zruší pasažierovi letenku.

Prípady užitia *SpravaLeteniek*, *KontrolaBatozinyAPasaziera* a *KontrolaLetenky* potrebujú na začiatku vyhľadať letenku. Keby som túto udalosť zobrazovala v každom zo zmienených prípadov užitia, došlo by k redundancii dát. Preto je lepšie pre túto funkcionalitu vytvoriť nový prípad užitia *VyhladatLetenku* a zmienené prípady užitia budú s novovytvoreným prípadom užitia *VyhladatLetenku* v relácii *include*. Smer relácie určuje smer šípky – určuje ktorý prípad užitia zahrňuje správanie iného prípadu užitia. Teda v uvedenom prípade prípad užitia *SpravaLeteniek* bude zahrňovať správanie sa prípadu užitia *VyhladatLetenku* – smer šípky je uvedený od prípadu užitia *SpravaLeteniek* k prípadu užitia *VyhladatLetenku.*



Obrázok č. 4 : Diagram prípadov užitia s pokročilými modelovacími technikami

Obrázok č. 5 zobrazuje opäť informačný systém letiska, ale tento krát sú v ňom zobrazené najčastejšie chyby pri tvorbe diagramu prípadov užitia.

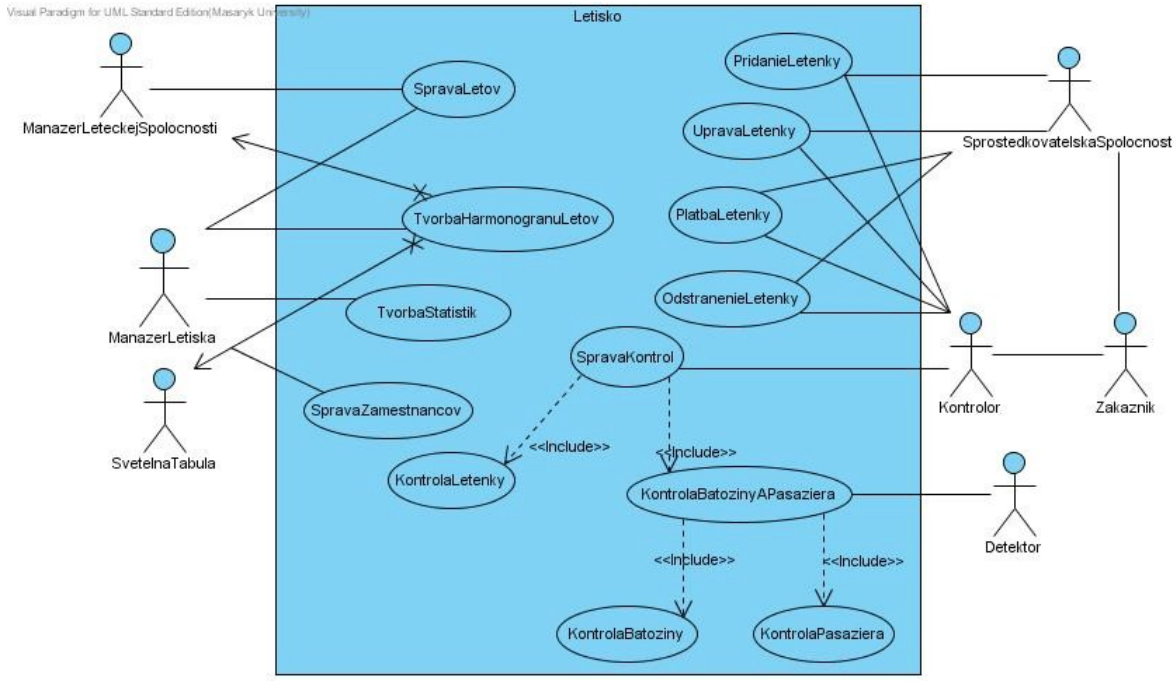
SpravaKontrol a z nej vychádzajúce relácie *include* do prípadov užitia *KontrolaLetenky* a *KontrolaBatožinyAPasažiera* a potom následné rozdelenie prípadu *KontrolaBatozinyAPasaziera* na dva ďalšie prípady vytvára funkčnú dekompozíciu. Snahou je vyhnúť sa takýmto konštrukciám nakoľko zneprehľadňujú celý diagram. Vhodným riešením je zobraziť uvedené prípady užitia do jedného prípadu užitia alebo po prípade do dvoch prípadov užitia, jeden s názvom *KontrolaLetenky* a druhý *KontrolaBatožinyAPasažiera,* ak by sa jednalo o zložité prípady, ktoré by neboli prehľadné v jednom prípade užitia.

Zakaznik na diagrame nekomunikuje priamo so systémom ale s ďalšími účastníkmi. Komunikácia medzi dvomi účastníkmi neovplyvňuje systém. Preto by sa *Zakaznik* ako účastník nemal zobrazovať na diagrame.

Prípady užitia *PridanieLetenky*, *OdstranieLetenky*, *PlatbaLetenky* a *UpravaLetenky* tak, ako sú uvedené na tomto diagrame zneprehľadňujú diagram – vzniká tu množstvo relácií. A navyše uvedený spôsob zápisu dovoľuje upravovať, zrušiť alebo zaplatiť letenku, ktorá ešte ani neexistuje. Dané prípady užitia nie sú až také zložité, aby museli byť uvedené samostatne. Najlepšie je ich napísať do jedného prípadu užitia.

Podobná spleť čiar je vytvorená na opačnej strane. *ManazerLetiska*, ktorý vytvára harmonogram letov by mal vedieť informácie o daných letoch. To je zobrazené vzťahom medzi ním a prípadom užitia *SpravaLetov*. Tento vzťah sa dá vyjadriť aj inak. Riešením je medzi prípad užitia *TvorbaHarmonogramuLetov* a *SpravaLetov* uviesť reláciu *include*. A kedykoľvek bude chcieť *ManazerLetiska* zobraziť informácie o letoch alebo tieto informácie zmeniť vstúpi do prípadu užitia *SpravaLetov*, ktorý mu tieto akcie umožňuje. Pôvodný vzťah sa už potom nezobrazuje.

Vhodným riešením uvedeného chybného diagramu na obrázku č. 5 je diagram na obrázku č. 3 alebo obrázku č. 4. Obe uvedené správne verzie diagramu prípadov užitia si navzájom odpovedajú, líšia sa len mierou detailov.



Obrázok č.5: Diagram prípadov užitia s častými chybami

4.2 Textová špecifikácia prípadov užitia (Textual specification)

4.2.1 Popis

Účel diagramu [3]: Textová špecifikácia prípadov užitia slúži na lepšie pochopenie diagramu prípadov užitia. Popisuje deje odohrávajúce sa v jednotlivých prípadoch užitia. Nepatrí medzi diagramy jazyka UML, jedná sa len o podporný nástroj pri tvorbe diagramov. Býva písaná z pohľadu

aktéra. Neexistuje žiadny štandard UML na písanie textovej špecifikácie prípadov užitia, ale každý popis typicky obsahuje názov prípadu užitia, mená účastníkov prípadu užitia a popis interakcie pomocou toku udalostí.

Komponenty [1]:*Účastník (Actor)* – externá entita, ktorá spúšťa konkrétny prípad užitia. Všetky prípady užitia sú spúšťané vždy jedným aktérom, ale rovnaký prípad užitia môže byť spustený rôznymi aktérmi v rôznych časových okamihoch. Z pohľadu konkrétneho prípadu užitia existujú dva typy účastníkov: *Hlavný účastník (Primary actor)* – aktér, ktorý môže spúšťať daný prípad užitia. *Vedľajší účastník (Secondary actor)* – aktér, ktorý je v interakcii s prípadom užitia po jeho spustený. Napr. v prílohe na obrázku č.24 je zobrazená textová špecifikácia prípadu užitia *SpravaLetov*. V uvedenej špecifikácii je zadaný ako hlavný aktér *ManazerLeteckejSpolocnosti* a *ManazerLetiska*, ktorí spúšťajú uvedený prípad užitia a vedľajším aktérom je *SvetelnaTabula*, ktorá je s daným prípadom užitia v interakcii tým, že zobrazuje údaje o letoch.

Vstupné podmienky (Preconditions) – kritéria, ktoré musia byť splnené ešte predtým, ako je možné spustiť prípad užitia. Sú to obmedzenia stavu systému. Všetky vstupné podmienky musia byť overiteľné systémom. V opačnom prípade systém nedokáže rozoznať, či je možné zahájiť daný prípad užitia.

Hlavný tok udalostí (Main Flow of events) – jednotlivé kroky prípadu užitia. Neexistuje žiadny štandard pre ich písanie, ale odporúča sa ich písať v číslovaných krokoch, lebo tento spôsob najlepšie vyjadruje jednoznačnosť interakcie. V toku udalostí by sa mali striedať akcie účastníka prípadu užitia s reakciami systému, pričom prípad užitia musí byť zahájený účastníkom. Systém len reaguje na jeho žiadosť. Každá udalosť by mala presne opisovať danú situáciu, preto by mala byť napísaná v tvare: <číslo><nejaká podmienka><určitá akcia>. Zápis toku udalostí by mal byť čo najjednoduchší, ale na druhú stranu, čo najjednoznačnejší. Nie je však nutné písať ho tak podrobne ako programovací kód.

Následné podmienky (Post-conditions) – kritéria, ktoré musia byť splnené na konci prípadu užitia. Musia to byť podmienky overiteľné systémom, aby systém rozhodol, či je možné daný prípad užitia opustiť.

Alternatívne toky (Alternative flows) – udalosti spúšťané z hlavného toku udalostí výnimočnými situáciami. Každý alternatívny tok by mal v sebe niesť podmienku, ktorá jednoznačne určuje, kedy má byť daný alternatívny tok spustený.

Časté chyby pri tvorbe [9]: Chybou pri tvorbe špecifikácie prípadov užitia je zadanie vstupnej alebo výstupnej podmienky neoveriteľnej systémom napr. „Zákazník si príde do sprostredkovateľskej spoločnosti rezervovať letenku.“ Zákazník nepracuje so systémom – systém nemôže tento fakt overiť.

Každá udalosť v hlavnom toku udalostí by mala byť jednoznačná. Keď je zadaná udalosť ako napr. „Sú zadané letenky.“, nie je možné odpovedať na otázky: „Kde sú zadané letenky?“ „O aké letenky sa jedná?“ „Kto zadáva letenky do systému.“

4.2.2 Príklady diagramov

Na obrázku č. 6 je zobrazená textová špecifikácia prípadu užitia *SpravaZamestnancov* uvedeného na diagrame prípadov užitia na obrázku č. 3 v kapitole 3.1.2.

Zmienený prípad užitia umožňuje *ManazeroviLetiska* spravovať databázu zamestnancov. *ManazerLetiska* spúšťa daný prípadom užitia t. j. vystupuje v úlohe hlavného aktéra. Okrem *ManazeraLetiska* už nikto iný nepracuje s daným prípadom užitia – nie je uvedený žiadny iný účastník.

ManazerLetiska pred zahájením tohoto prípadu užitia musí byť prihlásený do systému – jedná sa o vstupnú podmienku. Daná vstupná podmienka je ľahko overiteľná systémom. Manažér zadá prihlasovacie meno a heslo a systém prehľadá databázu prihlasovacích mien a hesiel a zistí, či má *ManazerLetiska* právo prístupu do systému.

Uvedený prípad užitia dovoľuje *ManazeroviLetiska* zadať do systému nového zamestnanca, zobraziť údaje o zadanom zamestnancovi ako aj zmeniť údaje zamestnanca, či vymazať zamestnanca z databázy zamestnancov. Všetky tieto možnosti sú uvedené ako hlavný tok udalostí očíslované podľa poradia v akom môžu nastať.

Okrem toho môže dojsť aj k iným situáciach, ktoré sú uvedené v sekcii alternatívne toky sú to udalosti ako napr. „Systém nemohol nájsť zamestnanca v databáze zamestnancov.“ Zo zadania alternatívneho toku by malo byť jasné, kedy sa má alternatívny tok spustiť. Zmienený alternatívny tok „Systém nemohol nájsť zamestnanca v databáze zamestnancov.“ sa spustí v prípade, keď *Manazer-Letiska* zadá jednoznačný identifikátor zamestnanca, ale systému sa nepodari nájsť v databáze zamestnancov zamestnanca s daným identifikátorom.

Výstupnou podmienkou v tomto prípade je aktualizácia databáze. Opäť sa jedná o systémom overiteľnú podmienku.

Uvedená textová špecifikácie je zakreslená pomocou diagramu aktivít v prílohe na obrázku č. 27.

Main	
Use Case ID	SpravaZamestnancov
Brief Description	UC umoznuje manazerovi letiska spravovat databazu zamestnancov.
Primary Actors	ManazerLetiska
Secondary Actors	-
Preconditions	ManazerLetiska je prihlaseny do systemu
Main Flow of Events	1. Pripad uzitia zacina, ked manazer letiska zvoli v hlavnom menu polozku "sprava zamestnancov". 2. System ponukne manazerovi 2 moznosti: "zadat noveho zamestnanca", "zobrazit udaje o zamestnancovi". 3. Ak manazer zvoli moznost "zadat noveho zamestnanca". 3.1 System zobrazi manazerovi formular udajov, ktore je potreba vyplnit. 3.2 Manazer udaje vyplni a potvrdi zmeny. 3.3 System zamestnanca ulozi do databazy zamestnancov. 4 Ak manazer zvoli moznost "zobrazit udaje o zamestnancovi" 4.1 System poziada manazera o zadanie indetifikacneho udaju zamestnanca. 4.2 Manazer zada identifikator zamestnanca. 4.3 System zobrazi zamestanca a ponukne manazerovi moznost "editacia udajov o zamestnancovi", "zmazat zamestnanca". 4.4 Ak manazer zvoli moznost "editacia udajov o zamestnancovi" 4.4.1 System zobrazi udaje o zamestnacovi. 4.4.2 Manazer edituje udaje a potvrdi zmeny. 4.4.3 System ulozi zmenene udaje o zamestnancovi do databaze zamestnancov. 4.5. Ak Manazer zvoli moznost "zmazat zamestanca". 4.5.1 System zmaze zamestnanca z aktualnej databaze zamestnancov.
Alternative Flows	1. Manazer sa moze kedykolvek vratit do hlavneho menu. 2. System nemohol ulozit udaje o zamestnancovi do databazy zamestnancov. 3. System nemohol najst zamestnanca v databaze aktualnych zamestnancov 4. Systemu sa nepodarilo zmazat zamestnanca z databaze zamestnancov.
Post-conditions	System aktualizoval databazu zamestnancov.

Obrázok č. 6 : Textová špecifikácia prípadu užitia SpravaZamestnancov

Na obrázku č. 7 je zobrazená textová špecifikácia toho istého prípadu užitia ako na obrázku

č. 6 t. j. prípadu užitia *SpravaZamestnancov*, s tým rozdielom, že na obrázku č. 7 je daná textová špecifikácia zobrazená s najčastejšie chybami.
Prvý nedostatok je v sekcii *Use case ID*. Táto sekcia mi mala jednoznačná určiť, aký prípad užitia je v nej špecifikovaný. Na danom obrázku je uvedené ako ID číslo 1, lenže z diagramu prípadov užitia na obrázku č. 3 v kapitole 3.1.2. nie je možné určiť, aký prípad užitia špecifikujeme. Najvhodnejšie je do sekcie *Use case ID* uviesť názov daného prípadu užitia, umožňujúci spárovanie so špecifikovaných prípadom užitia.

Popis (Brief Description) "UC umožňuje spravovať databázu zamestnancov." je nedostatočný, lebo sa nedá určiť komu dovoľuje daný prípad užitia spravovať databázu zamestnancov, hoci z diagramu prípadu užitia je to možné vyčítať, odporúča sa aj v popise uviesť účastníka daného prípadu užitia.

Main	
Use Case ID	1
Brief Description	UC umoznuje spravovat databazu zamestnancov.
Primary Actors	ManazerLetiska
Secondary Actors	-
Preconditions	1. Manazer prijal noveho zamestnanca. 2. Manazer chce zmenit udaje zamestnanca. 3. Manazer chce odstranit zamestnanca zo systemu.
Main Flow of Events	1. Pripad uzitia zacina, ked system ponukne manazerovi letiska 2 moznosti: "zadat noveho zamestnanca", "zobrazit udaje o zamestnancovi". 2. Ak manazer zvoli moznost "zadat noveho zamestnanca". 2.1 Manazer udaje vyplni a potvrdi zmeny. 2.2 Zamestnanec je ulozeny do databazy. 3 Ak manazer zvoli moznost "zobrazit udaje o zamestnancovi" 3.1 System poziada manazera o zadanie indetifikacneho udaju zamestnanca. 3.2 System zobrazi zamestanca a ponukne manazerovi moznost "editacia udajov o zamestnancovi", "zmazat zamestnanca". 3.3 Manazer potom moze udaje zamestnanca editovat alebo zamestnanca zmazat z databazy.
Alternative Flows	1. Manazer sa moze vratit do hlavneho menu.
Post-conditions	System aktualizoval databazu zamestnancov.

Obrázok č.7 : Textová špecifikácia prípadu užitia SpravaZamestnancov s najčastejšími chybami

Vstupné podmienky (Preconditions) nie sú vhodne zvolené. Vstupné ako aj výstupné podmienky by mali byť overiteľné systémom, čo v tomto prípade nespĺňa ani jedna vstupná podmienka. To, že *ManazerLetiska* prijal zamestnanca, či chce jeho údaje zmeniť alebo ho odstrániť z databázy systém nevie overiť. Príkladom dobre formulovanej vstupnej podmienky je podmienka z textovej špecifikácie na obrázku č. 6 „Manažér je prihlásený do systému.".

Hlavný tok udalostí (Main flow of Events) býva zahájený účastníkom, ktorý komunikuje s daným prípadom užitia. Nebýva zahájený činnosťou systému ako je to uvedené na obrázku č. 7. Systému by potom nebolo jasné, kedy má spustiť uvedený prípad užitia. Akcie účastníka so systémom by sa mali striedať resp. systém reaguje na nejakú udalosť, ktorú uskutočnil daný aktér. Preto bod 2 a bod 2.2 v textovej špecifikácii by nemali nasledovať po sebe. Malo by sa niečo odohrať medzi nimi ako napr. „Systém ponúkne manažérovi formulár s údajmi, ktoré je potreba vyplniť.". Bod 2.2 je opäť zle

definovaný, lebo nie je úplne jasné, kto uloží zamestnanca do databázy a ani nie je možné z daného popisu zistiť do akej databázy budú údaje o zamestnancovi uložené.

Medzi bodmi 3.1 a 3.2 by mala byť nejaká reakcia manažéra – „Manažér zadá identifikátor zamestnanca“. Nie len preto, že medzi systémom a účastníkom by malo dochádzať k výmene riadenia prípadu užitia ale koniec–koncov aj z dôvodu, že systém nevie, ktorého zamestnanca má zobraziť. Bod 3.3 je lepšie rozpísať a nenechávať takto v jednom bode, lebo sa jedná o akciu skladajúcu sa z množstva podakcií a pre prehľadnosť je ich lepšie uviesť.

Každý alternatívny tok (Alternative Flows) by mal byť určený nejakou boolovskou podmienkou, ktorá ho spúšťa. Preto „Manažér sa môže vrátiť do hlavného menu“ nie je dobre formulovaná podmienka, nakoľko z toho nevieme, kedy sa môže vrátiť do hlavného menu. Vhodnejšia formulácia je „Manažéra sa môže kedykoľvek vrátiť do hlavného menu.“

4.3 Diagram aktivít (Activity diagram)

4.3.1 Popis

Účel diagramu [3]: Diagram aktivít je jedným z UML diagramov, ktoré popisujú správanie systému. Tento diagram sa používa pre modelovanie procedurálnej logiky a procesov. Každý proces v diagrame aktivít je reprezentovaný sekvenciou jednotlivých krokov. Diagram aktivít je vzhľadovo podobný stavovému diagramu líši sa však použitím. Najčastejšie sa používa na popis prípadov užitia, ale môže sa použiť aj na modelovanie tokov medzi prípadmi užitia, popis toku dát v systéme a modelovanie obchodných procesov. Ja vo svojej práci budem pomocou diagramu aktivít prezentovať popis prípadov užitia.

Komponenty [1]: *Aktivita (Activity)* – celok, ktorý je v diagrame aktivít dekomponovaný. Názov aktivity by mal, čo najjednoznačnejšie určiť, čo sa odohráva v danej aktivite.

Akcia (Action) – najprimitívnejší prvok výpočtu. Nedá sa ďalej dekomponovať. Za akcie môžme považovať čokoľvek napr. vyhľadanie zamestnanca v databáze. Akcie nie je možné prerušiť. Názov akcie volíme podľa názvu konkrétneho postupu odohrávajúceho sa v danej akcii. UML zobrazuje akciu ako obdĺžnik so zaoblenými rohmi vnútri, ktorého je názov akcie.

Tok (Flow) – zobrazuje prechod z jednej akcie do druhej. Tieto prechody nastávajú automaticky po ukončení predchádzajúcej akcie. Symbolom je šípka smerujúca z jednej akcie do druhej. Tok môže byť ohodnotení booleovskou podmienkou. V prípade, že nie je splnená daná podmienka, nie je možné pokračovať v danej aktivite.

Rozhodovací uzol (Decision node) – rozdeľuje tok na viacej tokov podľa určitej podmienky. Vhodné je danú podmienku do diagramu zobraziť, aby bolo jasné ako sa toky rozdeľujú. Ak je z diagramu jasné podľa akej podmienky sa tok rozdeľuje, nemusí sa do daného diagramu zobraziť. Symbolom je kosoštvorec.

Zlučovací uzol (Merge node) – spája viacej tokov do jedného toku. Symbolom je kosoštvorec. Zo zlučovacieho uzlu pokračuje práve jeden tok.

Rozcestie (Fork) – rozdeľuje tok na viacej tokov, ktoré sa prevedú paralelne. Zobrazuje sa čiernou čiarou.

Spojenie (Join) – spája viacej paralelných tokov do jedného toku. Zobrazuje sa čiernou čiarou. Spája toky až v prípade, že všetky toky vstupujúce do tejto komponenty sa už odohrali. Nezáleží na poradí prevedenia tokov.

Počiatočný uzol (Initial node) – jedná sa o uzol, v ktorom začína výpočet aktivity. Jedna aktivita obvykle obsahuje jeden počiatočný uzol. Zriedkakedy ich môže byť viac, ak sa jedná o paralelný

výpočet nad danou aktivitou. Symbolom je čierny kruh.

Konečný uzol (Final node) – uzol, v ktorom končí výpočet aktivity. Aktivita môže obsahovať jeden alebo viacej konečných uzlov. Symbolom je čierny kruh vo vnútri kružnice.

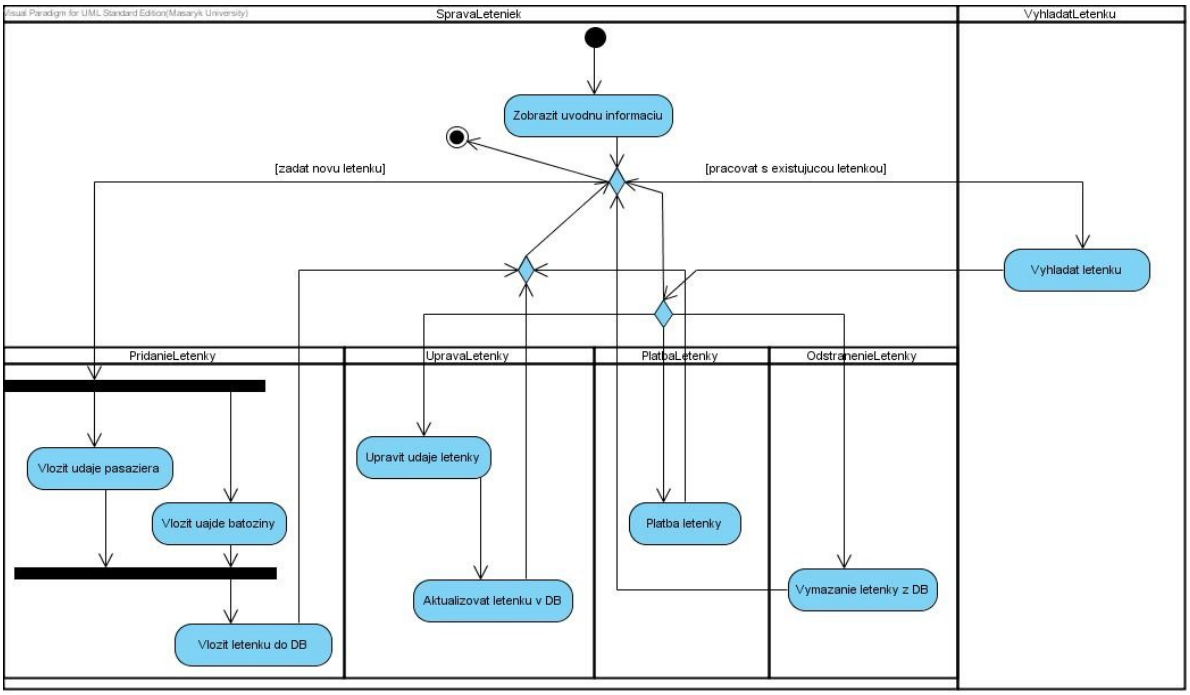
Oddiely (Swimlanes) – rozdelenie diagramu aktivít na viacej častí, aby sa tým znázornila zodpovednosť konkrétnej triedy, osoby, oddelenia za danú činnosť.

Časté chyby pri tvorbe [9]: Chybou pri tvorbe diagramu aktivít je, keď do jednej akcie vstupuje, či vystupuje viacej tokov reprezentujúcich nezávislé alternatívy, pretože UML takto uvedené toky pokladá za paralelné toky. Riešením je použitie rozhodovacieho alebo zlučovacieho uzlu. V prípade, ak sa jedná o toky vystupujúce z jednej akcie, zmením ich na jeden tok a ten nasmerujem do rozhodovacieho uzlu a z neho už zobrazím všetky pôvodné toky idúce do ďalších akcií. Ak ide o toky vstupujúce do akcie, pred akciu zobrazím zlučovací uzol, tam presmerujem všetky toky, ktoré by vchádzali do akcie a z rozhodovacieho uzlu pôjde len jeden tok do nasledujúcej akcie. Použitie zlučovacích a rozhodovací uzlov je dôležité preto, že keď nepoužijeme tieto uzly a necháme, aby do jednej akcie vstupovali viacej tokov, potom daná akcia čaká, kým sa prevedú všetky do nej vstupujúce toky, a až potom sa zahájí daná akcia, čo sa v prípade alternatívnych tokov nikdy nenastane.

4.3.2 Príklady diagramov

Na obrázku č. 8 je zobrazený diagram aktivít prípadu užitia *SpravaLeteniek* zobrazeného na diagrame prípadu užitia v kapitole 4.1.2 na obrázku č. 4.

Uvedený prípad užitia dovoľuje *SpotredkovateskejSpolocnosti* a *Kontrolorovi* pristupovať k databáze leteniek. Daný prípad užitia tak ako je znázornený na obrázku č. 4 komunikuje reláciou *include* s prípadom užitia *VyhladanieLetenky*, čo je na diagrame aktivít na obrázku č.8 zobrazené oddielom s názvom *VyhladanieLetenky*. Tento druhý oddiel reprezentuje prípad užitia *VyhladanieLetenky*. Keďže prípad užitia *VyhladanieLetenky* má len jednu akciu a to vyhľadanie letenky tak oddiel *VyhladanieLetenky* bude mať len jednu akciu. Keby mal prípad užitia, ktorý je v relácii *include* s daným prípadom užitia viacej akcií tak by boli všetky zobrazené v danom oddiely.

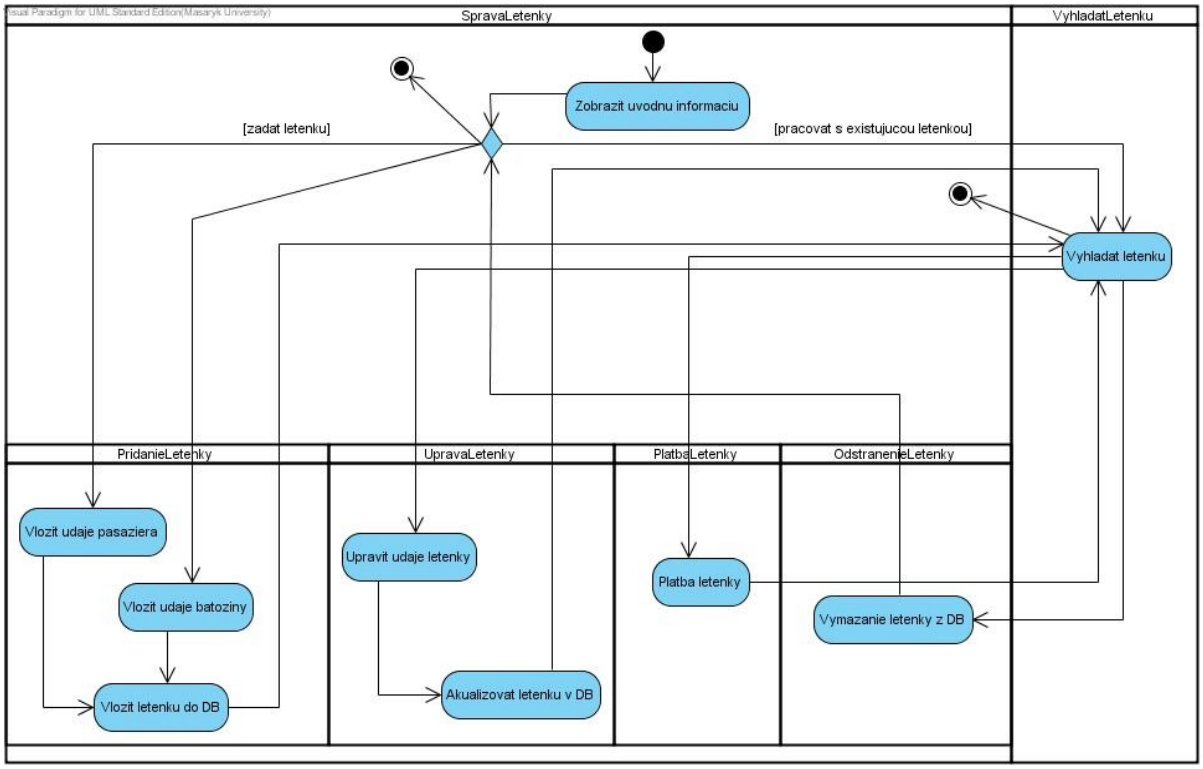


Obrázok č. 8: Diagram aktivít prípadu užitia SpravaLeteniek

Na začiatku tejto aktivity sa zobrazia úvodné informácie – akcia *Zobrazit uvodnu informáciu*. Následne sa ide do rozhodovacieho uzlu. Tu sa rozhoduje podľa toho, či je letenka zadaná alebo nie. V prípade, že letenka nie je zadaná, prechádza sa do oddielu *PridanieLetenky*. Vytvorí sa nová letenka, zadajú sa informácie o pasažierovi a o jeho batožiny. Keďže nezáleží na poradí, v akom sa tieto akcie odohrajú t. j. jedná sa o paralelné toky, použijem preto rozcestie (fork) a spojenie tokov (join). Po zadaní týchto údajov sa vloží nová letenka do databázy. Z akcie *Vložiť letenku do databázy* sa dá prejsť do akejkoľvek inej akcie na danom diagrame vďaka tokom, ktoré vedú z danej akcie a môžem dokonca aj danú aktivitu opustiť. Fakt, že je možné prejsť z jednej akcie to druhej je zobrazený orientovanými šípkami. Ak existuje orientovaná cesta medzi dvoma akciami, potom je možné prejsť z jednej akcie do druhej.

V prípade, že je už letenka zadaná tak letenku vyhľadám. Diagram aktivít nemá za úlohu presne opísať ako dané akcie prebiehajú, jeho úlohou je zobraziť, ako a za akých podmienok sa dá medzi jednotlivými akciami prebiehať. Preto napríklad vyhľadanie letenky zobrazím len ako akciu *VyhladanieLetenky* a neopisujem akým spôsobom letenku vyhľadávam, či je to pomocou mena alebo pasažiera alebo podľa nejakého jednoznačného identifikátoru letenky. Po vyhľadaní letenky prechádzam do rozhodovacieho uzlu, z ktorého môžem letenku buď upraviť – oddiel *UpravaLetenky* alebo zaplatiť – *PlatbaLetenky* alebo zrušiť – *OdstránenieLetenky*. Toky vychádzajúce z rozhodovacieho uzlu je dobré označiť nejakou podmienkou, podľa ktorej viem, ktorou cestou sa vydať. Tak ako je to zobrazené z prvého rozhodovacieho uzlu ak je letenka v databáze tak ju vyhľadám a ak nie je tak ju zadám. Ale prípade, že je jasné, o akú podmienku sa jedná, podmienka sa písať nemusí. Ako napríklad z rozhodovacieho uzlu, z ktorého môžem letenku buď upraviť, zaplatiť alebo zrušiť. Tu nie je nutné podmienku nejak zdôrazňovať, lebo si ju viem ľahko z diagramu odvodiť. Napr. do oddielu *PlatbaLetenky* budem pristupovať pod podmienkou, že chcem letenku zaplatiť.

Z danej akcie môžem opustiť aktivitu ak je možné sa dostať z danej akcie do koncového uzlu. Napríklad po vyhľadaní letenky môžem opustiť aktivitu *SpravaLeteniek*.



Obrázok č. 9: Diagram aktivít prípadu užitia SpravaLeteniek s častými chybami pri tvorbe

Na obrázku č. 9 je zobrazený diagram aktivít prípadu užitia *SpravaLeteniek* ako aj na obrázku č. 8 s tým rozdielom, že na obrázku č. 9 sú uvedené nejaké chyby, ktoré často vznikajú pri tvorbe diagramu aktivít.

Veľký rozdiel medzi obrázkom č. 8 a obrázkom č.9 je v sekcii *PridanieLetenky*. Podľa obrázku č. 8 by sa dala táto situáciach prezentovať nasledovne: Ak letenka nie je v databáze, vytvorí sa nová letenka, vložia sa údaje pasažiera a jeho batožiny. Nezáleží na tom v akom poradí ani nie je nutné zadať oba typy údajov, čo je pre systém letiska vyžadujúce, lebo každá letenka má svojho pasažiera, ale nie každý pasažier má batožinu. Na obrázku č. 9 je táto situácia znázornená tak, že sú zadané buď údaje o pasažierovi alebo údaje o batožine (pretože vychádzajú z rovnakého rozhodovacieho uzlu ale pod rôznymi tokmi) a letenka sa vloží do databázy až, keď sú vyplnené oba typy údajov (čaká až kým sa prevedú oba toky vstupujúce do danej aktivity), čo vlastne nenastane vôbec nikdy. Vhodné riešenie je ukázané na obrázku č. 8 použitým rozcestia a spojenia tokov.

Podobná situácia nastáva pri vstupe do akcie *VyhladatLetenku.* Tu je však vhodnejšie použiť zlučovací uzol, nakoľko sa jedná o toky alternatívne.

4.4 Sekvenčný diagram (Sequence diagram)

4.4.1 Popis

Účel diagramu [12]: Sekvenčný diagram patrí medzi interakčné diagramy. Reprezentuje interakciu, ktorá je predstavovaná množinou správ vymenených medzi objektmi na splnenie požadovanej operácie alebo získanie výsledku. Používa sa v prípadoch, kde je dôležité zobraziť časové súvislosti interakcií. Využíva sa na lepšie pochopenie diagramu tried, hlavne tých tried, ktoré menia svoje správanie v čase.

Komponenty [7]: *Účastník (Actor)* – podobne ako na diagrame prípadu užitia sa jedná o entitu, ktorá svojim chovaním ovplyvňuje systém. Sekvenčný diagram má jedného účastníka, ktorý spúšťa danú interakciu. Zobrazuje sa ako panáčik s názvom účastníka.

Objekt (Object) – zobrazuje sa podobne ako na diagrame objektov. Jedná sa o konkrétnu inštanciu nejakej triedy. Narozdiel od zobrazenia na diagrame objektov zobrazuje sa len názov objektu bez atribútov a metód. V prípade, keď sa jedná o konkrétny objekt, napíšem názov objektu v tvare *<názov objektu>:<názov triedy>* napr. *JozefMrkvicka:Pasazier*. Ak chcem zobraziť všetky objekty danej triedy napíšem názov objektu takto *:<názov triedy>* napr. všetky inštancie triedy *Pasazier* uvediem ako *:Pasazier*. Na sekvenčnom diagrame sú zobrazené len tie objekty z diagramu objektov, ktoré vstupujú do danej interakcie.

Čiara života (Lifeline) – ukazuje, kedy objekt žije (zobrazené čiarkovanou čiarou kolmou na objekt) a kedy objekt zaniká (čiarkovaná čiara je ukončená krížikom).

Správa (Message) – vyjadruje špecifický typ komunikácie medzi čiarami života behom interakcie. Spúšťa metódu v inom alebo rovnakom objekte. Je znázornená pomocou šípky. Metóda môže obsahovať typ návratovej hodnoty, či vstupné parametre, ako to bude zobrazené na konkrétnom diagrame záleží od miery detailu.

Najpoužívanejšie typy správ: *Synchrónna správa* – odosielateľ čaká na dokončenie úlohy príjemcom správy. *Asynchrónna správa* – odosielateľ odošle správu a pokračuje v behu – nečaká, až príjemca dokončí svoju úlohu. *Návrat správy* – príjemca skoršej správy vracia aktivitu jej odosielateľovi. Býva znázornená prerušovane. *Tvorba objektu* – odosielateľ vytvorí inštanciu klasifikátoru uvedeného príjemcom. Táto správa býva ohodnotená stereotypom *<<create>>* a vchádza do novovytvoreného objektu a nie do jeho čiary života ako je to v prípade ostatných správ.

Uvoľnenie objektu – odosielateľ ukončí život príjemcovi. Ukončenie života objektu je znázornené ukončenou čiarou života pomocou krížika. Táto správa býva ohodnotená stereotypom <<*destroy*>>.

Iterácia – zobrazuje opakovanie niektorej časti sekvenčného diagramu podľa určitej podmienky. Najčastejšie operátory iterácií: *Podmienené spustenie (Optional execution)* skratka *opt* – operácie vo vnútri bloku sú spustené po splnení podmienky. *Voliteľne spustenie (Conditional execution)* skratka *alt* – blok je rozdelený na viac častí. Každá časť má svoju podmienku. Uvedená časť sa prevedie ak je splnená podmienka. *Paralelné spustenie (Parallel execution)* skratka *par* – blok je rozdelený na viac častí. Uvedené bloky sa spúšťajú paralelne. *Cyklické spúšťanie (Loop execution)* skratka *loop* – blok sa opakuje pokým sa uvedená podmienka vyhodnotí ako pravda. *Odkaz na iný diagram (Reference)* skratka *ref* – zobrazuje názov aktivity, ktorá je spúšťaná separátnych diagramom. *Negácia (Negative)* skratka *neg* – ukazuje nesprávnu interakciu. Kritická sekcia (Critical region) skrakta region – blok môže mať spustenú v danom okamihu len jednu časť.

Časté chyby pri tvorbe [9]: Najčastejšou chybou pri tvorbe sekvenčného diagramu je, keď objekt posiela správu inému objektu, na ktorého nemá referenciu. Referenciu môže získať tromi základnými spôsobmi. Prvý spôsob, že jeden objekt má referenciu na druhý objekt priamo medzi svojimi atribútmi. Na diagrame tried je to zobrazené asociáciou medzi uvedenými triedami. Uvedený spôsob získania referencie je zobrazený na obrázku č. 10. Objekt *Letenka* má medzi svojimi atribútmi, atribút *Let* a teda má na let priamu referenciu. Ďalším spôsobom získavania referencie je, že objekt si o referenciu na daný objekt požiada tretí objekt. Tento tretí objekt riadi správu objektov t. j. zadáva nové objekty, vyhľadá objekty alebo ruší objekty. Tento objekt najčastejšie pomenovaný *SpravaObjektov* je v systéme jedinečný. Tento princíp získanie referencie je zobrazený na obrázku č. 10. Účastník *SprostredkovatelskaSpolocnost* požiada o získanie odkazu na žiadaný let objekt *SpravaLetov*. Tretí spôsob získania referencie je globálne sprístupnenie objektu napr. pomocou vzoru singleton.

Každý sekvenčný diagram zobrazuje objekty z diagramu objektov. Metóda, ktorú posiela jeden objekt pomocou správy druhému objektu, musí druhý objekt obsahovať. Napr. *ManazerLetiska* pošle objektu *SpravaLetov* správu ohodnotenú metódou *vyhladajLet*. Tak potom objekt *SpravaLetov* musí túto metódu obsahovať inak nemôže prebehnúť daná akcia. Porušenie konzistencie medzi sekvenčným diagramom a diagramom tried sa považuje sa chybu.

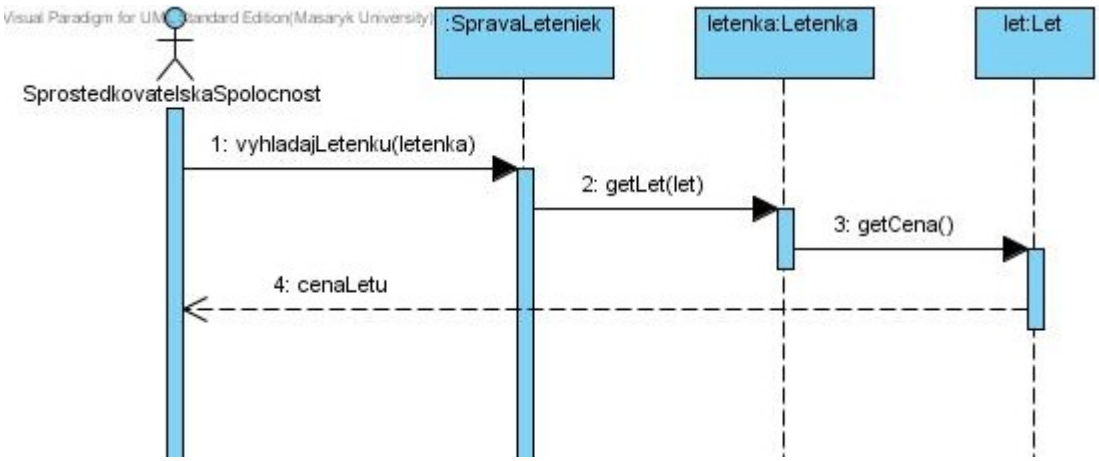
Keď chcem zrušiť objekt, ktorý má na seba naviazané iné objekty, zrušeným objektu sa k týmto objektom nedá dostať. Riešením tohto problému je povoliť zmazanie takéhoto objektu až po zmazaní objektov naň naviazaných alebo požiadať užívateľa o súhlas s ich zmazaním a zmazať automaticky.

Každú interakciu v sekvenčnom diagrame musí zahajovať aktér, tým že pošle nejakému objektu správu. Objekt pošle inému objektu správu len ak je to nutné na základe správy, ktorú obdržal. Objekt nemôže samovoľne poslať správu inému objektu.

4.4.2 Príklady diagramov

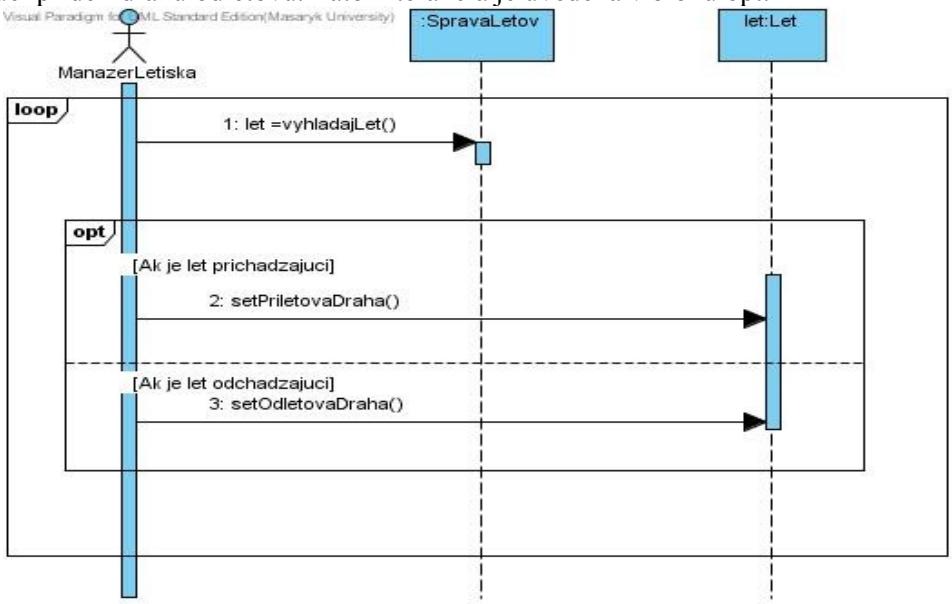
Nasledujúce sekvenčné diagramy vychádzajú diagramu tried zobrazeného v prílohe na obrázku č. 30.

Na obrázku č. 10 je zobrazená akcia získanie ceny letu konkrétnej letenky. Uvedenú akciu začína účastník *SprostredkovatelskaSpolocnost* tým, že požiada objekt *SpravaLeteniek* o vyhľadanie letenky, ktorú mu dal ako parameter metódy *vyhladajLetenku*. Objekt *SpravaLeteniek*, ktorý vedie správu leteniek, uvedenú letenku vyhľadá a pošle jej správu žiadajúcu o vyhľadanie letu uvedeného na letenke. Letenka let vyhľadá a pošle objektu letu správu, ktoré žiada o získanie ceny letu. Cena letu je následne poslaná účastníkovi interakcie návratovou správou.



Obrázok č. 10: Sekvenčný diagram interakcie získanie ceny letu na letenke.

Na obrázku č. 11 je zobrazená akcia prideľovania odletových a príletových dráh jednotlivým letom. Novými prvkami na uvedenom diagrame je použitie iterácií *loop* a *opt*. *Loop* značí cyklické opakovanie. V tomto prípade *ManazerLetiska* prideľuje letov dráhy, a keďže sa jedná o viacej letov celá akcia je uvedená v bloku *loop*. To, či ManazerLetiska pridelí letu odletovú ale príletovú dráhu záleží na tom, o aký let sa jedná. Ak je let prichádzajúci pridelí mu príletovú dráhu a ak je len odchádzajúci pridelí dráhu odletovú. Táto interakcia je uvedená v bloku opt.



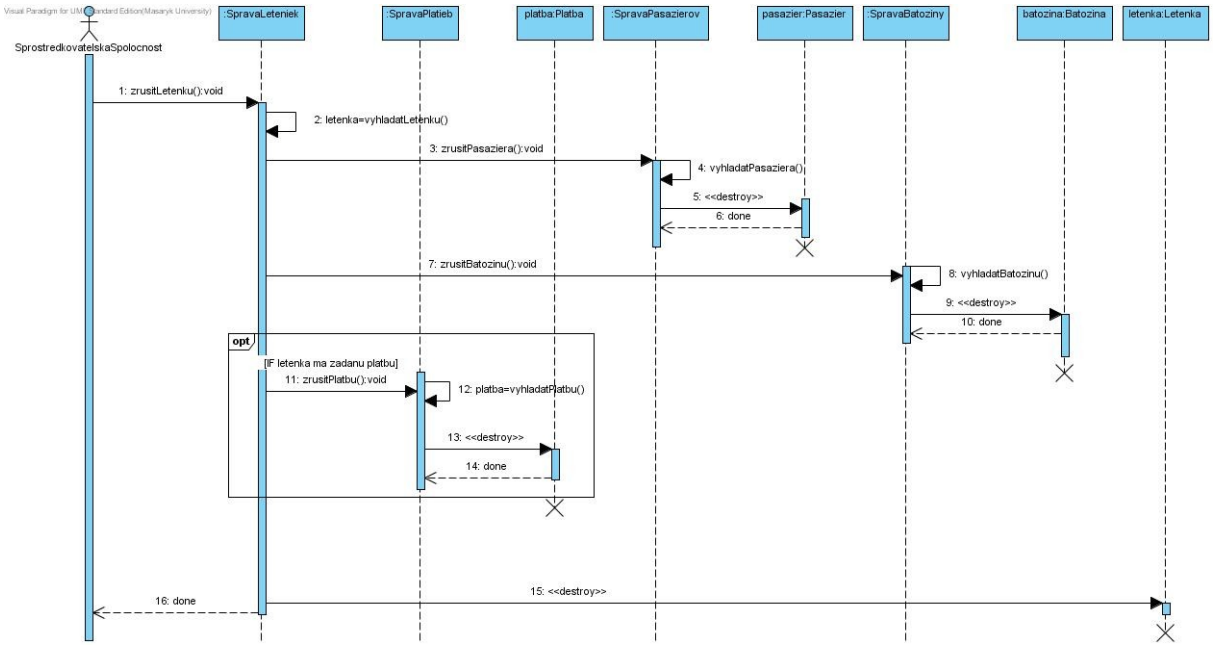
Obrázok č.11: Sekvenčný diagram zobrazujúci prideľovanie príletovych alebo odletových dráh letom

Na obrázku č. 12 je zobrazená interakcia zrušenie letenky, ktorá je súčasťou prípadu užitia *SpravaLeteniek*.

Pri rušení objektov je dôležité dať pozor na to, aby sa zrušili aj objekty, na ktoré daný objekt odkazuje. V tomto prípade, ak chcem zrušiť letenku, tak viem, že s danom letenkou je spojený objekt *Pasazier*, *Batozina* a poprípade aj *PlatbaLetenky* ak je letenka zaplatená. Ak chcem danú letenku zrušiť musím najskôr zrušiť tieto objekty, lebo bez letenky už nemajú v systéme zmysel a bez konkrétnej

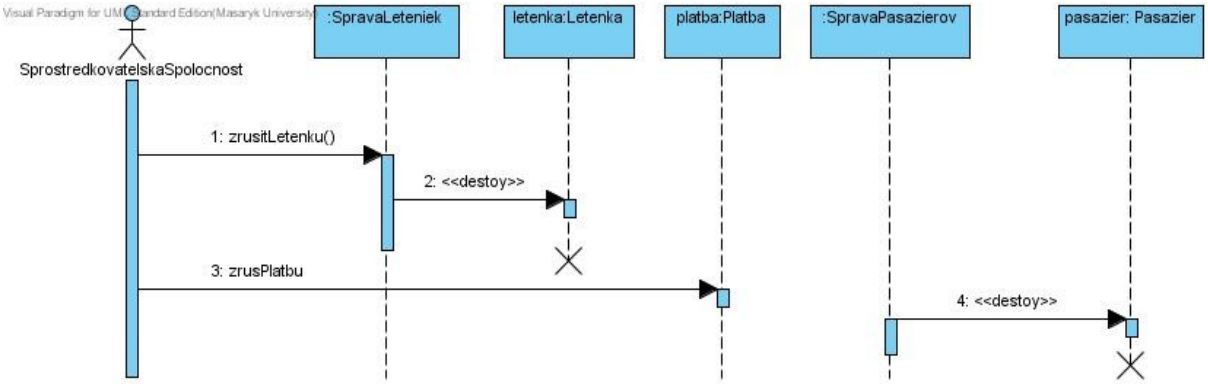
letenky sa nevieme k týmto objektom ani dostať.

K novým prvkom sekvenčného diagramu na obrázku č. 12 oproti predchádzajúcim sekvenčným diagramom je použitie správy *destroy* – zrušenie objektu zo systému. Napríklad *SpravaPasazierov* zruší pasažiera poslaním tejto správy. A od toho okamihu už daný pasažier neexistuje v systéme – jeho čiara života je ukončená. Objekt *SpravaPasaziera* zaháji zrušenie pasažiera po obdržaní správy *ZrusitPasažiera,* s atribútmi konkrétneho pasažiera, od objektu *SpravaLetov*, ktorý poslal túto správu na základe obdržanej správy *ZrusitLetenku* od účastníka tejto interakcie. Žiadny objekt nemôže samovoľne zahájiť nejakú interakciu.



Obrázok č. 12: Sekvenčný diagram zobrazujúci interakciu zrušenie letenky.

Na obrázku č.13 je zobrazená interakcia *ZrusenieLetenky* s častými chybami pri tvorbe sekvenčných diagramov. Korektné riešenie tejto interakcie je zobrazené na obrázku č.12.



Obrázok č. 13: Sekvenčný diagram zobrazujúci interakciu zrušenie letenky s chybami pri tvorbe.

Prvý nedostatok je uvedený v interakcii zrušenie letenky. Najskôr treba zrušiť všetky objekty,

na ktoré letenka odkazuje t. j. objekt *Pasazier*, *Batozina* a po prípade, ak je letenka zaplatená, tak aj objekt *Platba* a až potom odstrániť objekt *Letenka*. Keby sa to udialo tak, ako je to nakreslené na obrázku č. 13, tak tým, že najskôr zruším letenku sa už nikdy nedostanem k objektom, na ktoré letenka ukazovala.

Správa *zrusPlatbu* je uvedená chybne. Jednak sa nedá po odstránení letenky dostať ku platbe patriacej k danej letenke a navyše správa *zrusPlatbu* nemôže byť adresovaná konkrétnej platbe, ale objektu *SpravaPlatieb,* ktorý je v systému jedinečný a vedie správu platieb. Objekt *Platba* obsahuje len metódy, ktoré nastavujú atribúty konkrétnej platby, metódy týkajúce sa správy platieb ako je napr. metóda *vyhladaniePlatby* obsahuje objekt *SpravaPlatieb*.

Posledným uvedeným nedostatkom je interakcia *zruseniePasaziera*. Túto interakciu nezahajuje účastník ani iný objekt na základe obdržanej správy od účastníka. Takto ako je to uvedené na obrázku č. 13 to znamená, že objekt *SpravaPasazier* môže samovoľne odstraňovať pasažierov zo systému, čo je neprijateľné. Navyše uvedená interakcie nie je vôbec spojená so zvyškom interakcie, čo by znamenalo, že k uvedenej interakcii nepatrí a teda nemá byť zobrazená na uvedenom sekvenčnom diagrame.

4.5 Komunikačný diagram (Communication diagram)

4.5.1 Popis

Účel diagramu [14]: Komunikačný diagram patrí medzi interakčné diagramy. Popisuje komunikáciu medzi objektmi. Je izomorfný so sekvenčným diagramom – dajú sa automatizovane prevádzať z jedného tvaru na druhý, ale len v prípade, keď sa jedná o jednoduchý sekvenčný diagram bez použitia štruktúrovaných mechanizmov ako sú iterácie. Použitie komunikačného diagramu je vhodnejšie v prípadoch, keď chceme zdôrazniť štrukturálne aspekty komunikácie t. j kto s kým komunikuje a nezáleží nám až tak na zdôraznení časových súvislostí interakcie.

Komponenty [12]: *Aktér (Actor)* – podobne ako na sekvenčných diagramoch zahajuje interakciu a je znázornený pomocou panáčika.

Objekt (Object) – je zobrazený ako uzol na komunikačnom diagrame. Predstavuje konkrétny objekt z diagramu objektov podobne ako je tomu pri sekvenčných diagramoch, s tým rozdielom, že v komunikačných diagramoch nemá zobrazenú čiaru života.

Správa (Message) – komponenta, ktorú si objekty posielajú navzájom. Správa nesie názov, ktorý sa volí podobne ako u sekvenčných diagramov. Každá správa by mala mať svoj smer určený šípkou, aby bolo jasné, ktorý objekt posiela správu, a ktorý objekt danú správu prijíma. Čas tu nevystupuje ako zvláštna dimenzia ako je tomu v sekvenčných diagramom, kde čas plynie zhora dolu, preto musí byť sekvencia zasielaných správ určená pomocou poradového čísla sekvencie.

Časté chyby pri tvorbe: Chyby tvorby komunikačného diagramu sú podobné chybám pri tvorbe sekvenčného diagramu. Sú uvedené v kapitole 4.4.1, preto ich už tu nebudem zmieňovať.

Špecifickou črtou komunikačného diagramu je, že na rozdiel od sekvenčného diagramu nezobrazuje objektom čiary života, a preto na komunikačnom diagrame prijímaná správa smeruje do objektu a nie do jeho čiary života, ako je tomu na diagrame sekvenčnom. Z toho dôvodu nie je prehľadné, či v danom okamihu uvedený objekt existuje, alebo nie. Preto sa môže stať, že objekt alebo aktér pošle správu objektu, ktorý v tom danom okamihu v systéme neexistuje.

4.5.2 Príklady diagramov

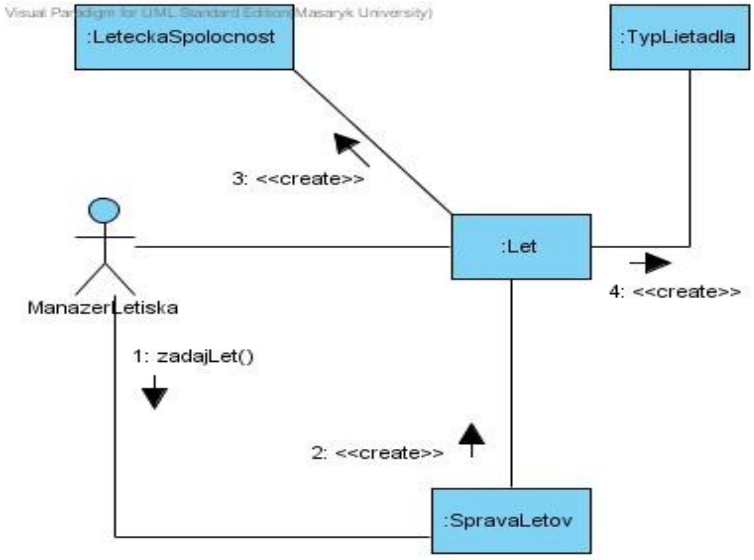
Na obrázku č. 14 je znázornená interakcia pridanie nového letu do systému pomocou komuni-

kačného diagramu. Uvedená interakcia je súčasťou prípadu užitia *SpravaLetov.*

Účastníkom tejto interakcie je *ManazerLetiska*, ktorý zadáva nové lety do systému prostredníctvom prípadu užitia *SpravaLetov.* Uvedený účastník zahajuje interakciu tým, že pošle objektu *SpravaLetov* správu žiadajúcu o zadanie konkrétneho letu. Požiada o to prostredníctvom metódy *zadajLet*. Objekt *SpravaLetov* je v systéme jedinečný a vedie správu letu t. j. umožňuje zadanie nového letu, vyhľadanie letu alebo zrušenie letu. Uvedená správa má zobrazenú šípku, ktorá určuje smer správy, v tomto prípade je odosielateľom správy *ManazerLetiska* a príjemcom objekt *SpravaLetov.*

Objekt *SpravaLetov* na základe obdržanej správy od účastníka vytvorí nový let pomocou správy ohodnotenej stereotypom <<*create*>>, ktorým býva ohodnotená správa vytvárajúca nový objekt. Novovzniknutý objekt *Let* následne vytvorí nový objekt *LeteckaSpolocnost* a objekt *TypLietadla*.

Nakoľko komunikačný diagram nemá pre čas určenú dimenziu je dôležité jednotlivé správy číslovať, aby bolo jasné, v akom poradí sa odohrávajú.



Obrázok č.14: Komunikačný diagram interakcie pridanie nového letu

Na obrázku č. 15 je zobrazená interakcia zrušenie letu zo systému pomocou komunikačného diagramu. Uvedená interakcia je súčasťou prípadu užitia *SpravaLetov.*

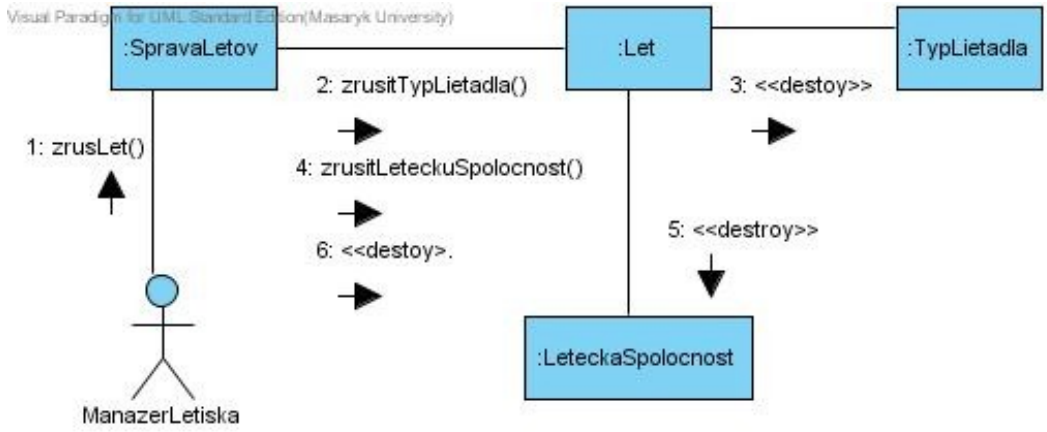
Novinkou oproti predošlému diagramu je zrušenie objektu pomocou stereotypu <<*destroy*>>. Uvedená interakcia prebieha nasledovne. Účastník požiada objekt *SpravaLetov* o zrušenie konkrétneho etu pomocou operácie *zrusLet*. Predtým ako sa zruší konkrétny let, je dobré, aby sa zrušili aj objekty, na ktoré let ukazuje t. j. objekt *SprostedkovatelskaSpolocnost* a objekt *TypLietadla*. Objekt *SpravaLetov* požiada o zrušenie týchto objektov a objekt *Let* na základe obdržanej správy zruší uvedené objekty pomocou správy ohodnotenej stereotypom <<*destroy*>>. A až keď sú zrušené tieto objekty zruší sa objekt *Let.*

Na obrázku č. 16 je zobrazená interakcia zrušenie letu s chybou, ktorej je možno sa dopustiť pri tvorbe komunikačného diagramu.

Keďže komunikačný diagram nemá pre zobrazenie času špeciálnu dimenziu, ako je tomu pri sekvenčných diagramoch môže sa stať, že je správa poslaná takému objektu, ktorý už v daný okamih

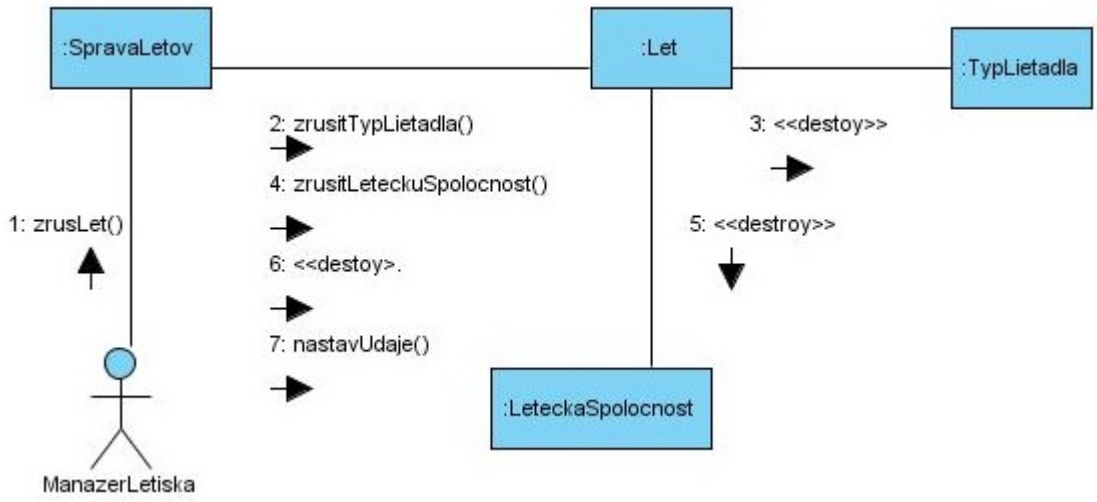
neexistuje v systéme, ako je to uvedené na obrázku č. 16. *ManazerLetiska* posiela správu žiadajúcu o zmenu údajov letu, ktorý už pred tým zo systému žiadal odstrániť. Korektné riešenie je ukázané na obrázku č. 15.

Všetky tri komunikačné diagramy vychádzajú z diagramu tried zobrazeného v prílohe na obrázku č. 30.



Obrázok č.15: Komunikačný diagram interakcie zrušenie letu

l



Obrázok č.16: Komunikačný diagram interakcie zrušenie letu s chybami pri tvorbe

4.6 Diagram prehľadu interakcií (Interaction overview diagram)

4.6.1 Popis

Účel diagramu [14]: Diagram prehľadu interakcií patrí medzi interakčné diagramy. Kombinuje koncepty diagramu aktivít a sekvenčného diagramu, kde hlavnú kostru tvorí diagram aktivít a jednotli-

vé akcie v diagrame aktivít sú zobrazené pomocou sekvenčného diagramu.

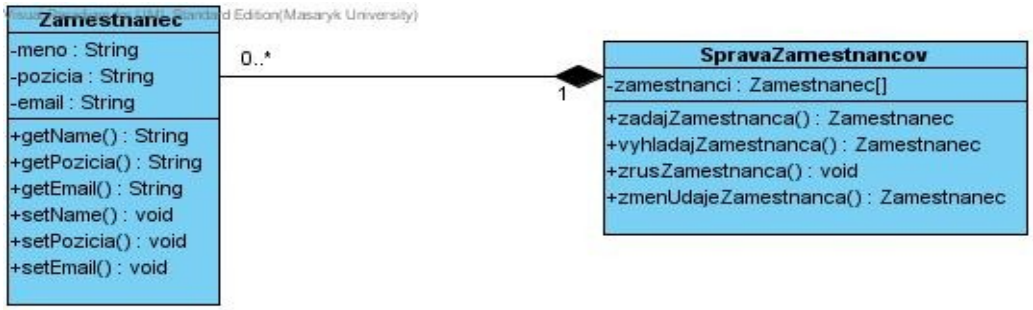
Komponenty [12]: Keďže uvedený diagram spája koncepty diagramu aktivít a sekvenčného diagramu, medzi jeho komponenty budú patriť komponenty uvedených dvoch diagramov. Tieto komponenty nebudem v tomto odstavci ďalej rozoberať, nakoľko sú rozobrané v kapitolách 4.3.1 a 4.4.1. Diagram prehľadu interakcií používa iba dve špeciálne komponenty a to:

Interakcia (Interaction) – slúži na popis akcie pomocou sekvenčného diagramu. V tomto prípade sa jedná o sekvenčný diagram, ktorý ešte nie je utvorený v procese analýzy. Po zvolení tejto možnosti UML zobrazí prázdny štvorec, ktorý má v ľavom hornom rohu napísanú skratku sd. Po vytvorení je sekvenčný diagram, ktorý rozoberá danú akciu zobrazený v uvedenom štvorci.

Použitie interakcie (Interaction use) – opäť slúži na popis akcie, ale tento krát pomocou sekvenčného diagramu, ktorý už je vytvorený. Keď vyberiem túto voľbu UML zobrazí štvorec, ktorý má v ľavom hornom rohu skratku ref a čaká na zadanie názvu sekvenčného diagramu, na ktorý sa bude odkazovať.

Časté chyby pri tvorbe: Častými chybami pri tvorbe daného diagramu sú chyby, ktoré sa dopúšťajú pri tvorbe diagramu aktivít a sekvenčného diagramu. Uvedené chyby sú zmienené v kapitolách 4.3.1 a 4.4.1.

4.6.2 Príklady diagramov



Obrázok č. 17: Diagram tried prípadu užitia SpravaZamestnancov

Na obrázku č. 17 je zobrazený diagram tried prípadu užitia SpravaZamestnancov. Nasledujúce diagramy prehľadu interakcií budú vychádzať z uvedeného diagramu tried.

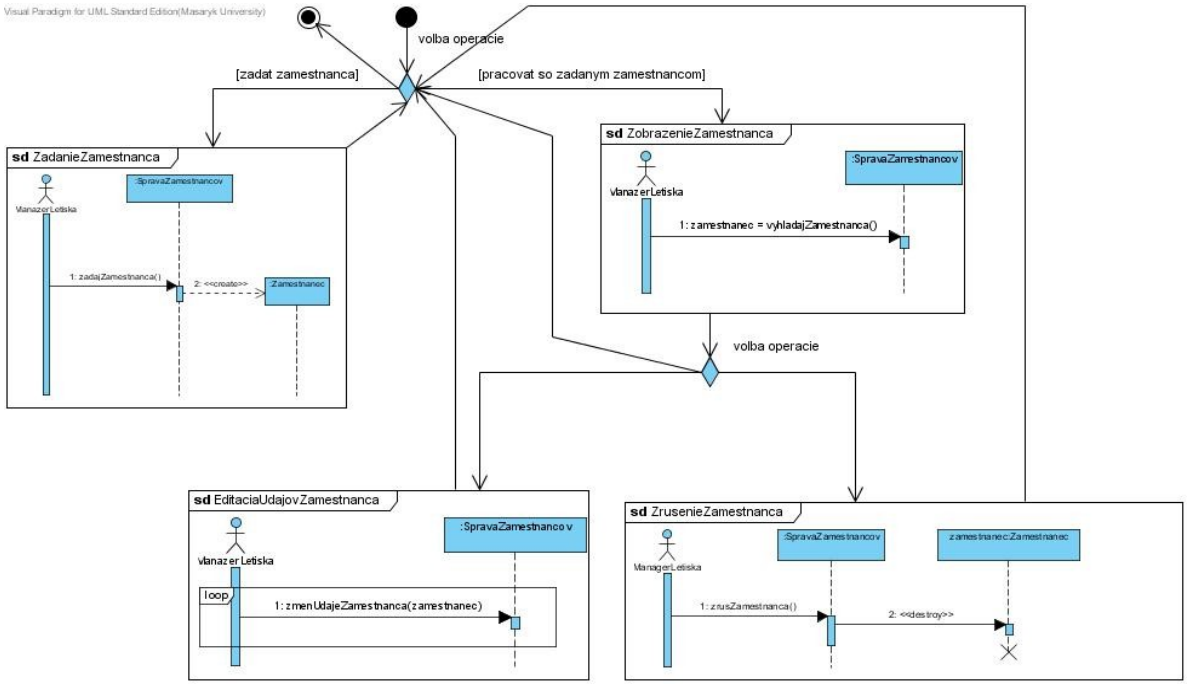
Na obrázku č. 18 je zakreslený prípad užitia *SpravaZamestnancov* pomocou diagramu prehľadu interakcií. Uvedený prípad užitia je zobrazený pomocou textovej špecifikácie prípadu užitia na obrázku č. 4 v kapitole 4.2.2 a pomocou diagramu aktivít v prílohe na obrázku č.27.
Na začiatku tohto diagramu sa účastník rozhodne, či je zamestnanec zadaný do systému alebo nie. Podľa toho vyberie jeden z tokov vychádzajúcich z prvého rozhodovacieho uzlu.

Ak zamestnanec nie je zadaný do systému, tak ho zadá, čo je zobrazené pomocou interakcie *ZadanieZamestnanca*. V tejto interakcií je zakreslená situácia pomocou sekvenčného diagramu. *ManazerLetiska* požiada objekt *SpravaZamestnancov* o vytvorenie zamestnanca a ten potom vytvorí nového zamestnanca. Uvedená interakcia tak ako je zobrazená na obrázku č.18 môže viesť k nejasnostiam. Existujú totiž dva spôsoby interpretácia zmieneného sekvenčného diagramu. A to, že metóda *zadajZamestnanca* vytvorí nový objekt zamestnanca a jeho údaje už nevyplňuje, čo je v uvedenom prípade zlé riešenie. Potom je dobré uviesť ďalšie metódy ako *nastavMeno* (metóda, ktorú vyvolá objekt *SpravaZamestnancov* na objekte konkrétneho zamestnanca), a tým nastaví jeho údaje. Druhou interpretáciou, ktorú som použila aj ja na obrázku č.18. je idea, že metóda *zadajZamestnanca*, ktorú volá *ManazerLetiska* na objekte *SpravaZamestnancov* obsahuje atribúty, ktoré slúžia na nastavenie

údajov o zamestnancovi po jeho vytvorení. Obidve uvedené možnosti sú správne, záleží len na konkrétnom analytikovi, ktorú možnosť považuje za vhodnejšiu.

V prípade, že zamestnanec je zadaný v databáze zamestnancov, tak sa vyhľadá – interakcia *ZobrazenieZamestnanca*. Následne môžme editovať údaje zamestnanca – interakcia *EditaciaUdajov-Zamestnanca* alebo môžme zamestnanca z databáze odstrániť – interakcia *ZrusenieZamestnanca*. To, ktorou vetvou pôjdeme sa rozhodne v druhom rozhodovacom uzle t. j po akcii *ZobrazenieZamestnanca*.

Statická štruktúra prípadu užitia *SpravaZamestnancov* podľa diagramu tried je zobrazená na obrázku č. 17.



Obrázok č.18: Diagram prehľadu interakcií prípadu užitia SpravaZamestnancov.

Na obrázku č.19 je zobrazený diagram prehľadu interakcií prípadu užitia *SpravaLeteniek*, ktorý je pomocou diagramu aktivít zobrazený na obrázku č. 6 v kapitole 4.3.2.

Novinkou tohto diagramu oproti predchádzajúcemu diagramu je použitie interakcie. Táto komponenta poukazuje už na vytvorené sekvenčné diagramy. Použitie interakcie *ZadatLetenku.* je zobrazené v prílohe na obrázku č. 33, *PlatbaLetenky* v prílohe na obrázku č.31. A *ZrusitLetenku* v kapitole 4.4.1 na obrázku č.11.

.

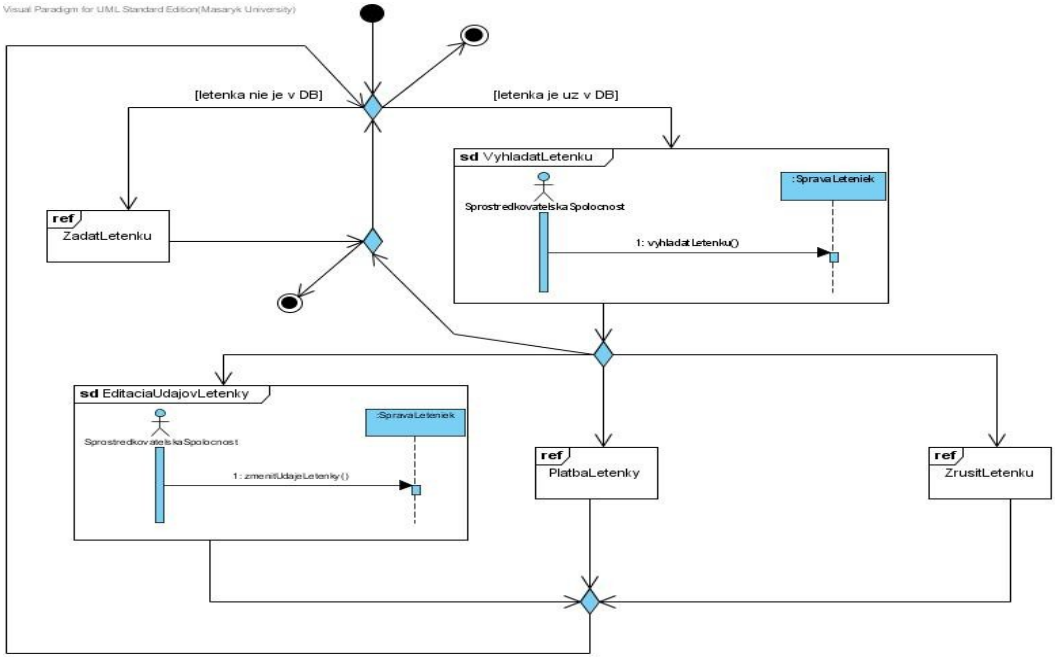
Na obrázku č.20 je uvedený diagram prehľadu interakcií prípadu užitia *SpravaZamestnancov* s uvedenými častými chybami pri tvorbe tohto diagramu.

Prvým veľkým nedostatkom na tomto diagrame je chyba, ktorá je typická pre diagramy aktivít. Z interakcie *ZobrazenieZamestnanca* vychádzajú celkom tri toky. Táto chyba sa opravuje pomocou rozhodovacieho uzlu. Z interakcie *ZobrazenieZamestnanca* povedie len jeden tok do rozhodovacieho uzlu a z neho sa tok rozdelí na tri rôzne toky.

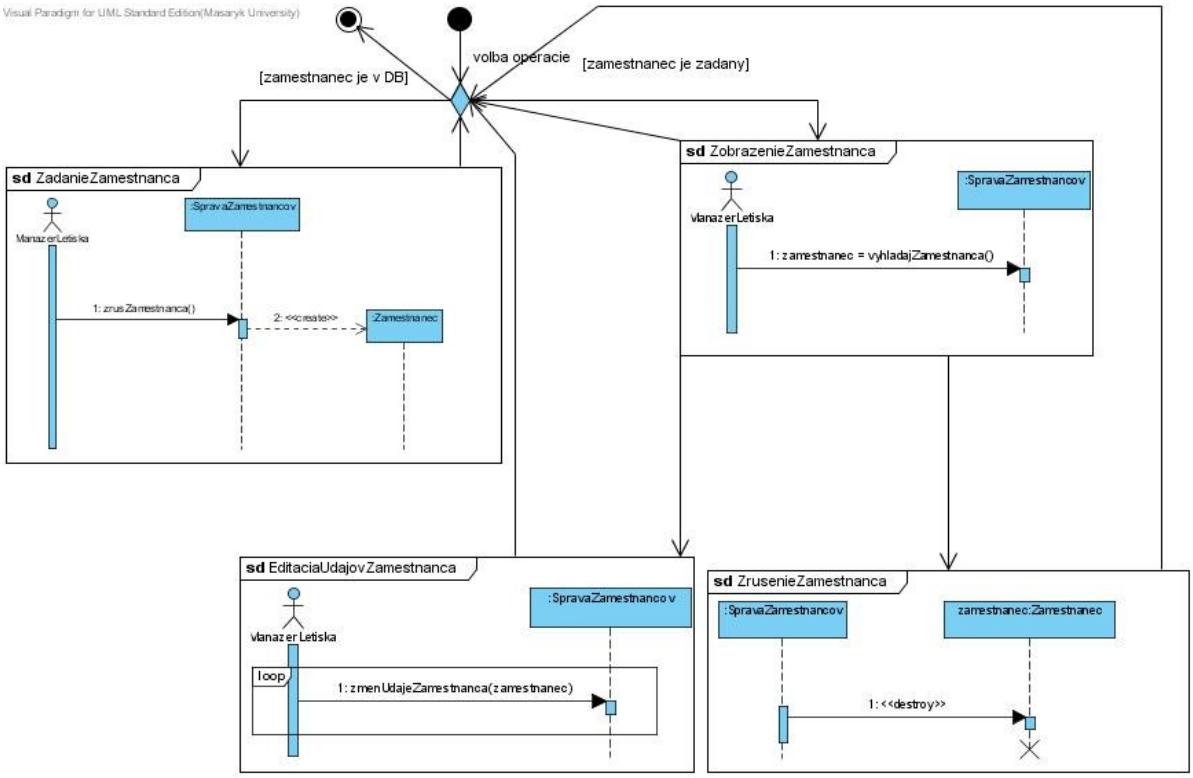
Ďalšou chybou na uvedenom diagrame je chyba v interakcii *ZrusenieZamestnanca*. Každá interakcia musí byť zahájená účastníkom. A nie ako je to uvedené v tejto interakcii. Interakcia sa nemôže sama spustiť. Objekt *SpravaZamestnancov* nepošle sám od seba správu, ktorou chce zrušiť konkrétny objekt. Riešením tohto nedostatku je zadanie účastníka. V tomto prípade *ManazerLetiska*,

ktorý zaháji uvedenú interakciu tým, že zašle správu *zrusZamestnanca* objektu *SpravaZamestnancov.*

Jedno z možných správnych riešení týchto nedostatkov je na obrázku č.18.



Obrázok č.19: Diagram prehľadu interakcií prípadu užitia SprávaLeteniek.



Obrázok č.20: Diagram prehľadu interakcií prípadu užitia SpravaZamestnancov s chybami pri tvorbe

4.7 Diagram časovania (Timing diagram)

4.7.1 Popis

Účel diagramu [12]: Diagram časovania je špeciálny typ interakčného diagramu, ktorý modeluje časové obmedzenia spojené so zmenou stavov rôznych objektov. Slúži na modelovanie systémov pracujúcich v reálnom čase. Existujú dva spôsoby notácie. Na obrázku č. 21 je uvedená kompaktná forma zápisu. V tejto podobe je dôraz kladený predovšetkým na stavy a relatívny čas a nie na čas absolútny modelovaný pomocou časovej osi, ako je tomu pri druhej forma zápisu zobrazenej na obrázku č. 22.

Komponenty[7]: *Čiara života (Lifeline)* – predstavuje čiaru života objektu, ktorého zmenu stavov modelujeme. Objekt nesie názov podľa názvu objektu v diagrame objektov.

Časová osa(Timeline) – zaznamenáva plynutie času. Čas na nej plynie zľava doprava.

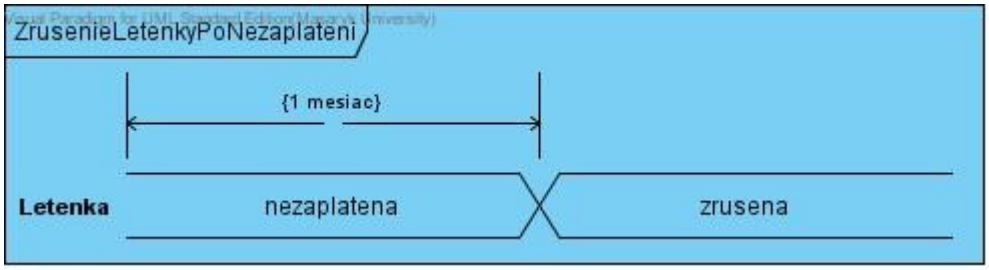
Stavy (States) – stavy, ktoré môže uvedený objekt nadobúdať v čase.

Časté chyby pri tvorbe: Prechody medzi stavmi čiary života objektu na diagrame časovania sú závislé na čase t. j. z jedného stavu sa prechádza do iného stavu po uplynutí určitého časového okamihu. Chybou pri tvorbe diagramu časovania býva zobrazenie prechodov medzi stavmi po splnení určitej podmienky. Takéto zmeny stavov sa zobrazuje pomocou stavového diagramu a nie pomocou diagramu časovania.

4.7.2 Príklady diagramov

Na obrázku č. 21 je zobrazená akcia zrušenie letenky po nezaplatení. Diagram sa snaží zachytiť nasledujúcu situáciu: Pasažier si zadá prostredníctvom sprostredkovateľskej spoločnosti do systému letenku. Pasažier je povinný zaplatiť letenku do jedného mesiaca od zadania letenky. V prípade, že tam neučiní je jeho letenka zrušená.

V úlohe čiary života vystupuje letenka. Čiara života zobrazuje objekt v diagrame objektov, ktorý mení svoje stavy v závislosti na čase. Na uvedenom diagrame má letenka zobrazené dva stavy a to: nezaplatená a zrušená. Po zadaní letenky do systému je stav letenky nastavený na nezaplatená. Ak pasažier po uplynutí jedného mesiaca letenku nezaplatí, je jeho letenka zrušená – letenka prechádza do stavu zrušená.



Obrázok č. 21: Diagram časovania zobrazujúci akciu zrušenie letenky po nezaplatení

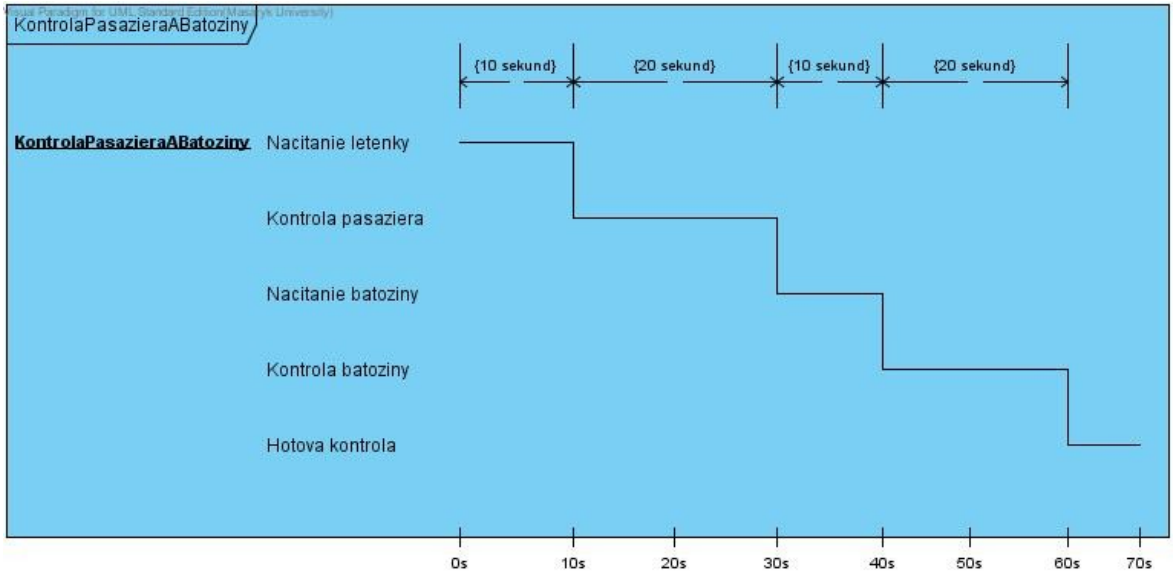
Na obrázku č. 22 je zobrazená akcia kontrola pasažiera a jeho batožiny. Uvedený diagram časovania je zakreslený v inej forme ako diagram na obrázku č. 21. Nástroj UML povoľuje obe formy a jednotlivé formy sa dajú automatizovane prevádzať.

Kontrola uvedená na obrázku č. 22 prebieha nasledovne. Na začiatku kontroly je potrebné načítať letenku podľa nejakého jednoznačného identifikátoru letenky. Vyhľadanie a overenie letenky trvá 10 sekúnd. Potom sa začne kontrolovať pasažier – prejde sa do stavu *Kontrola pasažiera.* Kontrola pasažiera trvá 20 sekúnd. Po uplynutí tejto doby sa načíta identifikátor batožiny. Načítanie

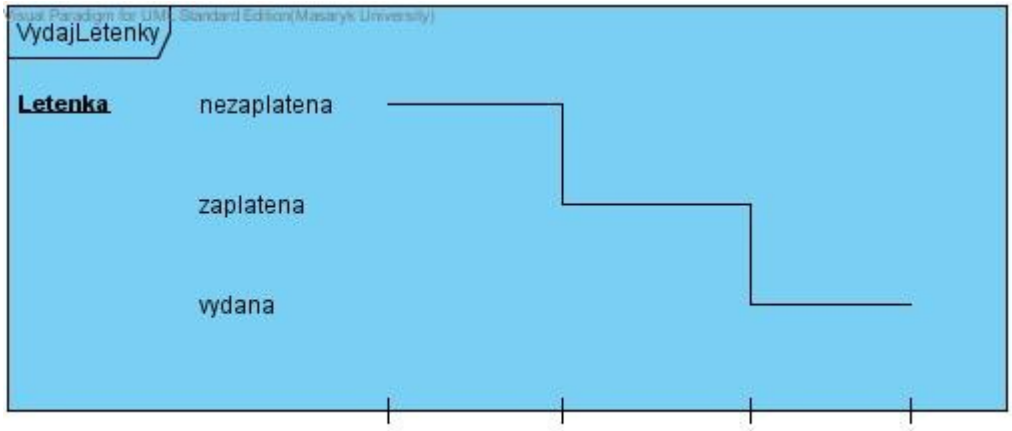
a overenie identifikátoru trvá 10 sekúnd. Potom začne kontrola batožiny. Kontrola batožiny trvá 20 sekúnd. A po kontrole batožiny sa prechádza do stavu *Hotová kontrola*.

Na uvedenom diagrame je zobrazený najčastejší prechod uvedenou kontrolou. Nie sú zobrazené alternatívy ako napr. keď sa systému nepodarí načítať identifikátor letenky, kontrola pasažiera nie je úspešná a podobne.

Uvedená situácia aj s alternatívnymi prechodmi je zobrazená pomocou aktivity diagramu v prílohe na obrázku č. 28.



Obrázok č.22: Diagram časovania zobrazujúci kontrolu pasažiera a jeho batožiny



Obrázok č.23: Diagram časovania akcie výdaj letenky s častými chybami

Na obrázku č. 23 je zakreslený diagram časovania akcie výdaj letenky s chybou pri tvorbe tohto diagramu. Prechody medzi stavmi uvedenými na obrázku nie sú závislé na čase, ale sú závislé na určitej podmienke. Zo stavu nezaplatená do stavu zaplatená sa letenka dostane po zadaní platby a zo stavu zaplatená do stavu vydaná sa letenka dostane potom, ako je pasažierovi vydaná na prepážke na

letisku. Uvedené prechody medzi stavmi na obrázku by mali byť zakreslené pomocou stavového diagramu, kde prechody medzi stavmi nastávajú po splnení nejakej podmienky. Diagram časovania zaznamenáva len také prechody medzi stavmi, ktoré sa menia v čase.

5 Záver

Táto bakalárska práca mala za cieľ prezentovať základné koncepty tvorby diagramov UML ako aj poukázať na najzásadnejšie chyby pri ich tvorbe. Keďže sa jedná celkovo o 14 diagramov, na uvedenej téme sme pracovali dvaja, aby sme si tak mohli diagramy rozdeliť a venovať viacej pozornosti jednotlivým vybraným diagramom. Ja som sa vo svojej práci rozpracovala nasledujúce diagramy: diagram prípadov užitia, textová špecifikácia prípadov užitia, diagram aktivít, sekvenčný diagram, komunikačný diagram, diagram prehľadu interakcií a diagram časovania. Ku každému diagramu som uviedla niekoľko modelov tak, aby som v každom modele poukázala na nejaký iný prvok tvorby diagramu a zároveň som uviedla aj model s najčastejšie sa vyskytujúcimi chybami pri tvorbe daného diagramu. Niektoré modely sú uvedené v texte práce spolu s popisom konceptov. Ostatné modely sú uvedené v prílohe. Všetky diagramy som prezentovala na ilustračnom systéme, ktorého textovú špecifikáciu sme si sami navrhli a jej znenie je uvedené v texte práce. Našou úlohou nebolo spraviť kompletnú a konzistentnú analýzu navrhovaného systému, ale prezentovať tvorbu diagramov UML.

Výsledky našej práce môžu byť použité ako pomôcka pri výuke predmetov zameraných na výučbu nástroja UML.

6 Literatúra

[1] Jim Arlow, Ila Neustadt: *UML a unifikovaný proces vývoje aplikácí.* Pearson Education Limited, 2002, 387 s., ISBN 80-7226-947-X.

[2] Joseph Schmuller: *Myslíme v jazyku UML*. Grada Publishing, 2001, 359 s., ISBN 80-247-0029-8.

[3] Martin Fowler: *UML Distilled*. Pearson Education Limited, 2007, 175 s., ISBN 0-321-19368-7.

[4] Perdita Stevens with Rob Pooley: *Using UML*. Pearson Education Limited, 2000, 256 s., ISBN 0-201-64860-1.

[5] Bernd Oestereich: *Developing software with UML*. Pearson Education Limited, 1999, 320 s., ISBN 0-201-36826-5.

[6] *Wikipedia The Free Encyclopedia*, URL <http://www.wikipedia.org>,[citované 2009-04-26].

[7] Arlow, Jim - Neustadt, Ila:*UML 2.0 and the unified process :practical object-oriented analysis and design. 2nd ed*. Boston : Addison-Wesley, 2005, 592 s., ISBN 0321321278.

[8] Jaroslav Ráček, Radek Ošlejšek: *Studijní materiály předmětu PB007.* Fakulta informatiky Masarykovej univerzity v Brně, 2008.

[9] Radek Ošlejšek: *Studijní materiály předmětu PA103.* Fakulta informatiky Masarykovej univerzity v Brne, 2009.

[10] Stanislav Hrabí: *Bakalárska práca*. Fakulta informatiky Masarykovej univerzity v Brne, 2007.

[11] Karla Petrlíková: *Diplomová práca*. Fakulta informatiky Masarykovej univerzity v Brne, 2005.

[12] *Objekty - Přehled OO notací a metodik - Diagramy UML*,<http://objekty.vse.cz/Objekty/MetodikyANotace-UMLDiagramy>, [citované 2009-05-02].

[13] *UML 2 a unifikovaný proces vývoje aplikací :objektově orientovaná analýza a návrh prakticky.* Edited by Jim Arlow - Ila Neustadt. Brno : Computer Press, 2007, 567 s., ISBN 978-80-251-1503.

[14] *UML: co je UML, historie UML,* URL <http://mpavus.wz.cz/uml/uml-uvod-1.php>, [citované 2009-04-04].

7 Prílohy

Main	
Use Case ID	SpravaLetov
Brief Description	UC umoznuje akterovi viest spravu letov
Primary Actors	ManagerLetiska, ManagerLeteckejSpolocnosti
Secondary Actors	SvetelnaTabula
Preconditions	Akter je prihlaseny do systemu.
Main Flow of Events	1. Pripad uzitia zacina, ked akter vyberie v hlavnom menu polozku "sprava letov". 2. System umozni akterovi vybrat z poloziek "zadat novy let", "editovat let", "zrusit let". 3. Ak akter vyberie volbu "zadat novy let". 3.1 System ponukne akterovi formular s udajmi, ktore je potreba vyplnit. 3.2 Akter vyplni potrebne udaje a zmeny potvrdi. 3.3 System ulozi novy let do databazy letov. EXTENSION POINT(ZmenaUdajovLetu). 4. Ak akter zvoli volbu "editovat let" 4.1 System poziada aktera o zadanie jednoznacneho identifikatoru letu. 4.2 Akter zada identifikator letu. 4.3 System let vyhlada v databaze letov a zobrazi vsetky dostupne informacie o lete. 4.4 Akter zmeni udaje a zmeny potvrdi. 4.5 System ulozi zmeny o lete do databazy letov. EXTENSION POINT(ZmenaUdajovLetu). 5. Ak akter zvoli polozku "zrusit let". 5.1 System poziada aktera o zadanie jednoznacneho identifikatoru letu. 5.2 Akter zada identifikator letu. 5.3 System zrusi vyhladany let z aktualnej databaze letov. EXTENSION POINT(ZmenaUdajovLetu).
Alternative Flows	Akter moze kedykolvek prejst do hlavneho menu. Systemu sa nepodarilo ulozit novy let do databazy letov. Systemu sa nepodarilo editovat udaje o lete. Systemu sa nepodarilo zrusit let z databaze letov.
Post-conditions	System aktualizoval databazu letov.

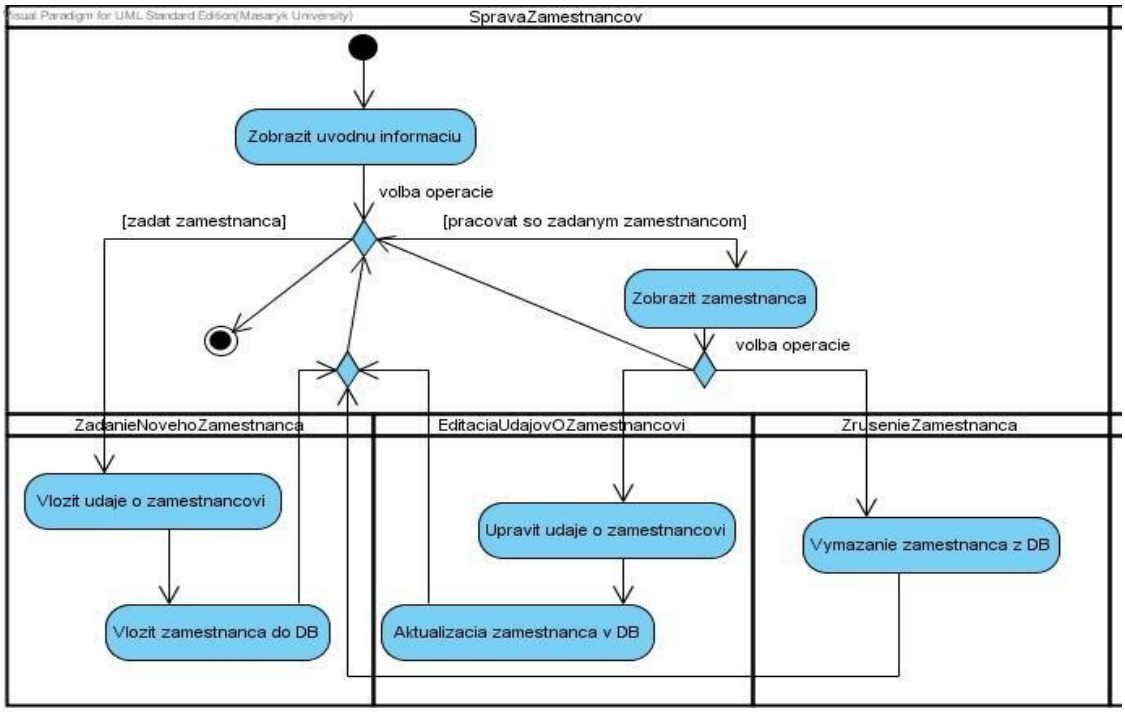
Obrázok č.24: Textová špecifikácia prípadu užitia SpravaLetov

Main	
Use Case ID	ZaslanieUpozornenia
Brief Description	UC zasle manazerovi letiska a manazerovi leteckej spolocnosti informaciu o zmene udajov letu.
Primary Actors	ManazerLetiska. ManazerLeteckejSpolocnosti
Secondary Actors	-
Preconditions	1. Pripad uzitia bol vyvovalany kvoli bodu rozsirenia ZmenaUdajovLetu v pripade uzitia SpravaLetov. 2. Manazer letiska alebo manazer leteckej spolocnosti zmenili nejake udaje o lete.
Main Flow of Events	1. System zasle manazerovi letiska a manazerovi leteckej spolocnosti email o zmene informacii o lete.
Alternative Flows	-
Post-conditions	-

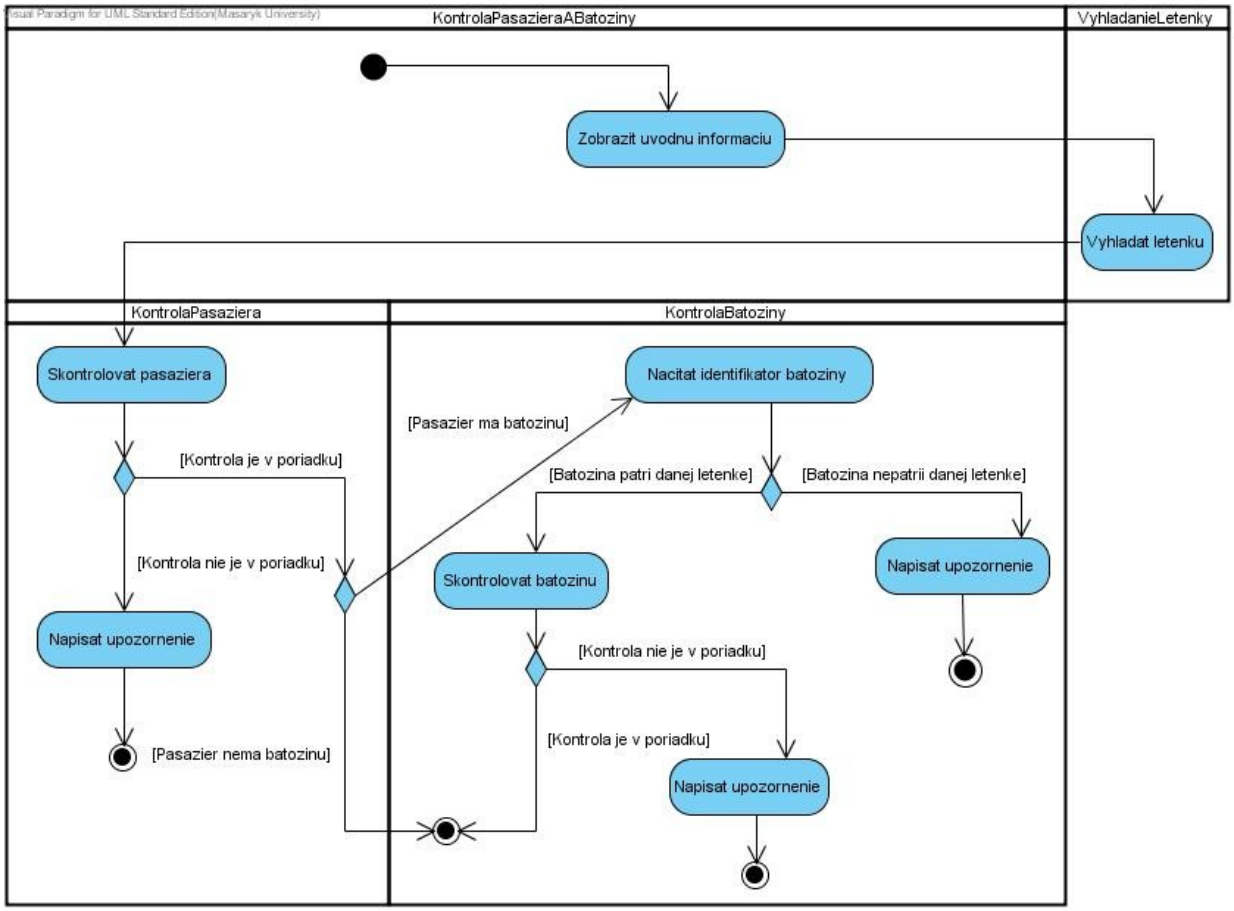
Obrázok č.25:Textová špecifikácia prípadu užitia ZaslanieUpozornenia

Main	
Use Case ID	TvorbaHarmonogramuLetov
Brief Description	UC TvorbaHarmonogramuLetov umoznuje ManazeroviLetiska vytvorit harmonogram letov
Primary Actors	ManazerLetiska
Secondary Actors	-
Preconditions	1. ManazerLetiska je prihlaseny do systemu 2. ManazerLetiska ma pristup k informáciam o letoch
Main Flow of Events	1. Pripad uzita zacina, ked ManazerLetiska zvoli v hlavnom menu polozku "tvorba harmonogramu letov". 2. System zobrazi ManazeroviLetiska aktualny zoznam letov zadanej bez odletovej alebo priletovej drahy a zobrazi aj aktualne obsadenie odletovych a priletovych drah. 3. Ak sa jedna o let odchadzajuci 3.1 ManazerLetiska prideli letu volnu odletovu drahu na cas. ktory zadal do udajov ManazerLeteckejSpolocnosti a cislo drahy zapise do udajov o lete. 3.2 INCLUDE(SpravaLetov) 4. Ak sa jedna o let prichadzajuci. 4.1 ManazerLetiska prideli letu volnu priletovu drahu na cas, ktory zadal do udajov o lete ManazerLeteckejSpolocnosti a cislo drahy zapise do udajov o lete.. 4.2 INCLUDE (SpravaLetov)
Alternative Flows	1. ManazerLetiska sa moze kedykolvek vratit do hlavneho menu. 2. Ak nie je volna draha na dany cas ManazerLetiska prideli najblizsiu volnu drahu a zmeni udaje o case odchodu alebo prichodu a zapise cislo drahy. INCLUDE(SpravaLetov).
Post-conditions	Pripad uzita vytvori harmonogram obsadenia priletovych a odletovych drah.

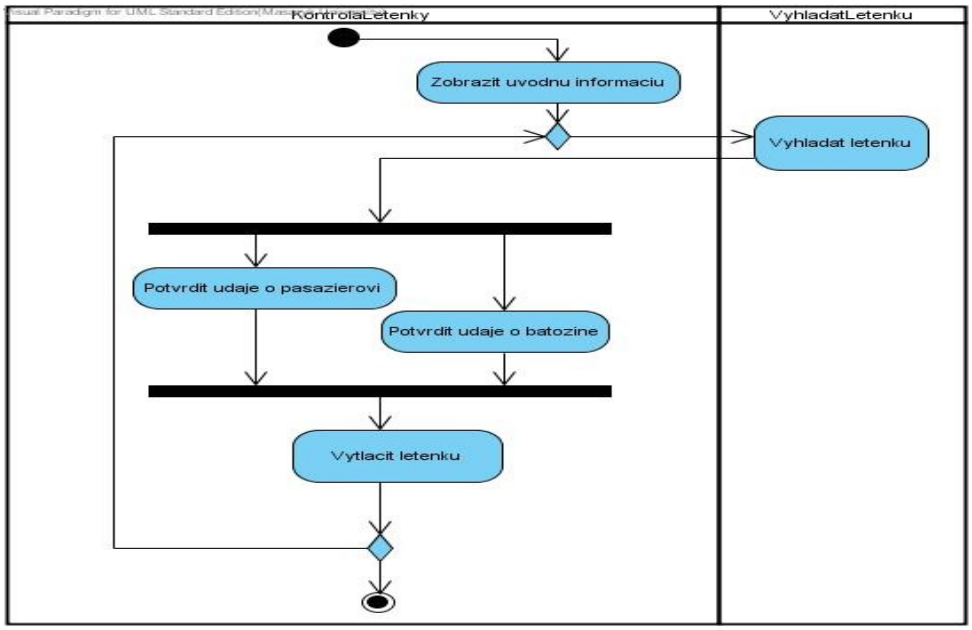
Obrázok č.26: Textová špecifikácia prípadu užitia TvorbaHarmonogramu



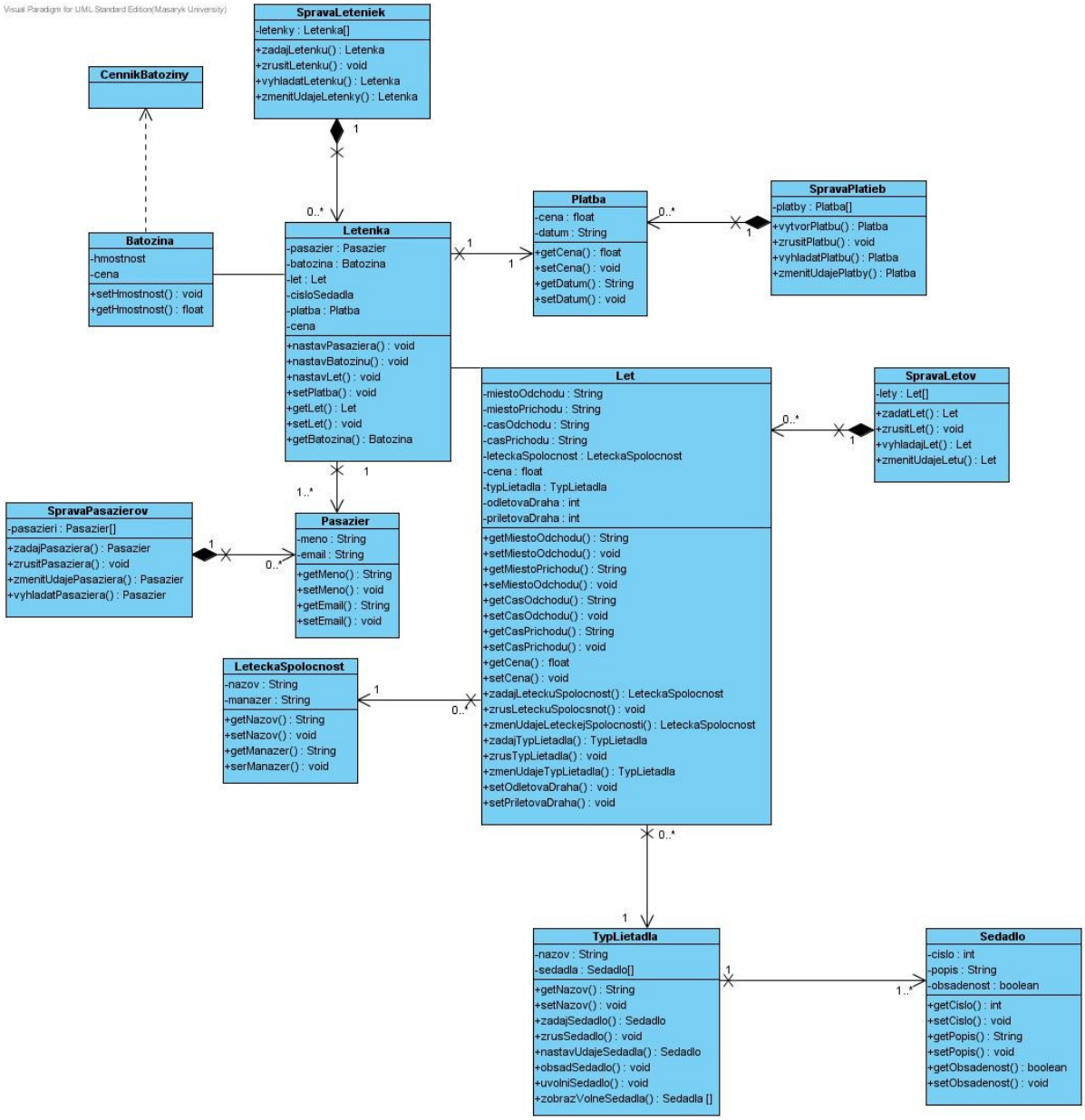
Obrázok č.27:Diagram aktivít prípadu užitia SpravaZamestnancov



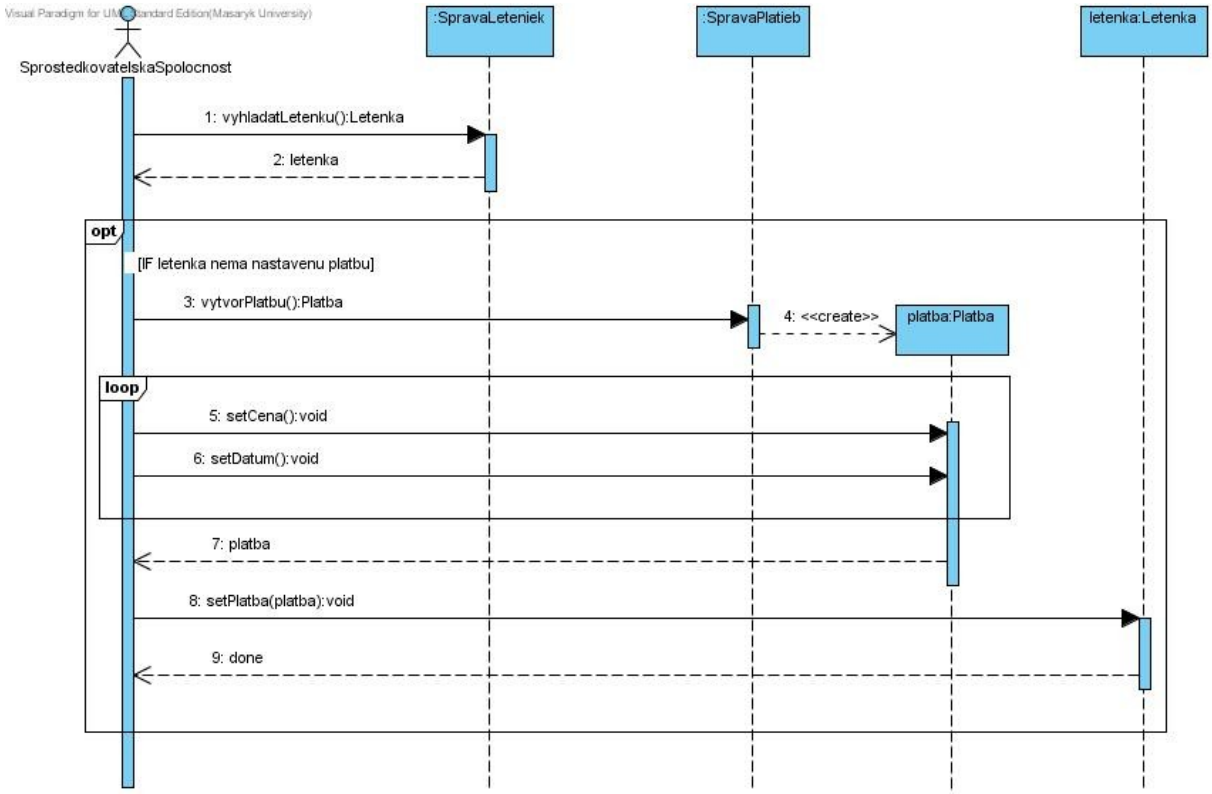
Obrázok č.28: Diagram aktivít prípadu užitia KontrolaBatozinyAPasaziera



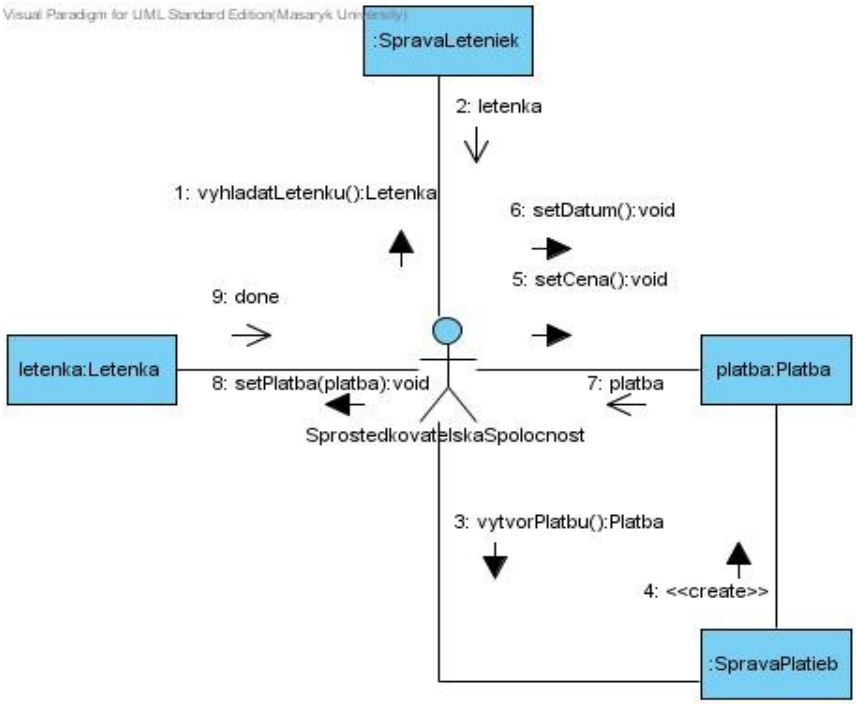
Obrázok č.29: Diagram aktivít prípadu užitia KontrolaLetenky



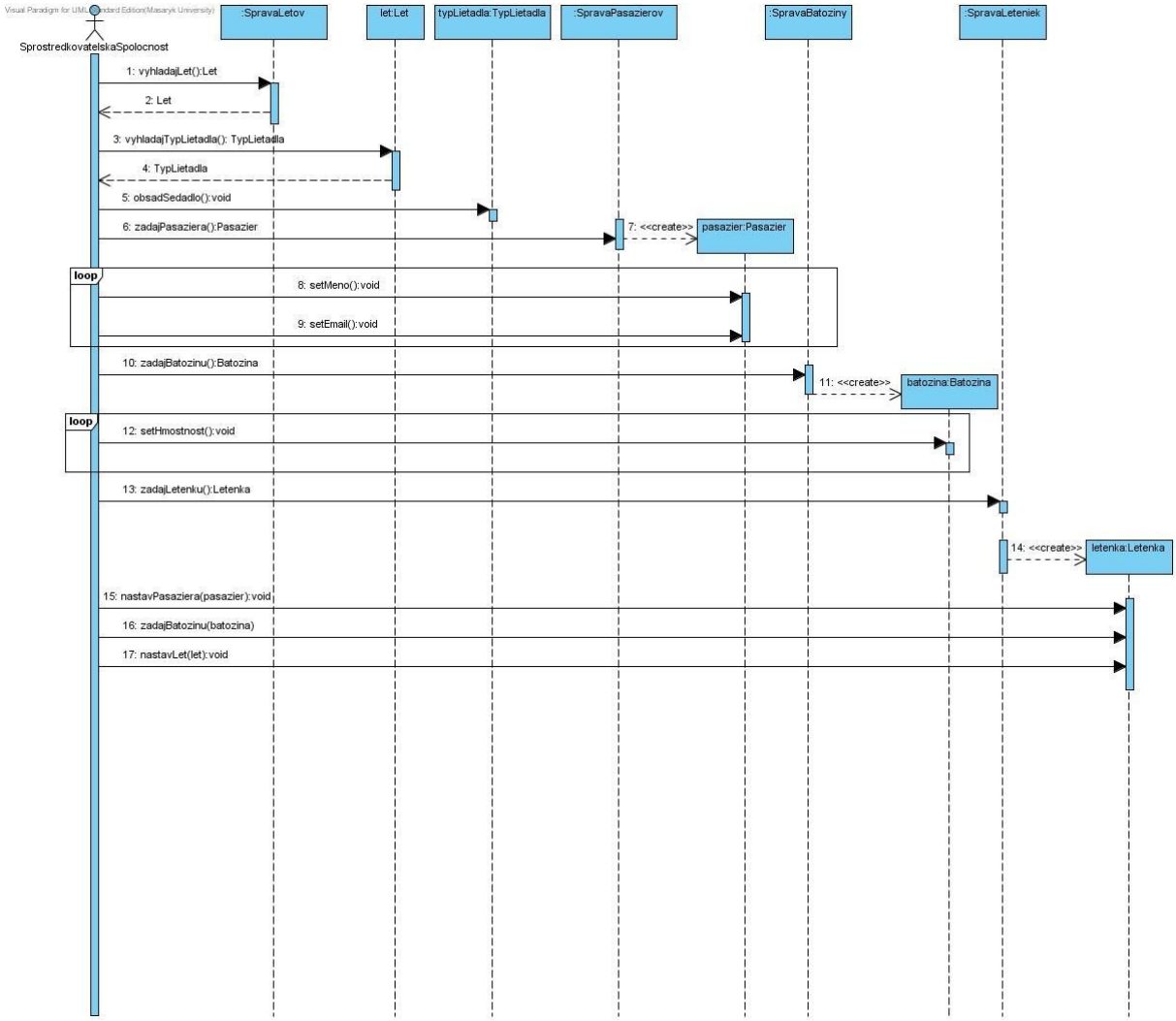
Obrázok č.30: Diagram tried prípadu užitia SpravaLeteniek



Obrázok č. 31: Sekvenčný diagram interakcie PlatbaLetenky



Obrázok č. 32: Komunikačný diagram interakcia PlatbaLetenky



Obrázok č.33: Sekvenčný diagram interakcie ZadatLetenku