XRVYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

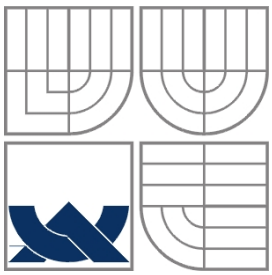
NÁVRH A ANALÝZA INFORMAČNÍHO SYSTÉMU PRO SPRÁVU POČÍTAČOVÉHO CENTRA

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

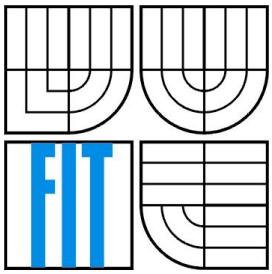
AUTOR PRÁCE
AUTHOR

PETR RENIERS

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁVRH A ANALÝZA INFORMAČNÍHO SYSTÉMU PRO SPRÁVU POČÍTAČOVÉHO CENTRA

DESIGN AND ANALYSIS OF COMPUTER CENTER MANAGMENT SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

PETR RENIERS

VEDOUCÍ PRÁCE
SUPERVISOR

ING. JIŘÍ KRAJÍČEK

BRNO 2008

Abstrakt

S příchodem internetu se začínaly objevovat nové typy webových aplikací a systémů, které byly vytvořené na standardních technologiích, a proto jsou dostupné na většině uživatelských počítačů. Bez těchto systémů si v dnešní době nedokážeme svět ani představit. Mezi uživateli a vývojáři si získali velkou oblibu a jsou vyvíjeny jak pro internetové aplikace, tak i pro firemní intranetové aplikace. Vzhledem k příchodu objektově orientovaného návrhu a návrhových vzorů je důležité si tyto moderní vývojové nástroje otestovat na vývoji informačního systému. Právě touto problematikou se zabývá tato práce. Vyvíjím informační systém pro správu počítačového centra, který slouží k lepší přehlednosti a správě firem.

Klíčová slova

Analýza, návrh, UML, návrhové vzory, MVC, model, pohled, řadič, objektová orientace

Abstract

With arrival of the internet new types of easily accesible web applications and systems developed on standard technologies have begun to emerge. Nowadays we cannot even imagine the world without these systems. They have garnished massive popularity among users and developers while being created for the internet applications as well as corporate intranet applications. Considering an emergence of object oriented desing and design patterns it is important to test these modern tools on developement of an information system. This thesis acts just this problem. I develope computer center managment system, which serves for better management of firms.

Keywords

Analysis, design, UML, design patterns, MVC, model, view, controller, object orientation

Citace

Reniers Petr: Návrh a analýza informačního systému pro správu počítačového centra. Brno, 2008, bakalářská práce, FIT VUT v Brně.

Návrh a analýza informačního systému pro správu počítačového centra

Prohlášení

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

……
Petr Reniers
14.5. 2008

Poděkování

V této sekci bych chtěl poděkovat Ing. Jiřímu Krajíčkovi za ochotu a čas k vedení této práce a za plnohodnotné připomínky, které vedly ke vzniku tohoto textu.

Obsah

1 Úvod

Čas jsou peníze a to ví i majitelé a manageři firem a společností. Účelem této práce je poskytnout firmám o desítkách až stovkách uživatelů možnost správy počítačového centra, která by ušetřila každodenní schůzky a porady, které jsou mnohdy zbytečné, možnost prohlížení firemních informací pro vedoucí a managery, lepší přehled o zaměstnancích apod. I ve firmě, která má stovky obyčejných zaměstnanců, vedoucích a managerů, nic takového není a nastává mnohdy zmatek a chaos. To se dá vyřešit informačním systémem. Šetří to tedy čas, peníze a i nervy.

Při vývoji informačního systému použiji objektově orientovaný "formální" návrh systému. Základem bude návrh, který bude postaven na softwarové architektuře Model View Controller.

Dále zmíním strukturu technické zprávy. Ve druhé kapitole jsou popsány konkrétní požadavky na informační systém. Ve třetí kapitole je podle požadavků důkladně navrhnut tento informační systém. Do detailů jsou rozebrány modelovací techniky UML. Pro návrh systému jsem použil různé typy diagramů. Přímo tyto diagramy jsou zobrazeny v této kapitole. Ve čtvrté kapitole jsou popsány technologie použité při vývoji. Pro implementaci této aplikace jsem zvolil kombinaci programovacího jazyka PHP a databázového systému MySQL. Nalezneme zde také i základní informace o ostatních použitých prostředcích. Pátá kapitola se zabývá možnými nápady pro vylepšení a rozšíření. V šesté, závěrečné kapitole se podíváme na zhodnocení práce a přínosu pro mě.

2 Analýza požadavků

Analýza požadavků je jednou z nejdůležitějších součástí procesu vývoje softwaru. Přesně specifikované požadavky nám pomáhají optimalizovat náklady (časové i finanční) na vývoj softwaru a jsou dobrým předpokladem vytvoření správnosti softwaru. Analýza požadavků analyzuje a definuje požadavky na software. Na základě analýzy je možné sestavit seznam požadavků, které musí informační systém splňovat a které uvádím v další podkapitole.

2.1 Požadavky na informační systém

- Přihlášení zaměstnance
- Prohlížení informací vlastního účtu
- Prohlížení informací o ostatních zaměstnancích
- Odesílání zpráv na Helpdesk (podpora problémů)
- Prohlížení stavu projektu
- Vyplňování reportů
- Zobrazení úkolů
- Měnění informací o zaměstnancích
- Měnění stavu projektu
- Rozdělování úkolů projektu
- Zadávání projektů
- Přijímání zpráv na Helpdesk
- Odpovídání zpráv na Helpdesk
- Spravování databáze
- Registrace zaměstnance
- Uchování historie projektů
- Uchování historie zpráv helpdesku
- Neumožnění měnit informace nadřízeného
- Smazání zaměstnance ze systému

2.2 Role uživatelů

Navrhovaný informační systém rozlišuje čtyři role uživatelů. Nejvýše postavený je IT manager (mohlo by se říct i třeba správce systému). Ten má nejvíce oprávnění. Dále může být v systému libovolný počet managerů jednotlivých oddělení firmy. Pod managerama jsou vedoucí, kteří podléhají managerům. A nakonec, nejníže postavený je zaměstnanec. Jednotlivé možnosti a práva každého z nich uvedu níže. Samozřejmostí je, že jak IT manager, tak i ostatní manageři a vedoucí jednotlivých oddělení jsou zároveň zaměstnanci.

1. Zaměstnanec
    - Každý zaměstnanec se může přihlásit do systému
    - Dále může prohlížet informace o sobě
    - Každý zaměstnanec může prohledávat seznam jeho kolegů. Vyhledávat jejich kontaktní údaje kromě soukromých informací
    - Každý zaměstnanec může zasláním zprávy na helpdesk požádat o podporu při nějakém problému
    - Taktéž může prohlížet stavy (průběh) projektů a zobrazovat úkoly, které mu byly přiřazeny
    - Poslední možností je vyplňování reportů a jejich odeslání

2. Vedoucí:
    - Každý vedoucí může prohlížet všechny informace o svých podřízených a může tyto údaje měnit
    - Taktéž může měnit stav (průběh) zadaného projektu
    - Navíc rozděluje a posílá úkoly svým podřízeným, co se týkají projektu

3. Manager
    - Všichni manageři můžou zadávat projekty
    - Můžou měnit všechny informace o svých vedoucích

4. IT Manager (správce systému)
    - IT manager spravuje databázi
    - Přijímá zprávy na helpdesk od zaměstnanců
    - Následně může odpovídat na zprávy na helpdesk
    - Dále registruje uživatele a dělá celkovou správu uživatelů jako např. smazání uživatele apod.

2.3 Požadavky na uživatelské rozhraní

Uživatelské rozhraní (prostředí) v informačním systému by mělo být jednoduché, přehledné a intuitivní. Součástí systému by měla být jasná a srozumitelná nápověda, protože systém budou používat i obyčejní uživatelé a nejen počítačový experti. Proto musí být systém navržen tak, aby jej mohl hned intuitivně uživatel využívat a nebylo potřeba většího zaškolení. S kvalitním uživatelským prostředím se totiž snižují i náklady na školení zaměstnanců pro práci s informačním systémem.

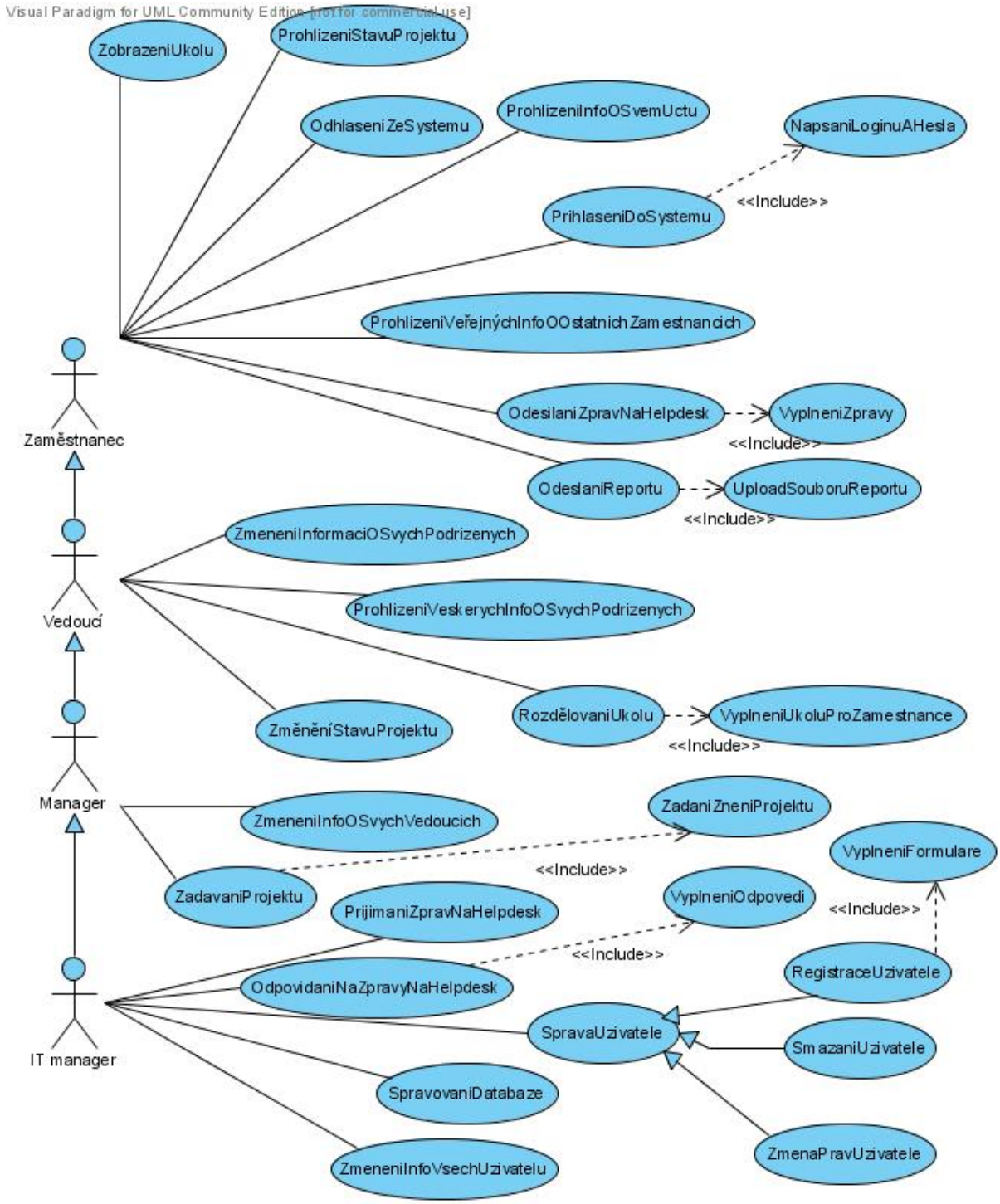
2.4 Use Case diagram

S odkazem na [1]. Use case diagram je zobrazení dynamické (funkční) struktury systému z pohledu uživatele. Je primárně určen k definici chování systému bez toho, že by odhaloval jeho vnitřní strukturu. Je to soubor scénářů pro používání systému. Každý scénář obsahuje:

- sekvenci (posloupnost) událostí, které v jeho rámci probíhají (včetně případných variant)
- popis interakce (komunikace) mezi uživatelem (aktorem) a systémem

Využití:

- specifikace požadavků na systém (podklad pro analýzu a návrh)
- komunikace se zákazníkem (uživatelem systému)
- podklad pro řízení projektu
- tvorba testovacích případů pro fázi testování systému (před uvedením do provozu)
- návrh uživatelského rozhraní (mezi každým use case a aktorem má být rozhraní)



Obrázek 2.1 – Use Case diagram informačního systému pro správu počítačového centra.

3 Návrh

Původní vývoj softwaru začal prostým programováním. Jak rostla velikost systému a zkušenosti programátorů, tak si lidé uvědomili, že prostý zápis kódu aplikace nedostačuje. I kdyby taková aplikace fungovala, její kód by byl hůře udržitelný a rozšiřitelný. Seznamme se proto nejprve s návrhem.

Návrh softwaru umožnil vytvořit plán racionálního uspořádání kódu ještě před vytvořením prvního řádku do kódovací desky. Tato radikální změna dokonce umožnila lidem odhadnout potenciální problémy se správou už ve fázi přípravy podkladů.

Aby byly naplněny požadavky uživatele, byla také zavedena uspořádanější a přesnější analýza. Po celou historii softwaru se lidé snažili dosáhnout možnosti opakovaného používání kódu. Bohužel však většina procedurálních jednotek kódu není natolik uzavřena sama do sebe, aby je bylo možné nezávisle opakovaně používat. Avšak prostřednictvím objektové orientace máme další možnost dosáhnout opakovaného využívání. Historie objektové orientace je podobná hlavnímu softwarovému proudu. Práce s objekty však od implementace k abstrakci dosáhla nebývalého tempa. Objektově orientované programování nabylo na popularitě poprvé v 80. letech. Totéž desetiletí zažilo uvedení jak objektově orientovaného návrhu, tak i objektově orientované analýzy [2].

V této práci se budu zabývat tedy objektově orientovaným návrhem a následně i objektově orientovanou implementací.

3.1 Objektová orientace

Než se podíváme na samostatný návrh v UML, tak popíšu nejdříve stavební kameny objektové orientace s odkazem na [2].

Objekty – spojují data a funkcionalitu společně do jednotek zvaných objekty, ze kterých se potom skládá výsledný objektově orientovaný program (na rozdíl od strukturovaného složeného z procedur a funkcí). Objekty jsou tedy základní jednotkou modularity i struktury v objektově orientovaném programu, který umožňuje problém intuitivně rozdělit na přímo realitě korespondující podčásti a díky jejich vzájemné komunikaci i tento problém řešit.

Abstrakce – je schopnost programu ignorovat některé aspekty informací, se kterými pracuje. Abstrakce je pohled na vybraný problém reálného světa a jeho počítačové řešení. Při vytváření takovéto abstrakce je vhodné mít možnost skrývat detaily do jakési černé skříňky, která je pro okolí definovaná pouze svým rozhraním, přes které komunikuje s okolím, a nikoli vnitřními detaily implementace, která může být podstatným zjednodušením reálného světa.

Zapouzdření – zajišťuje již na úrovni definice sémantiky jazyka, tak že uživatel nemůže měnit interní stav objektů libovolným (tedy i neočekávaným) způsobem, ale musí k tomu využívat poskytované rozhraní (operace nad objektem). Každý objekt tedy nabízí protokol, který určuje, jak s ním mohou ostatní objekty komunikovat. Ostatní objekty se tedy při komunikaci s tímto objektem spoléhají pouze na jeho externí (poskytované) rozhraní a skutečná implementace zůstává dokonale skryta. Právě tento koncept zásadně zjednodušuje vývoj nových vlastností a vylepšování stávající implementace našeho programu, protože nám stačí zachovat pouze zpětnou kompatibilitu rozhraní objektů a nikoli skrytých implementací.

Polymorfismus – je mnohotvárnost využívající mechanismus zaslání zpráv. Namísto běžného volání podprogramů (procedur a funkcí) ve strukturovaném programování se v objektově orientovaném jazyce využívá konceptu zasílání zpráv. Konkrétní použitá metoda reagující na zaslání zprávy závisí na konkrétním objektu, jemuž je tato zpráva zaslána. Například, pokud máme objekt *orel* a pošleme mu zprávu *rychle_se_přemísti*, tak implementace reagující metody bude pravděpodobně obsahovat příkazy pro roztažení křídel a vzlétnutí. Pokud ale budeme mít objekt *gepard*, tak implementace metody volané při obdržení tytéž zprávy bude obsahovat například příkaz pro rozběhnutí se. Obě reakce na zaslání stejné zprávy byly uzpůsobeny potřebám konkrétního objektu, který zprávu obdržel, a tudíž byly plně v jeho vlastní režii. Takto získáme polymorfismus, a tak může zaslání stejné zprávy vyvolat rozdílné reakce během různých kontextů.

Dědičnost – je způsob, jak implementovat sdílené chování. Nové objekty tak mohou sdílet a rozšiřovat chování těch již existujících bez nutnosti všechno znovu reimplementovat. Dědičnost se v praxi využívá ke dvěma účelům: k indikaci, že nové chování specializuje jiné již existující chování a pro sdílení kódu. Tato vlastnost je tedy velmi důležitá pro udržovatelnost a rozšiřitelnost objektově orientovaných systémů.

3.2 UML

Veškeré návrhy a analýza budou prováděny pomocí prostředků UML, který se stává standardem v oblasti objektového návrhu.

Na konci 90. let Grady Booch, Ivar Jacobson a Jim Rumbaugh přišli s definitivní verzí unifikovaného modelovacího jazyka, který je nyní standardem přijatých kromě jiných také skupinou OMG (Object-Management Group). Zápis UML se podobá zápisu techniky modelování objektu (Object Modeling Technique – OMT), kterou Rumbaugh a další vyvinuli na začátku 90. let [2].

UML (Unified Modeling Language) je grafický jazyk pro vizualizaci, specifikaci, tvorbu a dokumentaci prvků softwarových systémů. UML je však využitelný i pro business modelování a pro modelování ne-SW systémů. V UML lze modelovat jakýkoliv typ aplikace běžící na jakémkoliv typu HW. Lze také modelovat SW nezávisle na operačním systému a programovacím jazyku. Pro UML je nejpřirozenější modelování pro objektově orientované prostředí, ale lze modelovat v jakémkoliv jiném jazyku. [2]

V současné době má jazyk UML největší význam při návrhu softwarových systémů, protože objektově orientovaný návrh každé složitější aplikace je nezbytným předpokladem pro její úspěšnou a rychlou implementaci. [2]

Pro objektově orientovaný návrh je samozřejmě možné použít různé podpůrné prostředky (zejména další odlišné typy diagramů). UML je však významné také v tom ohledu, že přesně specifikuje, co má daný diagram obsahovat, což je velmi důležité zejména při sdílení informací mezi jednotlivými analytiky a vývojáři. Dále je již z principu UML nutné, aby vytvářené grafy měly vnitřní konzistenci a přesně danou sémantiku. UML diagramů existuje několik typů lišících podle toho, jaké se pomocí nich plánují či zpracovávají úlohy. Tyto diagramy se od sebe odlišují především repertoárem použitých značek, způsobem jejich vzájemného propojení a s nimi související sémantikou. [2]

3.3 Softwarová architektura

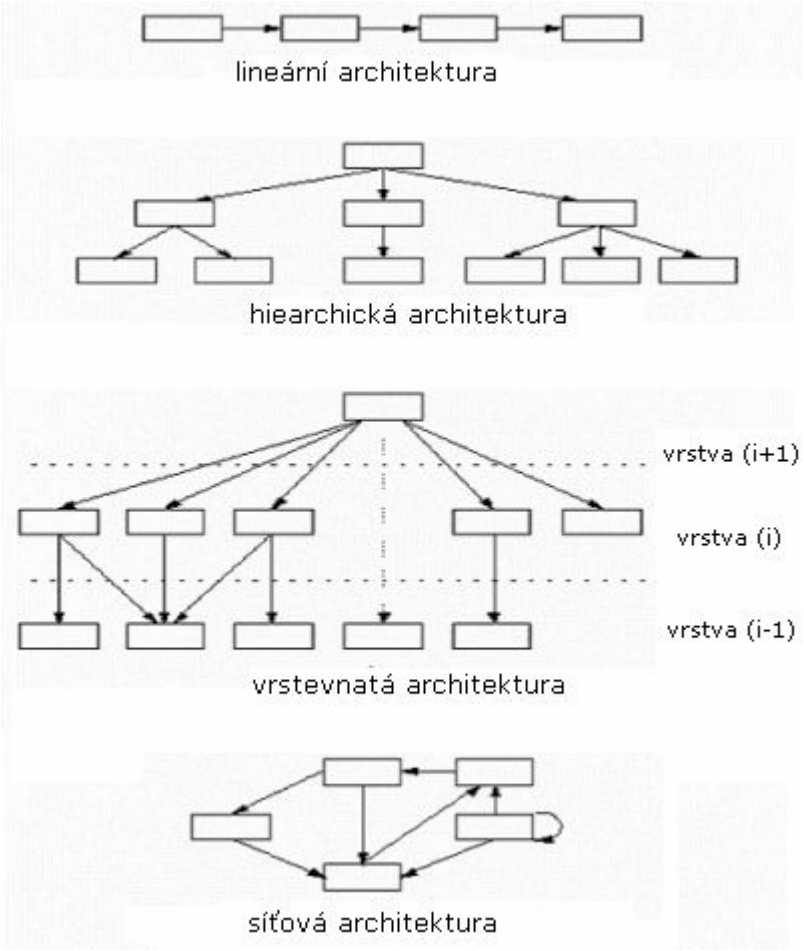
S rostoucí velikostí a složitostí softwarových systémů se návrh a specifikace celkové struktury systému stává významnějším problémem než volba algoritmů a datových struktur. Softwarová architektura je struktura komponent programu/systému, jejich vzájemné vazby, principy a předpisy určující jejich návrh a vývoj v průběhu času.

Z pohledu našeho zájmu lze činnosti prováděné při návrhu programového systému rozdělit do několika skupin. Jednu skupinu tvoří činnosti, které provádějí dekompozici systému do dekompozičních jednotek (míněno z pohledu návrhu) v potřebném počtu úrovní dekompozice. Činnosti, které provádějí specifikaci těchto dekompozičních jednotek. Činnosti navrhující spolupráci těchto dekompozičních jednotek. Tedy činnosti jednoznačně zřejmé, když se hovoří o návrhu. Pracovně tyto činnosti nazvěme vlastní návrh. Vlastní návrh má jednoznačnou návaznost na analýzu vyvíjeného systému v tom smyslu, že navrhuje realizaci analytického modelu vyvíjeného systému. Architektura programového systému má zásadní vliv jak na jeho fungování, tak na jeho vývoj, údržbu a další rozvoj. Uvádím zde stručný rozbor přínosů a možností, které softwarová architektura nabízí:

- dokumentace softwarové architektury představující jednotný přístup k strukturálním otázkám je nesmírně přínosná pro celý další život systému, tj. jeho vývoj a (hlavně) údržbu.

- softwarová architektura by měla systematicky odrážet kvalitativní nároky na vyvíjený systém (např. udržovatelnost, rozšiřitelnost) a naopak analýzou softwarové architektury lze o systému mnohé říci, tedy jde o jednoznačný přínos k zajišťování jakosti softwarového procesu a produktu.
- vědomé stanovení softwarové architektury nutí k jednotnému a vnitřně konzistentnímu učinění těch nejzásadnějších rozhodnutí designu.
- stanovená a dokumentovaná softwarová architektura umožní její sdělení všem zainteresovaným stranám při vývoji a údržbě softwarového systému.

Obrázek 3.1 (podle zdroje [21]) znázorňuje typy softwarových architektur.



Obrázek 3.1 – Typy architektur

3.4 Návrhové vzory

Základním kamenem pro návrhové vzory v oblasti informační technologie se stalo dílo Design Patterns: Elements of Reusable Object-Oriented Software. Autorem bylo seskupení zvané Gang of Four (pánové Gamma, Helm, Johnson a Vlissides) a poprvé bylo představeno na OPSLA´94, kde se

setkalo s velkým úspěchem. Tato práce je nazývána základní knihou v oblasti návrhových vzorů a dodnes se jedná o velmi uznávané dílo. Většina dnešních děl o návrhových vzorech vychází z rozdělení, které v tomto díle bylo popsáno. Vzhledem k tomu, že tato kniha si udržuje svoji hodnotu i v rychle se vyvíjejícím světě informačních technologií, jedná se opravdu o unikátní dílo. [3]

Návrhové vzory jsou doporučené postupy řešení často se vyskytujících úloh. Můžeme je považovat za vzory, které použijeme při návrhu architektury budoucí aplikace. Výrazně se sníží pravděpodobnost chyb, které uděláme, když bychom řešení sami vymysleli. Současně si často ulehčíme budoucí práci, protože návrhové vzory již dopředu počítají s typickými rozšířeními. Takže budoucí úpravy a rozšíření dají mnohem méně práce a výsledné programy budou navíc spolehlivější.[4]

Vzory lze pochopitelně využít i v jiných fázích vývoje softwaru než je návrh. Existují například analytické vzory používané ve fázi analýzy. Tato práce využívá softwarovou architekturu Model – View – Controller (MVC). O této architektuře si řekneme více v další kapitole.

S odkazem na [20] zmíníme v přehledu návrhové vzory. Toto rozdělení vychází z výše zmíněné knihy Design Patterns: Elements of Reusable Object-Oriented Software. Autoři vymezili tři základní skupiny návrhových vzorů, které jsou stále uznávané. Návrhově vzory se rozdělují do tří skupin a to: Creational Patterns (vytvářející), Structural Patterns (strukturální) a Behavioral Patterns (vzory chování).

Creational Patterns řeší problémy související s vytvářením objektů v systému. Snahou těchto návrhových vzorů je popsat postup výběru třídy nového objektu a zajištění správného počtu těchto objektů. Většinou se jedná o dynamická rozhodnutí učiněná za běhu programu. Mezi tyto návrhové vzory patří: Factory Method Pattern, Abstract Factory Method Pattern, Builder Pattern, Prototype Pattern, Singleton Pattern.

Structural Patterns představují skupinu návrhových vzorů zaměřujících se na možnosti uspořádání jednotlivých tříd nebo komponent v systému. Snahou je zpřehlednit systém a využít možností strukturalizace kódu. Mezi tyto návrhové vzory patří: Adapter Pattern, Bridge Pattern, Composite Pattern, Decorator Pattern, Facade Pattern, Flyweight Pattern, Proxy Pattern.

Behavioral Patterns se zajímají o chování systému. Mohou být založeny na třídách nebo objektech. U tříd využívají při návrhu řešení především principu dědičnosti. V druhém přístupu je řešena spolupráce mezi objekty a skupinami objektů, která zajišťuje dosažení požadovaného výsledku. Mezi tento typ vzorů můžeme zařadit: Chain Of Responsibility Pattern, Command Pattern, Interpreter Pattern, Iterator Pattern, Mediator Pattern, Memento Pattern, Observer Pattern, State Pattern, Strategy Pattern, Template Pattern, Visitor Pattern.

3.5 Architektura MVC

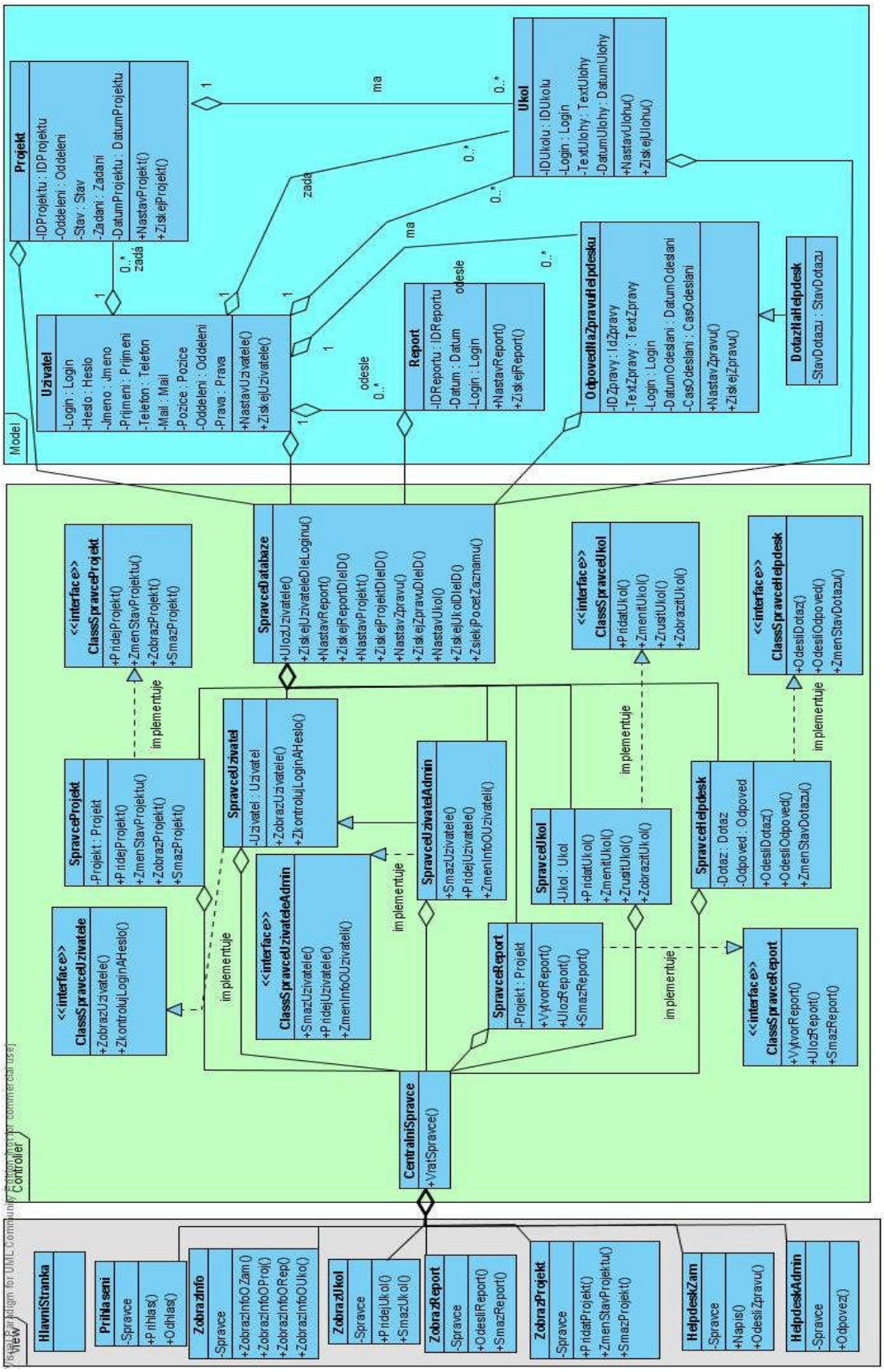
Hlavním bodem této práce a návrhu je softwarová architektura Model – View – Controller (Model – Pohled – Řadič). Tato architektura je ve velké míře využívána při vytváření prezentačně orientovaných systémů a může být publikována jako návrhový vzor.

MVC odděluje část programu mající na starost před zpracováním příkazu uživatele od části zabezpečující logiku programu, která uživatelovy příkazy zpracovává a části, která má na starosti zobrazení výsledků. MVC se stává ze tří druhů objektů: model představuje objekt aplikace, view (pohled) představuje prezentace na obrazovce a controller (ovládání) definuje způsob, jak rozhraní reaguje na vstup uživatele. MVC rozděluje objekty systému a tím zvyšuje flexibilitu a znovu-použitelnost celého řešení. Do části zabývající se prezentací výsledků patří nejenom vlastní GUI ale často i kód, který data na tuto prezentaci připravuje. Aplikace tohoto vzoru umožňuje snadnou implementaci zadávání požadavků různými způsoby (klávesnice, myš, hlas). Stejně jako různé možnosti prezentace výsledků (tabulka, graf, mluvené slovo). Další výhodou takovéhoto rozdělení je usnadnění případných budoucích změn. Aplikace vzoru usnadňuje přenositelnost mezi platformami.

Vzor je spíše obecným doporučením, které jednotlivé vazby nijak konkrétně nespecifikuje. V jednodušších aplikacích se některé části často slučují. Obdobně je to i při použití vzoru v jiných situacích. Např. pokud navrhneme lepší algoritmy na zpracování dat, tak inovujeme model a zachováme-li komunikační rozhraní se zbylými dvěma částmi, uživatel se o tom vůbec nedozví. Dostali jsme požadavek zobrazovat data v různých formátech. Definujme čtyři různé pohledy a s využitím vzoru nechme na uživateli, aby si vybral, který je pro něj v dané chvíli optimální. Oddělíme-li dobře jednotlivé části aplikace, můžeme získat aplikaci, která je velice snadno přenositelná mezi platformami. Aplikace mohou být naprogramovány na platformu JAVA SE a aplikace, která pro zobrazení používá knihovnu TVARY, lze prostou výměnou této knihovny převést na platformu JAVA ME, takže si je pak budeme moci pouštět třeba na mobilním telefonu. [4]

Tento návrhový vzor je takovým obecným doporučením, které je možné implementovat různými způsoby. Vzor například nijak nespecifikuje, jak přesně má komunikace mezi jeho jednotlivými částmi probíhat. V některých aplikacích posílá model v zobrazovacím modulu informace o tom, kdy a jak je má zobrazit. Jindy si naopak zobrazovací modul o tyto informace sám říká modelu. Jak jsem již uváděl, v řadě jednodušších aplikací se někdy zobrazovací a ovládací část slučují do jedné třídy. Protože použité grafické uživatelské rozhraní k tomu přímo vybízí. Stručně řečeno: nelze najít žádná přesná implementační doporučení. Pouze obecné zásady doporučující tyto tři činnosti oddělit, abych pak mohl s minimální námahou změnit kteroukoli z nich, aniž bych tím ovlivnil zbylé dvě. Taktéž často se používá aplikace vzoru pozorovatel. Pohled se přihlásí u modelu jako jeho pozorovatel a model jej v případě jakékoliv změny na tuto skutečnost upozorní. Pohled si pak od modelu vyžádá potřebné informace a aktualizuje výstup [4].

Na další straně je obrázek 4.1, který ukazuje výsledné navržení systému v architektuře MVC. Začnu popisem části zvané Model. V této části je 6 tříd. A to Uživatel, který může odesílat Report, ten představuje další třídu. Poté Uživatel může zadávat Projekt a Úkol, což jsou další dvě třídy. V neposlední řadě Uživatel komunikuje s třídou OdpověďNaZprávuHelpdesku a DotazNaHelpdesk. Je také využito dědičnosti mezi třídami OdpověďNaZprávuHelpdesku a DotazNaHelpdesk. Každá z těchto tříd má své atributy. Všechny tyto třídy komunikují se třídami části zvané controller pomocí třídy SprávceDatabáze, což je třída controlleru. Třídami controlleru jsou SprávceÚkol, SprávceUživatel, SprávceProjekt, SprávceHelpdesk, SprávceUživatelAdmin, SprávceReport a každá z těchto tříd implementuje vlastní interface, který má stejné metody. Každá metoda je ve třídě vyjádřena názvem, který přibližuje její chování. Např. metoda *ZkontrolujLoginAHeslo* má na starost kontrolu hesla a loginu přihlašujícího uživatele. Pro spojení a vymezení tříd části zvané view a části controller je tu třída controlleru CentrálníSprávce. V části view jsem navrhnul 8 tříd. Např. třída Přihlášení je zobrazení přihlašovacího okna anebo třída ZobrazProjekt je zobrazení projektu podle ID na obrazovce.

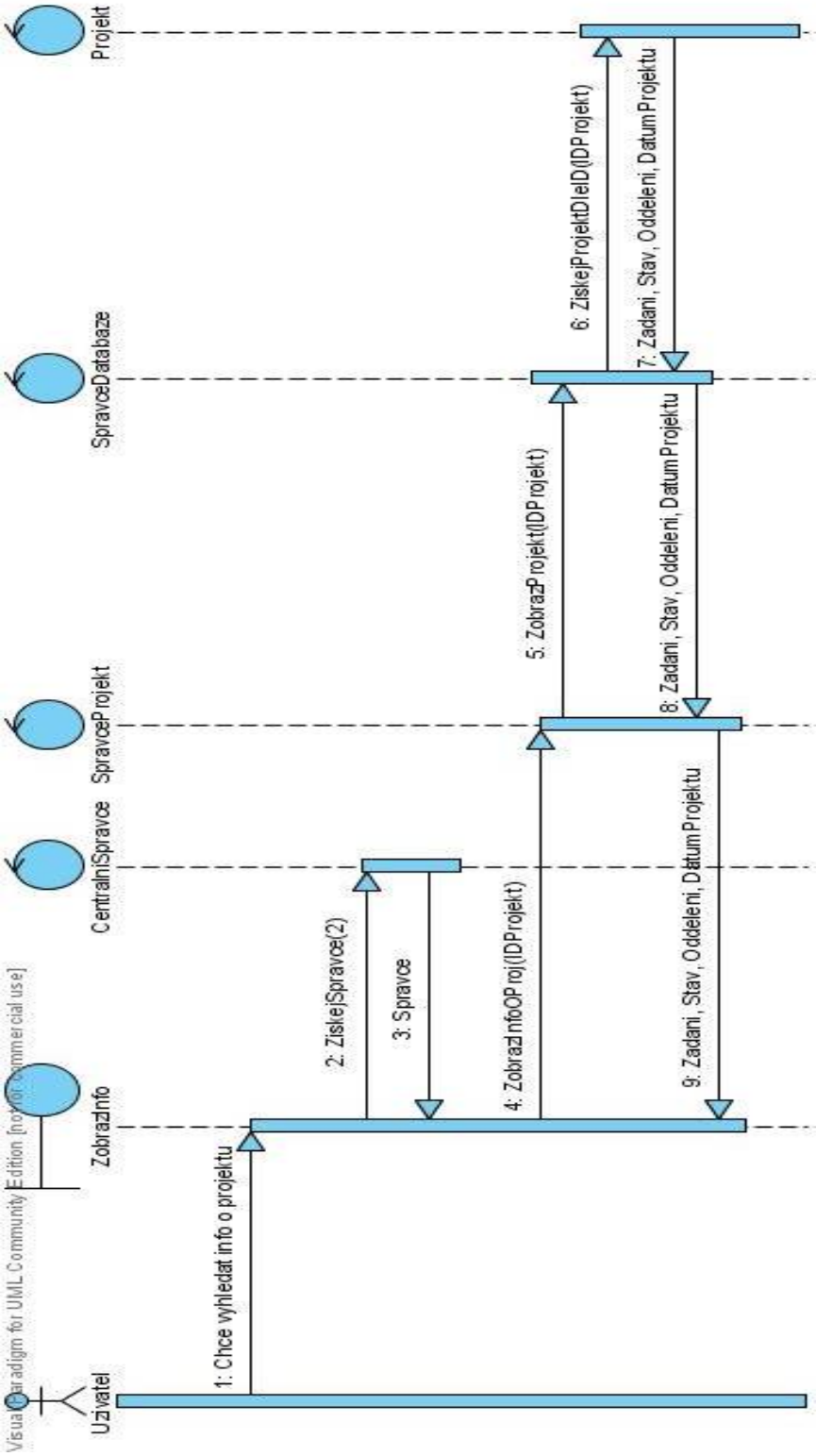


Obrázek 3.2 – Diagram tříd navrženého systému.

3.6 Sekvenční diagram

Pro názornost a návrh časování informačního systému jsem si vybral jeden z diagramů jazyka UML a to diagram sekvenční. Je to typ diagramu interakce objektu, který zdůrazňuje časovou posloupnost vztahů mezi statickými objekty. S pomocí sekvenčního diagramu je možné jednoznačně určit, jaké zprávy si objekty zasílají. Kdykoli se stanou procedurální nebo časovací záležitosti nejasnými, je vhodné použít tento diagram pro vyjádření časové posloupnosti vztahů. Jak ukazuje sekvenční diagram, čas se zvyšuje směrem dolů po svislé ose a seznam objektů, které budou přijímat zprávy, se rozšiřuje podél vodorovné osy horní části diagramu. Tyto objekty mohou být uvedeny pod svými názvy tříd, je-li to dostatečně přesné. Tělo diagramu zachycuje aktivované operace s velikostí odrážející jejich přibližné trvání: symbol aktivované operace svisle přibližně proporcionálně odpovídá době, po kterou je operace aktivní. Šipky mezi operacemi popisují zprávy. Volitelně použitelná sekvenční čísla na zprávách vyznačují časovou posloupnost.[3]

Na následující straně je obrázek 4.2, který ukazuje sekvenční diagram, který demonstruje vyhledání informací zadaného projektu v rámci MVC architektury. Uživatel chce vyhledat informace o projektu. View ZobrazInfo nejdříve získá Správce metodou *ZískejSprávce*(2). Poté pomocí view ZobrazInfo a metody *ZobrazInfoOProj*() si vyžádá zobrazení informací o projektu podle zadaného ID přímo třídě controlleru SprávceProjekt. V dalším kroku SprávceProjekt zavolá funkci *ZobrazProjekt*(), která v další třídě controlleru SprávceDatabáze zavolá metodu *ZískejProjektDleID*(). Tato funkce poté v třídě modelu zpět pošle ostatní atributy (Zadání, Stav, Oddělení, DatumProjektu) projektu dle dříve zadaného ID. Poté tyto atributy se navrací zpět až view ZobrazInfo. A uživateli se zobrazí hledané informace o projektu.



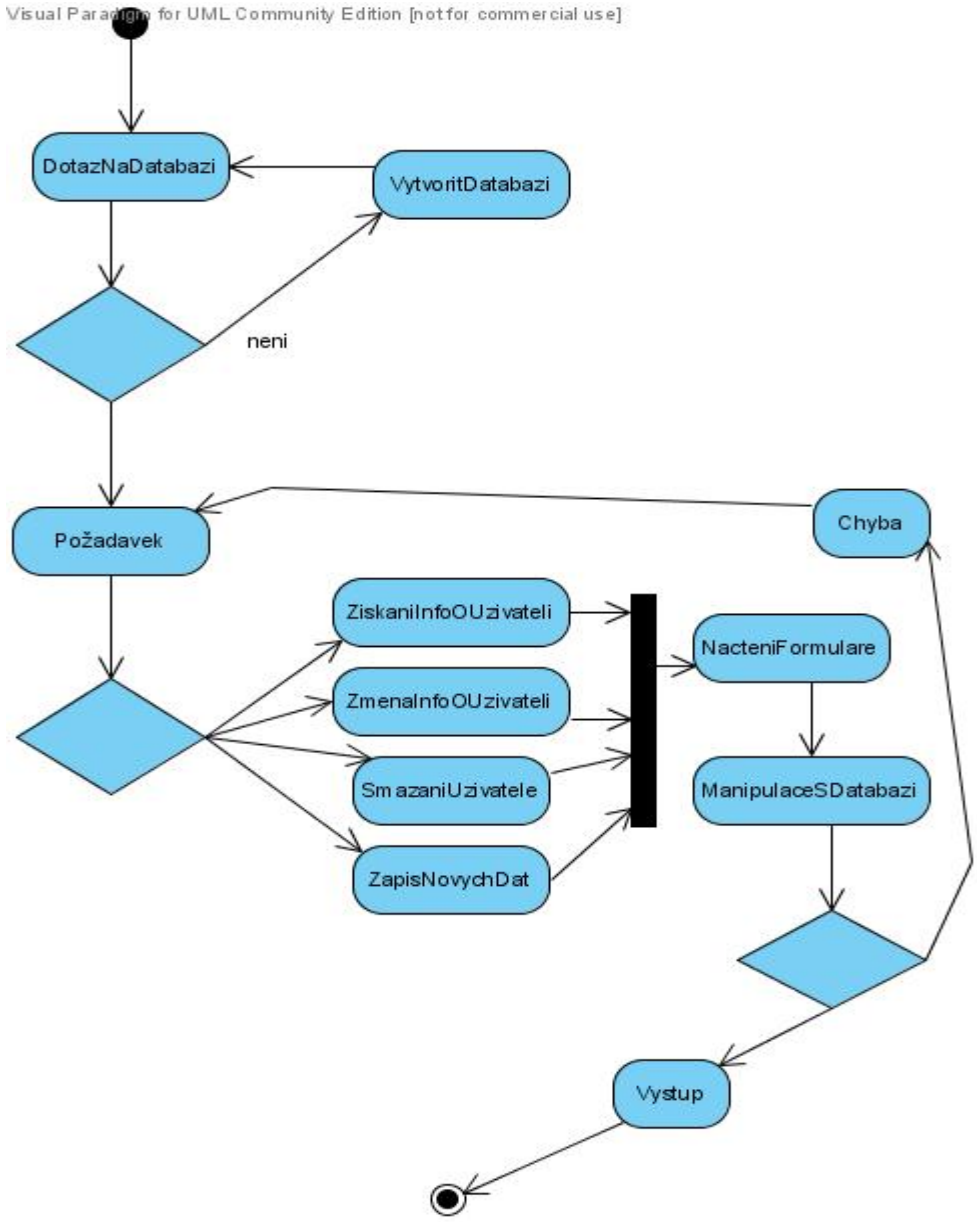
Obrázek 3.3 – Sekvenční diagram.

3.7 Diagram aktivit

S odkazem na [5]. Diagram aktivit je reprezentace struktury (dynamiky) počítačových a organizačních procesů v systému, zaměřená především na jeho vnitřní chování. Je to zobrazení řídících toků (přechodů) mezi akcemi (aktivitami) v systému od počátečního bodu po jeden nebo více koncových bodů. Důraz se klade na zobrazení pořadí aktivit. Využití:

- modelování průběhu jednotlivých Use Case a operací v třídách
- modelování podnikových procesů (business process modeling, business modelování)
- workflow

Následující obrázek 4.3 demonstruje diagram aktivit části navrženého systému pro akce spojené s databází a tabulkou uživatel. Po vstupním bodu je akce *DotazNaDatabázi*. Poté je rozhodnutí o existenci databáze, a pokud databáze neexistuje, nastane akce *VytvořitDatabázi*. Pokud databáze existuje, tak se přikročí k *Požadavku* na databázi. Poté zde může nastat několik akcí – *ZískáníInfoOUživateli*, *ZměnaInfoOUživateli*, *SmazáníUživatele*, *ZápisNovýchDat*. Po některé z těchto akcí se *NačteFormulář* a zkontroluje se *ManipulaceSDatabází*. Pokud není manipulace v pořádku, tak nastane akce chyba a přejde se zpět k akci *Požadavek*. Pokud je manipulace v pořádku, tak následuje *Výstup*.



Obrázek 3.4 – Diagram aktivit.

4 Implementace

Nyní je proveden již návrh informačního systému, a proto se podíváme na možnosti vytváření dynamických webových stránek. Navrhovaný systém bude vyvíjený jako webová aplikace. Postupně vyjmenuji a popíši stručný přehled technologií, které jsem použil při implementaci. Rozlišujeme dva hlavní přístupy: technologie na straně klienta a technologie na straně serveru. S odkazem na [8] si uděláme přehled těchto technologií.

4.1 Technologie na straně klienta

Technologie na straně klienta jsou velice často inovovány. Pro implementaci webových stránek je to veliká nevýhoda, neboť správce webového serveru nemůže ovlivnit software u každého klienta. Firmy chtějí obsáhnout největší množství uživatelů s různými prohlížeči. Tato skutečnost brání přijímání nových technologií, protože je podporují pouze nejnovější prohlížeče. Obecně se tyto technologie hodí pro nepříliš složité stránky, které nevyžadují speciální funkčnost.

Technologie na straně klienta přináší jednu výhodu. Jelikož jsou stránky zobrazovány (graficky interpretovány) na klientově systému, nezatěžují server. Jedná se např. o kontrolu formulářů, změny vzhledu stránek atd. Možné technologie jsou

- JavaSript
- XHTML
- Cascading Style Sheets (CSS)

4.2 Technologie na straně serveru

Technologie na straně serveru nabízí, oproti výše uvedeným způsobům, řadu výhod. Dochází k minimalizaci provozu na síti, díky omezení potřebné komunikace prohlížeče a serveru. Server zpracuje požadavek, vykoná potřebné úkony a odešle čistou XHTML stránku. To v konečné fázi znamená rychlejší načtení stránky a vyřešení problému s kompatibilitou prohlížečů. Dnes již by každý prohlížeč měl podporovat XHTML standardy. Další nespornou výhodou je možnost práce s daty na straně serveru. Server má k dispozici data a může je lehce poskytnout prohlížeči. Bezesporu největší výhodou je zabezpečení. Můžeme jednoduše zakódovat věci, které prohlížeč nikdy neuvidí.
Technologií na straně serveru existují více.

- Common Gateway Interface (CGI)
- API rozhraní webového serveru
- Active Server Pages (ASP)
- JavaScript na straně serveru

- Servlety a Java Server Pages (JSP)
- PHP: Hypertext Procesor (PHP)

V naší práci využiji pro definici dokumentu jazyk XHTML. Pro grafické upravení využijeme skriptovací jazyk JavaScript a kaskádové styly CSS. Funkčnost a práci s daty nám zajistí skriptovací jazyk PHP a databázový jazyk MySQL.

4.3 Použité prostředky

Dnes se už skoro žádná stránka neobejde jen s použitím klasického XHTML. Čas statických webových stránek je dávno pryč.

Systém správy pro počítačové centrum byl implementovaný jako webová aplikace hlavně pomocí dvou technologií a to PHP a MySQL.

Velmi oblíbená a často nasazovaná je kombinace MySQL, PHP a APACHE. Jednou z výhod použitých prostředků je v tom, že uživatel nepotřebuje žádný speciální software, aby mohl informační systém využívat. Postačí mu obyčejný webový prohlížeč. Taktéž údržba je mnohem jednodušší.

4.3.1 PHP

Začátky PHP se datují rokem 1994, kdy jeden nezávislý dodavatel vyvíjející software jménem Rasmus Lerdorf vyvinul jistý skript PerIICGI, který mu umožňoval zjistit, kolik návštěvníků čte jeho online resumé. Aby spouštění perlu tolik nezatěžovalo server, přepsal ho do jazyka C. Tento systém se stal ihned populárním, a proto ho autor rozšířil a uvolnil pod názvem Personal Home Page Tools, poté Personal Home Page. Postupně narůstali užitečnosti a možnosti této technologie, došlo i ke změně slov udávajících význam PHP. V poslední verzi PHP a to verzi 5.0 se PHP přiblížilo ostatním jazykům podporující objektově orientované programování [9].

PHP je dnes velmi rozšířená technologie, umožňující jednoduché programovaní na straně serveru. Je to serverový skriptovací jazyk (server-side) navrhnutý pro potřeby webových stránek. To znamená, že všechno co PHP vykoná, neprobíhá na straně klienta, jako je tomu například u JavaScriptu, ale interpretuje se na straně serveru. Vygeneruje stránku podle zadaných kritérií a výsledek, který uvidí uživatel, odešle volajícímu počítači stejným způsobem, jakým se odesílají běžné statické stránky. To je možné využít k tvorbě různých interaktivních webových stránek. Takto je možné vytvořit různá vylepšení, od těch jednodušších jako jsou různé ankety nebo i knihy návštěv až k složitějším redakčním systémům, grafickým aplikacím či aplikacím s elektronickou poštou.

V PHP je podporováno propojení s velkým množstvím databází. Dalším podporovaným protokolem je LDAP v podobě API rozhraní klientských programů. PHP podporuje také jazyk XML,

který je považován za jazyk budoucnosti. Dále podporuje protokoly IMAP a SMTP pro využívání mailových služeb serverů. Jazyk PHP zahrnuje podporu pro obrázky, PDF dokumenty a síťové služby. Výhodou jazyka PHP je, že lze vložit jednotlivé přídavné moduly, které mohou obsluhovat jakýkoliv protokol nebo práci s daty.

Z uvedeného vyplývají hlavní přednosti PHP – každý den můžeme aktualizovat obsah stránek na serveru tím, že změníme data v databázi a ne tím, že budeme každou stránku samostatně přepisovat.

Jazyk PHP se skládá se základních řídících struktur, operátorů, druhů proměnných, deklarací funkcí, deklarací tříd, objektů, atd. Patří mezi jazyky netypované, dále jakákoliv proměnná zde může kdykoliv změnit svůj typ [9].

4.3.1.1 Výhody jazyka PHP

- výkon
- těsná integrace s většinou databázových systémů
- stabilita
- je zdarma
- nejsou problémy s kompatibilitou prohlížečů
- prohlížeč může poskytnout data, která nejsou na straně uživatele k dispozici
- open source – volně šiřitelná technologie
- není závislý na platformě a není vázaný žádným konkrétním serverem – může tedy běžet kdekoliv
- objektově orientovaný programovací jazyk, který je svojí strukturou podobný jazyku C++
- lepší zabezpečení oproti HTML

4.3.1.2 Nevýhody jazyka PHP

- jestliže je stránka načítaná, pomocí PHP ji už není možné dále měnit

4.3.2 SQL

Databázi je v podstatě možné si představit jako prostor, do kterého se ukládají všechny potřebné údaje. Zpracováním a přístupem k databázi je pověřený program, kterému se říká Systém řízení báze dat (SRBD) [17]. Představitelem SRBD je například MySQL. Většina dnešních SRBD je založena na tzv. relačním modelu dat. Název tohoto modelu vychází z relační algebry, což je matematický aparát, na kterém relační model staví. V tomto modelu jsou údaje uspořádané do tabulek [17].

Při práci s SRBD programem se používá model klient/server. Jako server vystupuje SRBD program poskytující svoje služby. Je tak nepřetržitě spuštěný a čeká na požadavky od klientů – jejich aplikací [17].

Pro zápis těchto požadavků na databázový server se dnes nejčastější používá jazyk SQL. Je to zkratka znamenající Structured Query Language. Samotný SQL obsahuje všechno potřebné pro práci s databázemi – vytváření, rušení, modifikovaní tabulky, ale i nástroje pro práci se samotnými údaji – přidávání, změnu, rušení a vyhledávání údajů.

SQL se stal univerzálním jazykem pro programování databází. Pomocí tohoto jazyka určujeme, co chceme v databázi vykonat [17].

4.3.3 MySQL

MySQL je multiplatformní databáze. Komunikace s ní probíhá, jak už název napovídá, pomocí formy jazyka SQL. Podobně jako u ostatních SQL databází se jedná o dialekt tohoto jazyka s některými rozšířeními. Je to systém, který se etabloval především ve webových aplikacích, a který je hodně preferovaný při spolupráci s PHP.

Ve svojí podstatě je MySQL ořezaný o některé možnosti, které mají jiné databázové systémy. Důsledkem je nenáročnost MySQL na zdroje počítače a zvýšení rychlosti u některých operací. MySQL je jednoduše rychlý, jednoduchý a nenáročný na databázový systém. Níže uvádím další výhody.

4.3.3.1 Výhody MySQL

- open source
- cachování dotazů – uchovává si dotazy SELECT spolu s výsledky. Při následných dotazech jich porovnává s těmito cachovanými dotazy. Jsou-li shodné, MySQL obejde nákladné získávání dat z databáze, a prostě použije výsledky cachovaného dotazu
- fulltextové indexování a vyhledávání
- replikace – umožňuje, aby se databáze umístněná na jednom serveru MySQL zduplikovala na jiný, což přináší množství výhod
- zabezpečovací a konfigurační volby

4.3.4 Webový server Apache

Webový server Apache je nejrozšířenější webový server, i když podle statistik se jeho využívání snižuje, tak je stále v provozu asi na 50 % serverech na internetu. Velkou výhodou je, že to je to open

source projekt a je tedy zdarma. Mezi jeho přednosti patří možnost použití na mnoha operačních systémech – Linux, BSD, Microsoft Windows, je neustále vyvíjen a obsahuje nejnovější standardy. Apache je server velice dobře konfigurovatelný a může být doplněn o různé moduly i vlastní výroby.

4.3.5 XHTML a CSS

Dokument XHTML (Extensible HyperText Markup Language) je rozšiřitelný značkovací jazyk pro hypertext. Je jedním z jazyků pro vytváření stránek v systému World Wide Web, který umožňuje publikaci dokumentů na internetu. Jeho historie začíná příchodem HTML a datuje se k roku 1991. Je složen ze základních příkazů, kterým se říká tagy. Píše se v ostrých závorkách, tedy mezi znaménky menší (<) a větší (>), a dělíme je do dvou skupin — na tagy párové a nepárové [18].

Každý tag má speciální význam. Ovlivňovat chování tagu můžeme pomocí parametrů a jejich hodnot. Tyto parametry jsou uvedeny za názvem tagu. Př. . Tento tag vypíše text uvedený mezi značky tagu o velikosti písma 2.

Jednotlivé tagy lze zanořovat. Platí ovšem pravidlo, že tagy se nesmí vzájemně křížit. Nesmíme tedy ukončit nadřazený tag, dříve než ukončíme tag vnořený.

CSS je zkratka pro anglický název Cascading Style Sheets, česky tabulky kaskádových stylů. Je to jazyk pro popis způsobu zobrazení stránek napsaných v jazycích XHTML. CSS umožňují přiřadit současně mnoho vlastností všem prvkům na stránce, které jsou označeny konkrétním tagem. Hlavní výhodou je oddělení definice vzhledu od definice kostry stránek. Jsou tu ale také nevýhody. Největší nevýhodou je, že CSS nejsou plně podporovány žádným internetovým prohlížečem. CSS se skládají z tagu nebo-li selektoru a definice nebo-li deklarace nebo více definicí. Definice je uzavřena ve složených závorkách { } a mezi jednotlivými definicemi se píše středník [18].

Kaskádové styly lze rozdělit do dvou skupin, na interní a externí. Interní se zapisují v sekci READ. Vložení je způsobeno zápisem tagu <STYLE></STYLE>. Mezi tento tag se poté vkládají jednotlivé definice. Zápis externích CSS se provádí pomocí tagu <LINK RE#=“ stylesheet“ TYPE=“text/css“ HREF=“ soubor.css“>. Samotné definice jsou uvedeny v externím souboru soubor.css a zapisují se stejně jako v interní definici.

4.3.6 JavaScript

JavaSript je multiplatformní, objektově orientovaný skriptovací jazyk. Ale narozdíl od PHP se kód vykonává na straně klienta. Kód, který se má vykonat, se klientovi pošle uvnitř HTML stránky a kód vykoná prohlížeč během zpracovávání stránky.

JavaScript se používá pro generování dynamického obsahu podle různých okolností. Např. vložení aktuálního času, reakce na verzi prohlížeče. Také umožňuje interaktivní práci s dokumentem, reakce na události (pohyb myši, kliknutí, změna hodnoty) a v neposlední řadě může sloužit k ovládání prohlížeče (pohyb v historii, otevírání nových oken, ovládání stavového řádku). Výhoda spočívá v

tom, že není nutné pro změnu stránky neustále obnovovat obsah stránky. Omezení však spočívá v možnostech použití, taktéž i v jejich (ne)bezpečnosti. Protože skripty se vykonávají přímo na počítači uživatele, mohl by potencionální útočník bez problémů vykonat nebezpečný kód. Proto jsou v JavaScriptu vypnuté funkce, které by přímo pracovaly se soubory na disku. JavaScript se kvůli tomuto omezení taktéž používá jako doplněk při vytváření webové grafiky. V této práci je například JavaScript použit jen k zobrazení aktuálního času. I když čas je získávaný z počítače klienta, výhoda je v jednoduchosti implementace narozdíl od PHP, kde je tato operace složitější [19].

4.4 Implementace uživatelského rozhraní

Při vstupu do systému se musí každý uživatel nejprve přihlásit. Proto první stránka obsahuje pouze odkaz na přihlašovací formulář. Přihlašuje se zadáním uživatelského jména a hesla. Pokud uživatel nezadá platné údaje, tak mu není povolen vstup do systému a je vyzván k vložení nových údajů. Jedním ze způsobů, jak implementovat autentizaci uživatele, je použití standardní HTTP autentizace. Protokol HTTP nabízí jednoduché prostředky pro ověření totožnosti uživatele. Pokud uživatel požádá server o nějaký prostředek, k němuž je přístup omezen, odešle server klientovi zprávu 401 (neautorizovaný přístup). Klient tuto zprávu rozpozná a zobrazí uživateli okno s výzvou k zadání uživatelského jména a hesla. Po zadání těchto údajů je klient odešle serveru, kde se tyto údaje ověří (například porovnají s údaji z databáze). Po úspěšném ověření je uživateli povolen přístup. Po správném přihlášení se uživateli zobrazí úvodní stránka. V současné implementaci tato stránka zobrazuje pouze uvítací zprávu, ale jako rozšíření by zde mohl zahrnovat jistý systém upozornění, například na nejbližší události, nebo se zde mohou zobrazovat určité firemní aktuality.

Navigace je řešena pomocí hlavního menu, které se zobrazuje v levě části obrazovky. Protože se jedná o hlavní menu, je viditelné a přístupné z každé stránky celého systému. Celé menu obsahuje bloky s odkazy. V závislosti na tom, jaká má uživatel oprávnění, jsou mu zobrazeny příslušné bloky. Například pokud je uživatel zaměstnancem, ale není vedoucím žádného projektu ani managerem, zobrazí se mu pouze první blok. V něm jsou odkazy na různé stránky, tyto odkazy odpovídají jednotlivým případům použití z Use Case diagramu. Stejně tak pokud bude uživatel IT manager, přibude ještě poslední blok.

4.4.1 Vzhled

Při návrhu barevného provedení stránek jsem se snažil řídit několika zásadami. Barevné schéma by nemělo být příliš pestré, aby nerušilo a neodrazovalo uživatele, ale zároveň takové, aby

celkový vzhled systému působil obyčejně a přirozeně. Proto bylo předefinováno barevné nastavení prakticky všech HTML prvků, které se v systému objevují. Vše samozřejmě za použití kaskádových stylů (viz 4.3.5), aby se dal celý vzhled systému kdykoliv jednoduše změnit. Pomocí kaskádových stylů je řešeno i pozicování jednotlivých elementů, ať už relativní nebo absolutní.

Každý webový IS potřebuje vhodný návrh vzhledu a podpůrných funkcí. Při dodržování jistých pravidel můžeme dosáhnout velmi dobrého výsledku. Řekněme si sedm základních pravidel webdesignu [8].

- To, že něco můžete udělat, ještě neznamená, že byste to udělat měli.
  Webové technologie nabízejí mnohé možnosti a nástroje. Přidání určité technologie může často web zpomalit. Není vždy vhodné využití těchto technologií.
- Výjimka potvrzuje pravidlo.
  Pro webdesign téměř neexistují absolutní pravidla. Určitá technologie nehodící se na určitý web, viz výše, se může velice hodit na jiný web.
- Soudcem a porotou jsou koncoví uživatelé.
  Na webu by se nemělo vyskytovat nic, co by se většině uživatelů nelíbilo nebo čemu by nerozuměli. Každý uživatel má jiný pohled, ale určité základy jsou pro většinu společné.
- Webdesignér by měl vždy usilovat o získání co nejširších znalostí a zkušeností.
  Při řešení libovolného problému bude webdesignér vždy ve výhodě, pokud mu bude důkladně rozumět po technické stránce. Je vždy potřeba vzít v potaz technickou stránku věci. Např. rozdíly mezi prohlížeči.
- Nejlepším přístupem je skromnost.
  Vždy se najde atraktivnější webdesign. Pokud se nenechá webdesignér zaslepit vlastní pýchou, má spoustu příležitostí, jak se poučit od jiných.
- Není na světě člověk ten, aby se zavděčil lidem všem.
  Každý člověk může mít trochu jiný názor a rozdílné preference. Existují však jemné hranice mezi tím, že se váš design bude líbit většině lidí, a tím, že se pokusíte uspokojit všechny, ale ve skutečnosti úplně neuspokojíte nikoho.
- Snažte se udržovat si přehled o specifikacích a standardech.
  Webové specifikace a standardy se neustále mění a měnit budou. Webdesignér by ale měl chápat alespoň základní principy nejnovějších technik.

Dále se můžeme zaměřit na určité filozofie jednotlivých webů. Výsledkem těchto filozofií je dosažení co nejefektivnější komunikace s uživatelem a předávání informací. Můžeme rozdělit na tři hlavní body.

- Estetická stránka.

Působí web profesionálním dojmem? Je jeho vzhled konzistentní s identitou, jakou se snaží prezentovat daná společnost či jednotlivec?

- Použitelnost.

  Jak rychle a jednoduše může uživatel vyhledat a zpracovat potřebné informace? Jak složité činnosti přitom musí provést?

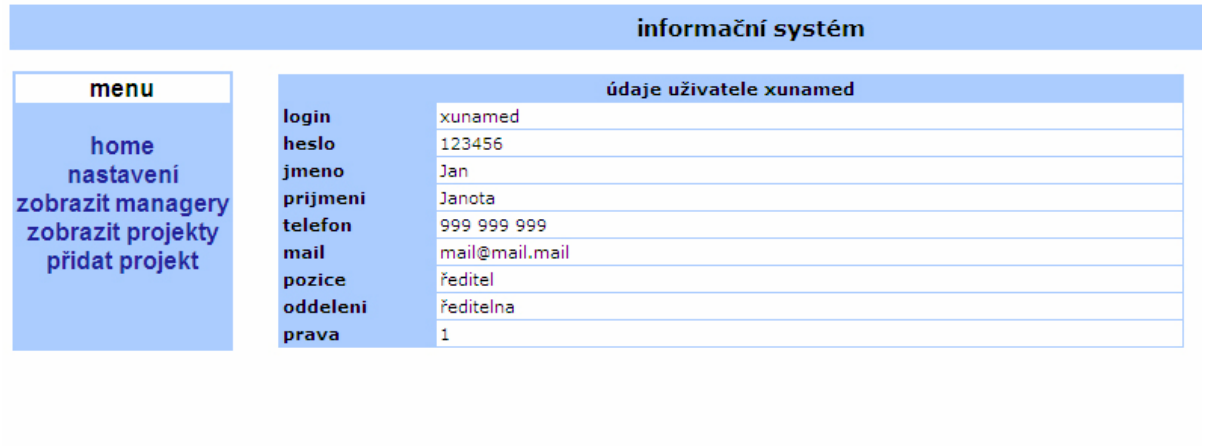
- Funkčnost.

  Správným naprogramováním by měly být zajištěny funkční aspekty webu, například správná činnost formulářů a textů dynamicky doplňovaných z databáze.

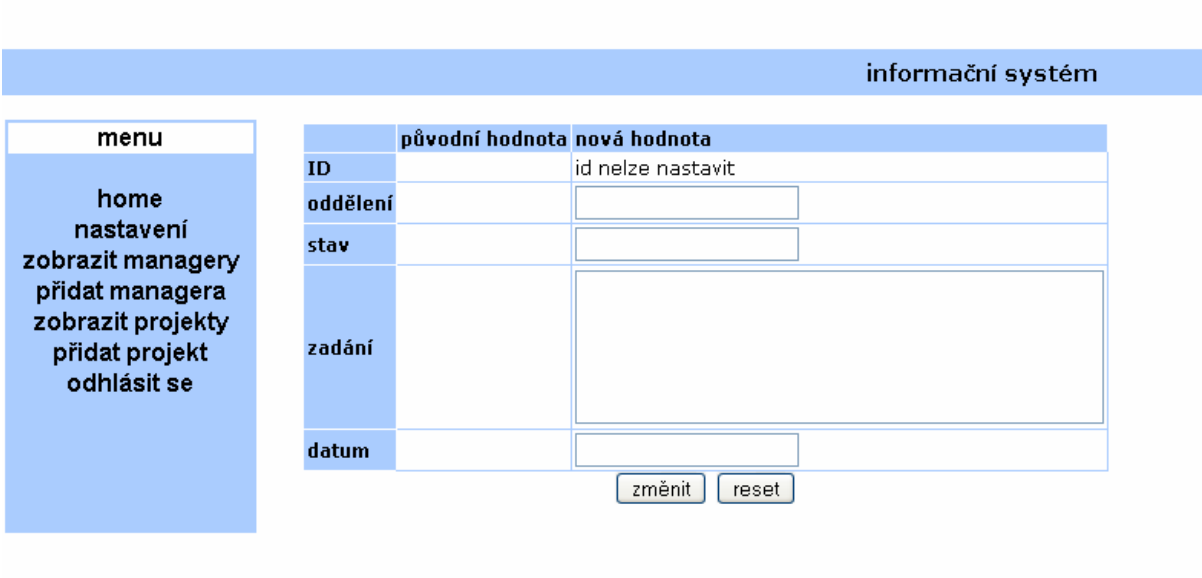
Vzhledem k obrovským rozdílům mezi různými uživateli Internetu a ve využívaném hardwaru a softwaru není žádná z uvedených filozofií ideální pro všechny situace. Jestliže webdesignér pochopí jednotlivé filozofie a jejich silné a slabé stránky, může se snadněji rozhodnout, která z nich povede k nejlepšímu naplnění požadavků na konkrétní navrhovaný web.

4.5 Ukázka implementace

V této kapitole jsou ukázky implementace v podobě screenshotů.



Obrázek 4.1 – Ukázka implementace – úvodní obrazovka po přihlášení.



Obrázek 4.2 – Ukázka implementace – přidání projektu.

Obrázek 4.3 – Ukázka implementace – změnění atributů uživatele.

Obrázek 4.4 – Ukázka implementace – zobrazení všech managerů.

4.6 Problémy při implementaci

4.6.1 Bezestavovost HTTP

Jedním z problémů, na které jsem během práce na systému narazil, bylo jak se vypořádat z bezestavovostí HTTP protokolu. Protokol HTTP (Hypertext transfer protocol) definuje pravidla pro přenos textu, grafiky, videa a dalších dat přes World Wide Web. Jak jsem již výše zmínil, jedná se o bezestavový protokol. To znamená, že server nemá k dispozici žádnou informaci o předchozích stránkách (dotazech), které si uživatel vyžádal. Taková zjednodušená implementace významnou měrou přispěla k všudypřítomnosti HTTP, bohužel při vývoji komplexních webových aplikací může způsobovat problémy. Následuje příklad.

V tomto informačním systému se neustále zpracovává a zobrazuje množství databázových tabulek. Výstup je vždy stránkován. Počet řádků tabulky, které se zobrazí na jedné stránce, je nastaven pomocí globální proměnné pagesize a tato hodnota je nastavena na třicet řádků. Pokud databázový dotaz vrátí více než třicet řádků, výstup se rozdělí do několika stránek a pod tabulkou se zobrazí navigace po těchto stránkách. Dále může uživatel výstup setřídit podle kteréhokoliv sloupce v tabulce.

A nyní k vlastnímu problému. Představme si situaci, kdy uživatel prohlíží tabulku projektů, je někde na druhé stránce výstupu, protože projektů je více než třicet a k tomu si nechal výstup seřadit podle loginu vedoucího projektu. Když už konečně našel projekt, který hledal, chce se podívat na seznam zaměstnanců, kteří na projektu pracují a popřípadě provést nějaké změny. Nyní klikne uživatel na odkaz zpět na projekty a bude chtít pokračovat v jejich prohlížení, ale bohužel výstup je opět v takovém stavu jako byl na začátku. To znamená před tím, než ho uživatel seřadil podle loginu vedoucího projektu a než se dostal na druhou stránku výstupu. To může být docela nepříjemné, zejména v případě, kdy uživatel prohlíží jednotlivé projekty a jejich zaměstnance jeden po druhém a po každé provedené změně bude muset znovu seřadit výstup a proklikat se na požadovanou stránku.

Jedním ze způsobů, jak tento problém řešit je neřešit ho a k navigaci používat historii navštívených stránek, která je k dispozici v prohlížeči (používat tlačítko zpět). Tento způsob má několik nevýhod, historie tlačítka zpět není neomezená a navíc se nejedná o pohodlné řešení.

Takže jsem se rozhodl tento problém řešit za použití cookies. Při prvním přístupu na stránku se do cookies uloží název sloupce, podle kterého se bude výstup řadit. Pokud se uživatel rozhodne seřadit tabulku podle jiného sloupce, přepíše se název sloupce v cookies. To samé platí ohledně stránek výstupu. Takže když uživatel opustí stránku a později se na ni vrátí, zobrazí se mu v takovém stavu, v jakém ji opustil. [6]

5 Náměty na rozšíření

V této kapitole uvádím pár nápadů, které mě napadly v průběhu práce na systému a tyto nápady by mohly vylepšit a přidat další funkce. Ačkoli již nyní informační systém zvládá řadu funkcí. Ty vycházejí ze základních požadavků na systém, které jsou specifikovány v kapitole 2.

5.1 Pokročilejší vyhledávání

Uživatel může v systému vyhledávat údaje podle jednotlivých klíčových slov. Například uživatel prohlíží seznam zaměstnanců a v tomto seznamu může hledat podle jména, příjmení, loginu, emailu, telefonního čísla nebo čísla, které používá na firemním chatovacím programu.

Toto jednoduché vyhledávání umožňuje vždy vyhledávat pouze podle sloupců jednotlivých tabulek nebo pohledů, ve kterých vyhledáváme. To má jistá omezení. Bylo by vhodné rozšířit možnosti vyhledávání o tvorbu komplexnějších požadavků. Například by mohl uživatel zadat více klíčových slov zároveň.

5.2 Použití XML

Extensible Markup Language (XML) patří do kategorie značkovacích jazyků. Tento jazyk se používá i mimo oblast internetu a dovolím si říct, že revoluční, co se týká přenosu dat.

Jazyk je určen především pro výměnu dat mezi aplikacemi a pro publikování dokumentů. Umožňuje popsat strukturu dokumentu z hlediska věcného obsahu jednotlivých částí, nezabývá se sám o sobě vzhledem dokumentu nebo jeho částí. Prezentace dokumentu (vzhled) se potom definuje připojeným stylem. Další možností je pomocí různých stylů provést transformaci do jiného typu dokumentu, nebo do jiné struktury XML [7].

XML také používá tagy jako XHTML. Ale výčet těchto tagů není pevně stanoven. Jde si tedy vytvářet vlastní tagy, které mají uživatelem definovaný význam – je na něm, jak tyto tagy zpracuje. XML dokumenty lze v určitých případech použít i jako zástupce databáze. Může se jednat např. o seznamy knih v eshopu. V souboru je definována struktura popisující knihu — autor, žánr, rok vydání, počet stran, jazyk, atd. Dokumenty XML mají určitě velkou budoucnost, proto informační systémy by měli zahrnovat jejich podporu.

5.3 Statistiky postupu práce

Dalším možným rozšířením by mohla být statistika postupu práce na projektech. Například bylo by výhodné najít vhodný způsob, aby vedoucí mohl být informován o stavu postupu jednotlivých pracích.

Například následujícím způsobem. Vedoucímu na základě hodnot od zaměstnanců, kteří by svou hodnotu někam ukládali (nejvhodnějším místem by byl další atribut v tabulce projektů), by se u každého projektu zobrazoval jistý progress bar, který by informoval, v jakém stavu se projekt (nebo jeho dílčí část) nachází.

.

6 Závěr

Rozvoj informačních systémů neustále posouvá představy o podnikání. Schopnost neustále rychle reagovat a jednoduše zadávat a získávat informace. To je jeden ze základů úspěšného podnikání v dnešní době. Tento systém umožňuje firmě využívat jeho hlavní výhody a to šetří času. Zvyšuje se i pohodlnost a jednoduchost tím, že systém redukuje často problematickou osobní komunikaci mezi zaměstnanci.

I když tento systém nebyl v praxi firemního prostředí ověřen, stanovené cíle byly dosaženy. Systém poskytuje jednoduché intuitivní ovládání, které by nemělo způsobovat problémy i méně zručným uživatelům.

Hlavní cílem této bakalářské práce bylo navrhnout systém pomocí objektově orientovaného návrhu a následně jej implementovat. Dalším cílem této práce bylo seznámit se s principy fungování firemních záležitostí a upevnit si znalosti s programováním v PHP a jiných jazycích. To nakonec bylo velkým přínosem této práce. Dalším přínosem této práce je zkušenost s tvorbou webu a tvorbou databázových tabulek a vztahů mezi nimi v prostředí databáze MySQL.

6.1 Zhodnocení práce

Na závěr bych se rád zmínil o celkovém přínosu této bakalářské práce. Díky několika měsíční práci na tomto projektu jsem si zlepšil své dosavadní znalosti z mnoha oblastí, jako například použití skriptovacího jazyka PHP, značkovacího jazyka XHTML nebo práce s databázovým relačním serverem MySQL. Také jsem nastudoval novou látku, která je na Fakultě informačních technologií Vysokého učení technického v Brně vyučováno až v pozdějším studiu (viz například předmět magisterského studijního programu „Návrh a analýza informačních systémů“), jako je např. objektově orientovaný návrh a s tím související použití návrhových vzorů. Mohu prohlásit, že tato bakalářské práce pro mě byla opravdovým přínosem.

Literatura

[1] Kučerová, H. Projektování informačních systémů 2007/2008 [online]. Poslední modifikace: 2007-03-04. [cit. 2008-05-07]. Dostupné na URL:<http://web.sks.cz/users/ku/pri/usecase.htm>.

[2] Jones, M. P. Základy objektově orientovaného návrhu v UML / Vyd. 1. Praha: Grada, 2001. 367 s. ISBN 80-247-0210-X.

[3] Pavlíček, L. Návrhové vzory [online]. Poslední modifikace: 2005-06-16. [cit. 2008-05-07]. Dostupné na URL: <http://objekty.vse.cz/Objekty/Vzory-historie>.

[4] Pecinovský, R.. Návrhové vzory / Vyd. 1. Brno : Computer Press, 2007. 527s. ISBN 978 0-251-1582-4.

[5] Kučerová, H. Projektování informačních systémů 2007/2008 [online]. Poslední modifikace: 2007-03-23. [cit. 2008-05-07]. Dostupné na URL: <http://web.sks.cz/users/ku/PRI/aktivity.htm>.

[6] Krčmář, Martin. Informační systém pro řízení projektu / Brno : Vysoké učení technické, 2006. 27 s.

[7] Soldát, M. Slabikář XML [online]. Poslední modifikace: 2002-05-13. [cit. 2008-05-07]. Dostupné na URL: <http://interval.cz/serialy/slabikar-xml/

[8] Mlynář, Petr. Informační systém družstva malé kopané / Brno : Vysoké učení technické, 2006. 66 s.

[9] Gilmore, Jason W. Velká kniha PHP 5 MySQL :kompendium znalostí pro začátečníky i profesionály / Vyd. 1. Brno : Zoner Press, 2005. 711 s. ISBN 80-86815-20-X

[10] Větrovská, P. Úvod do PHP [online]. Poslední modifikace: 2005-10-08. [cit. 2008-05-07]. Dostupné na URL: <http://www.webtvorba.cz/php/>.

[17] Ponkrác, M. Úvod do databází [online]. Poslední modifikace: 2004-07-29. [cit. 2008-05-07]. Dostupné na URL: <http://www.zive.cz/h/Programovani/AR.asp?ARI=1176558CA1=2O38>.

[18] Kosek, J. Jak pracují databáze na Webu Co je to databáze [online]. Poslední modifikace: 1999-02-02. [cit. 2008-05-07]. Dostupné na URL: <http://www.kosek.cz/clanky/iweb/12.html>.

[19] Janovský, D. Úvod do JavaScriptu [online]. Poslední modifikace: 2008-01-14. [cit. 2008-05-07]. Dostupné na URL: <http://www.jakpsatweb.cz/javascript/javascript-uvod.html>.

[20] Pavlíček, L. Návrhové vzory [online]. Poslední modifikace: 2005-06-16. [cit. 2008-05-07]. Dostupné na URL: <http://objekty.vse.cz/Objekty/Vzory-prehled>.

[21] Radovan, L. Úvod do informačních systémů [online]. Poslední modifikace: 2005-06-16. [cit. 2008-05-07]. Dostupné na URL: <http://radovan.bloger.cz/it/informacni-systemy/uvod-do-informacnich-systemu>.

Seznam obrázků

Seznam příloh

Příloha 1. – CD-ROM nosič.