VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

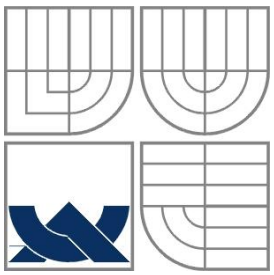
NÁVRH A ANALÝZA SYSTÉMU PRO SPRÁVU AUTOBUSOVÉ DOPRAVY

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

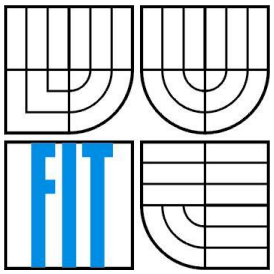
AUTOR PRÁCE
AUTHOR

MARTIN LEBEDA

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁVRH A ANALÝZA SYSTÉMU PRO SPRÁVU AUTOBUSOVÉ DOPRAVY

DESIGN AND ANALYSIS OF SYSTEM FOR BUS TRANSPORT MANAGEMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — MARTIN LEBEDA
AUTHOR

VEDOUCÍ PRÁCE — ING. JIŘÍ KRAJÍČEK
SUPERVISOR

BRNO 2008

Abstrakt

Cílem projektu bylo vytvořit Informační systém autobusové dopravy. Při detailnějším zamyšlení bylo zřejmé, že tento systém bude podporovat i tramvajovou, trolejbusovou a vlakovou dopravu. Podstatná část projektu je zaměřena na softwarové inženýrství, objektově orientovaný návrh aplikace.
Implementace je provedena v jazyce PHP za podpory databáze MySQL. Pro vyhledávání spojů bude použito metod umělé inteligence.

Klíčová slova

Informační systém, databáze, UML, jízdní řád, PHP, MySQL, CSS, XHTML, softwarová architektura, návrhové vzory, uživatelská práva, rozšiřitelnost, analýza, návrh, umělá inteligence, prohledávání stavového prostoru.

Abstract

The aim of this project was to create an Information system of bus transport. After further analysis it was certain, that this system will support tramway, trolley-bus and train transport as well. Essential part of the project is focused on software engineering, object-oriented application design. System is implemented in PHP with the support of MySQL database. Route searching engine will be using method of artificial intelligence.

Keywords

Information system, database, UML, time table, PHP, MySQL, CSS, XHTML, software architecture, design patterns, user rights, extensibility, analysis, design, Artificial intelligence, searching in state space.

Citace

Lebeda Martin: Návrh a analýza systému pro správu autobusové dopravy. Brno, 2008, bakalářská práce, FIT VUT v Brně.

Návrh a analýza systému pro správu autobusové dopravy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jiřího Krajíčka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Martin Lebeda
11. května 2008

Poděkování

Velice děkuji panu Ing. Jiřímu Krajíčkovi za odborné vedení a mnoho důležitých rad při tvorbě této práce.

Obsah

1 Úvod

Postupem času máme čím dál tím vyšší nároky na efektivitu, kvalitu a zároveň vyžadujeme co nejmenší náklady ve spojení s vykonanou prací. Z tohoto důvodu vytváříme s pomocí počítačové techniky a vývoje programovacích jazyků informační systémy. Snažíme se jimi nahrazovat lidský faktor (což je asi nejdražší element vcházející do tohoto procesu), spotřebovaný materiál apod.

Nároky na informační systémy jsou velmi vysoké. Uvedené systémy musí být uživatelsky co nejjednodušší, ale současně musí zvládat co nejvíce funkcí. Vyhovět těmto požadavkům není jednoduché. Dalším parametrem je rozšiřitelnost systému do budoucna, poněvadž si musíme uvědomit, že organizace, firmy a další instituce se postupem času rozvíjejí a mohou dokonce měnit i své zájmy.

Za těchto okolností stále více nabývají na významu objektově orientované jazyky. Jejich výhody jsou zřejmé, pomocí abstrakce si je dokážeme lépe představit. Objekt lze popsat třídou, která má své atributy a metody. Pro návrh objektově orientovaných systémů se používá UML (Unified Modeling Language), což je jazyk pro univerzální vizuální modelování systémů. Počátky UML se datují k roku 1997, kdy ještě neexistoval optimální modelovací jazyk pro objektově orientované metody. Výhodou jazyka UML je mimo jiné srozumitelnost i pro relativně nezkušeného uživatele. UML není závislé na programovacím jazyku; pro ten se dá rozhodnout až na konci návrhu.

Cílem této práce je nastudování analýzy a návrhu informačního systému, mezi což patří požadavky a případná rozšíření Informačního systému autobusové dopravy pomocí UML. Následně implementace a tato technická zpráva, v které se budou postupně uvádět popsané části systému.

Druhá kapitola je zaměřena na popis analýzy a návrhu zvoleného IS, kde se taky upřesní co to jazyk UML je a jaké je jeho použití. Třetí kapitola je věnována umělé inteligenci, její historii a zvolenému algoritmu. Popis samotné implementace se nachází ve čtvrté kapitole a závěrečná pátá kapitola se zaobírá dalšími možnostmi rozšíření navrženého systému.

2 Analýza a návrh

2.1 Požadavky

Původním cílem projektu bylo vytvořit informační systém autobusové dopravy. Během prací na projektu jsem však usoudil, že by bylo přínosné tento IS také na podporu spojů tramvajové, trolejbusové a vlakové dopravy. Značná část projektu byla zaměřena na softwarové inženýrství návrhu aplikace.

Aplikace má zobrazovat vývěsky různých spojů, vyhledávat nejrychlejší cestu z bodu A do bodu B s ohledem na zadanou dobu příjezdu, popřípadě spočítat cenu jízdného a zobrazit ceník jízdenek. Dále musí evidovat výluky a víkendové spoje.

Se systémem operují neregistrovaní uživatelé, kteří mohou z IS informace pouze číst, a zaměstnanci, kteří mohou měnit jízdní řády, výluky, jízdné a vozidla. Administrátor může měnit vše, co může měnit zaměstnanec, jen s tím rozdílem, že může vytvořit další administrátory a zaměstnance, popřípadě je editovat. Všichni mohou využívat informačních služeb systému. Také si mohou měnit přednastavené vzhledy rozhraní.

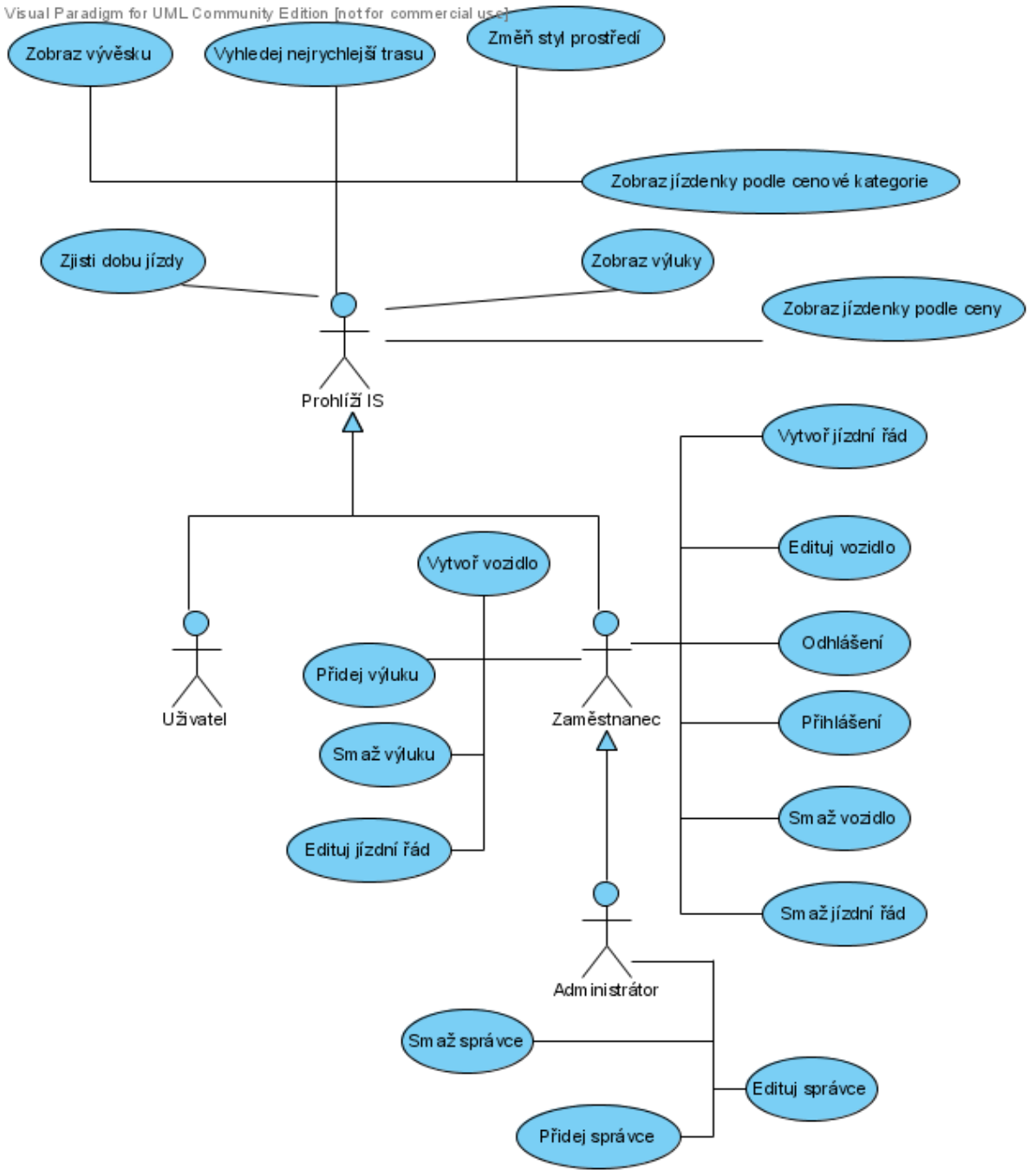
2.2 Analýza

Pro analýzu a návrh byl použit standardizovaný jazyk UML (viz kapitola 2.4) ve verzi 2.0, jejíž vydání se datuje na rok 2005. Jako první byl vytvořen diagram případu užití, ve kterém musely být znázorněny všechny interakce IS s uživatelem, administrátorem a zaměstnancem.

Z diagramu případů užití pak bylo možno přejít na diagram tříd, v kterém bylo jasně stanoveno, co má program obsahovat, popřípadě jaké závislosti budou mezi třídami.

2.3 Diagram případu užití

Diagram případů užití zobrazuje práva a přístupy různých typů uživatelů do systému viz obrázek 2.1.



Obrázek 2.1: Diagram případů užití

Diagram popisuje 3 druhy osob, které IS používají, což jsou uživatelé, zaměstnanci a administrátoři. Uživatelé si mohou zobrazit vývěsky, vyhledat nejrychlejší trasy, změnit styl prostředí a zobrazit jízdenky podle cenové kategorie. Zaměstnanci i administrátoři mohou využívat těchto služeb také, s tou výjimkou, že zaměstnanec může editovat, mazat a vytvářet jízdní řady, vozidla, jízdenky

a výluky. Absolutní kontrolu nad informačním systémem má administrátor, který může spravovat i zaměstnance (smaž, přidej, edituj správce). Více k obecné teorii diagramů případů užití se nachází v kapitole 2.8.

2.4 Návrh podle specifikace jazyka UML

Jazyk označovaný zkratkou UML neboli Unified Modeling Language je, jak již název napovídá, unifikovaný modelovací jazyk, který má, na rozdíl od převážně textově orientovaných programovacích jazyků, vlastní grafickou syntaxi (tj. pravidla pro sestavování jednotlivých elementů jazyka do větších objektů) a sémantiku (tj. jednoznačná pravidla určující jednotlivým syntaktickým výrazům jejich význam).

V současné době má jazyk UML největší význam při návrhu softwarových systémů, protože objektově orientovaný návrh každé složitější aplikace je nezbytným předpokladem pro její úspěšnou a rychlou implementaci. Pro objektově orientovaný návrh je samozřejmě možné použít různé podpůrné prostředky, zejména další odlišné typy diagramů. UML je však významné také v tom ohledu, že přesně specifikuje, co má daný diagram obsahovat, což je velmi důležité zejména při sdílení informací mezi jednotlivými analytiky a vývojáři. Dále je již z principu UML nutné, aby vytvářené grafy měly vnitřní konzistenci a přesně danou sémantiku, což u jiných typů grafů nemusí být obecně zaručeno. UML diagramů existuje několik typů lišících podle toho, jaké se pomocí nich plánují či zpracovávají úlohy. Tyto diagramy se od sebe odlišují především repertoárem použitých značek, způsobem jejich vzájemného propojení a s nimi související sémantikou.

Mezi velké přednosti jazyka UML i na něm postavených UML diagramů patří existence otevřeného a rozšiřitelného standardu, podpora celého vývojového cyklu aplikace či jiného (ne nutně programového) systému a velká podpora pro různé aplikační oblasti. Pro širší využití jazyka UML v praxi mluví také významný fakt, že je podporován celou řadou vývojových nástrojů, ať už samostatných aplikací určených pouze pro práci s UML, nebo i integrovaných vývojových prostředí (IDE), které v některých případech dovolují provádět převod informací mezi UML diagramem a algoritmem zapsaným v programovacím jazyce (a samozřejmě i opačným směrem, ten je však z pochopitelných důvodů složitější a ne vždy uskutečnitelný).

2.5 Historie UML

V roce 1996 zadala firma Rational Corp. tříčlennému týmu – tvořili jej Jacobson, Booch a Rumbaugh – úkol vytvořit jazyk UML. V roce 1997, byl už jazyk UML akceptován jako průmyslový standard a postupně začal vytlačovat ostatní analytické jazyky na pomyslné smetiště dějin. Verze 2.0 vyšla roku 2005.

2.6 Návrhové vzory

Návrhové vzory se užívají v SW inženýrství jako recyklovatelné řešení obvyklých problémů. Tím se dosahuje větší efektivity práce a dá se dopředu předpokládat, popřípadě se vyvarovat určitým komplikacím, které mohou nastat.

Dle GAM95 jsou návrhové vzory řešení, které se stále rozvíjí. Vymezení návrhových vzorů jako popisu řešení je velmi strohé. Jedná se spíše o jazyk, respektive o způsob komunikace. Jejich rozvoj je spojen s potřebou zachytit možnosti řešení netriviální problémů, které se opakovaně objevují při návrzích informačních systémů především v oblasti designu. Tyto dvě podmínky, tj. netriviální a opakující se situace, představují nutné podmínky pro vznik návrhových vzorů.

Návrhové vzory se dělí mezi 3 skupiny (viz níže a [1]).

2.6.1 Vzory tvorby (Creational patterns)

Pomáhají řešit problémy s vytvářením objektů v systému. Popisuje jak správně vybrat třídy objektů do návrhu.

Mezi tyto vzory patří:

a) **Abstract faktory Pattern** – řeší problém, jak vytvořit na základě rozhodnutí v běhu programu instanci třídy, která dále vytváří instanci souvisejících nebo závislých tříd.
b) **Builder Pattern** – řeší problém, jak oddělit vytváření složitých objektů od jejich prezentace, aby stejný proces konstrukce mohl mít za výsledek rozdílný způsob prezentace.
c) **Prototype Pattern** – řeší problém, jak vytvořit kopii existujícího objektu místo vytváření nové třídy.
d) **Singleton Pattern** – řeší problém jak zajistit existenci pouze jedné instance dané třídy a poskytnutí globálního přístupu k ní.

Zde byl použit Abstract faktory Pattern.

2.6.2 Strukturové vzory (Structural patterns)

Upravují stavbu tříd v návrhu systému. Mají na starost úpravu vztahů entit mezi sebou.

a) **Adapter Pattern** – již název tohoto návrhového vzoru napovídá, že se jedná o přizpůsobení určité třídy, aby ji bylo možné využívat i jiným požadovaným způsobem. Problémem je zajištění konverze rozhraní jedné třídy na rozhraní jiné třídy.
b) **Bridge Pattern** – představuje problém oddělení rozhraní třídy od její vlastní implementace, aby obě tyto části mohly být vytvářeny nezávisle na sobě. Tento princip zajistí, že může být změněna implementace třídy, aniž by byl změněn kód klienta.

c) **Composite Pattern –** představuje řešení, jak uspořádat jednoduché objekty a z nich složené (kompozitní) objekty. Snahou vzoru je, aby k oběma typům objektů (jednoduchým a složeným) bylo možné přistoupit jednotným způsobem.
d) **Facade Pattern** – představuje řešení, jestliže je nutné zjednodušit vstupní bod do systému
e) **Flyweight Pattern** – řeší problém, jakým způsobem zajistit efektivní správu velkého množství objektů, které se příliš neodlišují. Tento návrhový vzor se snaží využít sdílení stejných vlastností těchto objektů.
f) **Proxy Pattern** – využijeme tehdy, jestliže potřebujeme zajistit kontrolu nad přístupem k jinému objektu.

Zde byly použity Adapter Pattern a Bridge Pattern.

2.6.3 Vzory chování (Behavioral patterns)

Starají se o rozeznání běžných komunikačních vzorů mezi objekty a o jejich realizaci. Mohou být založeny na třídách nebo objektech. U tříd využívají při návrhu řešení především principu dědičnosti.

a) **Chain of responsibility Pattern** – řeší problém, jak zaslat požadavek bez přesného vymezení objektu, který jej zpracuje.
b) **Command Pattern –** odstíní klienta od procesu zpracování jeho požadavku. Klient pouze definuje požadavek a určí zpracovatele, ale už se nezajímá o způsob a čas vykonání jeho požadavku.
c) **Interpreter Pattern –** vytvoří jazyk, tzn., definuje gramatická pravidla a určí způsob, jak vzniklý jazyk interpretovat.
d) **Iterator Pattern** – řeší problém, jak se pohybovat mezi prvky, které jsou sekvenčně uspořádány, bez znalosti implementace jednotlivých prvků posloupnosti.
e) **Mediator Pattern** – určuje, jakým způsobem zajistit komunikaci mezi dvěma komponentami programu, aniž by byly v přímé interakci a tím musely přesně znát poskytované metody.
f) **State Pattern** – řeší problém jak změnit chování objektu, jestliže se změní jeho vnitřní stav. Po změně chování se objekt jeví jako instance jiné třídy.
g) **Styrategy Pattern –** určuje rodinu algoritmů, zapouzdřuje každý z nich do samostatného objektu a umožňuje jejich záměnu. Řeší problém změny algoritmu nezávisle na klientovi, který jej využívá.

Z těchto vzorů byly použity Chain of responsibility Pattern, Command Pattern a State Pattern.

2.7 Základní elementy jazyka UML

Základní stavební prvky jazyka UML jsou reprezentovány grafickými značkami v dvourozměrném grafu (viz [3] a [4]). Jsou to předměty, relace a diagramy.

2.7.1 Předměty

Mezi předměty se řadí chování, které zachycuje chování mezi objekty. Další je seskupení, které seskupuje části diagramu na nižší úrovni ve formě balíčků, a strukturní abstrakce v podobě programových tříd, případy užití, objektová rozhraní, komponenty anebo uzly. Posledním typem jsou poznámky, které blíže určují vlastnosti a chovaní elementů UML diagramu.

2.7.2 Relace

Relace je patří mezi základní elementy jazyka UML, určuje vztahy mezi diagramy. Existují 4 druhy relací:

a) **Asociace** – jejich pomocí se modeluje obecná souvislost předmětů, která je však v diagramu UML přesným způsobem definovaná. Speciální variantou asociace jsou takzvané kompozice a agregace, které jsou často používány v objektově orientovaných jazycích a návrzích databází.
b) **Závislost** – použije se, pokud změna v jednom předmětu způsobí změnu v předmětu jiném, nebo mu známým způsobem poskytne požadovanou informaci.
c) **Generalizace** – její pomocí se modeluje stav, v němž je jeden předmět specializací jiného předmětu. Tato relace je velmi často používána v objektově orientovaných jazycích, implementuje se většinou pomocí dědičnosti (inheritance).
d) **Realizace** – jedná se o druh vztahu, ve kterém jeden předmět představuje dohodu, za jejíž splnění je odpovědný jiný předmět. V objektově orientovaných jazycích se realizace vytváří pomocí rozhraní (interface) – samozřejmě za předpokladu, že daný OOP jazyk rozhraní podporuje.

2.7.3 Diagramy

Diagramy prezentují různé pohledy na systém, každý jej modeluje z jiného aspektu. UML používá 12 druhů diagramů. V rámci této práce budou uvedeny jen některé z nich (viz [5]).

2.8 UML diagramy navrženého systému

V této kapitole se zaměříme na popis podstatných UML diagramů, které hrají klíčovou úlohu při návrhu zde popisovaného informačního systému.

2.8.1 Diagram případu použití

Popisuje chování systému z pohledu uživatele. Zachycuje, požadavky uživatele na systém. Myšlenkou je zapojit uživatele do počátečních fází analýzy a návrhu systému. Zvyšuje to pravděpodobnost, že systém nakonec bude skutečně prospěšný pro ty lidi, pro které prospěšný být

má. Pro označení uživatele se používá název aktér, činnost systému jako odezva aktérovi (může to být osoba, jiný systém, hardware, plynutí času) se nazývá případ užití. UML používá ve svých diagramech pro znázornění slovně definovaných případů užití (vzniklých rozhovorem s uživateli) následující notaci. Aktér je vyjádřen symbolem osoby a případ užití (use case) symbolem elipsy. Aktér, který akci (resp. use case) spustil, se zakresluje vlevo, aktér, který obdrží výstup, se zakresluje vpravo. Případy užití a aktéři se propojují plnou čarou. Pro odlišení vnějšího světa a systému se systém ohraničuje obdélníkem.

Mezi jednotlivými případy užití mohou existovat vztahy vkládání (include) a rozšiřování (extend). Vztahy mezi nimi se znázorňují přerušovanou šipkou. Pro odlišení toho, zda šipka znázorňuje vkládání nebo rozšiřování, se používá stereotypů (např. « vkládá » a « rozšiřuje »). Vkládání umožňuje použít kroky definované jedním diagramem i v druhém diagramu.

I v oblasti diagramů případů užití může existovat dědičnost, viz [6]. Označuje se to jako zobecňování. Potomek dědí chování a význam svého předka a doplňuje své vlastní chování. Potomka je pak samozřejmě možné použít všude tam, kde lze použít jeho předka. Zobecnění se zakresluje šipkou směřující od potomka k předkovi.

Pro přehlednost nebo z důvodu vyznačení podsystémů se mohou jednotlivé případy užití seskupovat. Pro vyjádření skupiny se využívá balíček.

Diagram případů použití pro navrhovaný systém byl zmíněn již v rámci kapitoly analýza požadavků 2.3.

2.8.2 Diagram tříd

Diagram tříd UML vznikl spojením metod OMT, Booche a diagramu tříd většiny dalších metod.

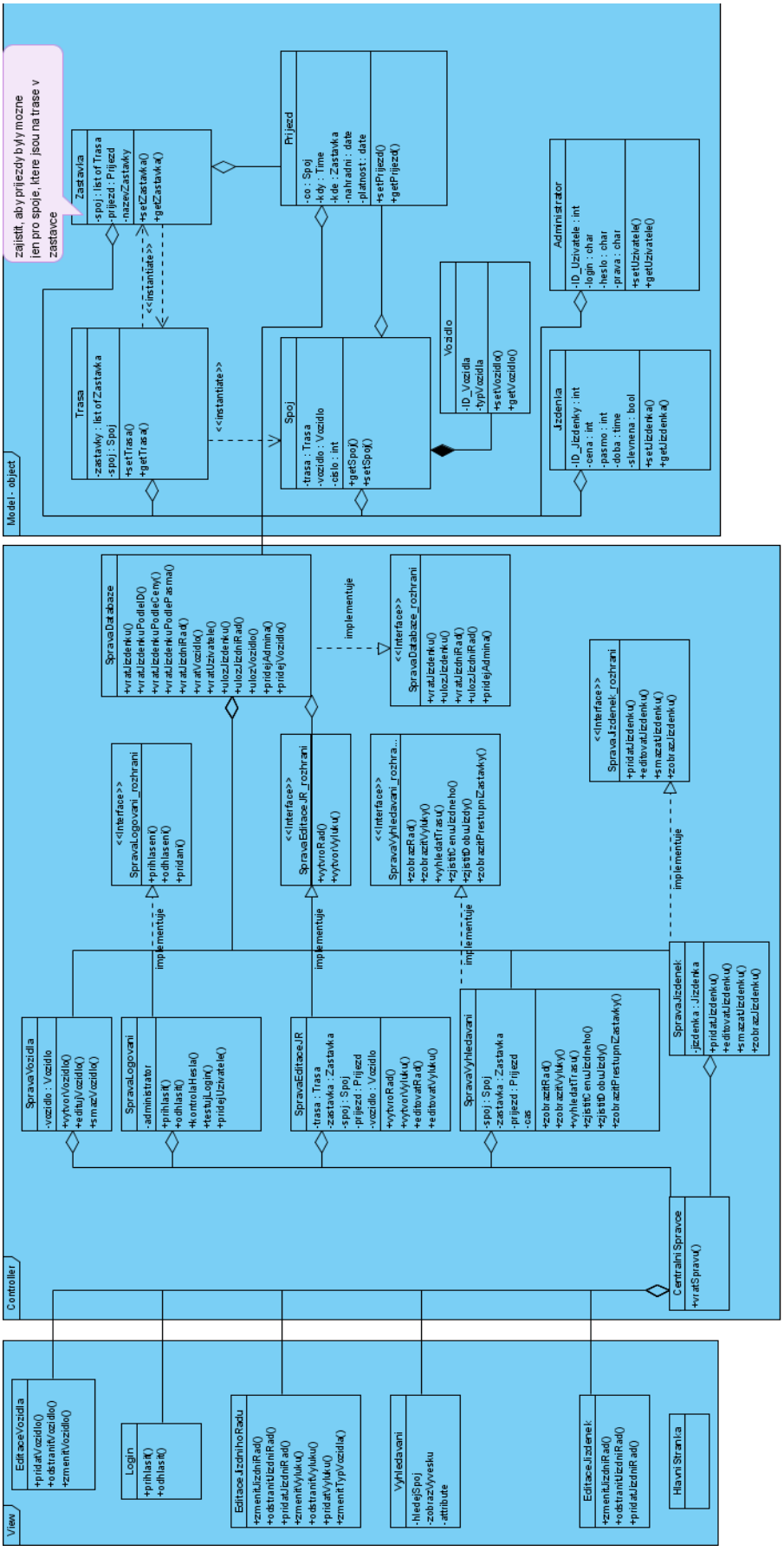
Diagram tříd zachycuje všechny třídy objektů (viz třída), které se týkají modelovaného systému (tj. celou oblast zájmu řešení) a vztahy mezi těmito třídami (viz asociace).

Tvorba diagramů tříd patří mezi klíčové problémy, dokonce lze prohlásit, že zvládnout objektový přístup často znamená správně využívat právě tento typ diagramů používaný průběžně napříč celým životním cyklem tvorby IS.

Značně ovlivněna svými přímými předchůdci je notace diagramu tříd v UML vzdáleně podobná klasickým ER diagramům obohaceným o některé objektové rysy. Atributy tříd jsou zobrazeny ve střední části prvku třídy, operace pak v části spodní.

Bohužel značná orientace na jedno konkrétní prostředí je jednou z méně šťastných vlastností UML. Na druhou stranu je nutné si opět uvědomit, že tyto prostředky spadají až do pozdního designu a dříve než v designu o nich nemá smysl uvažovat.

Diagram tříd navrhovaného systému s ohledem na použitou softwarovou architekturu (viz 2.7) je vidět na obr. 2.2.



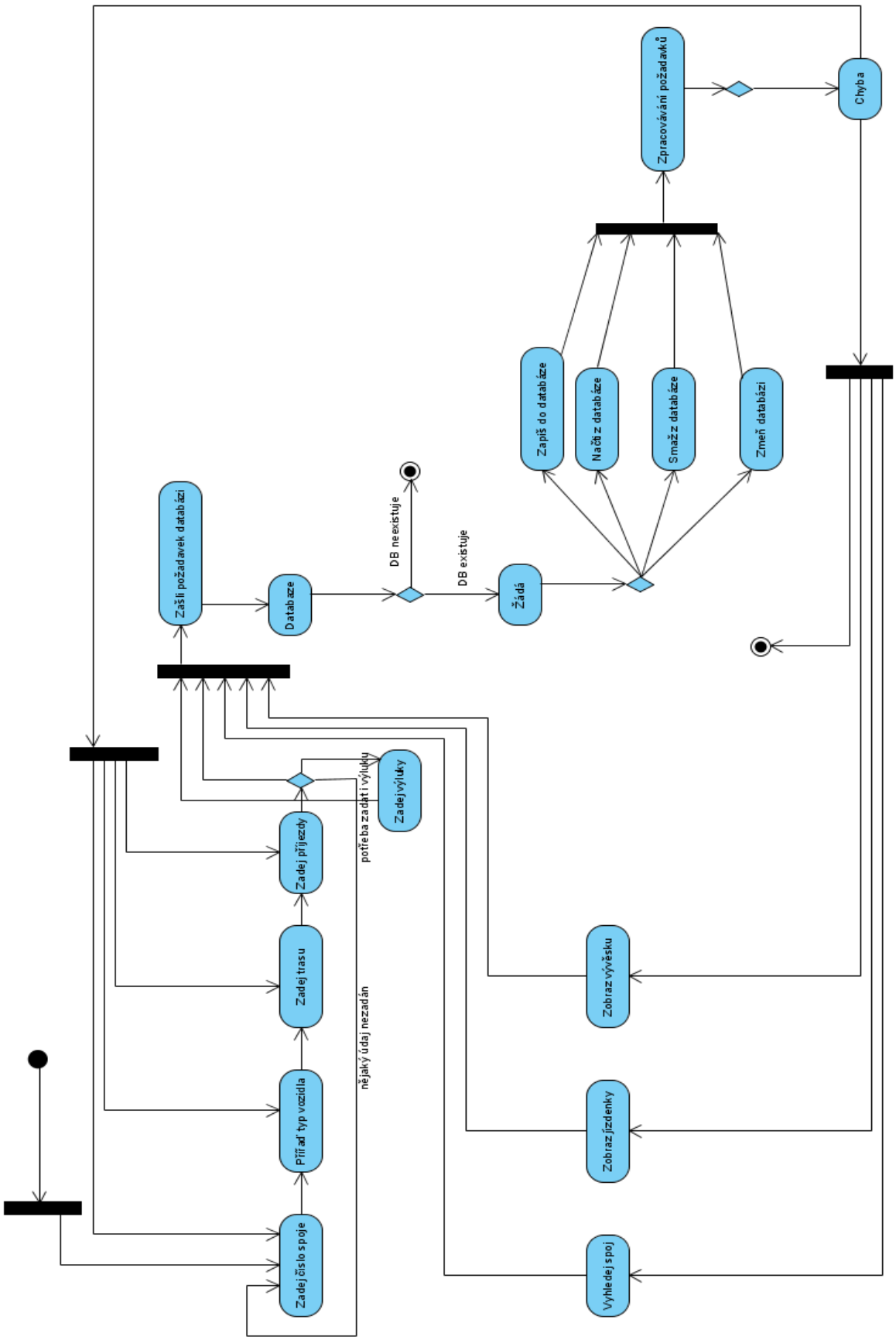
Obrázek 2.2: Diagram tříd (MVC: Model-Zobrazení-Řadič)

2.8.3 Diagram aktivit

Jako jistou obdobu stavových diagramů můžeme chápat i diagramy aktivit, kde jednotlivými stavy rozumíme aktivity a přechody mezi aktivitami je vyvolán dokončením aktivity stávající. Diagram aktivit se zpravidla vztahuje k jednomu případu užití, případně k jedné metodě objektu. Pomocí diagramu aktivit modelujeme tentokrát dynamický tok řízený nikoliv vnějšími událostmi ale interními podněty.

Poměrně elegantním způsobem je možné přiřazovat k jednotlivým aktivitám osoby (aktory), které jsou zodpovědné za provedení patřičných aktivit. Stejně dobře jako procesní modelování je možné používat diagramy aktivit i pro základní schematickou tvorbu grafického uživatelského rozhraní.

Diagram aktivit navrhovaného systému je vidět na obr. 2.3.



Obrázek 2.3: Diagram aktivit pro uložení jízdního řádu a aktivit nad vyhledáváním spojů, zobrazením jízdenky a vývěsky

2.8.4 Sekvenční diagram

Zachycuje časovou sekvenci interakce mezi objekty tříd, ke které dochází při komunikaci (předávání činnosti) v systému.

Objekty jsou umístěny v horní části diagramu vedle sebe zleva doprava. Směrem dolů od každého objektu směřuje přerušovaná čára, která se nazývá čáry života (viz [6]). Vykonání operace, kterou má objekt za úkol provádět se označuje jako aktivace a je znázorněna jako úzký obdélníček podél čáry života. Délka obdélníku naznačuje, jak dlouho aktivace trvá. Podél čáry života vertikálně dolů ubíhá čas.

Zprávy zasílané mezi objekty (přechody mezi aktivacemi objektů) se znázorňují šipkou. Zprávy dělíme na jednoduché, synchronní a asynchronní. Jednoduchá zpráva představuje přechod řízení z jednoho objektu na druhý. Synchronní zpráva je zpráva, kterou zasílá jeden objekt druhému, poté první objekt čeká na odpověď. Dokud ji nedostane, nebude nic dělat. Pošle-li objekt asynchronní zprávu, nečeká na odpověď, ale pokračuje ve své práci. Jednoduchá zpráva se znázorňuje obyčejnou šipkou, synchronní zpráva se znázorňuje vyplněnou šipkou a asynchronní jako šipka s polovinou hrotu.

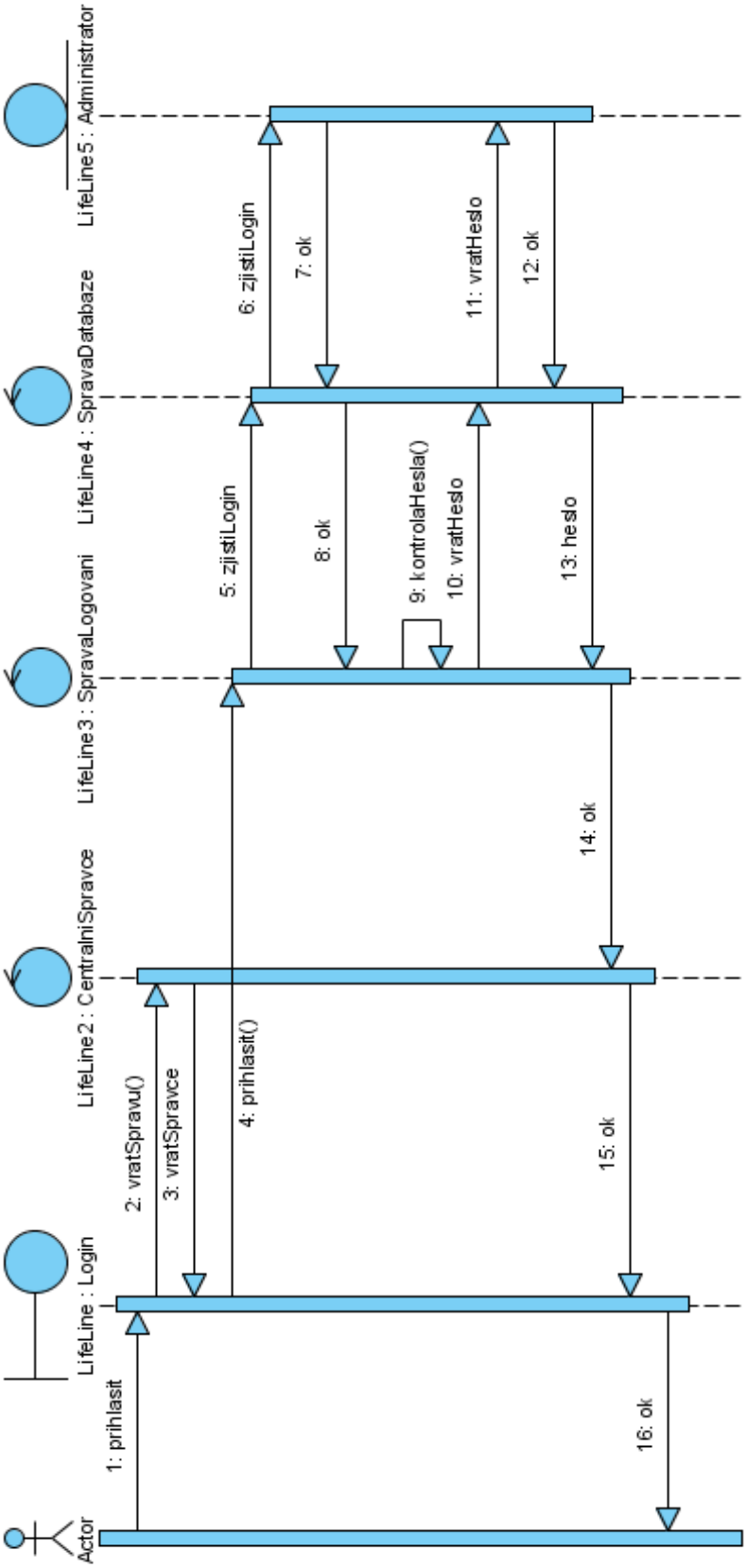
Sekvenční diagram v UML může být doplněn o účastníka, který sekvenci z vnějšku zahajuje. Tento účastník se pak znázorňuje jako panáček (obdobně jako aktér v diagramu případů užití). Tento symbol však není součástí symboliky sekvenčních diagramů (využívá se zde tedy vlastnosti pružnosti metodiky UML, jak bylo zmíněno na začátku kapitoly o UML).

Instanční sekvenční diagram popisuje pouze jednu instanci případu užití. Generický sekvenční diagram zachycuje všechny scénáře (instance) daného use case.

Sekvenční diagramy obsahují také konstrukty. Podmínky se zapisují do hranatých závorek nad příslušnou šipku. Podmínky pak způsobují selekci (rozdvojení) toku řízení. Příslušná zpráva se tedy rozdělí do dvou větví. Každá větev směřuje ke stejnému objektu, avšak tento objekt bude na každou z nich reagovat odlišně. To se znázorní rozdvojením jeho čáry života. Rozdvojené čáry života se později někde spojí. Dalším konstruktem je cyklus (iterace). Uzavírá se také do hranatých závorek, před levou závorku se navíc píše hvězdička.

Sekvence může vést k vytvoření nového objektu. Tento nový objekt se znázorní klasickým způsobem, avšak nezakreslí se v sekvenčním diagramu nahoru k dalším (původně existujícím) objektům, ale umístí se níže v diagramu tak, aby jeho pozice odpovídala času vzniku. Zpráva, která vede ke vzniku takového objektu, se pak může označit stereotypem « vytvořit ». Objekty mohou i zanikat. V sekvenčním diagramu se to znázorní tak, že na konec jeho čáry života se nakreslí „X“. Objekt také může provést operaci, kterou osloví sám sebe. Nazývá se to rekurze. Šipka tedy ukazuje na tu samou aktivace (operaci), ze které vychází.

Na obrázku 2.4 se nachází sekvenční diagram navrhované systému, který popisuje posloupnost operací, které jsou postupně volány při operaci přihlášení.



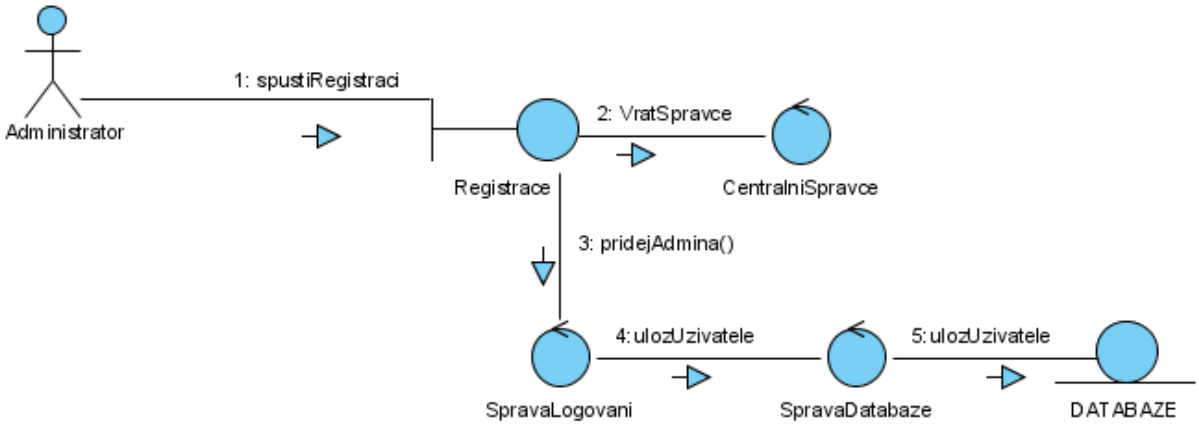
Obrázek 2.4: Sekvenční diagram (přihlášení do systému)

2.8.5 Diagram komunikace

Diagram komunikace se od sekvenčního diagramu odlišuje tím, že zdůrazňuje statickou strukturu propojení objektů, která se využije při interakci k dosažení požadovaného chování, což může být například definováno případem užití.

Diagram komunikace a sekvence si jsou podobny v syntaxi s tím rozdílem, že diagram sekvence je kreslen dvourozměrně. Podstatná je posloupnost zpráv. Vazby mezi spolupracujícími účastníky ukazují, jek odesílající zprávy získá spojení s adresátem. Diagramy komunikace se používají při návrhu prvních objektů, z tohoto důvodu se používají při počáteční komunikaci se zákazníkem, který si objednal softwarový projekt „na míru“.

Na obrázku 2.5 se nachází Diagram komunikace, který zobrazuje posloupnost zpráv při registraci nového administrátora anebo zaměstnance.

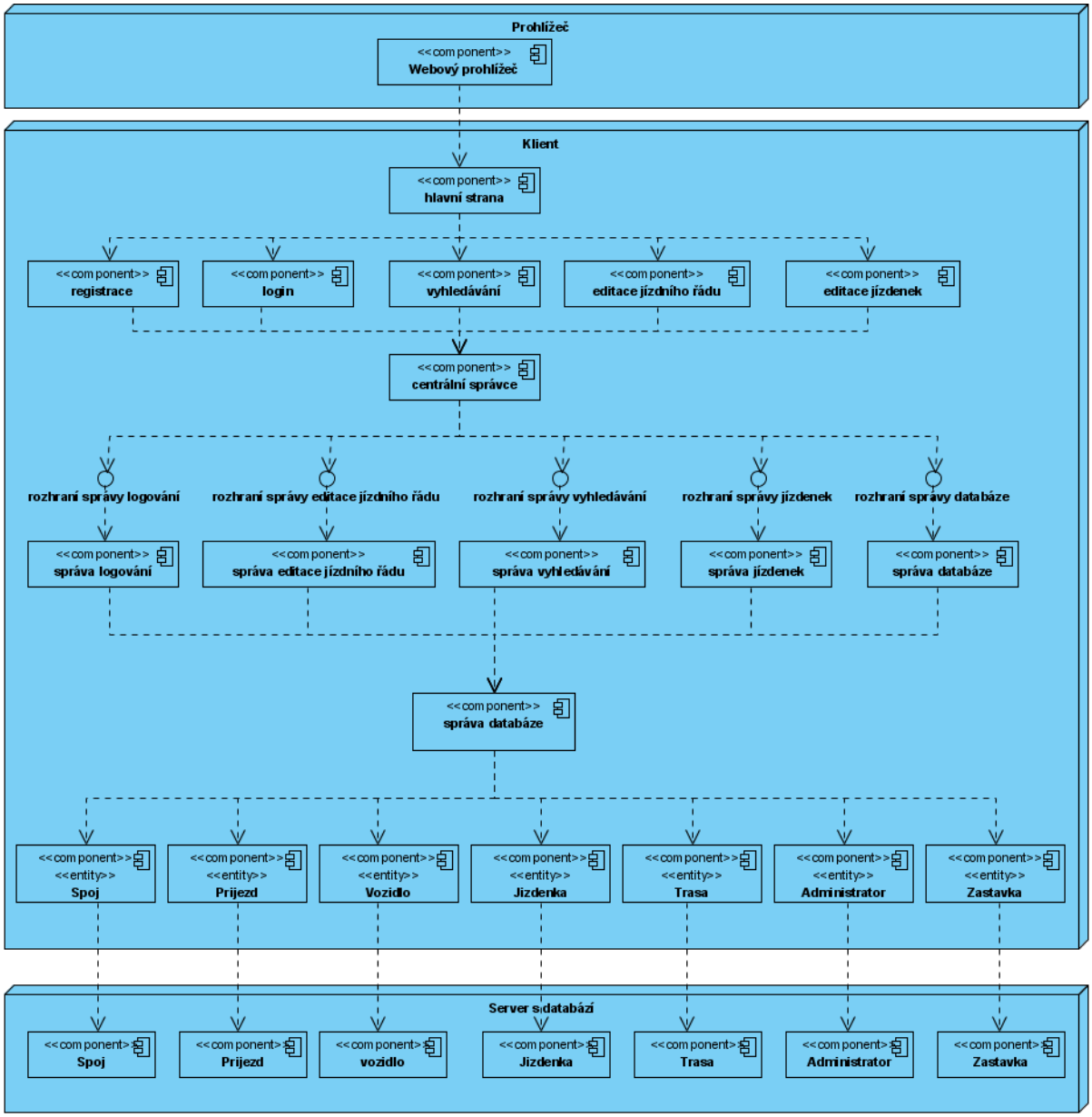


Obrázek 2.5: Diagram komunikace znázorňující registraci nového administrátora anebo uživatele

2.8.6 Diagram nasazení

Diagram nasazení je druhým typem diagramů určených pro implementační fáze. Jeho úkolem je především zobrazit vztahy mezi částmi systému tak, jak vypadají v době samotného vykonávání. Zobrazuje nastavení fyzických prvků na softwarovou architekturu a komunikaci mezi fyzickými prvky. Představuje topologie používaných sítí, druhy a využití komunikačních prostředků atd.

Obrázek 2.6 znázorňuje vzájemné oddělení částí klientské (SW části), webovým prohlížečem a serverem s databází (HW).



Obrázek 2.6: Výsledný diagram nasazení

2.8.7 Diagram činností

Diagram činností představuje sekvenční popis činností tak, jak postupně probíhají v systému. Vzniká rozšířením stavového diagramu. Diagram činností zachycuje především činnosti znázorněné ve stavovém diagramu.

Činnosti se znázorňují jako obdélníky se zaoblenými rohy. Po dokončení každé činnosti následuje automatický přechod na další činnost, což se zachytí šipkou. Začátek a konec diagramu činností se, podobně jako ve stavovém diagramu, znázorňuje pomocí kroužků.

Mnohdy je nutné rozhodnout, jaká činnost se bude vykonávat na základě určitých podmínek. Dochází tedy k větvení diagramu. Znázornit se to dá dvěma způsoby. Buď vychází možné cesty

přímo z některé činnosti, nebo se větvení znázorní pomocí malého kosočtverce, ze kterého pak větvené šipky vychází. Podmínky se uvádějí u větví (šipek) v hranatých závorkách.

Někdy je potřeba rozdělit tok událostí do dvou větví, které se budou provádět paralelně a později se zase spojí. Událost, kde dochází k větvení, a událost, kde se větve sbíhají, se označuje silnou příčnou čarou.

V průběhu sekvence činností je možné zasílat signály. Přijetí signálu způsobí to, že se začne vykonávat další (jiná) činnost. Zaslání signálu se v diagramu činností znázorňuje pětiúhelníkem ve tvaru šipky, přijetí signálu pětiúhelníkem ve tvaru šipky s hrotem šipky dovnitř. Přijetí a zaslání signálu se propojuje přerušovanou šipkou.

Z výše uvedeného je patrné, že se diagram činností podobá klasickému vývojovému diagramu.

V digramu činností je možné znázornit, který objekt odpovídá za kterou činnost. K tomu slouží prostředky pro zobrazení rolí. Diagram se rozdělí do oddílů, které se nazývají paralelní dráhy. Každá paralelní dráha je nadepsaná jménem role a obsahuje činnosti vykonávané danou rolí.

V diagramech činností je možné podle notace použít i symboly z jiných typů diagramů, jako jsou např. objekty (činnost směřuje k objektu, který ji vykoná). Tyto jiné prvky se znázorňují stejně jako v „mateřských“ diagramech. Takovéto diagramy činností se pak označují jako hybridní. Diagram činností je možné použít k popsání procesů.

2.8.8 Stavový diagram

Zachycuje stavy objektu, kterými může projít v průběhu svého životního cyklu v systému. Ke změně stavu může dojít konkrétní událostí nebo i pouhým vlivem času. Každý objekt má svůj počáteční stav a může mít konečný stav.

Stav se znázorňuje obdélníkem se zaoblenými rohy. Podobně jako o znázornění tříd je možné obdélník rozdělit na tři části, a to na název, stavové proměnné a činnosti stavu. Obvyklými činnostmi jsou vstup (co se stane, když se objekt dostane do tohoto stavu), výstup (co se stane, když objekt tento stav opustí) a proveď (co se provede, je-li systém v tomto stavu).

Přechody mezi stavy jsou symbolizovány šipkami. Přechody stavů mohou nést další informace. Spouštěcí událost je událost, která způsobí, že k přechodu dojde. Dále se uvádí seznam akcí, které přechod realizuje. Spouštěcí událost se od příslušné akce odděluje lomítkem.

Důležitým pojmem jsou podstavy. Jedná se o změny stavu v rámci jednoho stavu. Podstavy mohou byt dvojího druhu, sekvenční (podstavy na sebe navazují, spouštějí se postupně) a souběžné (podstavy probíhají paralelně).

Ukládaný stav je takový stav, který si pamatuje, v jakém podstavu se nacházel přesně před tím, než byl opuštěn. Používaná symbolika v UML je písmeno H (History) v kroužku se šipkou mířící k ukládanému podstavu.

2.9 Softwarová architektura – MVC

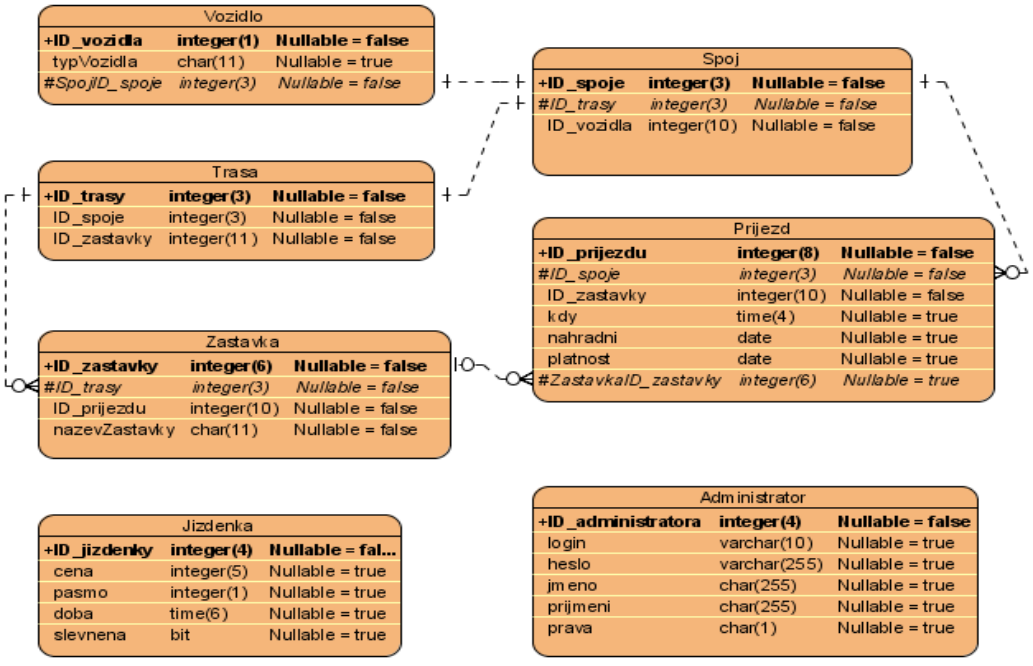
K tvorbě diagramu tříd byla zvolena softwarová architektura, která se skládá ze tří nezávislých modulů **Model-Zobrazení-Řadič** (Model-View-Controller) viz obrázek 2.2, neboť je typově vhodný pro zadání projektu a je přehledný.

Autorem MVC je Trygve Reenskaug, který jej poprvé popsal roku 1979. Originální implementace je popsána viz [2]. Touto architekturou získáme nejvyšší flexibilitu a rozšiřitelnost. Model není závislý na konkrétní aplikační logice, můžeme jej beze změny a snadno využít v jakékoliv aplikaci, jež pracuje se stejnými daty.

Navrhovaný diagram tříd dle architektury MVC byl zmíněn již v rámci kapitoly návrh podle specifikace UML 2.8.2.

2.9.1 Model

Tato část reprezentuje data samotné databáze, která bude později v implementaci uložena na MySQL serveru. Definuje pohled na všechna data (persistentní třídy), které se v IS nacházejí. Pro potřeby použití relační databáze (MySQL) se musel objektový model transformovat na ER diagram.



Obrázek 2.7: ER Diagram nad modelovanými daty

2.9.2 Řadič (Controller)

Termínem řadič se označuje část IS, která má na starosti komunikaci (správu) mezi persistentními daty (model) a uživatelským rozhraním (zobrazení). Zde se nacházejí všechny třídy a jejich metody, které přistupují k databázi přes správce databáze (obsahuje všechny potřebné SQL skripty).

2.9.3 Zobrazení

Poslední část IS, je grafické rozhraní, ve kterém operuje uživatel. Uživatel vyšle událost, požadavek a systém mu následně vrátí ta data, o která požádal. Komunikuje výhradně s řadičem.

3 Umělá inteligence

3.1 Úvod do UI

Pojem umělá inteligence (viz [9]) vymezuje Všeobecná encyklopedie DIDEROT (1996). Umělá inteligence je definována jako „modelování intelektuální činnosti člověka počítačem při řešení složitých úloh, kde postup vyžaduje schopnost výběru z mnoha nebo z nezřetelně popsaných variant; též samočinné rozpoznávání tvarů nebo předmětu, usuzování z jednoho výroku na jiný, vytváření analogií mezi jednotlivými úsudky, generování a ověřování hypotéz, tvorba a uplatnění znalostí na základě přijatých vstupních dat a informací, schopnost eliminovat nepříznivé reakce na podněty z okolí a usměrňovat činnost systému v probíhajících procesech s ohledem na měnící se a často nezřetelné vnější podmínky“.

3.2 Historie UI

Poprvé se o problematiku UI začali zajímat Warren McCulloch a Walter Pitts. V roce 1943 definovali umělý neuron a naznačili možnosti učení umělých neuronových sítí.

Roku 1949 bylo Donaldem Hebbem definováno základní pravidlo pro učení neuronových sítí, které se používá pro učení některých neuronových sítí dodnes. Další vývoj lze vymezit těmito fázemi:
1951 – první neuronový počítač (Marvin Minsky a Dean Almond)
1956 – poprvé použit pojem AI (Artificial intelligence – UI)
1958 – LISP – programovací jazyk
1960 – GPS (General Problem Solver, A. Newell) – první program k řešení obecných úloh
1962 – Perceptron (F. Rosenblatt) – neuronová síť rozpoznávající obraz s důkazem konvergence algoritmu pro učení této sítě.
1968 – A* Algoritmus (P. E. Hart, N. J. Nilsson, B. Raphael) – základní algoritmus pro řešení úloh

3.3 Přístupy k řešení úloh

I když je velmi obtížné, dokonce až nemožné vytvořit obecnou inteligenci, tak vědci za posledního půl století objevili množství postupů, jak řešit dílčí problémy. Jsou to například Neuronové sítě, Genetické programování, Expertní systémy a metody Prohledávání stavového prostoru.

3.3.1 Neuronové sítě

Neuronové sítě je běžný název pro technologii inspirovanou skutečnými neurony. Každý neuron má mnoho výčnělků, tzv. dendritů, jimiž přijímá signály od ostatních. A jeden velký výčnělek, axon, který tyto signály předává dál. Signály vysílá na základě zpracovaných informací na vstupech, takže funguje jako jakýsi procesor.

Modelování pomocí počítačových neuronových sítí je obdobné, neuron je nahrazen funkcí, která vstupy násobí váhami a vypočítá hodnotu pro výstup. Simuluje se propojení neuronů (v řádu desítek, stovek) a jejich sdružený výstup se zkouší využít pro různé účely. Užívají se pro rozpoznávání a kompresi obrazů nebo zvuků, předvídání vývoje časových řad (např. burzovních indexů), někdy dokonce k filtrování spamu. Síť se v první fázi „učí“, tedy vkládají se jí na vstup různá data a dolaďují se váhy tak, aby byl výstup správný. V druhé fázi se na vstup vkládají další data a síť je rozpoznává.

3.3.2 Genetické programování

Je to metoda strojového učení, která používá evoluční algoritmy. Umožňuje najít řešení klasických problémů a někdy také najde uspokojivé řešení některých velmi komplikovaných, nelineárních nebo kombinatorických úloh, například v oblasti data miningu nebo předpovídání počasí. Genetické programování může být u některých typů úloh náročnější na výpočetní výkon ve srovnání se specializovanějšími algoritmy a metodami.

3.3.3 Expertní systémy

Expertní systém je počítačový program, který má za úkol poskytovat rady, rozhodnutí nebo doporučit řešení v určité situaci. Jsou navrženy tak, aby mohly zpracovávat nenumerické a neurčité informace a řešit tak úlohy, které nejsou řešitelné tradičními algoritmickými postupy.

3.3.4 Prohledávání stavového prostoru

Jedním ze základních úkolů umělé inteligence je vytvořit metody pro strojové řešení úloh. Přes vysoký výpočetní výkon, kterým dnešní počítače oplývají, je pro naprostou většinu problémů zcela nemyslitelné, aby stroj hledal řešení postupným testováním všech možností. Vznikla potřeba řídit hledání efektivním způsobem. Pokud je řešená doména rozdělena do různých stavů a je definováno, že jeden z těchto stavů je počáteční, některé stavy jsou cílové a mezi různými stavy je možné aplikováním určitých akcí (operátorů) přecházet, vznikne stavový prostor. Je možné si jej představit jako orientovaný graf, jehož uzly jsou stavy a přechody udávají akci, jejímž vykonáním se lze dostat z jednoho stavu do druhého; nalezení řešení pak spočívá v nalezení cesty v grafu mezi počátečním a cílovým uzlem (viz Grafové algoritmy).

Otázkou však je, jakým způsobem tuto cestu hledat. Stavový prostor totiž může být pro řadu úloh příliš rozsáhlý, v některých případech dokonce nekonečný. Cílové stavy budou tedy od počátečního vzdáleny natolik, že počítač ani nemusí být schopen k žádnému z nich jednoduchými metodami v rozumném čase a s omezenou operační pamětí cestu najít. Navíc je nutné zvážit, že cílové stavy často nejsou známy – k dispozici je třeba jen představa o tom, jak by měly takové stavy vypadat.

Z těchto důvodů vznikly v uplynulých desítkách let různé metody (algoritmy) prohledávání stavového prostoru, které mají různé výhody a nevýhody. Nelze říci, že některá z metod je jednoznačně lepší než jiná. Záleží vždy na povaze řešené úlohy, požadavcích na řešení a dostupných prostředcích.

3.4 Metoda stejných cen (UCS)

Metoda UCS (Uniform Cost Search) pracuje s podobným algoritmem jako metoda BFS (Best First Search), s tím rozdílem, že uvažuje skutečné ceny přechodů (kladná čísla) a skutečné ceny cest (jsou dány součtem cen příslušných přechodů). Pro expanzi pak vybírá ze seznamu OPEN uzel s nejmenším ohodnocením, tj. uzel s nejnižší cenou cesty.

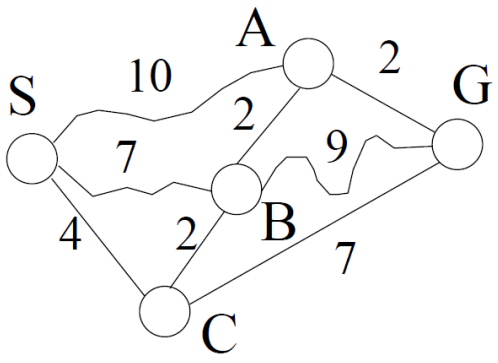
Základní algoritmus UCS je následovný:

1. Sestroj seznam OPEN (bude obsahovat všechny uzly určené k expanzi) a umísti do něj počáteční uzel včetně jeho (nulového) ohodnocení.
2. Je-li seznam OPEN prázdný, pak úloha nemá řešení a ukonči proto prohledávání jako neúspěšné. Jinak pokračuj.
3. Vyber ze seznamu OPEN uzel s nejnižším ohodnocením.
4. Je-li vybraný uzel uzlem cílovým, ukonči prohledávání jako úspěšné a vrať cestu od kořenového uzlu k uzlu cílovému (vrací se posloupnost stavů, nebo operátorů). Jinak pokračuj.
5. Vybraný uzel expanduj, všechny jeho bezprostřední následníky včetně jejich ohodnocení umísti do seznamu OPEN a vrať se na bod 2.

3.4.1 Příklad řešení úlohy metodou UCS

Na obrázku 3.1 je znázorněno řešení úlohy pomocí algoritmu UCS, kde vybere nejoptimálnější trasu (zdroj [9] strana 23).

Čísla u přechodů
značí jejich ceny.
Úlohou je nalézt
nejkratší cestu
z místa startu S
do cílového místa G.



Pořadí expanze	Seznam OPEN
0	[((S),0)]
1	[((A,S),10),((B,S),7),**((C,S),4)**]
2	[((A,S),10),**((B,C,S),6)**,((G,C,S),11)]
3	[((G,C,S),11),**((A,B,C,S),8)**]
4	[**((G,A,B,C,S),10)**] = cíl ⇒ cesta = (S,C,B,A,G)

Obrázek 3.1: Metoda UCS prohledávající stavový prostor a způsob eliminace cest s vyšším ohodnocením, než je optimální

4 Implementace

4.1 Prostředky k implementaci

4.1.1 XHTML

XHTML je zkratka extensible hypertext markup language (rozšiřitelný hypertextový značkovací jazyk). XHTML se používá při tvorbě hypertextových dokumentů (viz [7]). XHTML vzniklo v roce 2000 jako projekt firmy W3C (www.w3c.org). Existují 3 druhy XHTML:

a) **XHTML 1.0 Strict** – používá se, v případě, že je požadavek na strukturovaný dokument, ve kterém se ale nenachází formátovací značky, které mají na starosti rozvržení stránky. Předpokládá se jeho užití společně s CSS, které umožňuje dosáhnout potřebných grafických efektů. Nicméně i tato verze obsahuje formátovací elementy, například <b> nebo <i> a naopak zavrhuje některé sémantické elementy, například <menu>.

b) **XHTML 1.0 Transitional** – je přechodným definicí typu dokumentu pro webové stránky, který vám umožní používat překonané tagy. Nenahraditelný je u webových prohlížečů staršího data, které nepodporují CSS.

c) **XHTML 1.0 Frameset –** umožňuje používat zastaralé značky jako XHTML 1.0 Transitional a přidává podporu pro rámce. V dnešní době by se mělo rámcům vyhýbat použitím CSS, AJAXu nebo serverových skriptů jako například PHP.

4.1.2 CSS

CSS je zkratka pro Cascading Style Sheets (tabulky kaskádových stylů), je to jazyk, který určuje vzhled HTML/XHTML dokumentů. Hlavním devizou CSS je umožnit oddělení vzhled dokumentu od jeho struktury a obsahu. Původně to měl umožnit už jazyk HTML, ale v důsledku nedostatečných standardů a konkurenčního boje výrobců prohlížečů se vyvinul jinak.

Starší verze HTML obsahují celou řadu elementů, které nejenom že určují obsah a strukturu dokumentu, ale i způsob jeho zobrazení. Z hlediska zpracování dokumentů a vyhledávání informací není takový vývoj žádoucí, nehledě na případnou změnu už hotového vzhledu projektu. Výhodou CSS oproti starému formátování v HTML je, že kód a obsah webu je uložen v souboru *.html a veškerý design a formátování se načítá z jednoho souboru *.css, který je většinou společný pro celý web. To znamená, že pro změnu designu webu, stačí změnit pouze jeden soubor *.css a změna se aplikuje na celý web. Soubor CSS se uloží do mezipaměti prohlížeče a pokud není změněn, tak se načítá pouze jednou, čímž se zobrazení stránek se velmi urychlí.

CSS styl vytvořený pro tento informační systém je velice přehledný (viz obrázek 4.2).

4.1.3 PHP

PHP je skriptovací jazyk (Personal Hypertext Preprocessor) pro tvorbu dynamického webu, jehož počátky se datují od roku 1994. Tehdy pan Rasmus Lerdorf vytvořil jednoduchý systém pro počítání přístupu ke svým stránkám, který byl napsán v PERLu (interpretovaný programovací jazyk). Za nějakou dobu byl systém přepsán do jazyka C, protože kód v PERLu dost zatěžoval server. Sada těchto skriptů byla ještě později téhož roku vydána pod názvem „Personal Home Page Tools“, zkráceně PHP. Ještě potom se to jmenovalo „Personal Home Page Construction Kit“.

4.1.4 MySQL

Relační databázový server MySQL (viz [8]) byl vytvořen jako interní firemní projekt, v jehož čele stáli zaměstnanci Michael Widenius a David Axmark. Poprvé byl vydán pro veřejnost v roce 1995 softwarovou firmou TCX Data Konsult AB se sídlem ve Švédsku v Uppsale.

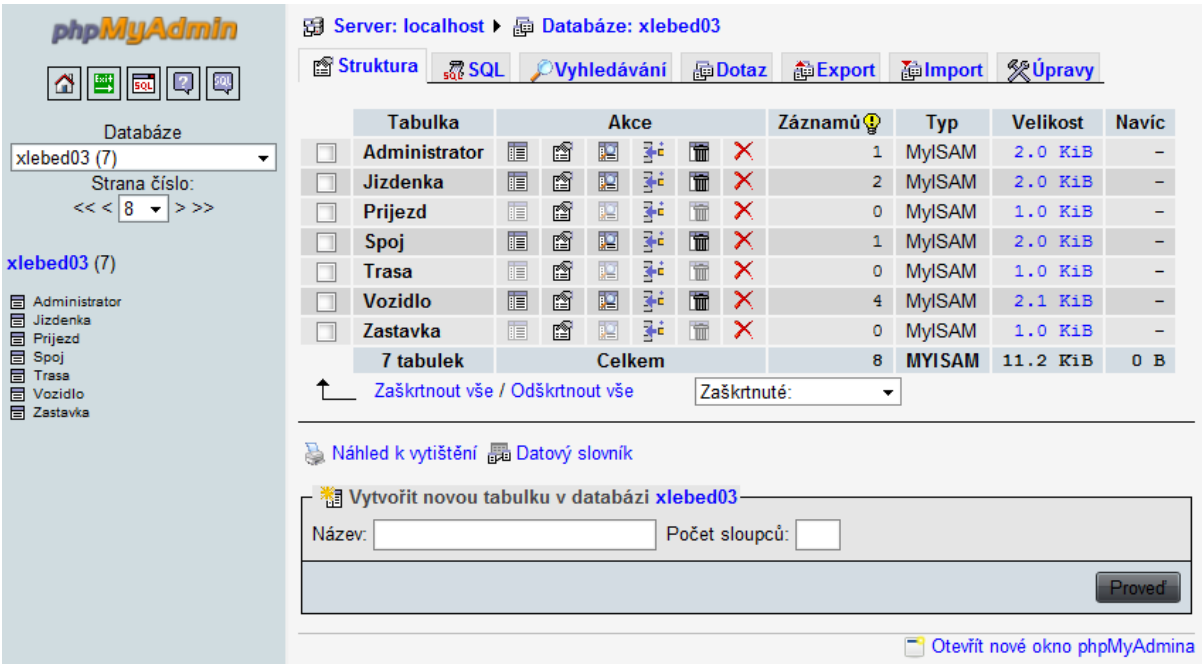
Od prvního veřejného vydání v roce 1995 kladli vývojáři MySQL značný důraz na výkon softwaru, což vedlo k vysoce optimalizovanému produktu, který ovšem postrádal mnohé schopnosti, které jsou například uložené procedury, triggery a transakce. Přesto produkt k sobě přitáhl pozornost nesmírného množství uživatelů, které spíše zajímala rychlost, než nejvyspělejší možnosti, které v mnoha případech stejně zůstanou nevyužité.

Práci databáze značně urychluje i fakt, že umí cachovat dotazy. Pokud je tato funkce aktivovaná, tak si MySQL uchovává dotazy select s jejich výstupními hodnotami. Pokud se začnou spouštět nové dotazy, tak se začnou porovnávat s cache předchozích dotazů a pokud budou shodné, tak MySQL nepoužije vyhledávání v databázi, ale použije hodnotu totožného dotazu. Pro prevenci k výstupům zastaralých dat jsou zabudované mechanismy, které automaticky odstraňují zastaralé cachované výsledky a nahrazují je při příštím požadavku novými.

Od verze 3.23.23 bylo MySQL obohaceno o schopnosti fulltextového indexování a vyhledávání. Tato schopnost také značně zvyšuje výkon, když se „dolují“ ze sloupců obsahujících texty (CHAR, VARCHAR, TEXT, atd.). Také dokáže vytvářet výsledky hledání seřazené podle relevance (jak moc se shoduje dotaz s nalezeným řádkem).

Jednou z dalších výhod MySQL jsou jeho replikační schopnosti. Replikací se zajistí přechod z DB na jednom serveru na přechod na jiný server, čímž se zvýší efektivita databáze. Další výhodou je zálohování, kdy se nemusí odpojit aplikace od databáze, poněvadž se zálohuje replika.

K účelům této BP se použila aplikace phpMyAdmin ve verzi 2.11.5 na školním serveru EVA, která umožňuje vytvářet/rušit databáze, vytvářet/upravovat/rušit tabulky, provádět SQL příkazy a spravovat klíče, viz obrázek číslo 4.1.



Obrázek 4.1: phpMyAdmin

4.2 Vzhled systému

Obrázek 4.2 reprezentuje úvodní obrazovku s pozdravem. Na obrázku 4.3 je vyobrazena finální verze registračního formuláře.

Obrázek 4.2: Úvodní obrazovka s pozdravem umístěným ve vrchní části

Obrázek 4.3: Finální verze registračního formuláře

5 Závěr

Tato technická dokumentace byla zaměřena na analýzu a návrh informačního systému.

Cílem toho informačního systému bylo vytvořit informační systém autobusové dopravy, který bude zobrazovat vývěsky a jízdenky, a také vyhledávat nejrychlejší spoj za pomoci umělé inteligence s tím, že se systémem budou pracovat 3 druhy uživatelů, což jsou neregistrovaný uživatel, zaměstnanec a administrátor (viz kapitola 2.3).

Prostřednictvím této práce jsem se seznámil s objektově orientovaným PHP, návrhovým jazykem UML, dále jsem si zlepšil znalosti v CSS, MySQL, XHTML a v oboru umělé inteligence.

Tento systém by se dal dále rozšířit o výpočet ceny jízdného při zadání trasy, což by bylo velmi přínosné pro uživatele, kteří nejsou místní a v cenách, popř. v dopravě se moc nevyznají. Další významné rozšíření by mohlo být v podobě prohlížeče IS bez nutnosti webového prohlížeče, zanesení zastávek a spojů do dvourozměrné mapy, z čehož je nasnadě zanesení vyhledané cesty i s přestupy (viz www.mapy.cz – plánovač tras). Všechna tato rozšíření by mohla být tématem případné diplomové práce.

Literatura

[1] Základní návrhové vzory - http://objekty.vse.cz/Objekty/Vzory-prehled

[2] Applications Programming in Smalltalk-80: How to use Model-View-Controller – http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html

[3] Návrh aplikací v jazyce UML – Unified Modeling Language – http://interval.cz/clanky/navrh-aplikaci-v-jazyce-uml-unified-modeling-language/

[4] Arlow Jim, Neustadt Ila, UML a unifikovaný proces vývoje aplikací, Computer Press, 2003

[5] Přehled OO metodik a notací – http://objekty.vse.cz/Objekty/MetodikyANotace-UMLDiagramy

[6] Schmuller, Joseph: Myslíme v jazyku UML, knihovna programátora, Grada Publishing, Praha 2001, první vydání

[7] Extensible HyperText Markup Languaghet – http://cs.wikipedia.org/wiki/Extensible_HyperText_Markup_Language

[8] W. Jason Gilmore, Velká kniha PHP5 a MySQL,Zoner Press, 2005

[9] doc. Ing. František Zbořil, CSc., Základy umělé inteligence, studijní opora k IZU (FIT), 2006

[10] Specifikace webových standardů – http://www.w3.org

[11] Jak psát web – http://www.jakpsatweb.cz

[12] Welling Luke, Thomson Laura, PHP a MySQL – rozvoj webových aplikací, druhé vydání, SoftPress, 2003

[13] ULLMAN, L.: PHP a MySQL: Názorný průvodce tvorbou dynamických WWW stránek. Computer Press, 2004

[14] XHTML eXtensible HyperText Markup Language – URL http://www.webtvorba.cz/xhtml/

[15] The Object Management Group – http://www.omg.org

[16] Manuál PHP – http://www.php.net/manual/cs

[17] MySQL 5.0 Reference Manual – http://dev.mysql.com/doc/refman/5.0/en

[18] CASTRO Elizabeth, HTML, XHTML a CSS, 2007, ISBN 80-251-1531-2.

[19] Tvorba kvalitních webových stránek – http://tvorba-www-stranek.kvalitne.cz

[20] Jak psát web – http://www.jakpsatweb.cz/

[21] Projektování informačních systémů II 2007/2008 – http://info.sks.cz/users/ku/PRI/UML.HTM

[22] Hruška, T., Křivka, Z.: Informační systémy (IIS, PIS) Studijní opora, 2006

Seznam příloh

Příloha 1. DVD