Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Návrh a tvorba webového informačního systému pro správu realit

Bc. Martina Lebdušková

Diplomová práce

2009

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra softwarových technologií
Akademický rok: **2008/2009**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Martina LEBDUŠKOVÁ**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Návrh a tvorba webového informačního systém pro správu realit**

Zásady pro vypracování:

V teoretické části práce budou popsány možnosti jazyka UML pro vizuální modelování objektových systémů a metodika unifikovaného procesu (UP) Dále je potřebné popsat možnosti tvorby webových aplikací (statické/dynamické, jazykové platformy atd.) a jejich technické prostředky pro jejich provozování Praktická část bude prvotně zaměřena na vypracování projektu informačního systému pomocí nástrojů jazyka UML zaměřenou na analýzu, návrh a design aplikace V další fázi se následně provede implementace systému pomocí vhodných technických prostředků

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

ARLOW, J.,NEUSTADT, I. UML a unifikovaný proces. Brno, Computer Press Books, 2005. SCHUMELLER, J. Myslíme v jazyku UML. Praha, GRADA, 2001. ULLMAN, L. PHP a MySQL. Brno, Computer Press Books, 2004.

Vedoucí diplomové práce: **Ing. Jan Fikejz**
Katedra softwarových technologií

Datum zadání diplomové práce: **31. října 2008**

Termín odevzdání diplomové práce: **22. května 2009**

Univerzita Pardubice Česká republika

L.S.

doc. Ing. Simeon Karamazov, Dr.
děkan

doc. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 4. listopadu 2008

Prohlašuji:

Tuto práci jsem vypracovala samostatně. Veškeré literární prameny a informace, které jsem v práci využila, jsou uvedeny v seznamu použité literatury.

Byla jsem seznámena s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 27. 5. 2009

Bc. Martina Lebdušková

Poděkování:

Na tomto místě bych ráda poděkovala panu Ing. Janu Fikejzovi za konzultace, které mi velmi ochotně poskytoval během vypracování diplomové práce.

ANOTACE

Tato práce se zaměřuje na oblast projektování informačních systému. Využívá se zde unifikovaný proces tvorby informačních systémů. Výsledkem této diplomové práce je webový informační systém správy realit, který byl vytvořen podle postupů unifikovaného procesu. Informační systém prošel všemi fázemi tvorby od analýzy přes návrh po testování.

KLÍČOVÁ SLOVA

unified modeling language; analýza; návrh; implementace

TITLE

Design and Creation of Web Information System for Property Management

ANNOTATION

This work focuses on the sphere of information systems design. Unified Process is applied to the creation of information systems. The result of this thesis is a web information system for property management which was created due to the procedure of Unified Process. The information system went through all phases of the creation from analysis through design to testing.

KEYWORDS

unified modeling language; analysis; design; implementation

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

UML	Unified Modeling Language
UP	Unified Process
IS	Informační systém
CASE	Computer-Aided Software (System) Engineering
IIS	Internet Information Services
PHP	Hypertext Processor
MVC	Model-View-Controller
ZF	Zend Framework
SW	Software, softwarový
RUP	Rational Unified Process
CASE	Computer-Aided Software (System) Engineering
(X)HTML	eXtensible Hypertext Markup Language
CSS	Cascading Style Sheets
JSP	Java Server Pages
ASP	Active Server Pages
URL	Uniform Resource Locator
RK	Realitní kancelář
HTTP	Hypertext Transfer Protocol
SQL	Structured Query Language
FTP	File Transfer Protocol
SMTP	Simple Mail Transfer Protocol

1 Úvod

Práce se zabývá tvorbou informačního systému. Informační systém se dá charakterizovat jako jakýkoli systém, ve kterém je možno shromažďovat, udržovat, zpracovávat nebo poskytovat data a informace. Pokud je informační systém přístupný na Internetu nebo intranetu, jedná se o webový informační systém. Takový systém může mít podobu např. obyčejného blogu, který určitým způsobem poskytuje data, nebo se může jednat o sofistikovaný systém, který je vytvořen přímo pro udržování a poskytování dat či informací.

Při tvorbě informačního systému je výhodou, když se využije unifikovaný proces a jazyk vizuálního modelování UML, protože vznikne dokumentace veškerých postupů při vývoji. V dnešní době, kdy je vývoj informačních systémů velmi rychle rostoucím odvětvím, se podcenění analýz a návrhů nevyplácí. Nekoordinovaná tvorba informačního systému by mohla mít za následek u firem zabývajících se tvorbou informačních systémů zvýšení nákladů a neschopnost dovést projekt do úspěšného konce.

Cílem této práce je vytvořit webový informační systém, který bude poskytovat a udržovat informace o nemovitostech, které jsou na realitním trhu k dispozici. Aby byl výsledný webový informační systém dobře navržen pro pozdější možnost rozšíření či úpravy, bude využit při jeho tvorbě unifikovaný proces s podporou modelování za pomoci vizuálního jazyka UML.

V práci budou popsány jednotlivé prvky vizuálního jazyka pro modelování UML a jednotlivé fáze tvorby informačního systému. Dále budou nastíněny druhy webových aplikací a popsány implementační jazyky, které mohou být využity pro samotnou implementaci informačního systému.

2 Modelování v jazyce UML

2.1 Co je UML?

Unified Modeling Language neboli zkráceně UML je univerzální jazyk pro vizuální modelování. Jedná se o souhrn grafických notací, které umožňují vyjádřit analytické a návrhové modely. Je využíván ve vývojové metodice unifikovaného procesu (UP) nebo v metodice RUP, ale sám o sobě nespecifikuje metodiku nebo proces.

2.2 Historie

Jazyk UML začal vznikat v roce 1995, před tímto datem existovalo několik jazyků pro vizuální modelování a zároveň i několik metodik. Oblast objektově orientovaných metod neměla stanovené standardy.

Následující rok 1996 sdružení Object Management Group (OMG) navrhlo specifikaci RFT (Request For Proposal), ve které byl jazyk UML navrhnut jako standard pro vizuální modelování objektově orientovaných systémů. O rok později OMG přijalo UML jako standard objektově orientovaného jazyka pro vizuální modelování.

V průběhu dalších let docházelo k rozšiřování jazyka UML. Dalším milníkem ve vývoji UML bylo ukončení specifikace jazyka UML 2.0 v roce 2005. Tato verze je obohacena o řadu nových prvků, které umožňují vyspělé modelování.

A kdo vlastně může za vznik jazyka UML? Hlavní zásluhu na jeho vzniku měli pánové Grady Booch, Ivar Jacobson a James Rumbaugh.

2.3 Struktura UML

UML je složeno z velkého množství grafických prvků. Tyto prvky lze spojit a kombinovat do podob různých diagramů.

2.3.1 Stavební bloky

Stavební bloky jsou pokládány za základní prvky modelu, jako jsou relace a diagramy. Jazyk UML se skládá ze tří stavebních bloků.

Předměty

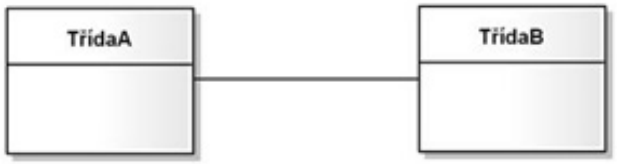
Předměty jsou definovány jako samotné prvky modelu. Předměty nebo také „věci“ se v jazyce UML dělí na:

- strukturní abstrakce – třídy, případy užití, komponenty, rozhraní, apod.,
- chování – interakce nebo stavy,
- seskupení – balíčky seskupující významově související prvky,
- poznámky – anotace, které lze přidat do modelu za účelem zachytit informaci.

Relace

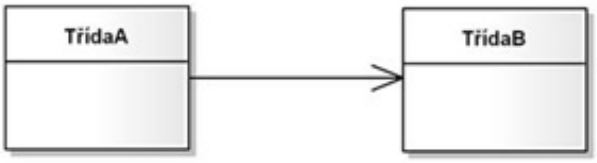
Vztahy neboli relace umožňují vyjádřit souvislost mezi dvěma nebo více předměty. Existuje několik druhů relací. Tyto relace jsou shrnuty v následujících bodech.

- Asociace - volnější vazba mezi třídami umožňující komunikaci mezi třídami. Třídy mají na sebe vzájemný odkaz. Obrázek 1 se dá interpretovat jako: *Třída A je ve vztahu s třídou B a naopak.*



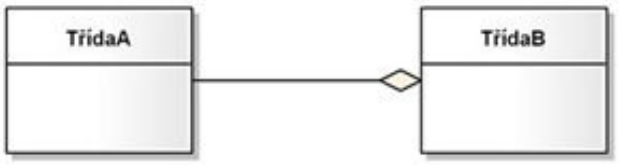
Obrázek 1: Vztah typu asociace. Zdroj: vlastní

- Jednosměrná asociace – typ vztahu odvozen od asociace s omezením směru komunikace. Obrázek 2 se dá interpretovat jako: *„Třída A komunikuje s Třídou B, ale Třída B nemůže komunikaci opětovat.“* Třída B nemá na třídu A odkaz.



Obrázek 2: Vztah typu jednosměrná asociace. Zdroj: vlastní

- Agregace - vztah části a celku. Část může existovat sama o sobě a může se stát součástí jiného celku. Obrázek 3 lze interpretovat jako: *„Třída B v sobě obsahuje třídu A.“* nebo jako: *„Třída A může být obsažena v třídě B“*.



Obrázek 3: Vztah typu agregace. Zdroj: vlastní

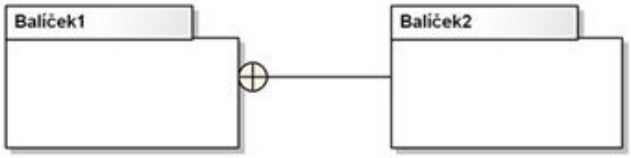
- Kompozice - silnější vazba mezi částí a celkem oproti agregaci. Část může existovat v rámci pouze jediného celku, pokud je celek zrušen, zaniká i jeho část. Obrázek 4 lze interpretovat jako: *„Třída B se skládá z třídy A.“* nebo jako: *„Třída A je složkou třídy B“*.

Obrázek 4: Vztah typu kompozice. Zdroj: vlastní

- Realizace - vztah mezi třídou a rozhraním je naznačen na Obrázek 5.

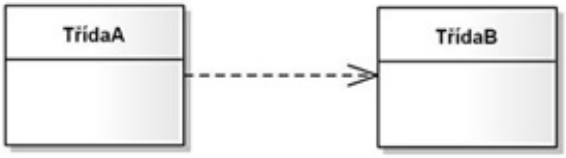
Obrázek 5: Vztah typu realizace. Zdroj: vlastní

- Ochranná nádoba – vztah mezi dvěma prvky je naznačen na Obrázek 6. Cílový prvek je obsažen ve zdrojovém. Využívá se v systému balíčků k namodelování jmenných prostorů.



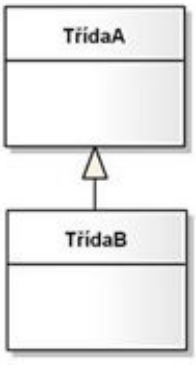
Obrázek 6: Vztah typu ochranná nádoba. Zdroj: vlastní

- Závislost – změna jednoho prvku se promítne do závislého prvku. Závislost je znázorněna na Obrázek 7.



Obrázek 7: Vztah typu závislost. Zdroj: vlastní

- Zobecnění – jeden prvek (*Třída A*) je zobecněním prvku druhého (*Třída B*). Tento vztah je zachycen na Obrázek 8.

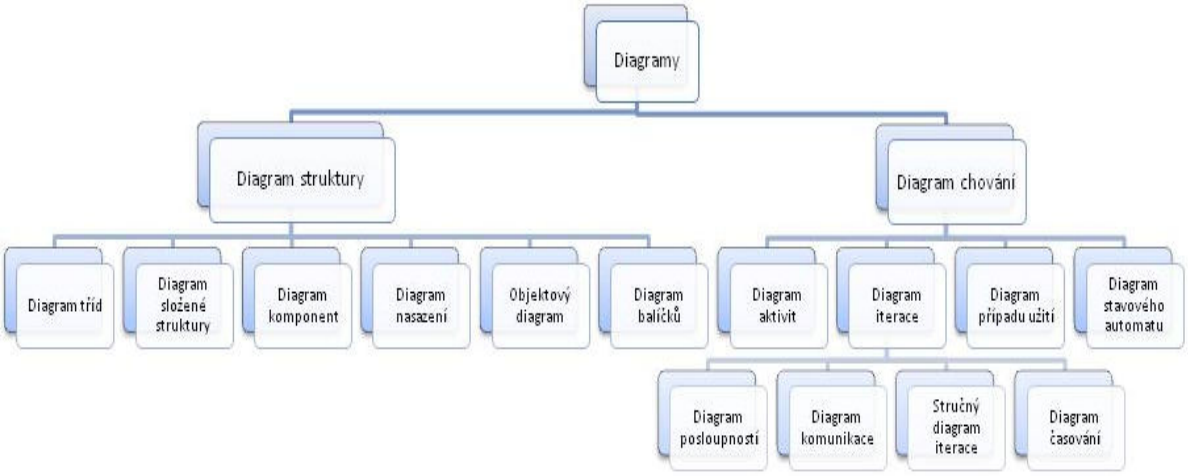


Obrázek 8: Vztah typu zobecnění. Zdroj: vlastní

Diagramy

Jsou považovány za pohledy na model. V rámci diagramu lze předměty odstranit, zatímco v modelu zůstávají. UML má k dispozici třináct diagramů, které slouží k znázornění

chování nebo struktury navrhovaného systému. Obrázek 9 zachycuje základní strukturu diagramů v UML.



Obrázek 9: Struktura diagramů v UML. Zdroj: vlastní

Diagramy struktury definují statickou strukturu modelu. Využívají se k modelování „věcí“, které tvoří model. Patří mezi ně třídy, objekty, rozhraní a fyzické komponenty.

- Diagram balíčků (Package diagram) slouží k rozdělení modelu do logických balíčků a popisuje vzájemnou spolupráci mezi nimi na nejvyšší úrovni.
- Diagram tříd (Class diagram) definuje základní stavební bloky modelu, jako jsou typy, třídy a základní součásti pro konstrukci kompletního modelu.
- Objektový diagram (Object diagram) znázorňuje, jaká rozhraní konstrukčních prvků jsou spojená a užívána v daném okamžiku.
- Diagram složené struktury poskytuje způsoby rozvrstvení prvků struktury a zahrnuje skryté detaily, konstrukci a vztahy.
- Diagram komponent je využíván pro modelování vyšší úrovně nebo komplexnější struktury, která je obvykle sestavena z jedné nebo více tříd a poskytuje dobře definované rozhraní.
- Diagram nasazení znázorňuje fyzické uspořádání podstatných artefaktů včetně nastavení.

Diagramy chování zachycují rozmanitost interakce a okamžitých stavů v modelu, jak jsou vykonávány během časové posloupnosti. Tyto diagramy sledují, jak se bude navrhovaný systém chovat v prostředí skutečného světa, a pozorují účinky operací zahrnujících jejich výsledky.

- Diagram případu užití je využíván k modelování vzájemného působení mezi uživatelem a systémem. Definují chování, požadavky a podmínky v rámci scénářů.
- Diagram aktivit má širokou škálu použití od základní definice toku programu k získání rozhodovacích bodů a akcí uvnitř jakéhokoli zobecněného procesu.
- Diagram stavového automatu je základem pro porozumění okamžiku.
- Diagram iterace zejména znázorňuje komunikaci.

- Diagram komunikace zobrazuje pořadí zpráv nebo komunikace mezi objekty v daném okamžiku během případu spolupráce.
- Diagram posloupností neboli sekvenční diagram blízce souvisí s diagramem komunikace a znázorňuje posloupnost zpráv, které prošly mezi objekty využívající čáru života.
- Diagram časování spojuje diagramy sekvenční a stavové, aby poskytl pohled na stav objektů v průběhu času a na zprávy měnící stav.
- Stručný diagram iterace spojuje diagramy aktivit a sekvenční diagramy.

Bližší informace k nejpoužívanějším diagramům při návrzích informačních systémů budou poskytnuty v následujících kapitolách.

2.3.2 Společné mechanismy

Specifikace

Specifikace jsou textovým popisem sémantiky jednotlivých prvků.

Ornamenty

Každý prvek v UML lze vyjádřit jednoduchým symbolem, který je možné obohatit o tzv. ornamenty, potřebujeme-li vyjádřit více informací. Jednoduše řečeno ornamenty rozšiřují minimalistický pohled na model.

Podskupiny

V rámci podskupin lze narazit na dva pohledy na podskupiny:

- skupina klasifikátorů a instancí – klasifikátor je abstraktní vyjádření předmětu, zatímco instance je konkrétní realizací předmětu,
- skupina rozhraní a implementace – rozhraní definuje co má předmět vykonávat, zatímco implementace určuje, jak to má vykonávat.

Mechanismy rozšiřitelnosti

Jelikož existující prvky jazyka UML nemusí plně vyhovovat všem uživatelům, byly do něj zahrnuty tři mechanismy, které umožňují jeho rozšiřitelnost.

- Omezení – specifikace podmínky nebo pravidla pro daný prvek modelu
- Stereotyp – definice nového prvku v rámci tvořeného modelu a to za podmínky, že je založen na existujícím prvku
- Označené hodnoty – rozšíření prvku o jeho vlastní vlastnosti

2.3.3 Architektura

Architektura je definována jako organizační struktura systému. Pohledů na architekturu může být po více, ale nejvýznamnějším a nejčastěji používaným pohledem je pohled „4+1“.

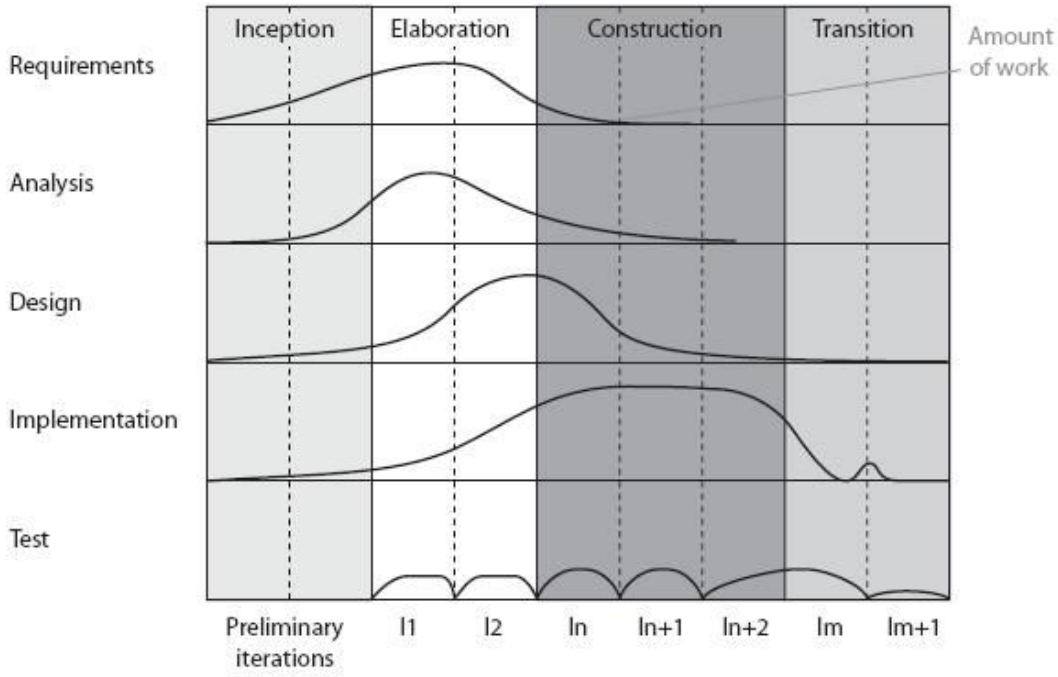
- Logický pohled se týká slovníku a funkcí systému.
- Pohled procesů se týká výkonu systému.
- Pohled implementace zachycuje soubory, které utvářejí hotový kód systému.
- Pohled nasazení se týká topologie systému, možností doručení a instalace.
- Pohled případů užití integruje všechny předchozí pohledy a zachycuje požadavky zadavatele.

3 Unified Process

Unifikovaný proces softwarového vývoje nebo zkráceně unifikovaný proces (UP) je oblíbeným iterativním a přírůstkovým procesem pro vývoj softwarových systémů. Iterativní znamená, že jeden velký projekt je rozdělen na menší podprojekty, které jsou řešeny v dávkách neboli přírůstcích, které vytváří systém postupně. Nejznámějším propracovaním UP s úplnou dokumentací a bohatým uživatelským prostředím je RUP neboli Rational Unified Process.

UP není jednoduchý proces ale spíše rozšiřitelný systém, který by měl být přizpůsobitelný pro specifické organizace a projekty. RUP je obdobně přizpůsobitelný systém. V důsledku toho je často nemožné určit, jestli zpracování procesu proběhlo pomocí UP nebo RUP. Proto mají názvy sklon k zaměnitelnosti.

Název UP, jako protiklad RUP, je obecně užíváno k popisu všeobecného procesu, zahrnujícího elementy, které jsou společné mnoha zpracováním. Název UP je tedy užíván k vyvarování se sporné otázce ohledně porušování autorských práv, protože RUP je ochranou známkou firmy IBM. První kniha, která popisovala UP se nazývala Unifikovaný proces softwarového vývoje a byla publikována Ivarem Jacobsonem, Grady Boochem a Jamesem Rumbaughem.



Obrázek 10: Fáze životního cyklu projektu. Zdroj: [1]

Iterativní proces umožňuje vracení se ke klíčovým postupům v průběhu tvorby systému i několikrát. Iterace se skládá z pěti základních pracovních postupů:

1. Požadavky (Requirements) odrážejí funkcionalitu systému.
2. Analýza (Analysis) upřesňuje požadavky.
3. Návrh (Design) vytváří architekturu systému na základě požadavků.
4. Implementace (Implementation) realizuje návrh.
5. Testování (Test) ověřuje správnou funkčnost implementace.

Ne každá iterace obsahuje všechny tyto postupy. Záleží na fázi životních cyklu projektu, ve které se daný projekt nachází. Jednotlivé postupy jsou zde jen nastíněny z důvodu podrobnějšího rozebrání problematiky v rámci praktické části této práce zaměřené právě na projektování informačního systému pomocí metodiky UP.

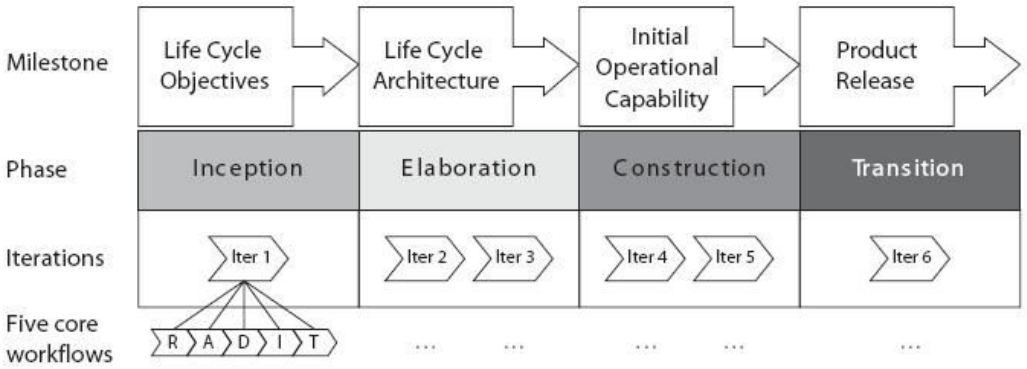
Výhoda iteračního přístupu, kterou využívá metodika UP, spočívá v možnosti běhu dvou a více iterací zároveň, což vede k urychlení dokončení projektu s efektivním rozvržením pracovních sil, za podmínky pečlivého plánování.

Životní cyklus každého projektu, zobrazen na Obrázek 10, se skládá z následujících fází, které mají své zaměření, cíl a milník:

1. Zahájení (Inception) je obdobím plánování.
   Odpovědná osoba: projektový manažer, systémový projektant.
   Cíl: zjištění rizik, zjištění proveditelnosti, definice nákladů a příjmů, zjištění rozsahu projektu.
   Milník: schválení projektu a rozpočtu, potvrzení proveditelnosti, náčrt architektury.
2. Rozpracování (Elaboration) je obdobím architektury.
   Cíl: spustitelný architektonický základ, zachycení případů užití, definice kvality.
   Milník: vytvoření spustitelného prototypu, vytvoření plánu celého projektu.
3. Konstrukce (Construction) je počátečním obdobím funkčnosti.
   Cíl: odhalení veškerých požadavků, dokončení analytického a návrhového modelu, zajištění provozní způsobilosti prototypu, testování hotových funkcionalit.
   Milník: produkt připraven k nasazení u uživatelů, revize příjmů a výdajů.
4. Zavedení (Transition) je obdobím nasazení produktu do reálného užívání.
   Cíl: oprava chyb, příprava prostředí pro nasazení produktu, tvorba dokumentací.
   Milník: dokončení testů, produkt předán zadavateli.

Milníkem jsou chápány pokroky v práci na projektu. Jedná se o cíle, které musí být splněny, aby byla fáze ukončena.

Metodika, jak už bylo zmíněno v úvodu, pracuje v iteracích. Jedná se o součásti, které kompletují celkový systém. V rámci jedné fáze se může objevit několik iterací zahrnujících výše popsané aktivity, tuto variantu zachycuje Obrázek 11.



Obrázek 11: Iterace. Zdroj [1]

4 Nástroje pro modelování

Nástroje pro modelování jsou označovány jako CASE nástroje. Jedná se o zkratku z anglického spojení Computer-Aided Software Engineering (počítačem podporované softwarové inženýrství) a někdy je tato zkratka interpretována jako Computer-Aided System Engineering (počítačem podporované systémové inženýrství).

Tyto nástroje je mají za úkol podporovat vývoj softwaru. Jsou tedy zaměřeny na podporu kompletního vývojového postupu od analýzy až po testování. Využívají UML pro vizuální modelování jako základní prvek. Elementy UML jsou tematicky rozděleny do skupin podle toho, ve kterém z druhů diagramů se používají.

Výhody CASE nástrojů:

- Přehled o vývoji a jednodušší údržba vyvíjeného systému
- Vznik kvalitní dokumentace
- Možnost spolupráce při vývoji
- Generování zdrojových kódů ve vybraném jazyce
- Tvorba SQL příkazů z datového modelu
- Import/export části nebo celého modelu systému

Následující výčet CASE nástrojů obsahuje ty nejpoužívanější:

- Powerdesigner od firmy Sybase
- Oracle Designer od firmy Oracle
- Rational Rose od firmy IBM
- Microsoft Visio od firmy Microsoft
- Enterprise Architect od firmy Sparx

Tyto nástroje jsou tedy dobrým zdrojem pokladů pro jakoukoli dokumentaci, neboť modely v nich vzniklé se za tuto dokumentaci mohou považovat. Jakýkoli vzniklý model či diagram je možné jednodušeji upravit, jelikož vše je uchováno v elektronické podobě.

V rámci této práce k tvorbě jednotlivých modelů je využit CASE nástroj Enterprise Architect od firmy Sparx.

5 Projektování informačního systémů

Projektování informačního systému, který má být zhotoven, začíná předložením nabídky k řešení situace či přihlášením do výběrového řízení na jeho zhotovení. V takovém případě je na vedení firmy rozhodnutí, jestli je daný projekt lukrativní a zároveň časově a materiálně zhotovitelný. To je zjištěno z předložené specifikace dosavadního řešení. Pokud nějaké řešení existuje, většinou ve firmě jsou již zavedené postupy, které zadavatel nebude chtít měnit, a proto se budou muset brát v úvahu při tvorbě systému.

V případě výhry výběrového řízení či přijetí předložené nabídky začíná proces projektování informačního systému.

5.1 Projektové řízení

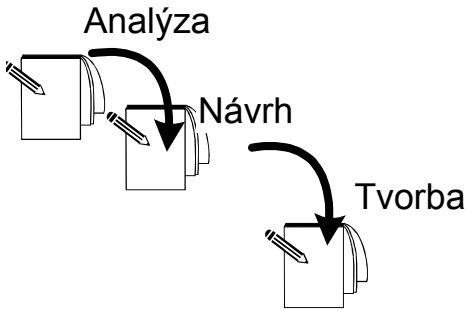
Projektové řízení je nezbytnou součástí při tvorbě systému. Kompetentní osoba, která se řízením projektů zabývá, se označuje jako projektový manažer a nese zodpovědnost za úspěšné či nespěšné dokončení projektu.

Následující výčet představuje důvody, které mohou být podnětem k zamýšlení, proč projekt řídit:

- Definice výsledku projektu
- Dodržování termínů
- Dodržení rozpočtu
- Podchycení veškerých rizik

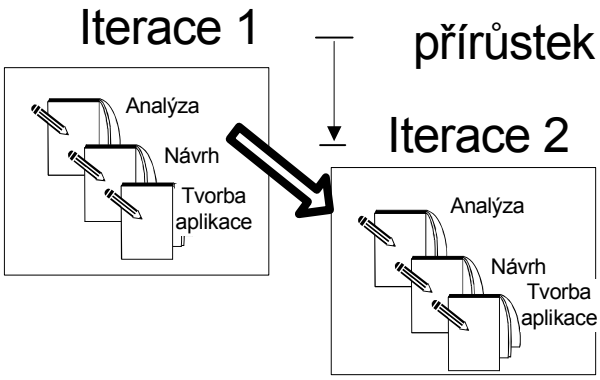
Existuje několik typů řízení:

- Model vodopád (Obrázek 12)
    - Vhodné pro malé projekty
    - Činnosti na sebe navazují (nutné dokončit jednu činnost, aby mohla být zahájena další)



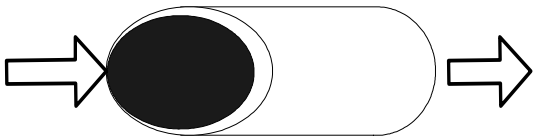
Obrázek 12: Typ řízení projektu – vodopád. Zdroj: [4]

- Iterativní přístup (Obrázek 13)
    - Vhodné pro velké projekty
    - V rámci jedné iterace se projde všemi činnostmi projektu
    - Činnosti na sebe nemusí navazovat, pokud další iterace není na předešlé závislá



Obrázek 13: Typ řízení projektu – přírůstek. Zdroj: [4]

- Žádné řízení označováno též jako extrémní programování (Obrázek 14)
  - Chybí koncepce
  - Chybí jakékoli řízení

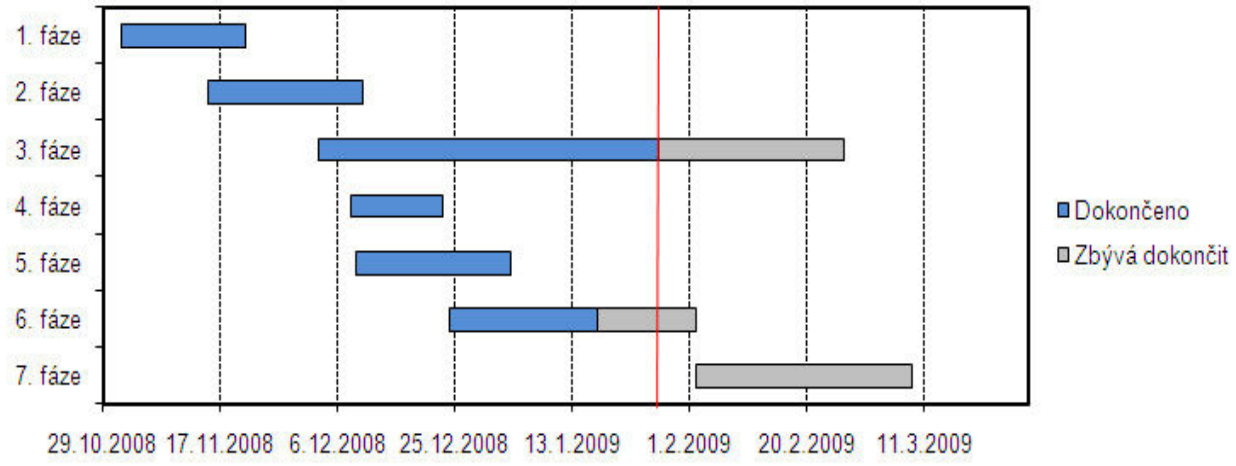


Obrázek 14: Typ řízení projektu – tunel. Zdroj: [4]

V rámci řízení projektu jsou přiřazeny role jednotlivým osobám, které se budou na vývoji informačního systému podílet. Jsou jim kromě rolí přiřazeny kompetence a zodpovědnosti. V různých časových horizontech má každá role jinou váhu podle toho, který postup projektu je aktuální. Pro přehlednost rozvržení jednotlivých fází se vytváří tzv. Ganttův diagram.

Ganttův diagram

Reprezentuje časový harmonogram vývoje informačního systému. Umožňuje znázornit odhadované časové zatížení v jednotlivých fázích projektu a tím možnost odhadnout využití zdrojů lidských i finančních. Vždy v milnících projektu, v době kdy dochází k přechodu mezi jednotlivými fázemi, dochází k prezentování dosavadní práce zadavateli. Obrázek 15 ilustruje příklad vzhledu jednoduchého Ganttova diagramu.



Obrázek 15: Ganttův diagram. Zdroj: vlastní

Dalším možným dělení řízení projektu je podle rozdělení práce určitému počtu pracovníků:

- Metoda příčný řez není příliš efektivní z toho důvodu, že veškerou činnost provádí jedna osoba úplně sama. Dochází ke splývání jednotlivých analýz a modely nejsou striktně oddělené nebo jsou neúplné. Jedná se o nejjednodušší postup i méně finančně náročný, ale existuje zde riziko nezastupitelnosti osoby. Nikdo jiný se v případné aplikaci nemusí vyznat, a tudíž nelze efektivně informační systém či jinou aplikaci upravovat.

- Metoda specializace umožňuje vytváření jednotlivých modelů specializovanými osobami. Analytickému modelu se věnuje analytik, návrhovému modelu se věnuje návrhář a implementaci má na starost programátor. Pokud je vše zdokumentováno je možné jednotlivé odpovídající osoby zastoupit. Nové osoby nemají problém se v modelech vyznat.

Důvody, proč se modeluje, byly zachyceny na začátku této podkapitoly. Je tu ještě jeden důležitý faktor, který se nazývá re-use.

Re-use

Re-use v překladu znovupoužití umožňuje využít již jednou vytvořených modelů ať již analytických či návrhových v podobných případech. Pokud je již jednou vytvořena analýza např. systému řízení vztahů se zákazníky a má se vytvořit další pro jiné odvětví, re-use umožňuje využit modelů, které mají společné rysy a upravit pouze to, co je potřeba. Ušetří se tak náklady při nové tvorbě analýzy i čas. Následuje shrnutí důvodů, proč re-use používat:

- Zamezení opakování prací při vývoji
- Zamezení opakování prací při opravách a změnách
- Přehlednost systému

5.2 Sběr požadavků

Sběr požadavků se považuje za stěžejní část celého projektu. Hlavními aktéry této fáze jsou analytik ze strany zhotovitele a na druhé straně uživatelé vyvíjeného informačního systému či samotný zadavatel. Na požadavcích může ztroskotat celý projekt, pokud analytik správně potřeby zadavatele neidentifikuje. Sběr může probíhat několika způsoby a to:

- Konzultacemi se zadavatelem
- Vyplňováním dotazníků
- Diskuzí neboli tzv. dílnou požadavků

V následujícím textu bude přiblíženo, co se v jednotlivých způsobech sběru požadavků odehrává.

Konzultace se zadavatelem

Tato metoda patří k velmi efektivní metodě oproti dotazníkové metodě. Pokud analytik nepochopí některý požadavek, může požádat zadavatele o upřesnění okamžitě na místě. Toto upřesnění může vést k dalším jinak skrytým požadavkům.

Dotazníky

Jedná se o seznam dotazů (otázek), které mají v odpovědích zachytit požadavky. Nevýhoda spočívá v dodatečném upřesnění některých nepřesně formulovaných odpovědích. Dotazníky se často kombinují s metodou konzultace se zadavatelem. Jedná se o doplněk. Do dotazníků se mohou zapsat otázky, které byly zodpovězeny zadavatelem. Tento dotazník může pak být předložen ostatním zainteresovaným osobám ke kontrole, zda jsou požadavky na systém pochopeny správně.

Dílna požadavků

Tato metoda patří k nejefektivnější. Jedná se o spontánní diskusi, kde veškeré nápady jsou považovány za dobré a zapíší se. Požadavky jsou určeny členy týmu a opět se zaznamenávají.

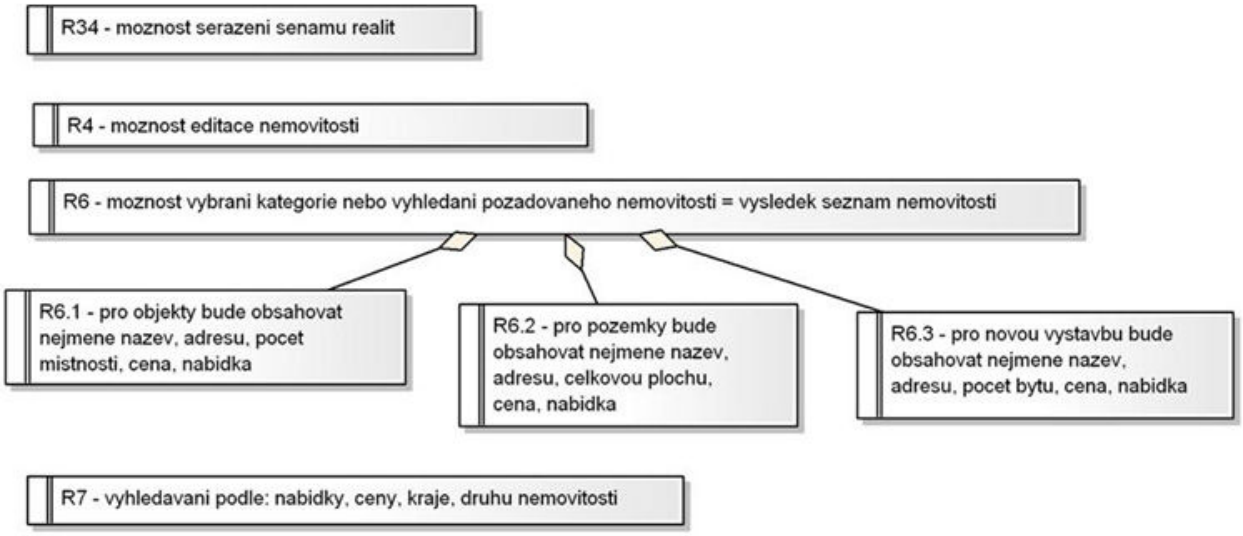
Obecně se požadavky dělí na funkční, ty které určují chování samotného systému, a nefunkční, tyto požadavky určují omezení vytvářeného systému. Dále se mohou požadavky dělit podle kategorie, do které patří. Priorita jednotlivých požadavků by měl být určena zadavatelem a podle této priority se řadí požadavky při modelování i při implementaci.

Dělení požadavků pro vytvářený informační systém, vypadá následovně:

- Funkční
  - Uživatel
    - Registrace nového uživatele (RK) pomocí formuláře
    - Možnost vložení nemovitosti do systému
    - Přihlášení do systému přihlašovacím údaji (e-mail a heslo)
    - Vyhledávání pomocí interaktivní mapy ČR
  - Nemovitost
    - Vyhledávání podle různých kritérií – výsledkem seznam odpovídajících nemovitostí
    - Zařazení do kategorií podle druhu nemovitosti, kraje a okresu
    - Vyhledávání a prohlížení podle kritérií: druh nemovitosti, kraj, okres
    - Identifikovatelné – jednoznačně přiřazené číslo nemovitosti
    - Stavby obsahují určité položky (např. počet místností, patro, …)
    - Pozemek obsahuje položky jako cena za m krychlový, apod.
    - Poklepáním na položku seznamu se zobrazí detailní popis dané nemovitosti včetně obrázků, které bude možné zvětšit
    - Detailní obrazovka umožňuje kontaktování inzerující osoby
- Nefunkční
  - Dostupnost
    - 24 hodin denně
  - Kapacita
    - Řádově desetitisíce nemovitostí v informačním systému
    - Počet nemovitostí uživatele „host“ omezen na jedinou
    - Velikost vložených obrázku max. 400kB
    - Počet vložených obrázku omezen na 5 u registrovaného uživatele a žádný u jednořádkové inzerce

- Shoda se standardy
  - Model-View-Controller
  - Program v PHP s využitím Zend Frameworku
- Výkon
  - Neomezený denní počet transakcí
  - Odezva na požadavek do 5 vteřin v 95 % použití

Při každé další konzultaci se zadavatelem se může narazit na další požadavky a to dokonce z části ve fázi konstrukce. Veškeré požadavky, které jsou zachyceny, vyplynuly v rámci konzultací se zadavatelem. Zbylé dvě metody nebyly použity. Výčet požadavků, který je zde prezentován, není kompletní. Kompletní sadu všech požadavků si lze prohlédnout v příloze B. Následuje Obrázek 16 zjištěných a zaznamenaných funkčních požadavků pro nemovitosti.



Obrázek 16: Funkční požadavky. Zdroj: vlastní

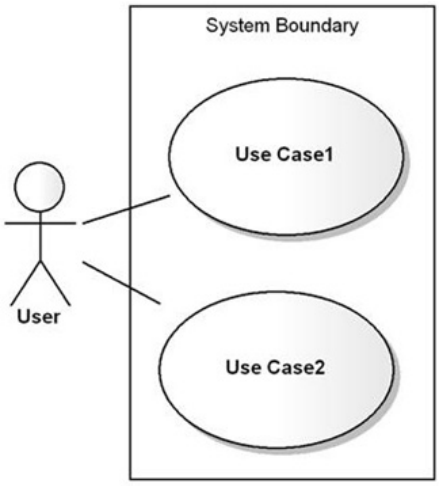
Sledování požadavků je realizováno CASE nástrojem pomocí vztahové matice. V rámci vztahové matice jsou přiřazeny funkční požadavky k případům užití. Mezi funkčními požadavky a odpovídajícím případem užití může existovat i více relací. Může nastat případ, že jeden požadavek je vyjádřen několika případy užití nebo obráceně jeden případ užití může být pokryt více požadavky. Neměl by nastat případ, kdy některý případ užití nelze přiřadit požadavku a naopak.

5.3 Use Case diagramy

Use Case diagramy, často překládány jako diagramy případu užití, jsou elementy jazyka UML, které popisují, jak uživatel navrhovaného systému spolupracuje se systémem tak, aby splnil samotnou část práce. Popisuje a naznačuje jednoduchou spolupráci v průběhu doby, která má význam pro koncového uživatele, kterým může být osoba, stroj nebo jiný systém.

Případy užití mají své požadavky a omezení, která popisují základní vlastnosti a pravidla, která musí dodržovat. Kromě toho můžou být spojeny se sekvenčním diagramem, který ilustruje chování během času. Tento diagram znázorňuje: *„Kdo udělá co komu a kdy“*.

Mezi aktéry zahrnujeme osoby, jiné systémy nebo i čas pracující s daným systémem. Podoba diagramu případu užití je zobrazena na následujícím Obrázek 17.



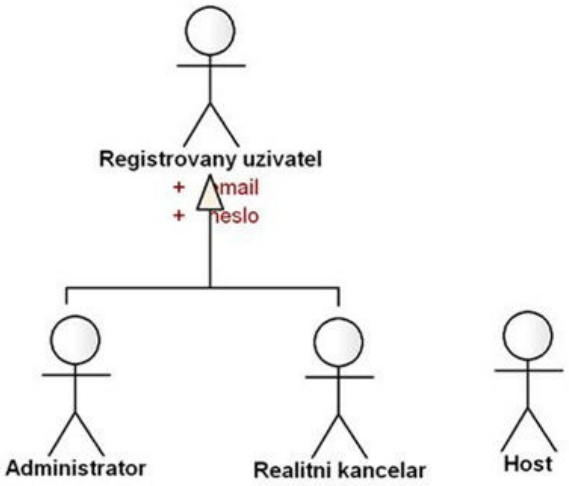
Obrázek 17: Struktura Use case diagramu. Zdroj: vlastní

Postava představuje aktéra. Obdélník, s názvem *System Boundary*, je ohraničení systému a elipsy představují samotný případ užití, který by měl ve svém názvu obsahovat sloveso. Posledním prvkem v rámci diagramu je komunikační relace aktéra s případem užití.

V případě navrhovaného informačního systému pro správu realit byli identifikováni následující aktéři:

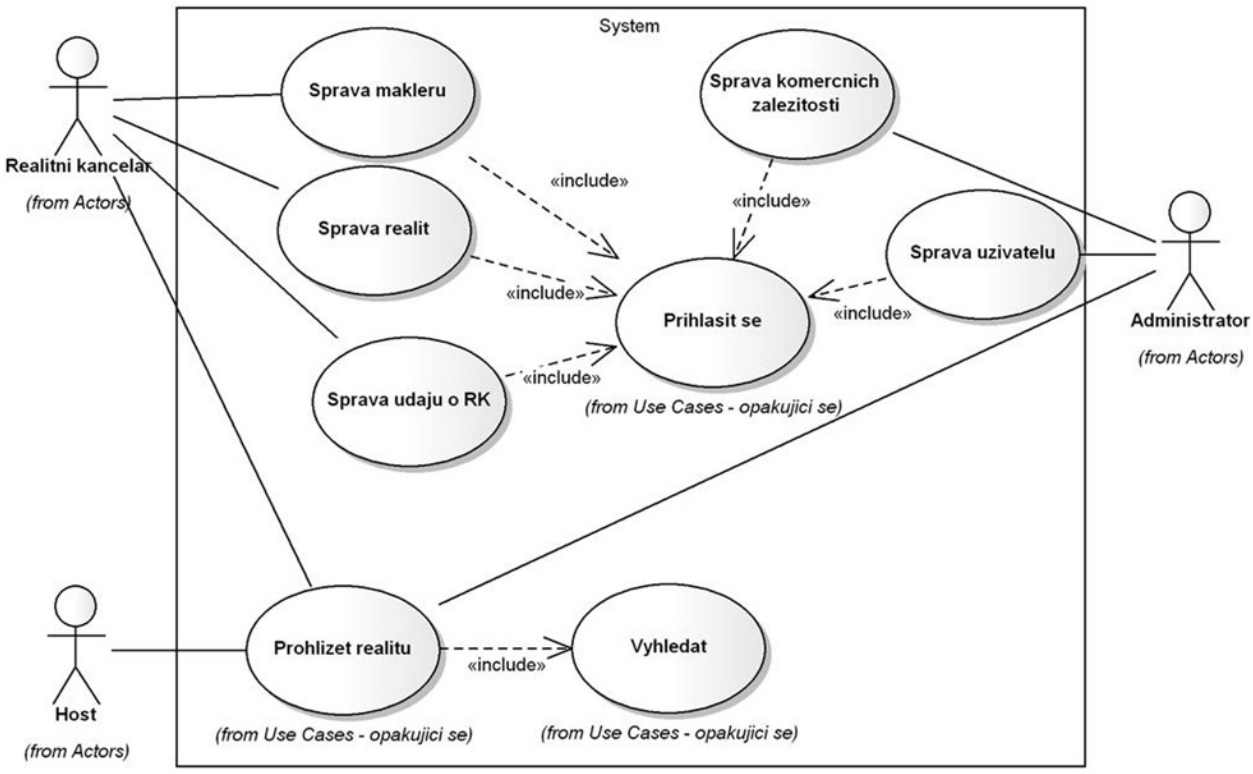
1. Administrátor – osoba spravující informační systém
2. Realitní kancelář – aktér využívající informační systém má přístupová prává
3. Host – tzv. návštěvník informačního systému

Obrázek 18 zachycuje vzájemný vztah mezi jednotlivými uživateli systému. Všichni aktéři jsou uživateli systému. Administrátor a realitní kancelář jsou registrovanými uživateli, kteří se mohou přihlásit do svých administračních center (neveřejné části informačního systému), zatímco host má přístup pouze do veřejné části informačního systému.



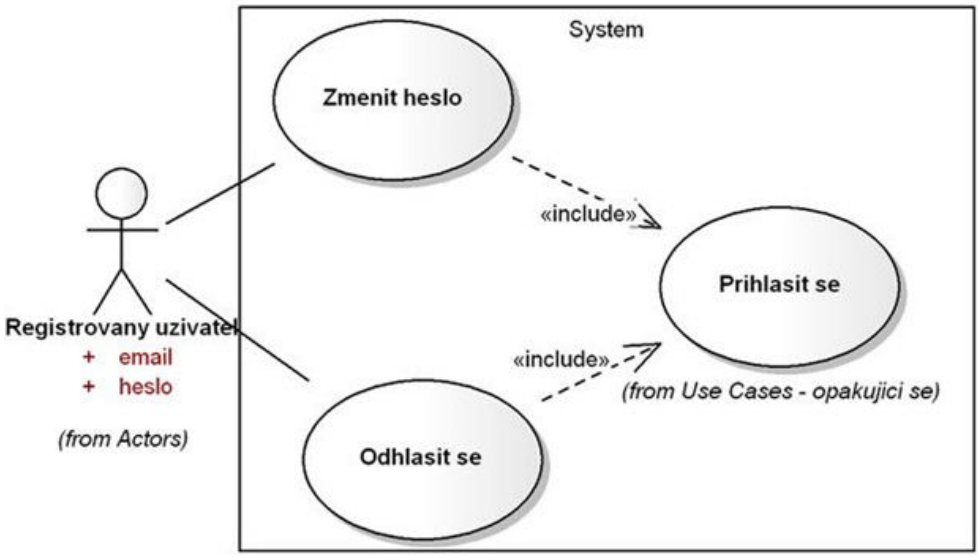
Obrázek 18: Uživatelé systému. Zdroj: vlastní

Jako hlavní diagram případu užití by měl být pořízený tzv. celkový pohled na systém. Případy užití v rámci tohoto uceleného pohledu jsou následně rozpracovány do dílčích diagramů případu užití. Obrázek 19 zachycuje již zmíněný celkový pohled na systém.



Obrázek 19: Celkový pohled na informační systém. Zdroj: vlastní

Případ užití *Správa makléřů* může být dále rozložen na podrobnější případy užití, jako je např. *Editace makléře*, *Přidání makléře* apod. Obrázek 20, který následuje, znázorňuje další možné funkcionality systému, jako jsou *Změna hesla* a *Odhlášení*, tyto případy užití jsou podmíněny *Přihlášením do systému*.



Obrázek 20: Základní operace uživatele. Zdroj: vlastní

Případy užití v sobě zahrnují i tzv. scénáře, které naznačují pracovní tok událostí, které vedou ke konečnému stavu či výsledku. Scénáře mají své základní dělení na scénář hlavní a scénář vedlejší (alternativní).

- Hlavní – popisují hlavní tok událostí. Tento scénář je pouze jeden jediný.
- Alternativní – jedná se o odchýlení od hlavního toku událostí. Když nastane nějaká výjimka, spustí se jiný náhradní postup. Nikdy se nevrací k hlavnímu scénáři. V rámci jednoho hlavního scénáře se může vyskytnout mnoho alternativních scénářů, jejich počet by však měl redukovat na nezbytné minimum. Vybírají se podstatné vedlejší scénáře, nebo pouze jeden hlavní alternativní, který se od ostatních jen minimálně liší. Do poznámky jsou poté zaznamenány rozdíly od dalších možných, které se již neuvedou.

Celkový počet scénářů by měl být takový, aby bylo pochopeno, jak systém pracuje a jak se chová, než zachytit kompletní model všech případů užití.

Tabulka 1 popisuje hlavní scénář v rámci navrhovaného systému pro případ užití *Registrace*. Tento hlavní scénář v sobě obsahuje odkazy na dva alternativní scénáře, které jsou zachyceny v následujících tabulkách: Tabulka 2 a Tabulka 3.

Tabulka 1: Hlavní scénář. Zdroj: vlastní

Případ užití: Registrace
Stručný popis: Registrování nového uživatele - realitní kancelář
Hlavní aktéři: Host
Vedlejší aktéři: žádní
Vstupní podmínky: žádná
Výstupní podmínky: oznámení o úspěšnosti registrace
Hlavní scénář 1. Host vyplní údaje o realitní kanceláři 2. Pokud chce nahrát logo 2.1 Host uloží cestu k logu 3. Jinak 3.1 Nemusí použít 4. Jestliže má RK jiné fakturační údaje 4.1 Vyplní jiné fakturační údaje 5. Host vybere kraje pro zaslání poptávek 6. Opíše kontrolní kód 7. Dokud nejsou vyplněny povinné údaje 7.1 Systém vyzývá k vyplnění neplatných nebo nevyplněných údajů 7.2 Systém ověří údaje 8. Systém uživatele zaregistruje
Alternativní scénář Neplatná e-mailová adresa Kód není správný

Tabulka 2: Alternativní scénář 1. Zdroj: vlastní

Alternativní scénář: Neplatná e-mailová adresa
Stručný popis: Špatně zadaný formát e-mailové adresy
Hlavní aktéři: Host
Vedlejší aktéři: žádní
Vstupní podmínky: žádná
Výstupní podmínky: oznámení o špatně zadané e-mailové adrese
Alternativní scénář 1. Alternativní scénář začíná krokem 7.2 hlavního scénáře. 2. Systém informuje hosta, že zadaná e-mailová adresa není platná.

Tabulka 3: Alternativní scénář 2. Zdroj: vlastní

Alternativní scénář: Kód není správný
Stručný popis: špatně opsaný kontrolní kód
Hlavní aktéři: Host
Vedlejší aktéři: žádní
Vstupní podmínky: žádná
Výstupní podmínky: oznámení o špatném kódu
Alternativní scénář 1. Alternativní scénář začíná krokem 7.2 hlavního scénáře. 2. Systém informuje hosta, že ověřovací kód neodpovídá předloze. 3. Systém zobrazí jiný kód.

Veškeré diagramy umožňují lepší pochopení struktury a chování navrhovaného systému pro zadavatele. Jeden obrázek může vyjádřit víc než několik slov.

5.4 Analytický model

Analytický model vzniká v rámci pracovní činnosti analýza. Odpovědnou osobou v tomto období je systémový analytik, který již figuroval při sběru požadavků u zadavatele. Jelikož zná problémovou doménu z popisu zadavatele v rámci konzultací, dokáže nalézt základní analytické třídy. V rámci analýzy se může vytvářet diagram aktivit.

Analytický model vychází z požadavků, modeluje hlavní a nejdůležitější chování systému a je na vysoké úrovni abstrakce. V tomto modelu se nezachycují detaily. Při tvorbě analytického modelu je třeba si odpovídat na otázku: *„Co má informační systém umět nebo dělat?“*. V této fázi projektování systému se využívá již dříve zmíněných diagramů případu užití.

Grafický vzhled výsledného informačního systému by měl být znám již v období analýzy, aby se v rámci návrhu mohl nechat ještě upravit, pokud by bylo potřeba. Vzhled může být dodán na základě smlouvy se subdodavateli, může být vytvořen interním grafikem, nebo jej může zadavatel dodat sám.

Při tvorbě analytického modelu lze využít stereotypy k namodelování chování aplikace, která je založena na modelu Model-View-Controller.

5.4.1 Analýza MVC aplikace

Elementy stereotypů jako jsou boundary, control a entity mohou být využity k znázornění uživatelského rozhraní, controllerů a databázových položek.

Enterprise Architect má v sobě zabudované stereotypy, které lze využít během analýzy MVC aplikace. Grafické znázornění těchto stereotypů je odlišné než obvyklé znázornění pomocí následujícího zápisu <<název>>. Obrázek 21 znázorňuje tyto stereotypy.

Obrázek 21: Stereotypy MVC. Zdroj: vlastní

Objekt entity není nic víc než informace nebo data, které vyhledávají objekty Boundary. Entity mohou být databázové tabulky, soubory apod. Je ekvivalentem vrstvy Model v rámci modelu MVC.

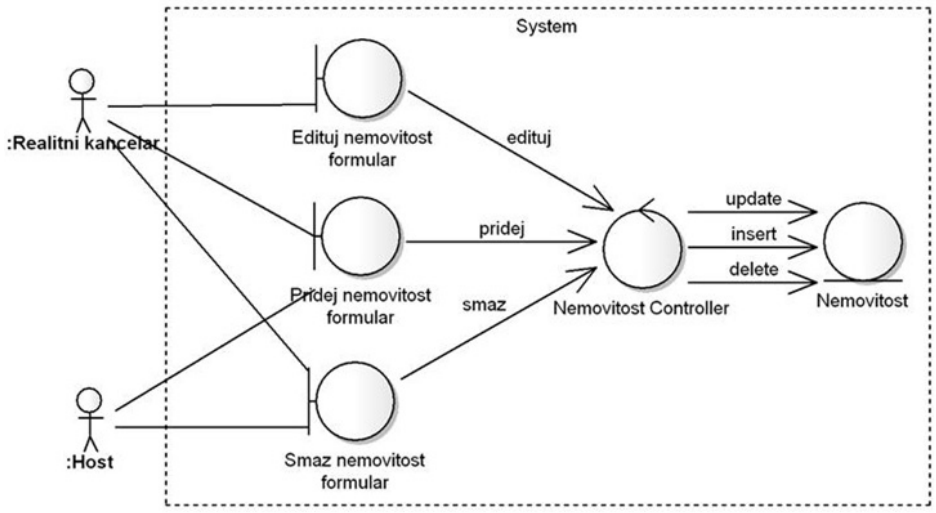
Objekt boundary je objekt, se kterým komunikuje aktér v rámci navrhovaného systému. Mezi tyto objekty patří dialogy, formuláře, menu, okna a obrazovky a jiná uživatelská rozhraní v systému. Zastupují vrstvu View v rámci modelu MVC.

Control je řídící objekt. Jedná se o pravidla, která filtrují data, aby mohla být prezentována uživateli, jež si vyžádal. Reprezentuje vrstvu Controller v rámci modelu MVC.

Následující čtyři základní pravidla určují, jakým způsobem dochází ke komunikaci mezi objekty v rámci modelování analytického modelu vycházejícího z diagramů případu užití:

- Aktéři mohou komunikovat pouze s objektem Boundary neboli pomocí uživatelského rozhraní.
- Boundary objekty mohou komunikovat pouze s objekty Control a uživateli.
- Entity objekty by měly komunikovat pouze s objekty Contol.
- Objekty Control mohou komunikovat s objekty Boundary, Entity a jinými objekty typu Control, ale nemohou komunikovat s aktéry.

Obrázek 22 názorně ukazuje, jak se tyto stereotypy v rámci analýzy mohou využít. Více těchto diagramů lze nalézt v příloze B.



Obrázek 22: Analýza pomocí MVC. Zdroj: vlastní

5.4.2 Analytické třídy

K identifikaci analytických tříd můžeme dospět několika způsoby. Nejvyužívanější jsou následující dva a to metoda štítků CRC a metoda podstatných jmen a sloves.

Metoda podstatných jmen a sloves

Metoda analýzy podstatných jmen a sloves je založena na existenci textového popisu navrhovaného systému. Jde o specifikaci, která obsahuje údaje potřebné k pochopení, co má systém umět. Z této specifikace můžou být identifikovány i požadavky. Pokud specifikace

existuje, označí se slovesa jako adepti na metody a podstatná jména jako adepti na třídy nebo atributy.

Metoda CRC štítků

Metoda CRC štítků zapojuje do hledání analytických tříd i uživatele. Tabulka 4 názorně popisuje štítek, který je rozdělen na tři části.

První fází při této metodě je shromažďování informací, kdy uživatelé mají pojmenovat předměty v jejich problémové doméně. Dalším krokem je přiřazení odpovědností těmto předmětům. Posledním krokem v první fázi je označení tříd, které by mohly společně navzájem spolupracovat. V druhé fázi se analytici a odborníci z dané domény rozhodnou, které z lístečků budou třídy a které jejich atributy.

Tabulka 4: Rozvržení štítku. Zdroj: vlastní

Název třídy (**C**lass)	
Odpovědnosti (**R**esponsibilities)	Spolupracovníci (**C**ollaborators)

Pokud se ve výsledku analytické třídy shodují, může se jednat o náznak, že se jde správnou cestou. Pokud se výsledky odlišují jen velmi málo lze tyto dva výsledky posoudit jako celek a nepotřebné třídy vyloučit. Pokud ovšem jsou úplně jiné, měli bychom raději ještě jednou zamyslet.

Dalšími zdrojem analytických tříd můžou být fyzické objekty v rámci reálného světa problémové domény (nemovitost, realitní kancelář, …). Modelování vztahů v analytickém modelu mezi třídami je jednoduché. Analytik nebere v úvahu, jak třídy spolu souvisí, ale jednoduše znázorňuje pouze spojení mezi třídami. Jak je vidět na Obrázek 23.

Základní podoba analytické třídy může být shrnuta v následujících bodech:

- Název je povinný
- Atributy jsou povinné, ale stačí uvést pouze množinu nejdůležitějších atributů
- Operace je obecná odpovědnost, nemusí být uvedeny parametry a návratové typy
- Typ viditelnosti není určen

Dobrá analytická třída se vyznačuje následujícími body:

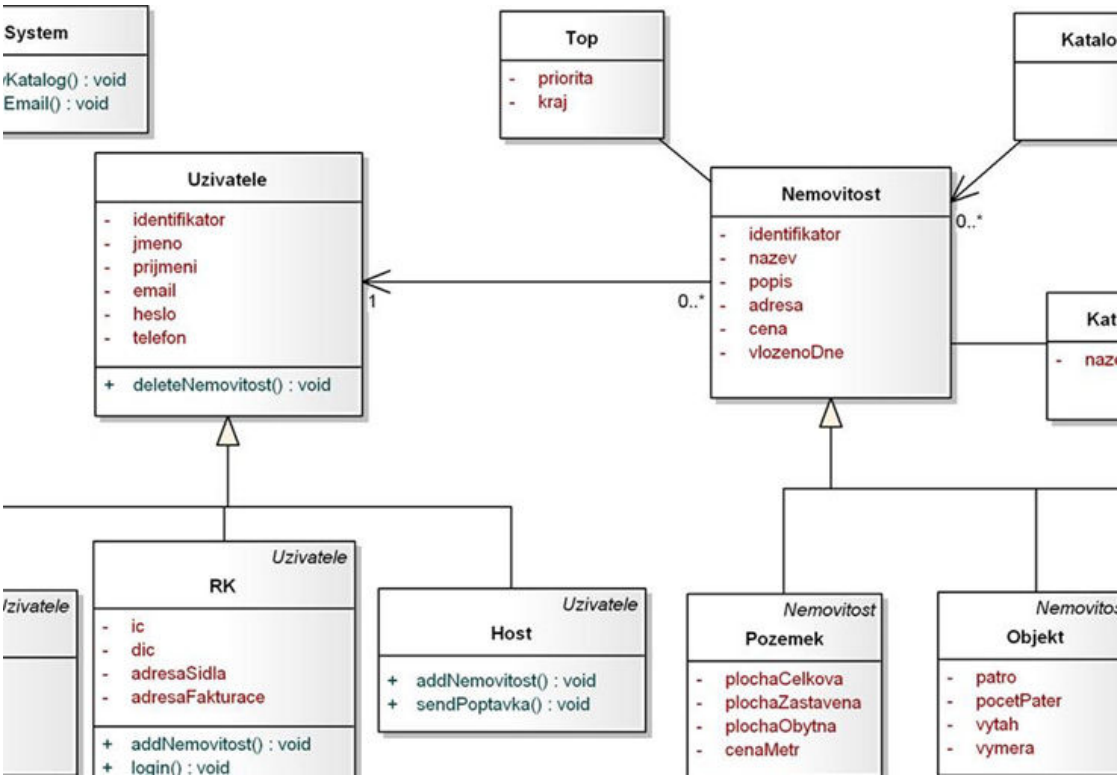
- Výstižný název
- Obsahuje množinu odpovědností (3-5)
- Obsahuje minimum vazeb na ostatní třídy

Obrázek 23 zachycuje výřez z analytických tříd navrhovaného systému. Kompletní analytický návrh je obsažen v příloze B.

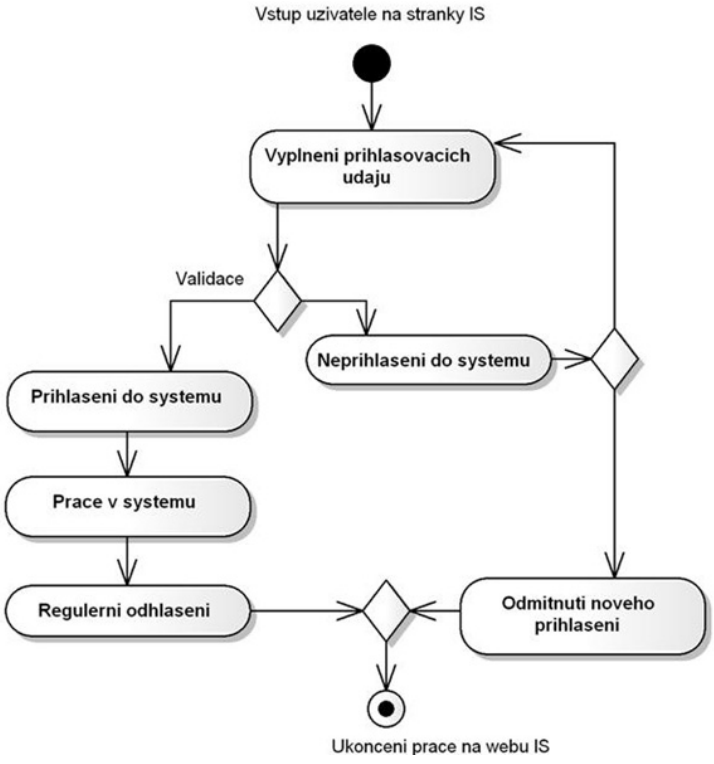
5.4.3 Diagramy aktivit

Diagram aktivit, jak již bylo naznačeno v jedné z předchozích kapitol, slouží k modelování chování v systému. Aktivity mohou probíhat buď sériově, nebo paralelně. Diagram začíná počátečním bodem a je ukončen jedním nebo několik koncovými body, mezi kterými jsou aktivity.

Obrázek 24 znázorňuje dynamiku při přihlašování do navrhovaného systému. Další diagramy aktivit jsou k nahlédnutí v příloze B.



Obrázek 23: Analytické třídy – výřez. Zdroj: vlastní



Obrázek 24: Přihlášení. Zdroj: vlastní

5.5 Návrhový a datový model

Za výsledné modely v rámci návrhu je odpovědný návrhář. Tuto roli může zastupovat v jistém úhlu pohledu sám analytik, jelikož zná detaily systému a nemusí je vysvětlovat další osobě. V rámci vytváření návrhu lze vytvářet sekvenční a komunikační diagramy znázorňující interakce.

5.5.1 Návrhový model

Návrhový model vychází z modelu analytického. Avšak již je na takové úrovni specifikace, že podle něj lze implementovat. Nástroje CASE podporují celou řadu implementačních jazyků, do kterých může být výsledný návrhový model vygenerován. Možnosti nastavení implementačního jazyka v rámci Entreprise Architectu jsou následující:

- C,
- C#,
- C++,
- Delphi,
- Java,
- PHP,
- Python,
- Visual Basic.

Vzhledem k velkému množství implementačních jazyků a jejich různým specifickým potřebám, by se nejprve měl vytvořit abstraktní návrhový model, který je zcela zproštěn od jakéhokoli kód. Tento abstraktní model je také základem již zmiňovaného re-use, kdy navrhovaný systém stejného typu avšak jiných požadavků na implementační jazyk je zcela stejný, a tudíž model lze využit v obou dvou případech. Na tento návrhový model by měl navazovat implementační model, který již zahrnuje specifika zvoleného implementačního jazyka. K tomuto závěru jsem došla po zjištění, že pokud chci v návrhovém modelu využívat tak zvané gettery resp. settery, funkce získávající resp. nastavující atributy, nelze to provést ve všech implementačních jazycích stejně. Například C# (CSharp) využívá pro nastavování a získávání hodnot datových složek tzv. vlastnosti ale jiným jazykům se musí explicitně vytvořit tyto funkce.

Mohlo by se dlouze polemizovat a správnosti tohoto řešení. Každý má svůj subjektivní pohled na věc. Z pohledu na re-use se toto může zdát jako přijatelné řešení.

5.5.2 Návrhové třídy

Návrhové třídy jsou třídy, které jsou více specifikovány než třídy analytické. Návrhové třídy vychází z dvou oblastí:

1. Doména problému zahrnuje upřesnění analytických tříd, které vzešly z analytického modelu.
2. Doména řešení obsahuje knihovny již existujících komponent a tříd.

Při tvorbě tohoto modelu je důležité umět si odpovědět na otázku: „Jak to má informační systém vykonat“. Operace z analytických tříd jsou převedeny na kompletní sadu metod v návrhových třídách. Tyto metody mají specifikované návratové typy a seznam parametrů. Atributy z analytických tříd jsou upřesněny o typ, viditelnost (public, private, protected) a

nepovinně se můžou uvést i implicitní hodnoty. Může nastat případ, kdy třída z analytického modelu se v rámci modelu návrhového rozpadne do dvou a více návrhových tříd.

Správně vypadající návrhová třída by měla disponovat následujícími vlastnostmi:

- úplnost – třída musí poskytovat vše, co se po ní požaduje,
- jednoduchost – třída poskytuje jednoduché nedělitelné služby,
- bez těsných vazeb – třída by měla komunikovat pouze se třídami, které nezbytně potřebuje,
- soudržnost – třída obsahuje pouze metody určené pro její účely.

Návrhová třída by měla obsahovat přibližně tři až pět metod. Neměly by vznikat univerzální všeobsahující třídy. Pokud je míra úplnosti návrhových tříd na vysoké úrovni, lze využít generování zdrojového kódu přímo z CASE nástroje. Nevýhoda generování tříd spočívá právě v detailní specifikaci, jelikož generátor si sám neumí domyslet nedostatečné specifikace. Další možností implementace návrhových tříd je samotnými programátory, kteří si dokážou některé detaily domyslet, ale i tak by měl být návrh na co nejvyšší úrovni.

Mezi jednotlivými třídami se v rámci návrhu upřesňují relace z analytického modelu. V rámci analýzy se detailně nezkoumá, jaký má třída vztah k druhé třídě. Vztah je určen jako asociace. Tyto asociace musejí být upřesněny, aby se dal návrhový model implementovat, protože např. nelze implementovat obousměrné asociace, asociace typu M:N. Upřesnění relací obsahuje následující kroky:

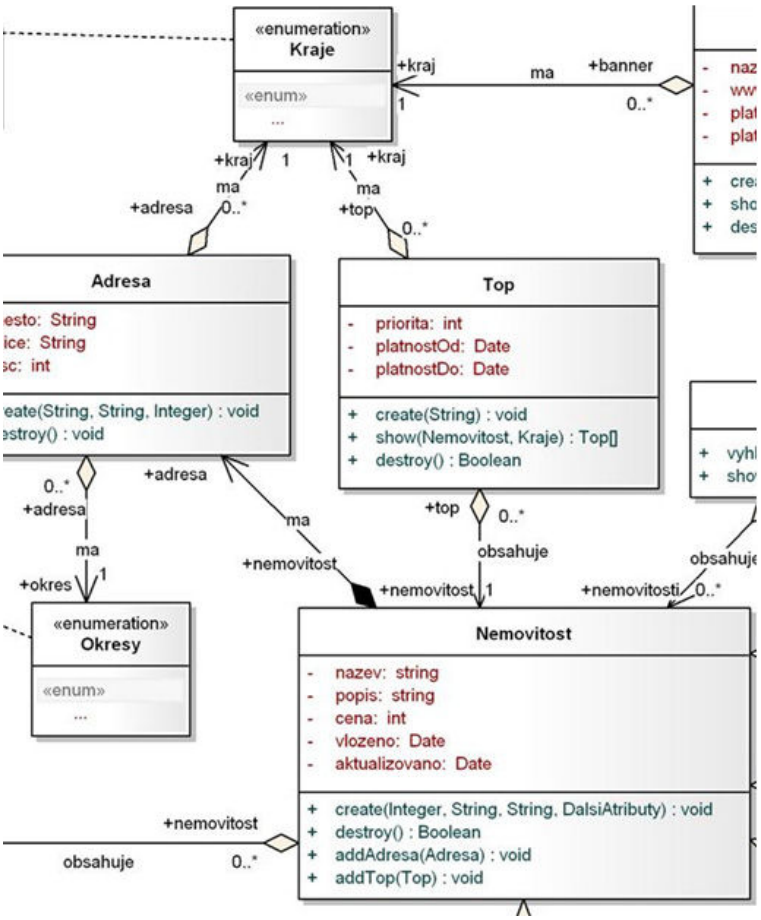
- určení násobnosti a názvů rolí,
- určení stran části a celku (agregace, kompozice, asociace neexistuje),
- určení průchodnosti, která musí být jednosměrná.

Pokud se narazí na třídy, které mezi sebou mají asociaci typu M:N, je nutno vytvořit tzv. asociační třídu. Tato asociační třída má s první třídou vztah typu M:1 a s druhou třídou vztah 1:N. V případě obou směrné asociace, se musí vytvořit dvě relace mezi třídami. Tyto dvě relace jsou průchodné vždy pouze jedním směrem.

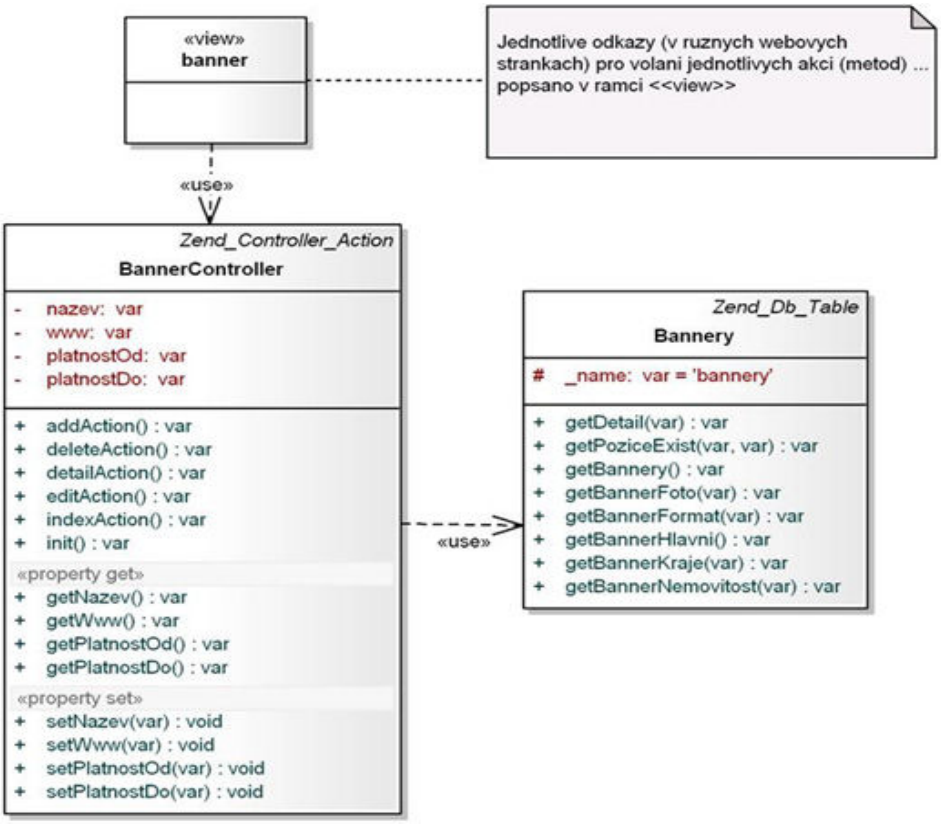
V rámci návrhových tříd návrhového modelu tvořeného informačního systému jsou zobrazeny i využívané výčtové konstanty. Namodelování vztahu mezi třídou a výčtovým typem není potřeba znázorňovat, opět záleží na úlohu pohledu daného návrháře.

Obrázek 25 je výřezem návrhových tříd v rámci navrhovaného systému. Celý návrhový model je k dispozici v rámci přílohy B.

Aby mohl být abstraktní návrhový model naimplementován dle specifik daného požadovaného jazyka, měl by být vytvořen implementační model navrhovaného systému. Část implementačního modelu je znázorněna v rámci Obrázek 26. Kompletní implementační model je opět k dispozici v příloze B.



Obrázek 25: Návrhový model - výřez. Zdroj: vlastní



Obrázek 26: Implementační model - výběr. Zdroj: vlastní

5.5.3 Návrhové vzory

Návrhové vzory jsou nástrojem pro vytvoření dobrého návrhu. Vzory zachycují řešení opakujících se problémů. Vzor lze považovat za obecný návod řešení problémů, které se často vyskytují v různých obdobách.

Pokud je tvořen systém se standardní situací pro použití vzoru, je nutno odhalit, jaký vzor by měl být použit. Často se stává, že si člověk neuvědomuje, že by mohl nějaký vzor použít, pokud vzory a jejich konkrétní situaci k použití dobře nezná. Každý vzor ve své specifikaci obsahuje tzv. smysl, což je jasné vystihnutí problému a řešení pomocí daného vzoru. Využívá se k rychlému rozpoznání použitelnosti vzoru na danou situaci a k rychlému rozpoznání vzoru.

Existuje celá řada návrhových vzorů. Pro nastínění problematiky budou popsány pouze ty nejpoužívanější, jelikož popsání všech exitujících vzorů je obsáhlá oblast a dala by na samostatnou kapitolu. Existují tři druhy návrhových vzorů:

- vzory tvorby objektů (*creational patterns*)
- vzory struktur (*structural patterns*)
- vzory chování (*behavioral patterns*)

Z první skupiny vzorů zaměřených na tvorbu objektů je nejznámější *Singleton*. Jedná se o vzor, který zabezpečuje, aby třída měla právě jednu instanci, která je viditelná globálně.

Druhou oblastí vzorů jsou vzory struktury. *Decorator* přidává dynamicky další funkcionalitu objektu, což může být alternativa k dědění. *Facade* zavádí jeden nebo více interfaců pro přístup k podsystému.

Poslední skupinou vzorů jsou vzory řešení chování. Zástupcem této skupiny je *Observer*, který umožňuje sledovat změny u objektů tím, že pokud objekt změní svůj stav, následuje reakce na změnu ostatními objekty.

V rámci této práce nebyly identifikovány žádné situace, ve kterých by se měl využít návrhový vzor. Avšak zkušenější návrhář by mohl oponovat a nějaký vzor doporučit k zahrnutí do návrhového modelu.

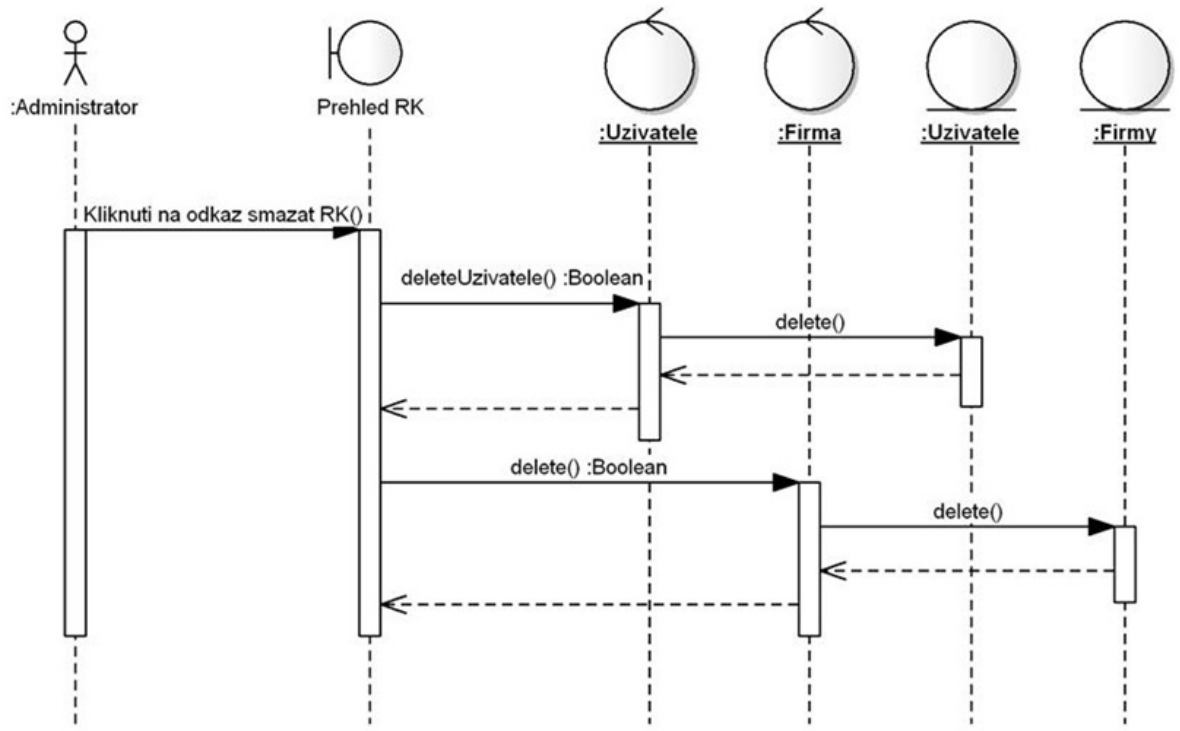
5.5.4 Sekvenční diagramy

Sekvenční diagramy znázorňují vzájemnou spolupráci mezi objekty v systému. Tato spolupráce je popisována v závislosti na čase.

Objekty jsou umísťovány zleva doprava. Od každého objektu míří od shora dolů přerušovaná čára, která znázorňuje dobu života objektu neboli plynoucí čas. V průběhu čáry života se objevuje tzv. aktivace, která znázorňuje čas a práci na operaci, která je po objektu vyžadována.

Zprávy mezi objekty jsou znázorněny šipkou. Pokud objekt vyvolá operace jiného objektu lze říci, že odeslal zprávu. Důraz je kladen na pořadí přenosu zpráv. Zprávy, které se mohou zasílat, jsou:

- synchronní, jež čeká na odpověď a nemůže se pokračovat v činnosti, dokud není odpověď doručena,
- asynchronní, kdy odpověď není vyžadována a činnost pokračuje,
- jednoduché, které pouze předávají řízení.



Obrázek 27: Sekvenční diagram

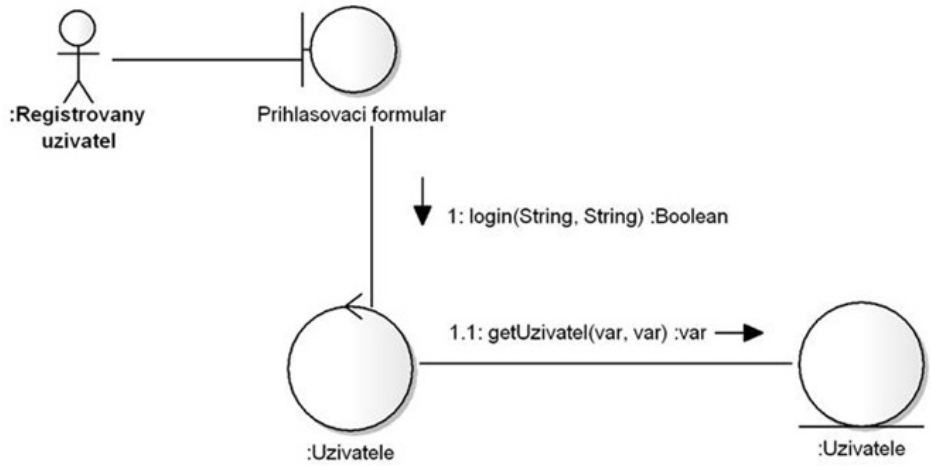
Elementy v rámci sekvenčního diagramu (Obrázek 27) pro navrhovaný informační systém jsou:

- Aktér – instance aktéra v čase (*Administrátor*),
- Čára života – doba existence objektu,
- Boundary – reprezentuje uživatelské rozhraní (*Přehled RK*),
- Entity – trvalý prvek typicky implementován jako tabulka databáze,
- Control – činná komponenta, která kontroluje jak, co a kdy je provedeno.

5.5.5 Komunikační diagramy

Komunikační diagram, původně nazýván diagram spolupráce, je obdoba sekvenčního diagramu. Jak komunikační, tak sekvenční diagram obsahují stejné informace, proto je možné je mezi sebou převádět. Komunikační digram zdůrazňuje uspořádání objektů, které mezi sebou komunikují. Zachycuje zprávy stejně jako sekvenční diagram.

Tento diagram upřednostňuje uspořádání a kontext vzájemně spolupracujících objektů, zatímco sekvenční diagram upřednostňuje pořadí jednotlivých událostí. Stručně a výstižně řečeno: Sekvenční diagram znázorňuje komunikaci v čas a komunikační diagram znázorňuje, co se děje v prostoru. V rámci tohoto diagramu se zprávy znázorňují pouze jednou, i kdyby se vyskytly několikrát, to činní diagram přehledným. U asociační čáry se vytvoří šipka s popisem zprávy, závorkou s argumenty a číslem, v jakém pořadí půjde na řadu. Číslo se zapisuje před název zprávy a je odděleno dvojtečkou.



Obrázek 28: Komunikační diagram

V tomto diagramu (Obrázek 28) stejně jako v diagramu sekvenčním se vyskytuje element Aktér, Boundary, Control a Entity.

5.5.6 Datový model

Cílem návrhu datového modelu je vytvořit takovou datovou strukturu pro informační systém, která bude uchovávat potřebná data ve zvolené databázi. Databáze se stala neoddělitelnou součástí uchovávání dat u informačních systémů, jelikož se musí zpracovávat velké množství data a využívání souborů není výhodné. Existují dva pohledy na datový model:

- Logický datový model neboli konceptuální datový model, je takový model, který nebere v úvahu konkrétní relační databázi.
- Fyzický datový model zahrnuje v sobě již konkrétní relační databázi.

Kroky analýzy datového modelu

1. Identifikace entit – objektů shromažďujících související data
2. Identifikace klíčů – identifikátor entity
3. Určení vazeb mezi tabulkami a jejich kardinalit
4. Tvorba datového modelu

Správné stanovení a navržení klíčů je v rámci datové analýzy velice nezbytné pro databázový návrh. Rychlost zpracovávání dat v rámci navrhovaného informačního systému je závislá na správně určených klíčích. Klíče jsou typu:

- Primární – nezaměnitelný klíč, v rámci entity jedinečný
- Alternativní – jednoznačný klíč, který není zvolen jako primární
- Cizí - atribut jedné entity (*DETAIL*), který se objevuje v jiné entitě (*MASTER*) jako klíč primární
- Unikátní - v rámci entity jsou jednoznačné, podmínka primárního klíče

Tvoření datového modelu

Datový model by měl být vytvořen na základě diagramu tříd z fáze návrhu. Využívá se několika mapování pro zjištění tabulek.

- Mapování tříd na tabulky
  Atributy třídy se stanou sloupci tabulky. Řádky tabulky odpovídají instancím objektů. Nesmí se zapomenout na konverzi datových typů, v různých databázích se lze setkat s různým pojmenováním datových typů. Pokud v rámci třídy neexistuje atribut jednoznačné identifikace, je v tabulce vytvořen, nejčastěji má formu celočíselného datového typu s pojmenováním *id*.
- Mapování asociací a agregací
  V rámci datového modelu se můžeme setkat s následujícími typy asociace:
  1:1 – tyto tabulky bývají sjednoceny v jedinou
  1:N – nejčastější typ asociace. Princip spočívá ve vytvoření cizího klíče v jedné tabulce odkazující na řádek tabulky, kde je tento klíč, klíčem primárním.
  M:N – tato asociace vede k vytvoření asociační tabulky, která s oběma původními tabulkami má asociační vztah 1:N.
  Agregace se v datovém modelu modeluje stejně jako asociace.
- Mapování dědičnosti
  Je jednou z nejsložitějších relací z pohledu mapování. Existují tři druhy:
  - Mapování 1:1 představuje nejjednodušší možnost mapování dědičnosti. Každá třída je mapovaná do samostatné tabulky, které mají stejný primární klíč.
  - Zahrnutí do nadtříd je metoda, která zahrnuje veškeré atributy z podtříd do nadtřídy. Vzniká jediná tabulka, ve které jsou sloupce z podtříd volitelné.
  - Rozdělení do podtříd je metodou přesunující veškeré atributy z nadtřídy do jednotlivých podtříd. Vznikne tolik tabulek kolik je podtříd s opakujícími se některými atributy.

Datový model, viz příloha B, je vytvořen tak, aby vyhovoval 3. normální formě. Normální formy se užívají ke správnému navrhování tabulek a vzájemných relací. Datový model se skládá z několika číselníků, které umožňují jednodušší správu celé aplikace tabulek potřebných pro udržování dat a tabulek pro ukládání potřebných vkládaných dat.

Výhoda číselníků spočívá v možnosti aktualizace např. názvu jedné položky pouze v tabulce obsahující tyto data. Číselník je vlastně takovou entitou *MASTER* a tabulka, kde je číselník využit, je entitou *DETAIL* z tohoto úhlu pohledu. V rámci pohledu na jiné dvě entity lze konstatovat, že entita *DETAIL* v předchozím případě může být entitou *MASTER* v dalším a její klíč bude obsažen v jiné entitě *DETAIL*.

Hlavními tabulkami jsou *Nemovitosti*, *Uzivatele*.Mapování dědičnosti mezi nemovitostí a jejími potomky bylo využito mapování 1:1. Vzhledem k předpokládanému objemu dat a počtu vzniklých sloupců, by byla většina nevyužita.

Asociační tabulky byly vytvořeny mezi číselníkem *TypIngSiti* a tabulkou *Nemovitosti*, kdy jedna nemovitost může mít více inženýrských sítí a zároveň jednu síť může mít více

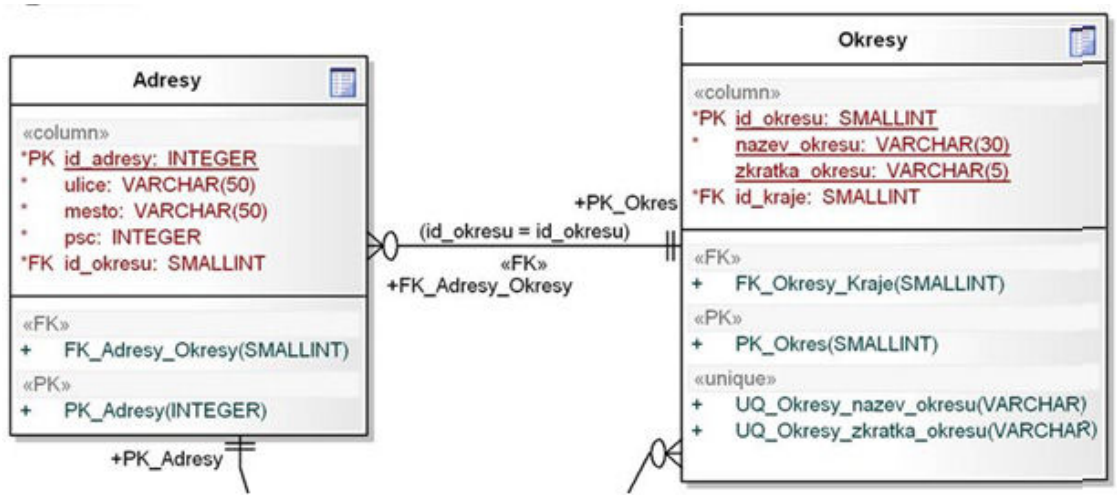
nemovitostí. Další asociační tabulka je vytvořena mezi tabulkou *Kraje* a tabulkou *Top*, jelikož bylo vyžadováno, aby nastavená top nemovitost mohla figurovat ve více krajích.

Pro zajištění mazání souvisejících dat v různých tabulkách byly vytvořeny triggery, které toto zajišťují. Názorná ukázka triggeru demonstruje smazání souvisejících dat v tabulce *Top* a v asociační tabulce *Top_kraj*.

```
CREATE TRIGGER smazTop
BEFORE DELETE ON top
FOR EACH ROW
BEGIN
DELETE FROM top_kraj WHERE id_topu =OLD.id_topu;
END;
```

Tigger je definován jako softwarový prostředek zajišťující automatické spouštění elementárních úkonů na úrovni tabulky. [3]

Obrázek 29 demonstruje část datového modelu z přílohy B, pro názornou ilustraci modelování data modelu v rámci CASE nástroje.



Obrázek 29: Výřez data modelu. Zdroj: vlastní

Jak již bylo zmíněno, CASE nástroje umějí vygenerovat SQL příkazy pro práci s tabulkami, které je možné importovat do dané databáze, která byla nastavena jako výchozí a podle této databáze je zvolen i druh syntaxe a jsou přizpůsobeny datové typy, které se mohou v různých databázích lišit. V rámci generování lze nastavit mnoho předvoleb záleží na tom, které části jsou zapotřebí. Následuje ukázka kódu vygenerovaného pro tabulky *Okresy* a *Kraje*.

```
CREATE TABLE Okresy
(
      id_okresu SMALLINT NOT NULL AUTO_INCREMENT,
      nazev_okresu VARCHAR(30) NOT NULL,
      zkratka_okresu VARCHAR(5),
      id_kraje SMALLINT NOT NULL,
      PRIMARY KEY (id_okresu),
      UNIQUE (nazev_okresu),
      UNIQUE (zkratka_okresu),
      KEY (id_kraje)
```

```
);
ALTER TABLE Okresy ADD CONSTRAINT FK_Okresy_Kraje
      FOREIGN KEY (id_kraje) REFERENCES Kraje (id_kraje);

CREATE TABLE Kraje
(
      id_kraje SMALLINT NOT NULL AUTO_INCREMENT,
      nazev_kraje VARCHAR(30) NOT NULL,
      PRIMARY KEY (id_kraje),
      UNIQUE (nazev_kraje)
);
```

5.6 Implementace

V této kapitole budou představeny možnosti a jazyky pro tvorbu webové aplikace. Největší důraz je kladen na skriptovací jazyk PHP. V dalších podkapitolách bude představen Model-View-Controler a Zend Framework, jelikož tyto náležitosti byly specifikovány jako nefunkční požadavky v rámci konzultací se zadavatelem.

Programátor je odpovědnou osobou v rámci této pracovní činnosti. Snaží se vytvořit systém podle návrhového modelu. Pokud má pochyby o možnosti implementace znázorněným návrhovým modelem, konzultuje je s autorem návrhového modelu. Pokud je návrhový model dobře vytvořen lze využít vlastnosti CASE nástroje pro generování kostry kódu jednotlivých tříd. Takto vygenerovaný kód pro jazyk PHP může vypadat následovně:

```
<?php
require_once ('code.php');
require_once ('Role.php');
/**
 * @author st13963
 * @version 1.0
 * @created 21-V-2009 23:19:13
 */
class Uzivatele
{
      private $jmeno;
      private $prijmeni;
      private $email;
      private $heslo;
      private $aktivni = false;
      private $registrace;
      private $telefon;
      public $m_Firma;
      public $m_Role;

      public function logout()
      {
      }

      /**
       *
       * @param heslo
       * @param email
       */
```

```
        public function login(String $heslo, String $email)
        {
        }

        /**
         *
         * @param telefon
         * @param heslo
         * @param email
         * @param prijmeni
         * @param jmeno
         */
        public function add(String $telefon, String $heslo, String
        $email, String $prijmeni, String $jmeno)
        {
        }

        /**
         *
         * @param kriterium
         */
        public function index(String $kriterium)
        {
        }

        public function delete()
        {
        }
}
?>
```

Před samotným generováním kódu lze nastavit rozšířené možnosti nastavení pro generování např. metody pro nastavení a získání soukromých atributů.

5.6.1 Webové aplikace

Webová aplikace je aplikace, která je prostřednictvím Internetu či intranetu k dispozici uživatelům z webového serveru. K aplikaci se lze připojit odkudkoli, kde je k dispozici připojení k Internetu, avšak není-li připojení k dispozici, webová aplikace nemůže být využívána. V případě těchto aplikací není potřeba provádět žádné instalace, aplikace je zprostředkována pomocí prohlížeče.

Pro tvorbu webových aplikace je možno zvolit implementační jazyk z mnoha možností. Je třeba brát v úvahu několik faktorů: rozšiřitelnost do budoucna, přenositelnost na různé platformy, apod. Krom předchozích faktorů je nutné brát v úvahu, o jakou webovou aplikaci půjde z pohledu statičnosti obsahu stránek.

Statické stránky

Statické stránky jsou charakterizovány statickým neboli neměnných obsahem. Jedná se o HTML stránky propojené pomocí odkazů. Jsou to vlastně uložené soubory s daným obsahem. Pokud se má obsah změnit, musí se zasáhnout do kódu s daným textem. Z toho plyne nevýhoda obtížné zpracovatelnosti obsahu a vyšší nároky na další rozšíření. Stránky může tvořit pouze osoba, která zná HTML tagy (značky) alespoň na základní úrovni.

Tyto stránky jsou vhodné pro malé webové prezentace, které neobsahují mnoho dat a nepotřebují častou modifikaci obsahu.

Dynamické webové stránky

Jak už název dynamické stránky napovídá, obsah těchto stránek je generován automaticky. Data mohou být obsažena buď v databázi, což je nejčastější případ, nebo v souborech. Aby data mohla být zpracována, je potřeba programovací technologie a program, který je uložen na webovém serveru.

Při příchodu požadavku na zobrazení stránky, je dynamicky sestavena a odeslána do prohlížeče. K sestavení stránky dochází za pomocí programu, který běží na technologiích jako jsou PHP, ASP, JSP nebo ASP.NET. Tyto stránky na rozdíl od statických nemají uložený obsah a jsou generovány po každém požadavku znova. Jedná se o interaktivní přístup.

Technologie pro dynamické stránky

Jak bylo zmíněno v přešlé odstavci pro tvorbu dynamických stránek je potřeba využívat programovací technologie, které umožňují generování těchto stránek. Mezi tyto technologie patří:

- **JSP (Java Server Pages)**

  Jedná se o Java technologii, která může obsahovat HTML, XML prvky. Tyto stránky mohou také obsahovat kód jazyka Java. Syntaxe JSP přidává další tagy, které jsou nazývány JSP akce, používané pro přidání vestavěné funkcionality.

  Stránky JSP jsou kompilovány pomocí JSP kompilátoru do podoby Java servletu. JSP kompilátor může generovat servlet v Java kódu, který je přeložen Java kompilátorem. Stránky mohou být interpretovány za běhu, což snižuje čas pro znovu načtení změn.

- **ASP (Active Server Pages)**

  Jedná se o skriptovací technologii vytvořenou společností Microsoft. Je velmi podobná PHP a má jinou koncepci než její následník ASP.NET. Koncepce ASP spočívala v strukturovaném programování, zatímco koncepce ASP.NET je zaměřena na objektově orientované programování. Stránky jsou prováděny interpretem.

  Jazyky, které se využívají při tvorbě aplikace v ASP, jsou VBScript (založeno na Visual Basicu) a JSript (serverový Javascript). ASP.NET jich podporuje již přes dvacet. Verze této technologie jsou tři a to ASP 1.0 (1996), ASP 2.0 (1997) a ASP 3.0 (2000), jež je poslední verzí ASP, poté nastupuje na scénu technologií pro dynamické stránky ASP.NET.

- **ASP. NET**

  Aby webová aplikace vytvořená v technologii ASP.NET mohla fungovat, je potřeba mít k dispozici komponentu Microsoftu .NET Framework, což je prostředí pro vytváření, nasazení a běh webové aplikace a webových služeb. .NET Framework obsahuje tzv. Common Language Runtime (CLR), který má na starosti správu paměti, správu vláken, bezpečnost kódu a načítání potřebných komponent do paměti. Může být provozována i na jiných platformách než je Windows, ale zprovoznění a následný běh může být problematický. Běží výhradně na webovém serveru Internet Information Services.

Existuje i oficiální framework, který je vybudován na technologii ASP.NET, nazývá se ASP.NET MVC a je určen pro vývoj webových aplikací založených na modelu MVC.

Vytvořené soubory mají příponu „.aspx“. Tyto soubory jsou stejné jako HTML soubory, mohou obsahovat HTML, XML a skripty. Nejznámější jazyky, které jsou v rámci ASP.NET využívány, jsou C# a Visual Basic.NET. Další jazyky, které lze zmínit, jsou JScript.NET a Managed C++. První verze ASP.NET 1.0 vyšla v roce 2002, následovaly verze ASP.NET 1.1 (2003), 2.0 (2005) a 3.0 (2006). ASP.NET 3.0 není novou verzí ASP.NET. Je to jen název pro verzi ASP.NET 2.0, která je funkcionálně rozsáhlejší.

- **PHP (Hypertext Preprocessor, původně Personal Home Page)**

  Je open source skriptovací technologie od PHP Group. Syntaxe jazyka obsahuje prvky několika jiných jazyků např. C, Java, Perl a Pascal. Jedná se o technologii hojně využívanou z důvodů:

  - jednoduché syntaxe,
  - volného využívání (je zdarma),
  - dostupných hostingů i freehostingů,
  - multiplatformního použití,
  - podpory mnoha databázových systémů (MySQL, Oracle, MSSQL, PostgreSQL),
  - slušně zpracované dokumentace.

  Nevýhody:

  - PHP je stále v aktivním vývoji, proto u novějších verzí může nastat situace, že funkce z předchozí verze se bude chovat jinak.
  - Unicode je podporován pouze přes knihovnu.
  - PHP skript se při každém požadavku překládá znovu.

  PHP disponuje také frameworky, které ulehčují programátorů práci při vývoji. Mezi PHP frameworky patří:

    - Zend Framework,
    - CakePHP,
    - CodeIgniter,
    - Symfony.

  První verze PHP 1 vyšla v roce 1995, následovala verze PHP 2, která odstraňovala chyby z verze 1. V roce 1997 byla vytvořena verze PHP 3, ale uvolněna byla až 1998. Ještě v roce 1997 byl význam PHP ve smyslu Personal Home Page změněn na Hypertext Preprocessorr. Předposlední verze PHP 4 pochází z roku 2000. Poslední verze PHP 5 byla uvolněna v roce 2004, ale jako stabilní prohlášena až 2008. Celkově verze PHP 5 má lepší podporu objektově orientovaného programování. Chystá se verze PHP 6, která by měl řešit problém s Unicode.

5.6.2 Nástroje pro tvorbu dynamického webu

Tvorba webových aplikací vyžaduje několik nástrojů k vytvoření odpovídajícího běhového prostředí. Jedná se především o webový server, databázový server, příp. prostředí pro samotnou implementaci.

Webový server

Webový server je počítač nebo počítačový program, který je zodpovědný za vyřizování HTTP požadavků od klientů pomocí webových prohlížečů (např. Mozilla Firefox, Internet Explorer, Opera atd.). Odpovídá na požadavky a vrací požadovaný dokument nebo obrázek. Nejčastěji využívanými webovými servery jsou:

- Apache HTTP Server vyvíjený Apache Software Foundation, který je velmi oblíben z důvodu možnosti běhu pod různými operačními systémy. Má otevřený kód a většina existujících webových serverů je právě Apache.
- Internet Information Services (IIS) od Microsoftu je skupina internetových služeb jako FTP, SMTP, apod. IIS je zaměřen pouze na operační systém Microsoft Windows, proto zaostává za Apachem.

Databázový server

Databázový server je počítačový program, který poskytuje služby databáze jiným počítačovým programům nebo počítačům. Pro přístup k datům je potřeba ovládat dotazovací jazyk, nejčastěji SQL. Mezi nejznámější databázové servery patří:

- MySQL
  Jedná se o volně šiřitelný, multiplatformní databázový systém vytvořený firmou MySQL AB.
- Microsoft SQL Server
  Jedná se o relační databázový systém vytvořený firmou Microsoft. MS SQL běží na operačním systému Microsoft Windows.
- Oracle Database
  Je multiplatformní databázový systém vytvořený firmou Oracle Corporation.
- PostgreSQL
  Jedná se databázový systém, který může běžet na operačním systému typu Unix nebo Windows.

Vývojové prostředí

Vývojové prostředí je software, který slouží k usnadnění práce při programování. Většinou se zaměřuje na jeden konkrétní programovací jazyk, ale existují i vývojová prostředí podporující více jazyků. Obsahuje kompilátor příp. interpret, debugger a prostředí pro editaci zdrojového kódu.

- Eclipse – překladač více jazyků,
- NetBeans – překladač více jazyků,
- Zend Studio – jazyk PHP,
- JDeveloper – jazyk Java,
- Microsoft Visual Studio – jazyky C, C++, C#,

Možnosti jak si pořídit webový a databázový server jsou dvě a to buď jako samostatné instalace, které je nutno si pořídit nebo jsou součástí tzv. balíčku.

Balíky předkonfigurovaných instalací, umožňují snadnou instalaci kompletního vývojového prostředí na uživatelském počítači. Tato možnost je určena především osobám, které nechtějí nebo neumějí nakonfigurovat jednotlivé aplikace, tak aby spolu vzájemně spolupracovali. Po dokončení instalace je vše plně funkční a připraveno pro první použití. Nevýhoda balíků spočívá v neznalosti provázanosti komponent, a tudíž neodborný zásah může mít důsledky v podobě nefunkčního prostředí.

Samostatné instalace jednotlivých serverů vyžadují určité znalosti a více času. Lze si je pořídit na webových stránkách dané firmy, pokud jsou ovšem volně k dispozici (nejčastěji v sekci download).

V rámci vytváření informačního systému pro správu realit byl využit před instalovaný balík konfigurací XAMPP. Jedná se o distribuci Apache, která obsahuje MySQL, PHP a Perl. Vývojovým prostředím byl zvolen částečně poznámkový blok a částečně Eclipse.

5.6.3 Model-View-Contorller

Model-View-Controller zkráceně MVC začíná být čím dál tím více populárnější modelem v posledních letech. Je to téma budoucího vývoje aplikací. MVC umožňuje oddělit datovou vrstvu od vrstvy logické (Controller), která zajišťuje logiku aplikace, a vrstvy vzhledové (View) mající nastarost vzhled stránky. Model umožňuje práci s daty a přístup k nim.

Dříve a možná ještě v některých případech dnes se používá tzv. drag&drop vývoj. Kód, který má zajišťovat vzhled, je promísen s kódem, který pracuje s daty z úložiště a poté dochází k jeho zpracování. Nevýhodami takto strukturovaných kódů jsou především nepřehledně oddělené pasáže a zdlouhavý kód. Nevýhody lze shrnout do následujících bodů:

- Odladění kódu s tisíce řádky při nalezení chyby
- Zobrazování stejných dat
- Opakování kódu na různých místech
- Vyšší finanční náklady na údržbu

MVC je výhodným řešením v rámci rozsáhlejších projektů, u kterých je vyžadována možnost úprav a údržby. Pokud budou usnadněny předchozí dvě potřeby, budou náklady na ně nižší a ušetří se i čas strávený nad úpravami.

MVC se skládá ze tří druhů objektů. Model je aplikační objekt, View je jeho obrazová prezentace a Controller definuje způsob, jakým uživatelské rozhraní reaguje na uživatelský vstup. Tyto tři objekty jsou odděleny a tím se zvyšuje znovu použitelnost výsledného řešení.

- **Model** je v tomto případě nejjednodušší část, protože je identický s Modelem v desktopových technologiích. (Obsahuje data a business logiku, s konkrétní prezentací nemá nic společného.) [9]
- **View** se někdy říká, že u webové aplikace je to HTML, ale to je jako říci, že u desktopové aplikace se View rovná pixelům na obrazovce, respektive nízkoúrovňovým vykreslovacím instrukcím. View je ve skutečnosti serverový kód, který se o generování HTML stará, tj. PHP, C#, Java, Ruby a podobně.

Navíc ani u webové aplikace není nutné, aby bylo výstupem HTML – klidně to mohou být formáty jako XML nebo JSON. [9]

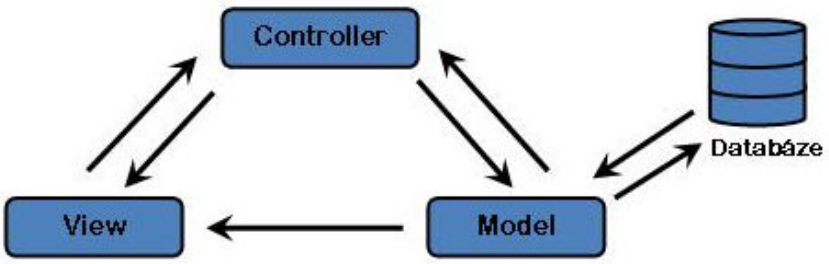
- **Controller** se v prostředí webu nejčastěji skládá ze dvou hlavních částí. První je tzv. Front Controller, který zachytává všechny HTTP požadavky, ty následně zpracuje a přepošle konkrétním Controllerům, což je ona druhá část. Konkrétní Controller potom typicky přijme data původně pocházející z HTTP požadavku, uloží je do Modelu a ten prováže s konkrétním View, které už se umí o vyrenderování HTML nebo jiného formátu postarat. [9]

MVC se řadí spíše mezi architektonický vzor než návrhový, protože se využívá při kompletním návrhu architektury. Nejznámější frameworky podporující MVC architekturu:

- Zend Framework od firmy Zend
- ASP.NET od Microsoftu

Obrázek 30 znázorňuj princip komunikace objektu, který se dá interpretovat v následujících bodech.

- Controller získává z Modelu data, která zpracuje a pošle na výstup do View.
- Controller získá data z View a předává je do Modelu nebo je volána jiná metoda pomocí View na základě které je výstupem odlišné View.
- View může zobrazit data jak z Controlleru, tak z View, ale data získaná pomocí View (např. formulářová data) lze zpracovat pouze pomocí Controlleru.
- Model dostává data od Controlleru, která zpracovává do databáze (update, insert, delete).
- Model získává data z databáze dle požadavků a odesílá je do Controlleru nebo do View.



Obrázek 30: MVC. Zdroj: vlastní

5.6.4 Zend Framework

Zend Framework (ZF) je výtvorem společností Zend Technologies Ltd. Tvořící IDE pro vývojáře v PHP. ZF se dá popsat jako seskupení knihoven, které slouží k ulehčení práce vývojářům PHP aplikací.

ZF se stále vyvíjí, hlavními tvůrci jsou odborníci s rozsáhlou praxí ve vývoji. ZF obsahuje vlastně to nejlepší z jiných frameworků jako jsou .NET nebo Ruby. ZF klade velký

důraz na objektově orientované programování. V ZF lze vytvářet aplikace založené na MVC modelu. Umožňuje jednoduchou autorizaci, autentifikaci uživatelů, práci s více druhy databází, práci s webovými službami jako je Amazon, Yahoo, Google apod.

Za použití ZF se nemusí platit ani při komerčním užití, je vytvořen pod BSD licencí, která umožňuje dále si rozšiřovat framework, avšak dodatečně vytvořené třídy nesmí být označeny prefixem „Zend“. Tato licence vyžaduje uvedení informace o licenci, uvedení autora a zřeknutí se odpovědnosti za dílo.

Vlastnosti, kterými ZF disponuje, jsou uvedeny v následujícím výčtu:

- objektově orientované komponenty,
- modulární architektura,
- MVC,
- podpora několika databázových systémů.

ZF je složen z několika komponent. V rámci aplikace lze využít jen ty, které jsou třeba. Nejznámějšími komponentami jsou:

- Zend_Acl – spravuje uživatelská oprávnění
- Zend_Auth – autentifikování uživatelů
- Zend_Config - nastavování aplikace pomocí konfigurační soubory
- Zend_Controller – implementuje MVC architektury
- Zend_Date - pracuje s datem
- Zend_Db – podporuje více databází (MySQL, Oracle, IBM DB2 MSSQL Server, PostgreSQL, SQLite a Informix Dynamix Server)
- Zend_Filter – filtrování dat
- Zend_Form – vytváření webových formulářů včetně filtrování a validace
- Zend_Layout – spravuje layoutů
- Zend_Mail – podporuje tvorbu e-mailů
- Zend_Paginator – pracuje se stránkováním
- Zend_Pdf – podporuje vytváření PDF souborů
- Zend_Registry – uchovává objekty a hodnoty v aplikační vrstvě
- Zend_Translate – podporuje překlady a jazykové mutace
- Zend_View – systém šablon
- ZendX_JQuery – podporuje framework jQuery

ZF doporučuje určitou adresářovou strukturu, ale nejedná se o striktní nařízení. Nejprve je nutno vytvořit kořenový adresář *jmeno_aplikace*. Dále se doporučují vytvořit následující podadresáře *application*, *library* a *public*. V adresáři *application* se vytvoří následující podadresáře *controllers*, *models* a *views*. *Views* muže obsahovat podadresáře *filters*, *helpers* a *scripts*. Do adresáře *library* se zkopíruje složka *Zend* z adresáře *library*, který je obsažen v komprimovaném souboru, který je nutné stáhnout na adrese http://framework.zend.com/download/current. Poslední adresář *public* je určena pro podadresáře *image*, *scripts* a *styles*.

Jak je zřejmé, byly vytvořeny samostatné adresáře pro soubory Modelu, View a Controlleru. Potřebné CSS soubory pro styl aplikace se nacházení v adresáři *styles*, obrázky

obsažené v aplikaci jsou umístěny v adresáři *images*. Pokud budou potřeba i jiné knihovny např. *ZendX*, tak se umísťují do adresáře *library* společně s adresářem *Zend*, který obsahuje soubory ZF.

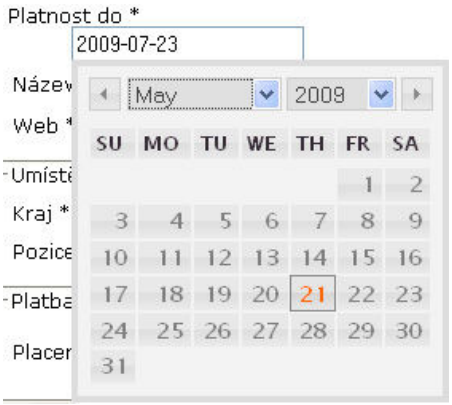
ZendX neboli Zend Extras Library, je knihovna, která integruje *jQuery*. *jQuery* je knihovna, která usnadňuje práci s JavaScritpem. Tato knihovna byla zapotřebí, aby se mohlo jednodušeji pracovat s datem, které bylo nutno zadávat, a však samotný ZF usnadnění práce s datem nepodporuje na takové úrovni.

Následující kód byl vložen do souboru *index.php*. Poté bylo možné pracovat s elementem pro datum v rámci celé aplikace.

```
$view= new Zend_View();
$viewRenderer = new Zend_Controller_Action_Helper_ViewRenderer();
$view->addHelperPath('ZendX/JQuery/View/Helper/',
                                'ZendX_JQuery_View_Helper');
$viewRenderer->setView($view);
Zend_Controller_Action_HelperBroker::addHelper($viewRenderer);
```

Následující kód je již kódem pro vkládání elementu pro datum, který nabízí k výběru den, měsíc a rok. Při použití CSS stylu pro tento element může výsledek vypadat jako na Obrázek 31.

```
$do = new ZendX_JQuery_Form_Element_DatePicker("do",
                            array("label" => "Platnost do "));
$do->setJQueryParam('dateFormat', 'yy-mm-dd');
```



Obrázek 31: DatePicker. Zdroj: vlastní

MODEL

Model má na starosti data. V informačním systému se pracuje s MySQL databází, proto bude potřeba využít třídu *Zend_Db_Table*. Tato třída je vytvořena pro usnadnění práce se zvolenou databází. Aby se mohlo se třídou pracovat je nutno ji nakonfigurovat. Je nutné nastavit typ databáze a přihlašovací údaje. Vytvoříme soubor *config.ini*, který obsahuje následující kód:

```
[general]
db.adapter = PDO_MYSQL
db.host = localhost
db.username = uživatel_DB
db.password = heslo_do_DB
db.dbname = jmeno_DB
```

V rámci *index.php* je nutno přidat následující řádek, který odkazu na vytvořenou konfiguraci: `$config=new Zend_Config_Ini('./application/config.ini', 'general')`. Zde je vidět že byla využita další třída ZF.

Třída *TypSluzby*, která je níže vypsána jako ukázková, je potomkem abstraktní třídy *Zend_Db_Table*. Nastaví se *protected* proměnná, která obsahuje název tabulky.

```
class TypSluzby extends Zend_Db_Table
{
  protected $_name = 'typ_sluzby';

  public function getNazev($id)
  {
    $db = Zend_Db_Table::getDefaultAdapter();
    $select = $db->select()
                ->from(array('ts'=> $_name),'nazev_sluzby')
                ->where('ts.id_sluzby = ?', $id);
    if ($result = $db->fetchAll($select))
    {
      return $result;
    }
  }
 …
}
```

Tuto třídu uložíme do adresáře *models*. Takto postupujeme při tvorbě dalších tříd, které odpovídají tabulkám v databázi.

CONTROLLER

Skoro každá stránka, která se zobrazuje uživateli, je vytvářena většinou pomocí *action*. Tyto *actions*, jež jsou veřejnými funkcemi, jsou umístěny v Controlleru. V jednom Controlleru jsou sdružovány *actions*, které mohou být vykonány nad jedním a tím samým objektem. Každý Controller musí obsahovat *indexAction*, která je vždy použita jako výchozí pokud není nastaveno jinak. ZF obsahuje i speciální Controller nazývaný *IndexController*, ten se využívá jako výchozí, když není specifikována cesta k jinému.

Controller v ZF je třída, která je potomkem *Zend_Controller_Action*. Název controlleru je pevně dán a skládá se z prefixu udávající název a slova *Controller*. Důležité je, aby prefix začínal velkým písmenem a ostatní písmena byla malá. Takto vytvořená třída, musí být umístěna v adresáři *controller*. Název akce (*action*) má také svá pevná pravidla. Celý prefix, určujícího zaměření akce musí být napsán malými písmeny a následuje slovo *Action*. Následuje příklad struktury třídy controlleru.

```
<?php
class NazevController extends Zend_Controller_Action
{
    function indexAction()
    {  //přehled  }

    function addAction()
    {  //přidávání  }

    function editAction()
    {  //editování   }
```

```
    function deleteAction()
    {  //mazání  }
}
```

Další kód zobrazuje konkrétní použití v rámci navrhovaného systému.

```
class TypsluzbyController extends Zend_Controller_Action
{
  function indexAction()
  {
    $ts = new TypSluzby();
    $this->view->typ_sluzby = $ts->fetchAll();
    //z tabulky TypSluzby veme všchny záznamy
    //odešle je jako výstup do index.phtml
  }

  function addAction()
  {
    $ts = new TypSluzby();
    $form = new Reality_Form_AddTypForm();
    if (!$this->getRequest()->isPost())
    {
      $this->view->form = $form;
      return;
    }
    elseif (!$form->isValid($_POST))
    {
      $this->view->failedValidation = true;
      $this->view->form = $form;
      return;
    }
      $values = $_POST;
      $data = array('id_sluzby'=> 0,
                    'nazev_sluzby' => $values['nazev']);
      $ts->insert($data);
      $this->view->entrySaved = true;
  }
  …
}
```

VIEW

Protože View úzce souvisí s Controllery a vychází z nich, je pojednání o prezentaci dat umístěna až za vysvětlením Controlleru. Všechny view soubory patřící k danému Controlleru musí být umístěny v adresáři pojmenovaném stejně jako prefix Controlleru, vše opět malými písmeny.

Komponenta pracující s View je nazvána *Zend_View* a umožňuje oddělení kódu pro zobrazení stránky od kódu v příslušné akci Controlleru. Tím dochází k oddělení nejčastěji HTML kódu od ostatního kódu. Stránky, které obsahují data z Controlleru z dané *action* mají stejný název jako prefix action. Tedy pokud se jmenuje funkce *addAction* odpovídající stránka ponese název *add.phtml* v adresáři pro View daného Controlleru .

Aby byla data z funkce zobrazena na dané stránce, musí se tam předat. Toto předání je pouhé přiřazení dat proměnné `$this->view->nazev_promenne`. Konkrétním příkladem může být již dříve zmíněná funkce *indexAction* Controlleru *TypsluzbyController*.

```
function indexAction()
  {
    $ts = new TypSluzby();
    $this->view->typ_sluzby = $ts->fetchAll();
    //z tabulky TypSluzby veme všchny záznamy
    //odešle je jako výstup do index.phtml
  }
```

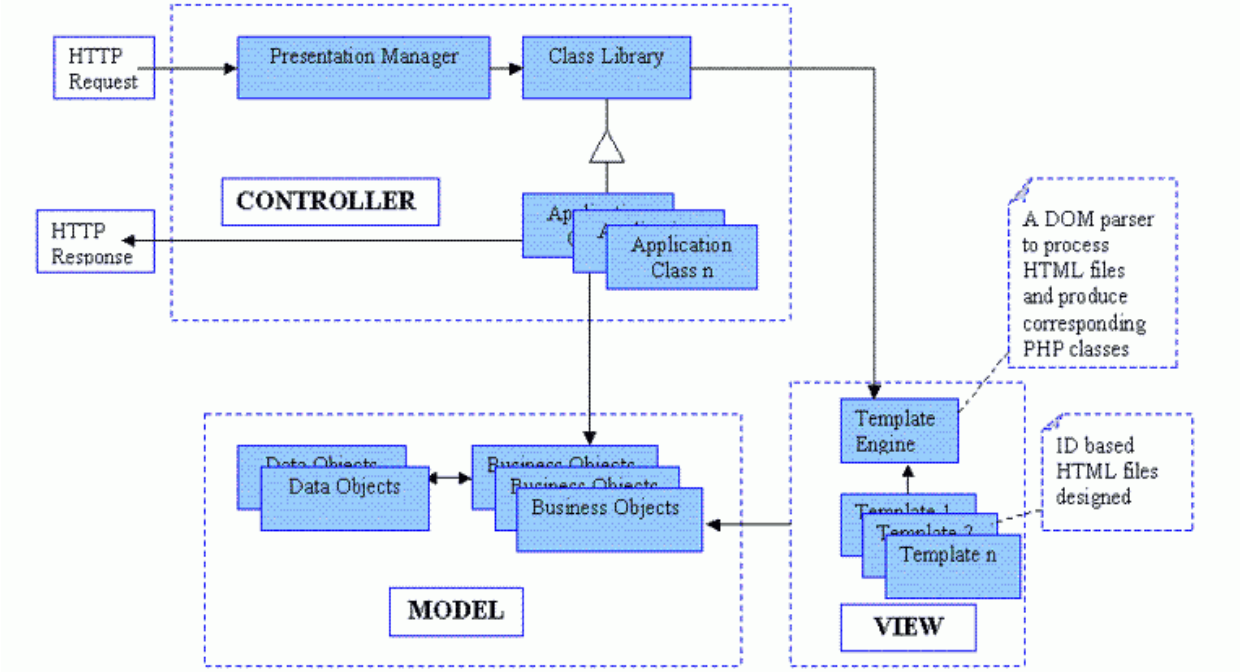
Zde je vidět přiřazení výsledku vyhledání v databázi proměnné:

```
$this->view->typ_sluzby = $ts->fetchAll();.
```

Následující ukázka kódu patří k akci *indexAction* ve třídě *TypsluzbyController*, kdy dochází k zobrazení všech položek číselníků typ služby z tabulky *TypSluzby* do stránky *index.phtml*. Proměnná `$this->typ_sluzby` nese vybraná data.

```
<div id = "oblaststred">
<div id ="text">TYPY SLUŽEB</div>
<a href="/index.php/typsluzby/add">Přidat</a>
<table class="tablecont">
  <tbody>
    <tr>
      <th>možnost</th>
      <th>název</th>
    </tr>
    <?php $n=0; ?>
    <?php foreach($this->typ_sluzby as $it) : ?>
    <tr>
      <td>
        <?php $smaz = 'typsluzby/delete/sluzba/'.$it['id'];?>
        <a href="<?= $smaz ?>" class="icon" title="smazat">
          <img src="public/images/icons/trash.png" alt="smazat">
        </a>
        <a href="typsluzby/edit/sluzba/<?= $it['id'] ?>"
                                              title="edit">
           <img src="public/images/icons/edit.png" alt="edit">
         </a>
      </td>
      <td><?= $it['nazev'] ?></td>
    </tr>
    <?php $n++; ?>
    <?php endforeach; ?>
  </tbody>
</table>
<a href="/index.php/ciselniky/index">Zpět</a>
</div>
```

Obrázek 32 znázorňuje rozvržení jednotlivých souborů v rámci využití PHP s ZF.



Obrázek 32: MVC PHP Frameworku. Zdroj: [16]

5.7 Testování

Úkolem testování softwaru je zjisti a ověřit požadované funkcionality tvořeného systému. Čím je testování důkladnější, tím méně je poté potřeba lidských a finančních zdrojů na jejich opravu a následnou údržbu. Zdroje, které se ušetří na opravách, se mohou vynaložit na další rozvoj systému. Dalším důležitým důvodem testování je minimalizace rizika výpadku systému při ostrém zavedení a tím možnosti vzniku finanční ztráty.

Existuje několik druhů testování systémů. Jedním z druhů testování je testování nejvyšší úrovně, které se dělí podle způsobu pohledu na systém.

Black-box testování

Předpokladem toho testu je neznalost testera vnitřní struktury systému. Takovými testery mohou být nezainteresované osoby. Tyto osoby bývají většinou budoucí uživatelé systému, jsou schopny nalézt chyby při přepokládaném využívaní systému. Testují se vstupy a jsou vyžadovány kompetentní výstupy.

White-box testování

Předpokladem toho testu je znalost vnitřní architektury systému. Testují se jednotlivé metody, procedury apod. V tomto případě by tester měl mít znalost programovacího jazyka, ve kterém je systém programován.

Kritériem dalšího dělení testování je fáze vývoje, kdy se provádí:

- Unit test – testují se dílčí části kódu. Jedná se o test v době vývoje a provádí ho obvykle sami programátoři, aby zjistili, jestli část programu dodaná do aplikace funguje správně.
- Assembly test – programátoři zjišťují funkční spolupráci částí kódů otestovaných v unit testu.
- Smoke test – ověřovací test pro další fázi testování.

- Integrační test – zjišťuje a ověřuje, zda veškeré části systému spolu navzájem komunikují.
- Systémový test – ověřuje se, že systém funguje správně jako celek. Testují se situace, které mohou nastat jen zřídka, dále se testuje správnost výstupů při daných vstupech. Tento test má několik opakování, pokud jsou nalezeny chyby, pak jsou opraveny a začíná se znovu testovat.
- Akceptační test – tento test je prováděn samotným zákazníkem nebo uživateli, kteří budou systém používat. Ověřují si, zda systém obsahuje veškeré funkcionality, které byly zákazníkem požadovány. Pokud systém projde akceptačním testem vývoj systému je ukončen a může být plně využíván.

Jedním z posledních kritérií, podle kterého se může testování dělit, je funkčnost.

Funkční test – obsahuje všechny testy, které mají ověřit správnost výsledků jednotlivých funkcionalit systému, které jsou po něm vyžadovány. Tyto funkcionality jsou určeny požadavky zákazníka.

Nefunkční test – zahrnuje veškeré testy, které nesouvisí s funkcionalitou systému. Testované vlastnosti musí vykazovat také správné výsledky pro správné fungování systému.

Scénáře případů užití se považují za výhodný zdroj tvorby testovacích scénářů. Scénáře případů užití určují testovací požadavky na fungování informačního systému. Jedná se o posloupnost kroků, které by měly být provedeny, a následných výstupů na vstupy od uživatele.

Testování probíhalo průběžně během implementace. Vždy po ukončení implementace části aplikace bylo otestováno správné fungování. V případě nekorektního chování se okamžitě učinila náprava, aby se chyby nekupily a nedocházelo k nekorektnímu chování jiných částí aplikace, jelikož chyba mohla mít vliv na jiné funkce.

Ke konečnému testování byly přizvány nezainteresované osoby, aby byly vyzkoušeny náhodné myšlenkové procesy potencionálního uživatele. V následující Tabulka 5 znázorňuje počty nalezených chyb jednotlivými testery.

Tabulka 5: Výsledky testování. Zdroj: vlastní

Osoba	***Testovací oblast***	***Počet nalezených chyb***
Tester 1	Veřejná část	7
	RK Admin centrum	3
	Admin centrum	6
Tester 2	Veřejná část	10
	RK Admin centrum	2
	Admin centrum	2
Tester3	Veřejná část	20
	RK Admin centrum	0
	Admin centrum	0

Z výsledků systémového testování se dospělo k názoru, že tester 3 se zaměřil pouze na veřejnou část informačního systému, ale ostatní oblasti vykazuje jako bezchybné. K těmto oblastem se pravděpodobně nedostal v termínu potřebném k vykázání výsledků, proto se jeho výsledky nemohou brát jako stěžejní. Tester 1 a tester 2 stihli alespoň částečně otestovat všechny oblasti, tudíž jejich výsledky mají pro programátora vypovídající hodnotu.

Veškeré nahlášené chyby byly opraveny a v následujícím testování aplikace prošla testovacím procesem bez chyb. Aplikace tak mohla být nasazena a předána k akceptačnímu testování zadavateli.

5.8 Nasazení

Nasazení softwaru umožňuje zpřístupnění systému uživatelům, pro které je vytvořen. Základní proces nasazení se skládá z několika souvisejících činností s možností přechodu mezi nimi. Tyto činnosti se mohou vyskytnout buď na straně zákazníka neboli zadavatele, nebo na straně výrobce neboli dodavatele, nebo dokonce na obou zmíněných stranách současně. Protože každý systém je jistým způsobem specifický, musí být jasně definována každá činnost. Nasazení by mělo být považováno za základní proces, který musí být přizpůsoben podle specifických požadavků. Proces nasazení se tedy může skládat z následujících činností:

- Vydání – následuje po dokončení vývoje, zahrnuje všechny operace pro přípravu systému k přemístění na stranu zákazníka, proto musí být určeny zdroje, aby mohl fungovat u zákazníka a shromážděny informace pro provedení následujících činností souvisejících s nasazením systému.
- Zavedení a spuštění – spuštění je činnost uvedení do provozu spustitelných komponent systému.
- Ukončení – vztahuje se k odstavení spustitelných komponent systému, je třeba pro vykonání jiných činností v rámci nasazení např. před aktualizací.
- Přizpůsobení – je proces přizpůsobení nebo změny systému, který byl zaveden nebo nainstalován. Na rozdíl od aktualizace je adaptace zahájena událostí jako například změna prostředí na straně zákazníka, zatímco aktualizace je většinou iniciována vzdáleně stranou výrobce.
- Update neboli aktualizace – nahrazuje původní verzi systému nebo pouze její část novou verzí.
- Vestavěný update – mechanismus pro aktualizace je vestavěny v některých systémech. Aktualizace může být buď plně automatická, nebo uživatelem iniciovaná a kontrolovatelná. Jiné systémy využívají dotazovací mechanismus, kdy uživatel je upozorněn na novou dostupnou aktualizaci a je zcela na uživateli, zda aktualizaci povolí.
- Odinstalování – opak instalace, je to odstranění systému, který již dlouhou dobu není potřebný. To vyžaduje rekonfiguraci jiných systémů, aby mohly být odstraněny systémové soubory a jejich závislosti.
- Stažení – systém je označen jako zastaralý a podpora již není zajišťována. Toto je definitivní konec životního cyklu softwarového produktu.

V rámci nasazení výsledného informačního systému pro správu realit bylo nutné nahrát aplikaci na server a nastavit základní URL. Vzhledem k tomu, že tvorba informačního systému byla převážně tvořena na localhostu s právy uživatele root, bylo nutné do databáze na serveru nechat zavést triggery, importovat potřebné tabulky a potřebná data do tabulek typu číselník. Po nasazení bylo nutno odladit ještě pár nedostatků, ale nebyl třeba žádný složitý zásah do celé aplikace. Jednalo se o dodatečné vzhledové úpravy, které si zadavatel vyžádal.

Veškeré služby spojené s provozem aplikace na webovém serveru, včetně využívání databáze, jsou outsourcovány. Firma, u které je hosting zařízen, garantuje dostupnost aplikace 24 hodin denně 7 dní v týdnu. Tato doba přístupnosti byla hlavním faktorem pořízení tohoto hostingu, jelikož nefunkčním požadavkem v oblasti přístupnosti byla požadována právě tato doba možného provozu. Veškeré výpadky v provozu jsou v režii poskytovatele hostingu a tím i zodpovědnost za případný ušlý zisk a odpovědnosti náhrady škody.

6 Závěr

Cílem této práce bylo vytvořit webový informační systém pro správu realit pomocí UML a UP. Práce na webovém informačním systému pro správu realit začala zjištění požadavků zadavatele na daný systém. Tyto požadavky byly upřesňovány i na dalších schůzkách při prezentaci dosavadního řešení.

Na základě zjištěných požadavků byly vytvořeny diagramy případů užití, které byly rozděleny do několika oblastí podle zaměření a s podrobnějším popisem. Každý případ užití obsahuje alespoň základní scénář a některé obsahují i scénáře alternativní. Na základě vytvořených případů užití a požadavků byla vytvořena vztahová matice.

Grafický vzhledem informačního systému byl určen dodanými podklady, které si nechal zadavatel připravit u externí firmy zaměřené na grafický návrh. Později bylo nutné poupravit tento návrh podle požadavků zadavatele.

V rámci návrhu analytického modelu bylo využito metody štítku pro nalezení analytických tříd a částečně znalost fyzických objektů problémové domény. Z analytického modelu se poté vycházelo při tvorbě návrhového modelu, kde bylo nutno upravit relace mezi třídami a více specifikovat atributy a metody daných tříd.

Dalším krokem před začátkem implementace bylo nutné vytvořit datový model, jelikož data měla být uchovávaná v databázi. Tento model opět navazuje na předchozí model tedy návrhový model. Zde bylo použito mapování návrhových tříd do tabulek. Pro jednodušší správu celého systému bylo vytvořeno několik číselníků pro různé druhy statických dat.

Při implementaci bylo občas nutné pozměnit návrhový model, protože se došlo k závěru, že tímto způsobem to jednoduše nepovede k cíli. Během implementace bylo využíváno tzv. systémového testování, kdy po každé nově implementované funkcionalitě byla ihned vyzkoušena její správná funkčnost.

Před konečným nasazením na server, který byl zajištěn. Proběhlo tzv. white-box testování pro zjištění chyb systému. Následné nasazení na server si vyžádalo export datového modelu z CASE nástroje, import vygenerovaného kódu na databázový server a zajištění možnosti přidání triggerů. Dále byly naplněny potřebné číselníky. Ohledně uploadu aplikace na aplikační server nabyly zaznamenány velké problémy. Jen musely být provedeny úpravy vzhledem k základní URL adrese.

Konečný akceptační test si provedl zadavatel sám. Upozornil na několik nedostatků, které byly ihned napraveny. Výsledný systém obsahuje všechny požadované funkcionality, a proto byl daný systém zadavatelem přijat. Tím, že byl informační systém přijat, bylo dosaženo cíle této práce.

Cíl práce byl splněn mimo jiné i díky využití Zend Frameworku, který je založen na oddělení prezenční, logické a datové vrstvy, tudíž je výsledný informační systém jednoduše modifikovatelný pro pozdější možná rozšíření.

Použitá literatura

[1] ARLOW, J.; NEUSTADT, I. *UML 2 a unifikovaný proces vývoje aplikací*. 1. vydání. Brno: Computer Press, 2007. 568 s. ISBN 978-80-251-1503-9

[2] SCHMULER, J.; *Myslíme v jazyku UML*. 1. vydání. Praha: Grada Publishing, 2001. 360 s. ISBN 80-247-0029-8

[3] KANISOVÁ, H.; MÜLLER, M. *UML srozumitelně.* 2. vydání. Brno: Computer Press, 2007. 176 s. ISBN 80-251-1083-4

[4] KRAVAL, I. *Zásady pro řízení projektů tvorby IS v objketově orientovaném prostředí za použití UML.* [e-book]. 2005 [cit. 2009-3-12] Dostupné z WWW: <http://www.objects.cz/produkty/produkty.html>

[5] Data & Object Factory. *Design Patterns.* [online]. 2007 [cit. 2009-4-7] Dostupné z WWW: <http://www.dofactory.com/Patterns/Patterns.aspx>

[6] Zoner software. *Zend Framework – otázky a odpovědi.* [online]. 2007 [cit. 2009-4-23] Dostupné z WWW: <http://php.interval.cz/clanky/zend-framework-otazky-a-odpovedi/>

[7] Zoner software. *Zend Framework – přehled knihoven.* [online]. 2007 [cit. 2009-4-23] Dostupné z WWW: <http://php.interval.cz/clanky/zend-framework-prehled-knihoven/>

[8] Zdroják. *Úvod do architektury MVC.* [online]. 2009 [cit. 2009-5-8] Dostupné z WWW: <http://zdrojak.root.cz/clanky/uvod-do-architektury-mvc/>

[9] Zdroják. *Prezenční vzory z rodiny MVC.* [online]. 2009 [cit. 2009-5-11] Dostupné z WWW: <http://zdrojak.root.cz/clanky/prezentacni-vzory-zrodiny-mvc/>

[10] Linux EXPRES. *Srovnání databázových serverů.* [online]. 2007 [cit. 2009-5-9] Dostupné z WWW: <http://www.linuxexpres.cz/software/srovnani-databazovych-serveru>

[11] CWUT. *Testování programových systémů.* [online]. 2008 [cit. 2009-4-30] Dostupné z WWW: <http://www.student.cvut.cz/cwut/index.php/Testování_programových_systémů>

[12] The Code Project. *Applying Robustness Analysis on the Model-View-Controller (MVC) Architecture in ASP.NET Framework, using UML.* [online]. 2004 [cit. 2009-2-4] Dostupné z WWW: <http://www.codeproject.com/KB/architecture/ModelViewController.aspx>

[13] Buzzle.com. *Software Testing.* [online]. 2005 [cit. 2009-4-30] Dostupné z WWW: <http://www.buzzle.com/articles/software-testing/>

[14] Wikipedia, the free encyclopedia *Software deployment.* [online]. 2008 [cit. 2009-4-29] Dostupné: <http://en.wikipedia.org/wiki/Software_deployment>

[15] Zend Framework. *Zend Framework Quick Start.* [online]. 2008 [cit. 2009-1-13] Dostupné z WWW: <http://framework.zend.com/docs/quickstart>

[16] Zend Framework. *Programmer's Reference Guide.* [online]. 2008 [cit. 2009-1-15] Dostupné z WWW: <http://framework.zend.com/manual/en/>

[17] Sparx Systems. *UML Models.* [online]. 2009 [cit. 2009-1-6] Dostupné z WWW: <http://www.sparxsystems.com.au/resources/tutorial/uml_models.html>

[18] Pari - weblog o webdesignu a všem možném. *Začínáme se Zend Framework 1.5.* [online]. 2008 [cit. 2009-2-12] Dostupné z WWW: <http://pari.cz/293/zaciname-se-zend-framework-15>

[19] W3Schools. *ASP.NET Introduction* [online]. 2009 [cit. 2009-5-22] Dostupné z WWW: <http://www.w3schools.com/aspnet/aspnet_intro.asp>

[20] W3Schools. *ASP Introduction* [online]. 2009 [cit. 2009-5-22] Dostupné z WWW: <http://www.w3schools.com/asp/asp_intro.asp>

[21] W3Schools. *PHP Introduction* [online]. 2009 [cit. 2009-5-22] Dostupné z WWW: <http://www.w3schools.com/php/php_intro.asp>

[22] The PHP Group. *What can PHP do?* [online]. 2009 [cit. 2009-5-22] Dostupné z WWW: <http://cz2.php.net/manual/en/intro-whatcando.php>

PŘÍLOHY

Seznam příloh

Příloha A

Uživatelská rozhraní

Informační systém pro správu realit

Samotný informační systém je rozdělen na tři části. První část je veřejná a zbylé dvě části jsou chráněné přístupovými údaji.

Veškeré položky ve formulářích označené * jsou povinné a bez jejich vyplnění nebude provedena požadovaná akce.

Veřejná část informačního systému

Tato část informačního systému je orientována na vyhledávání, ale obsahuje i další funkcionality jako registrace, vložení nemovitosti, zaslání poptávky realitním kancelářím, apod. Obrázek 33 znázorňuje veřejnou část informačního systému.

Vyhledávání

Možnosti vyhledávání jsou následující:

- *Vyberete kraj* (interaktivní mapa) umožňuje vyhledávat podle zvoleného kraje (1)
- *Kategorie* (v levém menu) umožňuje vyhledávat podle kategorie nemovitosti (2)
- *Rychlý výběr* umožňuje vyhledávat podle více kritérií: kraj, druh nemovitosti, nabídka, minimální cena (3).

Registrace (4)

Registrace je určena pro realitní kanceláře a probíhá pomocí formuláře, kde je nutné vyplnit vyžadované údaje.

Vložit inzerát (5)

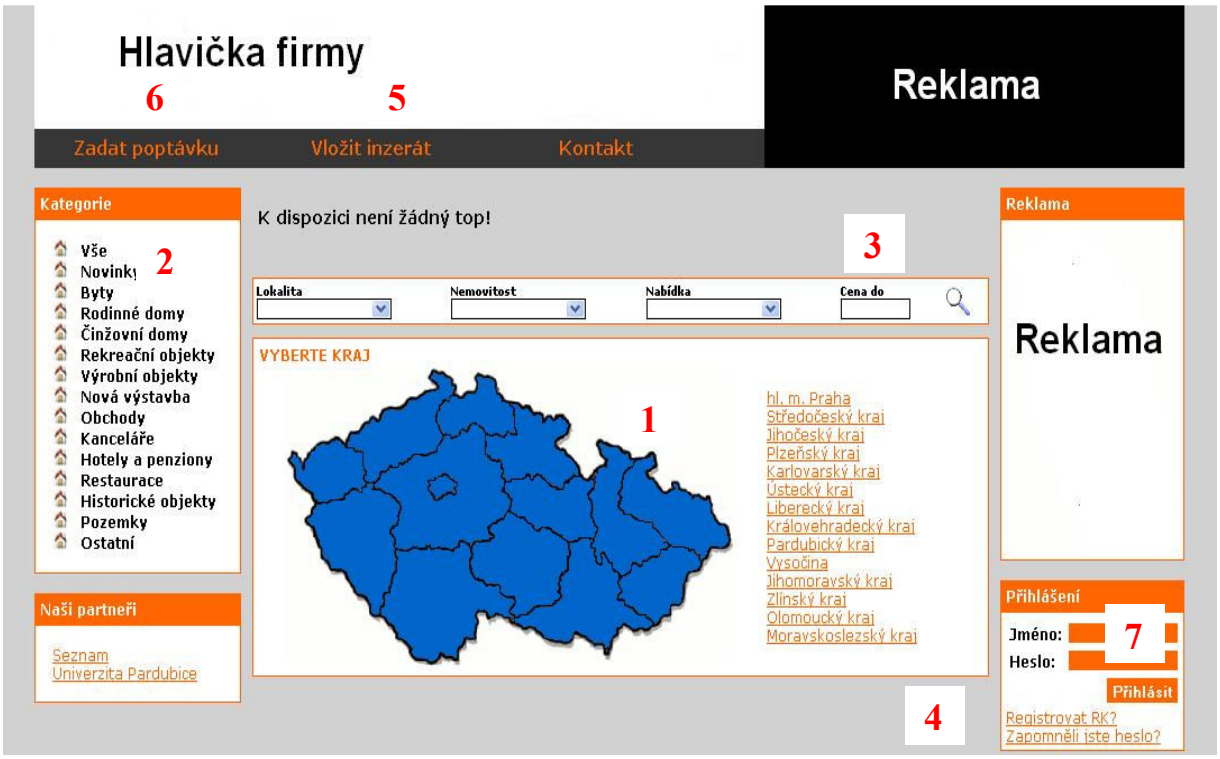
Jedná se o vložení nemovitosti do informačního systému hostem. Použit je vícekrokový formulář, ve kterém je nutno vyplnit potřebné údaje.

Zadat poptávku (6)

Návštěvník stránky neboli potencionální kupující nemovitosti má možnost po vyplnění formuláře zaslat poptávku po nemovitosti. Poptávka se odešle všem realitním kancelářím, které chtěly při registraci zasílat poptávku z kraje, který je určen u poptávky.

Přihlášení (7)

Je podmíněno úspěšnou registrací nebo vložením administrátorem. Je nutné znát přihlašovací jméno a heslo (zadáno při registraci), které přijde realitní kanceláři také na e-mail. Pokud některý z registrovaných uživatelů zapomněl heslo, lze zaslat nové pomocí odkazu *Zapomenuté heslo*, který se nachází pod odkazem *Registrace*.



Obrázek 33: Veřejná část systému. Zdroj: vlastní

Neveřejné části systému

RK Admin Centrum

V rámci *RK Admin Centra* lze zvolit z následujících možností v menu:

- *Správa nemovitostí* umožňuje vkládání, editování a mazání nemovitostí.
- *Správa makléřů* umožňuje vkládání, editování a mazání makléřů.
- *Správa údajů* umožňuje editovat určité údaje realitní kanceláře.
- *Zaslat výpis* odešle e-mail se seznamem nemovitostí dané realitní kanceláře na její e-mailovou adresu zadanou při registraci.
- *Změna hesla* mění heslo na nové, které je nutno zadat dvakrát (jednou řádně a podruhé pro kontrolu).
- *Odhlásit* ukončuje práci a přesměruje činnost na hlavní stránku veřejné části.

Obrázek 34 zachycuje úvodní obrazovku *RK Admin Centra*.

Obrázek 34: Úvodní obrazovka RK Admin Centra. Zdroj: vlastní

Admin Centrum

V *Admin Centru* lze zvolit z následujících možností umístěných vlevo v menu:

- *RK k aktivaci* obsahuje přehled nově registrovaných realitních kanceláří, které čekají na schválení.
- *Správa RK* slouží k zobrazení veškerých registrovaných realitních kanceláří v rámci informačního systému. Lze pouze zobrazit detail a smazat nekorektní realitní kancelář.
- *Správa hostů* slouží k zobrazení veškeré jednořádkové inzerce. Lze zobrazit údaje osoby, detail nemovitosti, případně smazat nekorektní nemovitost včetně inzerenta.
- *Správa nemovitostí* umožňuje vyhledávání nemovitostí podle kritérií, zobrazení nemovitosti a případné její smazání.
- *Správa adminů* umožňuje vkládání, editování a mazání administrátorů.
- *Správa bannerů* umožňuje vkládání, editování a mazání bannerů.
- *Správa TOPů* umožňuje editování, mazání topů a nastavení nemovitosti jako TOP.
- *Správa partnerů* umožňuje vkládání, editování a mazání partnerů.
- *Správa číselníků* umožňuje vkládání, editování a mazání položek jednotlivých číselníků.
- *Změna hes*la mění heslo na nové, které je nutno zadat dvakrát (jednou řádně a podruhé pro kontrolu).
- *Odhlásit* ukončuje práci a přesměruje činnost na hlavní stránku veřejné části.

Obrázek 35 zachycuje úvodní obrazovku *Admin Centra*.

Obrázek 35: Úvodní obrazovka Admin Centra. Zdroj: vlastní