VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁVRH A ANALÝZA SPORTOVNÍHO INFORMAČNÍHO SYSTÉMU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAROSLAV STRUŽKA
AUTHOR

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁVRH A ANALÝZA SPORTOVNÍHO INFORMAČNÍHO SYSTÉMU

DESIGN AND ANALYSIS OF SPORT INFORMATION SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAROSLAV STRUŽKA
AUTHOR

VEDOUCÍ PRÁCE Ing. JIŘÍ KRAJÍČEK
SUPERVISOR

BRNO 2008

Abstrakt

Tato bakalářská práce obsahuje popis analýzy, návrhu a následné implementace informačního systému. Informační systém je určena pro záznam a zveřejňování výsledků zápasů, tabulek skupin, informací o týmech, hračích a trenérech, statistik týmů a hráčů. Mimo výsledků zápasů systém eviduje také podrobnější události v zápasu jako počet diváků, autory branek, udělené karty, provedená střídání a časy těchto událostí. Dále uchovává informace o stadionech, na kterých se jednotlivé zápasy hrají. Součástí systému má být prvek umělé inteligence. Ten dokáže na základě podmínek, které popisují zápas, přiřadit sestavu rozhodčích k jednotlivým zápasům.

Klíčová slova

Informační systém, databáze, Mistrovství Evropy ve fotbale, ME 2008 ve fotbale, EURO 2008, návrh, analýza, návrhové vzory, softwarová architektura, diagram tříd, diagram případů užití, sekvenční diagram, diagram nasazení, diagram aktivit, entita, entitní množina, atribut, index, PHP, MySQL, CSS, JavaScript, XHTML, umělá inteligence, administrátor, uživatelská práva, 3-vrstvá architekura, MVC

Abstract

This bachelors thesis contains description fo analyze, design and futher implemantation of an information system. Information system is designed for evidence and publish scores of matches, group table, informations about teams, players and coaches, statistics of teams and players. Within the score system stores more detailed information about the game like number of viewers, authors of goals, number of cards, substitutions and times of theese actions. It stores informations about stadiums, where are the matches played. Part of the system is item of artificail intelligence, which links referees to the matches that should by played.

Keywords

Information system, database, European footbal championship, EURO 2008, design, analyze, design paterns, software architekture, class diagram, use case diagram, sequence diagram, deployment diagram, activity diagram, entity, atribut, index, PHP, MySQL, CSS, JavaScript, XHTML, artificial inteligence, administrator, users rights, 3-level architecture, MVC

Citace

Jaroslav Stružka: Návrh a analýza sportovního informačního systému, bakalářská práce, Brno, FIT VUT v Brně, 2008

Návrh a analýza sportovního informačního systému

Prohlášení

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

. . . . . .
Jaroslav Stružka
11. května 2008

Poděkování

Děkují panu Ing. Jiřímu Krajíčkovi za vedení a pomoc při tvorbě této bakalářské práce.

Obsah

Kapitola 1

Úvod

V dnešní době informační systémy nabývají na významu. Jsou používány stále ve větší a větší míře. A to zejména kvůli jejich víceméně jednoduché obsluhovatelnosti, přehlednosti a rozšířitelnosti. Další nespornou výhodou je možnost propojit informační systém s databázovým systémem, což umožňuje firmám přejít ze složité papírové agendy na podstatně jednodušší elektronickou. A tím ušetří nejen finanční prostředky, ale zejména čas a počet lidí potřebný ke správě.

V důsledku stále častějšího užívání informačních systémů se neustále zvyšují nároky na tvorbu těchto systémů. Je to způsobeno zvláště tím, že jsou uživatelé zvyklí na určitý typ standardu, který je založen na pohodlné obsluze a snadném pochopení při začínání s tímto systémem. Dalším kritériem je jednoduchost zavedení systému do firmy a velice důležitá je i možnost pozdějšího rozšíření systému podle požadavků uživatele.

Vytvoření informačního systému není, jak tomu bylo dříve v minulosti, jednoduchou záležitostí. V minulosti byl vývoj postaven na Vodopádovém modelu. Ten patří mezi první ucelené metodiky vývoje. Základní fáze tohoto modelu jsou definice problému, analýza, návrh, implementace, testování a provoz. Pro současné vývojové postupy se však ukázal jako nepružný a příliš těžkopádný. Dále nebyla potřeba používat objektově orientovaný návrh, neboť návrhové jazyky jako UML a implementační jazyky tento druh návrhu nepodporovaly, protože byly procedurální. Bohužel údržba takovéhoto systému byla noční můrou mnoha správců. Dnes je takovýto přístup k tvorbě nemožný. Základem vzniku opravdu kvalitního systému je použití nových návrhových modelů a vývojových modelů např. agilní vývojový model nebo testy řízený vývoj.

S těmito modely souvisí relativně nová věc, která se na programátorském poli objevila. Objektově orientované programování. Lze říci, že nejprve se formuloval objektový jazyk a později pro něj vzniklo UML a metody návrhu, které jsou mu blízké. Pokud se tedy rozhodneme pro implementaci touto metodou, návrh se stává nedílnou součástí tvorby systémů. Jakmile totiž v rukou držíme diagram tříd a sekvenční diagramy, implementace není složitá. V tomto případě vůbec nezáleží na OO jazyku, zvoleném pro implementaci systému, protože návrh např. prosřednictvým modelovacího jazyka UML je tvořen obecně pro všechny OO jazyky.

Cílem této práce je nastudování problematiky související s návrhem a analýzou informačního systému se zaměřením na vhodné přístupy objektového návrhu (softwarová architektura, návrhové vzory, atd.). Dále analyzovat požadavky sportovního informačního systému Mistrovství Evroby ve fotbale 2008 a pomocí jazyka UML provést detailní návrh. Další částí práce je podle tohoto návrhu daný systém naimplementovat.

1.1 Struktura práce

Práce je rozdělena do několika kapitol, v nichž jsou postupně popsány jednotlivé části tvorby systému.

V následující kapitole bude provedena analýza systému.

Ve třetí kapitole se budeme zabývat vhodnými přístupy objektového návrhu zejména si povíme o jazyku UML, navrhových vzorech apod.

Ve čtvrté kapitole se zaměříme podrobněji na třívrstvou architekturu MVC. A bude zde proveden podrobný návrh systému.

V paté kapitole se budeme zabývat provedenou implementací. Seznámíme se s problémy, na které bylo v průběhu implementace naraženo.

V závěrečné kapitole zhodnotíme dosažené výsledky, porovnáme systém s podobnými systémy a pokusíme se navrhnout další rozšíření do budoucna.

Kapitola 2

Analýza požadavků na systém

V této části práce se budeme věnovat analýze požadavků na IS. Díky analýze jsme schopni zjistit jednotlivé problémy. Způsob jejich řešení se provede v dalších kapitolách zejména potom v kapitole zaměřené na návrh.

2.1 Popis aplikace

Aplikace je určena pro záznam a zveřejňování výsledků zápasů, tabulek skupin, informací o týmech (fotografie, seznamy hráčů apod.), hračích (fotografie, osobní informace apod.) a trenérech (ty samé informace jako u hráče), statistik týmů (počet odehraných zápasů, počet vstřelených a obdržených gólů, průměr vstřelených a obdržených golů na zápas) a hráčů (počet odehraných minut, počet odehraných zápasů, počet vstřelených branek, průměr vstřelených branek na zápas, počet obdržených karet, průměr obdržených karet na zápas). Mimo výsledků zápasů systém eviduje také podrobnější události v zápasu jako počet diváků, autory branek, udělené karty, provedená střídání a časy těchto událostí. Dále uchovává informace o stadionech, na kterých se jednotlivé zápasy hrají.

Součástí systému má být prvek umělé inteligence. Ten dokáže na základě podmínek, které popisují zápas, přiřadit sestavu rozhodčích k jednotlivým zápasům.

2.2 Role uživatelů v systému

Aplikace rozlišuje 4 druhy rolí, jimiž jsou administrátoři, editoři, registrovaní návštěvníci a neregistrovaní návštěvníci. Každý z těchto uživatelů má svá práva, která se hierarchicky zvyšují a dědí, tzn. že uživatel, který je výše postavený má stejná práva jako jeho předchůdce a získavá další práva. Posloupnost od uživatele s nejmenšími oprávněními po uživatele s nejvyššími oprávněními je následující: neregistrovaní návštěvníci, registrovaní návštěvníci, editoři a administrátoři.

Neregistrovaní návštěvníci mají možnost prohlížet stránky (např. výsledky, podrobnosti ze zápasu, statistiku hráče atd.). Další část systému, do které mohou vstoupit, je *registrace uživatele*. Zde si po vyplnění unikátního přihlašovacího jména, hesla, jména, příjmení a kontrolniho čísla (zabraňuje, aby se registrovali tzv. roboti a zahltily tak databázi nepoužitelnými daty) vytvoří účet a získají stejná práva jako registrovaní návštěvníci.

Registrování návštěvníci mají stejné možnosti jako neregistrovaní a navíc mají možnost změnit vzhled systému. Mimo možností, které má již zmíněný neregistrovaný návštěvník, také možnost změnit vzhled systému. Díky svým právům mají přístup do části *Nastaveni*,

kde si mohou vybrat z několika implementovaných CSS stylů, nastavit oslovení či vybrat vlajku na pozadí. Dále mohou změnit své heslo a odhlásit se ze systému.

Editoři jsou podstatnou částí systému, protože mají práva přidávat a měnit vetšinu informací v systému. Vytvářejí, editují a mažou zápasy. Mohou měnit informace o týmu, hráči, trenérovi a rozhodčích; přidávat nebo odebírat hráče do/z databáze. A také mají možnost přidávání fotografií do fotogalerie. Jedinná možnost jak získat editorská práva je ta, že jsou přidělena správcem celého systému tzn. administrátorem.

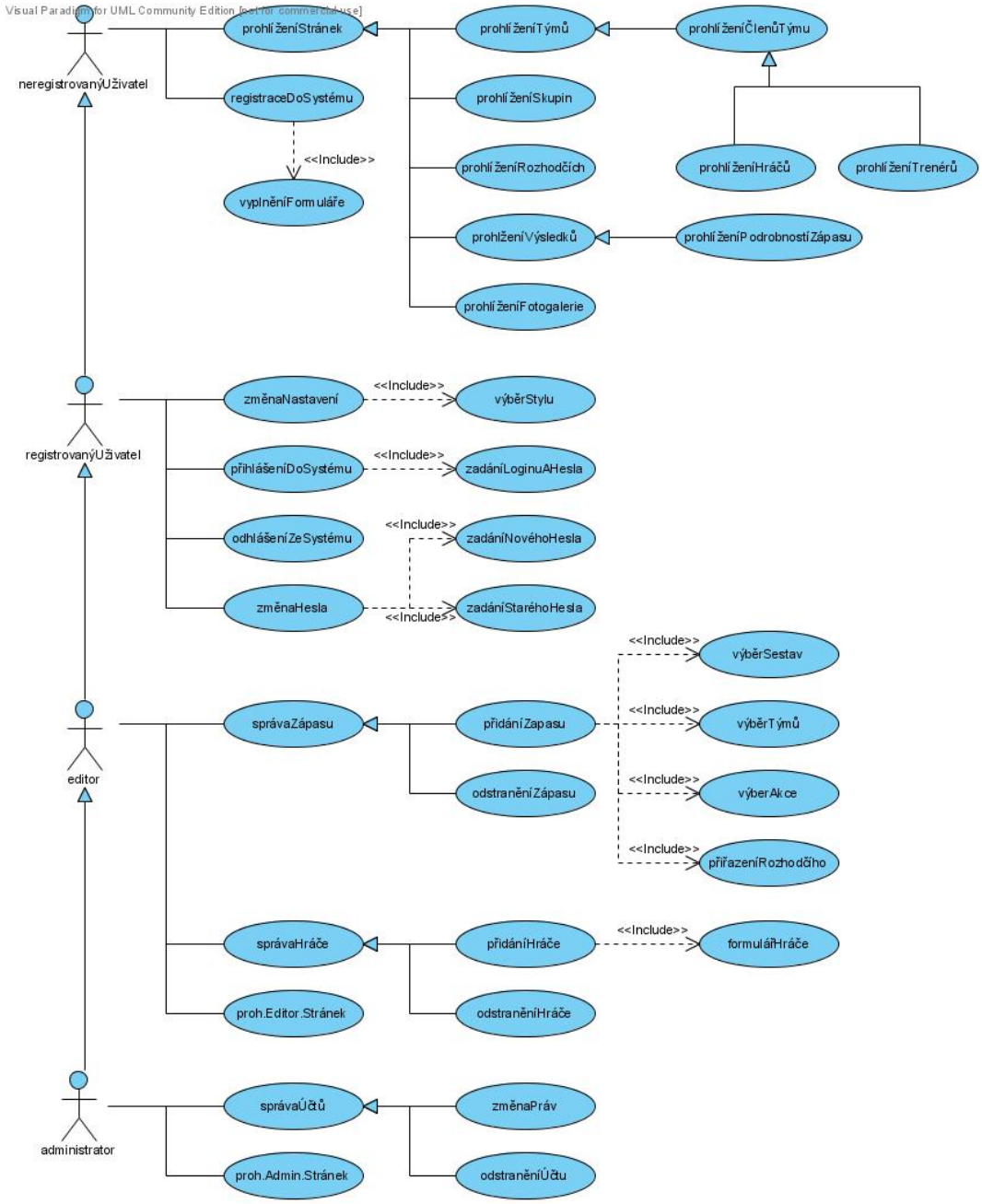
Administrátoři mají přehled o všech ostatních uživatelských účtech. Mají možnost přidávat, mazat a měnit všechny účty.

2.3 Analýza

Na základě uvedených požadavků a za použití jazyka UML (jedná se o grafický jazyk pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů) došlo k vytvoření diagramu případu užití (tzv. Use-case diagram) systému. (2.1)

Případy užití zachycují přesně funkčnost, která bude budoucím IS pokryta, a vymezují tak jednoznačně rozsah prací. Diagram užití popisuje požadovanou funkčnost systému. Systém nebude tedy obsahovat nic jiného než popisují případy užití.

Rozložení na podproblémy (tzv. dekompozice) celého úkolu bylo nedílnou součásti analýzy. Rozložení je založeno na diagramu případu užití a bylo tvořeno v závislosti na implementační metody řešení jednotlivých problémů a na další detailnější návrh pomocí jazyka UML.



Obrázek 2.1: Diagram případů užití systému

Kapitola 3

Objektový návrh

Jak jsme se již v úvodu dozvěděli, projekty nelze řešit bez kvalitního návrhu. V této kapitole se budeme věnovat vhodným přístupům objektového návrhu. S tím souvisí jazyk UML, softwarová architektura, návrhové vzory apod.

3.1 UML

Pro modelování objektově orientovaných softwarových systémů se v současné době používá jazyk UML (Unified Modeling Language). Je to otevřený rozšiřitelný průmyslový standardní vizuální modelovací jazyk uznaný skupinou OMG. První verze jazyka přijatá OMG je z roku 1997, poslední verze 2.0 je z roku 2005.

Dříve než se podrobněji podíváme na UML podrobněji, zopakujeme si velice stručně některé základní pojmy objektově orientovaného paradigmatu.

3.1.1 Základní pojmy

Objekt

Základním pojmem je objekt, ten budeme chápat jako část software, která má stav, chování a identitu. Stav objektu je určen hodnotami jeho atributů. Chování objektu je definováno službami(operacemi, též zvané metody), které může objekt provádět, když je o to požádán jinými objekty. Identita objektu (ObjectID) je specifická vlastnost objektu, podle jejíž hodnoty lze libovolné dva objekty systému odlišit a to i tehdy, když jsou hodnoty všech atributů obou objektů stejné a oba sdílí stejné operace.

Zapouzdření

Objekty klasifikujeme do tříd, třída tedy určuje typ objektu – definuje, jaké atributy objekt má a jaké operace může provádět. O objektu potom hovoříme jako o instanci třídy. Každý objekt třídy nese konkrétní hodnoty atributů definovaných třídou a poskytuje operace určené třídou. Vlastnost objektu, že ostatní objekty mohou ovlivnit stav objektu pouze prostřednictvím těchto operací, se nazývá zapouzdření.

Dědičnost

Dalším důležitým pojmem je dědičnost. Jde o vztah mezi třídami, kdy jedna třída dědí všechny vlastnosti (atributy, operace, atd.). Vlastnosti nadtřídy může dědit několik podtříd.

Pokud podtřída dědí vlastnosti jen jedné nadtřídy, hovoříme o dědičnosti jednoduché, má-li nadtříd více, hovoříme o dědičnosti vícenásobné.

Polymorfismus

Posledním základním pojmem, který si zopakujeme, je polymorfismus. Týká se operací. Pokud má jedna operace se stejnou hlavičkou několik implementací, pak jde o polymorfní operaci. Může k tomu dojít tak, že několik podtříd zdědí tutéž abstraktní operaci a dodefinuje vlastní implementaci nebo podtřída redefinuje zděděnou konkrétní operaci. Při požadavku na provedení operace se potom automaticky vybere implementace té třídy, na jejíž objekt se operace aplikuje.

3.1.2 Struktura jazyka

Pro pochopení struktury je důležité jednak pro pochopení jeho celkové filosofie a možností, které nám pro vizuální modelování nabízí. Filozofie dobrého návrhu je stejně duležitá jako filozofie bojových umění. Bez jejího pochopení jde jen o práci, formu bez obsahu. Podobně jako se ruzné bojové styly liší svou formou, základni obsah je stejný. Podobně i tak se metody návrhu mohou lišit, duležitá není však forma ale obsah.

Struktura jazyka je tvořena:

- Stavebními bloky – jsou to základní modelovací prvky jazyka, vztahy a diagramy.
- Obecnými mechanismy – obecné způsoby, které nabízí UML k dosažení konkrétních cílů.
- Architektura - pohled jazyka UML na architekturu modelovaného systému.

Stavební bloky

Mezi *stavební bloky* UML patří:

- Prvky – jsou to samotné modelovací prvky jazyka.
- Vztahy – slouží k modelování vazeb mezi prvky.
- Diagramy – pohledy do modelů v UML. Představují náš způsob vizualizace toho, co systém bude dělat (diagramy úrovně analýzy) nebo jak to bude dělat (diagramy úrovně návrhu)

Prvky Mezi *prvky* patří modelovací prvky, které slouží k modelování struktury systému (např. třída, případ použití, komponenta), k modelování chování (např. interakce, aktivita, stavový automat), k seskupování prvků (balíček (package)) a k okomentování (poznámka).

Vztahy *Vztahů* existuje v UML několik typů. Jako příklad můžeme uvést vztah závislosti, který modeluje skutečnost, že nějaký prvek závisí na prvku jiném a může být ovlivněn jeho změnou.

Diagramy *Diagramy* UML lze rozdělit do dvou skupin - pro modelování statické struktury (statický model) a pro modelování dynamické struktury (dynamický model) systému. Do prvé skupiny patří diagram tříd, diagram objektů, diagram balíčků, diagram komponent, diagram složené struktury a diagram nasazení. Do druhé skupiny potom diagram případů použití, diagramy interakce (diagram sekvence, diagram komunikace, přehledový diagram interakce, diagram časování), diagram stavového automatu a diagram aktivity.

Obecné mechanismy

Obecné mechanismy jazyka UML popisují čtyři strategie, které se zde používají v různém kontextu.

Specifikace Jde o textové informace doplňující grafickou podobu modelu. Hovoří se o grafické a textové dimenzi modelů UML. Specifikace umožňují vyjádřit sémantiku prvku modelu v kontextu řešeného problému. Typickým příkladem je specifikace případu použití, kdy pouhý diagram případů použití bez jejich specifikace má velice malou vypovídací schopnost.

Doplňky V UML má každý modelovací prvek přiřazen jednoduchý grafický symbol, který lze doplnit řadou doplňků a tak uzpůsobovat množství prezentované vizuální informace aktuálním potřebám. Například u třídy lze zobrazovat jenom její jméno nebo i atributy, nebo i operace a podobně.

Obecná dělení Jde o způsob uvažování o modelovaném světě. V UML jsou dvě obecná dělení.

1. Klasifikátor/instance – Zde klasifikátor značí abstraktní pojem - typ prvku a instance konkrétní výskyt. Například při modelování fakultního informačního systému může být takovým abstraktním pojmem student a jeho konkrétním výskytem student Jan Novák. Jistě jste poznali, že klasifikátorem je v tomto případě třída a instancí konkrétní objekt této třídy. Třída ale není zdaleka jediným klasifikátorem v jazyce UML, patří se také případ použití, komponenta, aktér, rozhraní a řada dalších. V UML existují diagramy, ve kterých figurují klasifikátory a jiné obsahující instance. Instance mají v UML obvykle stejný grafický symbol jako odpovídající klasifikátor, pouze jméno na symbolu je podtržené.
2. Rozhraní/implementace – Podstatou je oddělit to, co něco dělá (jeho rozhraní), od toho, jak to dělá (jeho implementace). Dá se říci, že rozhraní definuje „kontrakt“, který konkrétní implementace garantuje dodržet. Přitom způsobů, jak implementovat dané rozhraní může být celá řada. Principu oddělení rozhraní od implementace má u objektové orientace velký význam a moderní programovací jazyky, např. Java, poskytují rozhraní jako samostatný konstrukt. Podobně je tomu v UML.

Mechanismy rozšiřitelnosti Autoři jazyka si uvědomili, že není možné navrhnout zcela univerzální modelovací jazyk, jehož výrazové prostředky by pokryly všechny současné i budoucí potřeby. Proto do UML zahrnuli tři jednoduché mechanismy rozšiřitelnosti. Tyto mechanismy rozšiřitelnosti umožňují vytvářet tzv. *profily* seskupením stereotypů, připojených hodnot a omezení definovaných pro určitou aplikační oblast. Tak

již byly vytvořeny profily pro modelování systémů pracujících v reálném čase, pro modelování .NET aplikací apod.

Využití mechanismů rozšiřitelnosti ale vyžaduje příslušnou podporu modelovacího nástroje, který nám musí umožnit vytvářet stereotypy, definovat připojené hodnoty a omezení a umožnit práci s nimi.

1. Omezeni – Doplňující textová informace, která definuje nějakou podmínku nebo pravidlo vztahují se ke konkrétnímu modelovacímu prvku v modelu, které musí být dodrženo. UML obsahuje řadu předdefinovaných omezení, ale uživatel může libovolně vytvářet omezení vlastní. Přestože UML nevyžaduje zápis omezení v nějakém konkrétním jazyce, definuje jako své standardní rozšíření jazyk OCL (Object Constraint Language), který je určen i pro tyto účely. Omezení se píší v UML do složených závorek.
2. Stereotypy – Jde o nové, uživatelem definované prvky, které jsou odvozeny z existujících modelovacích prvků jazyka UML modifikací jejich významu. Samotný UML již obsahuje některé předdefinované stereotypy (např. podsystém jako stereotyp komponenty). Stereotypy se nejčastěji zobrazují použitím symbolu prvku, ze kterého je stereotyp odvozen, s uvedením názvu stereotypu ve dvojitých lomených závorkách.
3. Připojené hodnoty – Většina modelovacích prvků v UML má předdefinovány určité vlastnosti, z nichž některé se zobrazují v diagramu, jiné ne. Uživatel ale může definovat nové vlastnosti modelovacích prvků zavedením klíčových slov pro ně. Ke konkrétnímu prvku modelu jsou potom připojeny hodnoty těchto vlastností ve tvaru klíčovéSlovo1 = hodnota1, klíčovéSlovo2 = hodnota2, ... Uživatelem definované připojené hodnoty se uplatňují hlavně u stereotypů.

3.2 Softwarová architektura

3.2.1 Popis

Softwarová architektura programu nebo počítačového systému je struktura systému, který se skládá ze softwarových komponent, externě viditelných částí těchto komponent a vztahu mezi nimi. Termín také udává dokumentaci systémové architektury. Dokumentovaná softwarová architektura napomáhá komunikaci mezi analytiky a zákazníky. Dokumenty urychlují rozhodnutí o vysokolevelovém návrhu a povolují znovupoužití návrhových komponent a vzorů mezi projekty.

Dřívější problémy komplexnosti byly řešeny vývojáři výběrem správných datových struktur, vývojových algoritmů a aplikováním konceptu rozložení na podproblémy. Přesto je termín SW architektura v průmyslu relativně nový. Základní principy již byly sporadicky aplikovány softwarovými inženýry od poloviny 80tých let. Dřívější pokusy o zachycení a vysvětlení softwarové architektury systému byly však nepřesné a zmatené. V průběhu 90tých let byla snaha definovat a utřídit základní aspekty. Tehdy byly vyvinuty počáteční nastavení návrhových vzorů, stylů, nejlepších praktik, popisu jazyka a formální logika.

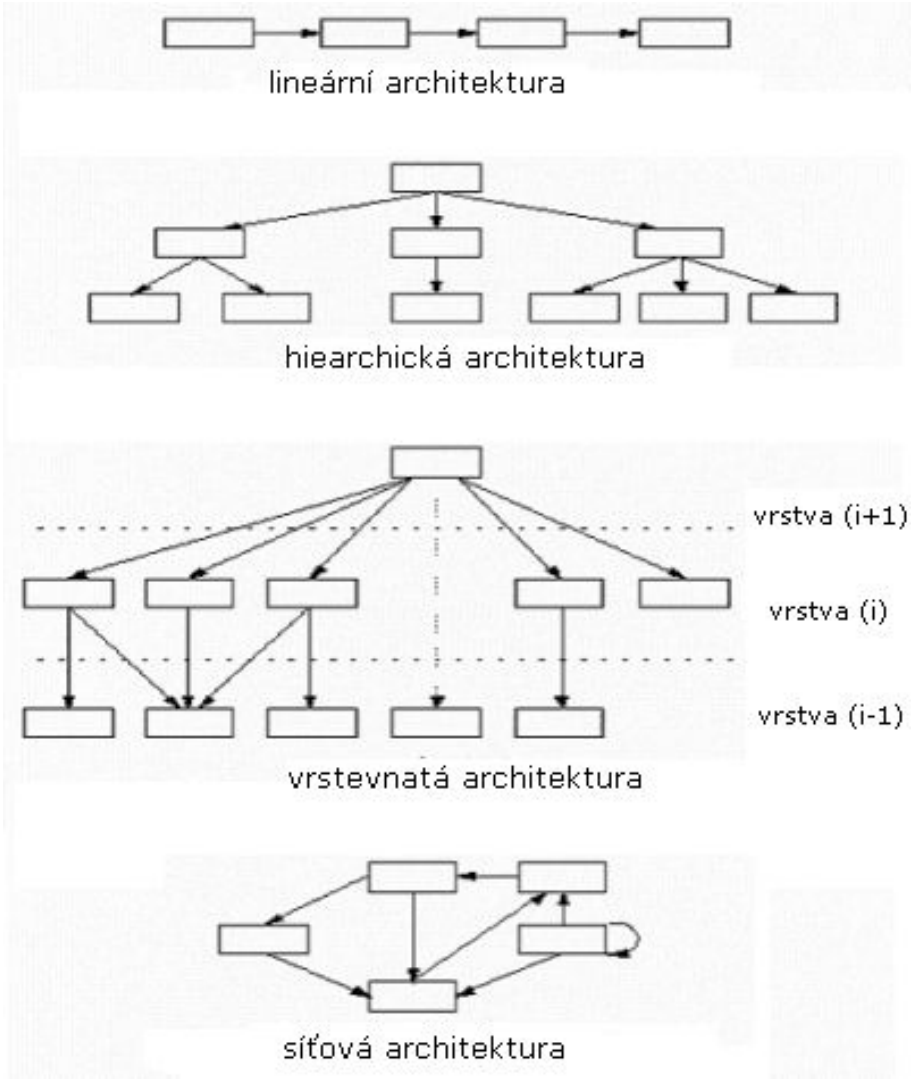
Obor SW architektury je založen na myšlence zredukování složitosti pomocí abstrakce a rozdělení na podproblémy. Dodnes ovšem stále neexistuje přesná definice termínu SW architektura.

Tento zrající obor bez jasně daných pravidel a správné cesty k vytoření systému a návrhu SW architektury je stále směs vědy a umění. Aspekt umění SW architektury podporuje

kvůli komerčnímu SW systému nektěré aspekty úkolu nebo poselstvi. Jak systém podporuje klíčové informace k dosažení cíle, je popsáno pomocí scénářů jako nefunkčních požadavků na systém. Také známé jako atributy kvality, které rozhodují, jak se systém bude chovat. Každý systém je vyjímečný kvůli povaze informací, kterými je specifikován – jako tolerance chyb, zpětná kompatibilita, rozšiřitelnost, udržovatlnost, bezpečnost, použitelnost apod.

3.2.2 Typy architektur

Určuje SW komponenty IS a jejich vzájemné vazby. Definuje vnitřní strukturu SW komponent - určení modulů, jejich vazby a charakteristiky. Je realizována pomocí monolitické, dvouvrstvé nebo třívrstvé architektury. (3.1).



Obrázek 3.1: druhy architektur

Z těchto čtyř druhů jsou univerzálně použitelné jen vrstvená a síťová architektura, zbývající dvě (lineární a hierarchická) lze použít pouze pro specifické aplikace. Síťová architektura je preferována, jestliže preferujeme nízké náklady provozu před nízkými náklady

na tvorbu, údržbu a užití. V ostatních případech je vhodnější užít architekturu vrstvenou.

3.3 Návrhové vzory

3.3.1 Popis

V SW inženýrství jsou návrhové vzory znovupoužitelným řešením obvyklých problémů. Návhové vzory nejsou vytvářeny tak, aby byly přímo převáděny na kód, ale slouží k popisu nebo jako šablona pro řešení problémů v mnoha rozdílných situacích. Objektově orientované návrhové vzory znázorňují vztahy a působení mezi třídami a objekty bez specifikace konečných aplikačních tříd a objektů, které jsou zapojeny do řešení.

Návrhové vzory mohou urychlit vývojový proces poskytováním otestovaných a dokázaných vývojových paradigmat. Efektivní SW návrh vyžaduje zvažování problémů, které by se neměly objevit až při pozdější implementaci. Znovupoužití návrhových vzorů pomáhá zabránit tomu, aby drobné problémy nezpůsobovaly velké problémy. A zároveň vylepšuje čitelnost kódu pro ostatni programátory a architekty, kteří jsou s technikou vzorů obeznámeni.

Lidé často rozumějí jen tomu, jak určitou SW návrhovou techniku aplikovat na určitý problém. Tyto techniky jsou težké pro aplikaci na problémy širšího rozsahu. Návrhové vzory poskytují celkové řešení doložené ve formátu, který nepotřebuje specifikace vázané na jednotlivé problémy.

3.3.2 Části návrhových vzorů

Podle knihy "Design paterns" od "Gang of four" (Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides) se návrhové vzory skládají z několika částí:

Jméno vzoru a klasifikace - Popisující a unikátní jméno, které pomáhá v identifikaci a odkazování na vzor.

Účel – Popis cíle návrhu a důvod pro použití...

Motivace – Scénář skládající se z problému a kontextu, v kterém lze tento vzor použít...

Aplikovatelnost – Situace, ve kterých je tento vzor použitelný; kontext pro vzor...

Struktura – Grafické vyjadření vzoru. Za tímto účelem se používají Digramy tříd a Sekvenční diagramy...

Účastníci – Seznam použitých tříd a objektů a jejich úloha v návrhu...

Spolupráce – Popis, jakým způsobem se použité třídy a objekty mezi sebou ovlivňují...

Důsledky – Popis výsledků, vedlejších účinků a vzájemných porovnání způsobených použitím návrhu...

Implemtace – Popis implementace návrhu; výsledná část návrhu...

Příklad kódu – Ilustrace toho, jak může být návrh použit v programovacím jazyce...

Známá použití – Příklady skutečnho použití návrhu...

Související vzory – Ostatní vzory, které mají podobný vztah s návrhem, diskuze rozdílů mezi vzory a podobné vzory...

3.3.3 Základní rozdělení

Rozdělení návrhových vzorů vychází z GAM95. Autoři vymezili tři základní skupiny návrhových vzorů, které jsou stále uznávané.

- Creational Patterns (vytvářející)
- Structural Patterns (strukturální)
- Behavioral Patterns (chování)

1. 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

2. Structural Patterns představují skupinu návrhových vzorů, zaměřujících se na možnosti uspořádání jednotlivých tříd nebo komponent v systému. Snahou je zpřehlednit systém a využít možnosti strukturalizace kódu. Mezi tyto návrhové vzory patří:

   - Adapter Pattern
   - Bridge Pattern
   - Composite Pattern
   - Decorator Pattern
   - Facade Pattern
   - Flyweight Pattern
   - Proxy Pattern

3. 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

3.4 Literatura

V této kapitole byly použity následující publikace [12, AIS], [13, IDS], [4, IIS], [1, UML], [5, OOP], [2, Design patterns] [6, NZ] a webové stránky [8, Objekty].

Kapitola 4

Návrh systému

V této kapitole si provedeme podrobný návrh systému pomocí diagramu tříd. Přiblížíme si třívrvstvou architekturu MVC z které vychazí návrh.

4.1 MVC architektura

Model-View-Controller architektura je základním prvkem navrhovaného systému i přesto, že realizace se odlišuje od jejího klasického chápání. Tato architektura je ve velké míře využívána při vytváření prezentačně orientovaného systémů a již je publikována jako návrhový vzor. Jejím principem je rozdělení systému do tří logických částí:

- *View* – slouží k vytvoření grafického rozhraní pro interakci s uživatelem. Každý ucelený systém by měl obsahovat jednotné grafické ovládání, které plně vyhovuje uživatelům. Při návrhu grafických standardů pro vznikající systém je nutné brát v úvahu nejenom požadavky uživatelů, ale i standardy Internetu. Účelem této vrstvy je oddělení prezentační logiky od věcné a datové, což zvyšuje její udržovatelnost a umožňuje jednodušší zapracování změn, které se objevují ve větší míře právě v prezentační vrstvě. Jestliže využíváme J2EE technologii, je pro tuto vrstvu možné využít servlety, JSP stránky a transformace XML pomocí XSL.

- *Model* – zapouzdřuje věcnou a datovou logiku systému. Jelikož se většinou jedná o nejsložitější vrstvu systému, která obsahuje množství komponent, je vhodné poskytnout přehledný přístup k této vrstvě pomocí ucelené sady aplikačních rozhraní (API). Z návrhových vzorů, které se hodí pro tento účel, můžeme použít Facade ve spolupráci s Command vzorem. Model, dle doporučení tvůrců J2EE platformy, by měl být realizován pomocí EJB komponent. Podobné snahy můžeme, ale nalézt i u rodiny jazyků .NET. Tyto jsou využity především v situacích, kdy je kladen důraz na bezpečnost, transakční zpracování, možnou rozšiřitelnost. Snahou ovšem je celou logiku rozdělit do samostatných komponent, které tvoří realizaci určité části, kterou je možné přesně vymezit a ohraničit. Vytvořením sady komponent zvyšujeme možnosti jejich znovupoužití a tím úsporu pracnosti. Každá komponenta by měla představovat zapouzdření funkcí a dat v ní uložených. Tímto můžeme i uvažovat o distribuovaném zpracování nebo využití více datových úložišť. Komponenty jsou uspořádány do větších celků (modulů, subsystémů), které prezentují logickou, samostatně provozovatelnou část. Pří návrhu je kladen důraz na snížení vzájemné provázanosti modulů, subsystému i samotných komponent.

- *Controller* – je vrstva mezi prezentační (View) a vrstvou aplikační (Model). Jejím účelem je zpracování událostí v prezentační vrstvě a vyvolání metod objektů tvořících Model. Controller vrstva redukuje závislost mezi věcnou a prezentační logikou. Základem vrstvy Controller je jeden servlet, který slouží jako jednotný přístup z prezentační logiky. Na základě vyhodnocení přijatého požadavku dochází k rozhodnutí, která část věcné logiky je odpovědná za jeho vyřízení.

MVC architektura je částečně postavena na principu návrhového vzoru Observer. Model představuje pozorovaný objekt, ke kterému se jako pozorovatelé přihlašují objekty z prezentační vrstvy a Controller. Jestliže nastane změna v Modelu (například změna dat), je o této skutečnosti informován Controller i zaregistrované prezentační objekty pomocí metody update(), která je definována v rozhraní Observer. Controller může vlastnit referenci na více komponent Modelu a i na více objektů z prezentační vrstvy. Návrhový vzor MVC byl vytvořen pro technologii client–server. Problém tohoto vzoru v internetové technologii je nemožnost okamžité notifikace prezentační logiky. Tato technologie pracuje na principu požadavek-odpověď, kdy zahájení komunikace vždy vychází z GUI. Proto je tato vrstva informována o proběhlých změnách pouze pomocí odpovědí na její požadavky.

4.2 Diagramy

4.2.1 Diagram aktivit

Často se označují jako „objektově orientované vývojové diagramy“. Vychází z osvědčených modelovacích technik vývojových diagramů, Petriho sítí a modelování workflow. Často se používají pro modelování business procesů, workflow, datových toků a operací. Aktivita, kterou modelují, ale může ukazovat i chování případu použití, skupiny případů použití, tříd, rozhraní nebo komponent. Ve srovnání s jinými diagramy pro modelování chování, například diagramy interakce, mají diagramy aktivit vlastnost, že modelují chování, aniž by bylo potřeba specifikovat statickou strukturu tříd a objektů, které aktivitu realizují. Toho lze s výhodou využít v počátečních stádiích analýzy. Ukázka viz Obr.(4.1)

4.2.2 Diagram tříd

Základním diagramem UML pro modelování logického pohledu na statickou strukturu systému je diagram tříd, který vizualizuje třídy a rozhraní, jejich interní strukturu a vztahy k ostatním třídám.

Třída je v UML definována jako popis množiny objektů, které sdílejí tytéž specifikace rysů, omezení a sémantiky. Rysy třídy jsou atributy a operace.

Vztahy v diagramu tříd jsou významná spojení tříd (případně rozhraní). Existují tři typy vztahů, které používáme v diagramu tříd: asociace, agregace a generalizace.

- *Asociace* ve skutečnosti reprezentuje vazby objektů daných tříd, podobně jako vztahy v ER diagramu. Nejčastějším případem asociací jsou binární asociace, což jsou asociace reprezentující vazby mezi dvojicemi objektů, zpravidla dvou různých tříd. Jde-li o objekty téže třídy, nazývá se asociace reflexivní. V případě tří objektů hovoříme o ternární asociaci. U asociace můžeme definovat její jméno, jméno role, násobnost a navigovatelnost. Poslední tři vlastnosti se vztahují vždy ke konkrétnímu konci asociace. Viz Obr.(4.2)

- *Agregace* je v UML speciální druh asociace modelující vztah celek – část, tj. kde instance jedné třídy (celek, agregát) obsahuje instance jiné třídy (části). Na Obr.(4.3) je třída `TřídaA` celkem a třída `TřídaB` jeho částí. Vztah celku a části modelovaný agregací je volný. Příkladem může být počítač a periferie. Celek někdy může existovat nezávisle na částech, jindy ne. Podobně části mohou existovat nezávisle na agregátu. U agregace může být dokonce část sdílená více agregáty, tj. násobnost u celku může být větší než jedna.

  V UML existuje silnější forma agregace, která vyjadřuje těsnější vazbu celku a částí. Nazývá se *kompozice* viz Obr.(4.4). Graficky se liší od agregace tím, že symbol u celku je vybarven. U kompozice mohou části patřit vždy jen jednomu celku, celek je zodpovědný za vytvoření a zrušení částí, a když je zrušený celek, musí být zrušeny všechny jeho části. Je zřejmé, že sémantika kompozice je velice podobná sémantice atributů a modeluje vlastně totéž. Atribut bychom měli použít, jde-li o primitivní typy nebo třídy, které jsou součástí jazyka nebo v případě, kdy nechceme třídu explicitně ukazovat v modelu. Základním pravidlem by měla být jasnost, užitečnost a srozumitelnost modelu.

- *Generalizace* je vztahem mezi obecnějším prvkem a specielnějším prvkem, kde specielnější prvek je zcela konzistentní s obecnějším prvkem, ale obsahuje více informací. Oba prvky se řídí principem nahraditelnosti. Ten znamená, že můžeme použít specielnější prvek všude tam, kde je očekáván prvek obecnější bez narušení systému. V diagramu tříd půjde o vztah mezi třídami. Obecnější třída se nazývá nadtřídou a specielnější třída podtřídou. Struktura vzniklá uspořádáním tříd podle vztahu generalizace se nazývá hierarchie generalizace. Určitě si uvědomujete, že z podstaty generalizace vyplývá dědičnost - podtřída dědí všechny vlastnosti (atributy, operace, vztahy a omezení vztahující se na objekty) svých nadtříd. Je rovněž zřejmé, že jde o mnohem silnější vztah tříd než asociace. Z generalizace vyplývá silná závislost obou tříd. Na Obr.(4.5) třída `TřídaA` je nadtřídou třídy `TřídaB`.

4.2.3 Diagram sekvence

Také známý jako Sekvenční diagram, vizualizuje posloupnost zpráv zasílaných mezi účastníky interakce. Můžeme v něm rozlišit dvě dimenze. Prvou (zleva doprava) je dimenze účastníků interakce. Jde o instance klasifikátorů v UML, tedy typicky jde o instance aktérů, tříd, balíčků nebo komponent. Graficky jsou znázorněny stejným symbolem jako odpovídající klasifikátor (např. u objektu třídy) a svislou čárkovanou čarou. Ta je vlastní „čarou života“ účastníka interakce. Na ní lze znázornit, kdy je instance aktivní a kdy nevlastní žádné vlákno řízení. Druhou dimenzí (shora dolů) je čas. Není vyjádřen explicitně, ale zasílané zprávy se v diagramu znázorňují tak, jak jdou po sobě v čase.

Zprávy reprezentují komunikaci mezi dvěma účastníky interakce, která může znamenat volání metody, vytvoření nebo zrušení instance (účastníka interakce) nebo zaslání signálu. Vyznačují se horizontálními šipkami mezi účastníky. V případě volání metody se stává přijímající účastník interakce aktivní po dobu provádění metody. Doba aktivace je vyznačena malým obdélníkem na „čáře života“ účastníka. Diagram sekvence lze ale kreslit i bez vyznačených aktivací, resp. některé nástroje je ani nepodporují. Doporučuje se znázorňovat aktivace pouze tehdy, když nezpůsobí menší přehlednost diagramu. Způsob znázornění účastníků interakce, zasílání zpráv jsou naznačeny na Obr.(4.6)

4.2.4 Diagram komunikace

Narozdíl od diagramu sekvence zdůrazňuje statickou strukturu (propojení objektů), která se využije při interakci k dosažení požadovaného chování, například definovaného nějakým případem použití.Ukázka viz Obr.(4.7)

Diagram komunikace má podobnou syntaxi jako diagram sekvence s tím rozdílem, že není kreslen „dvoudimenzionálně“. Účastníci interakce nemají čárkované čáry pro časovou dimenzi. Proto je zde podstatné hierarchické číslování zpráv, které definuje pořadí, včetně zanoření. Vazby mezi účastníky interakce ukazují, jak odesílatel zprávy získá „kontakt“ na adresáta. Může to být díky asociační vazbě, může jít o globální či lokální objekt apod.

V diagramu lze vyjádřit iterace i podmínky. Zejména u podmínek je ale jejich praktické využití podstatně menší než u diagramů sekvence, protože podmínky se rozšíří po celém diagramu a ten se brzy stane nepřehledný. Doporučuje se proto na diagramu komunikace ukazovat jen velmi jednoduché větvení. Mnohem jednodušší je ukázat složitá větvení na diagramu sekvence.

Diagramy komunikace jsou velmi vhodné k prvotnímu rychlému načrtnutí spolupráce objektů a při iterativním modelování metodou pokusů a omylů. Proto jsou užitečné při diskuzích a brainstormingu.

4.2.5 Diagram nasazení

Ukazuje nasazení softwarových prvků na fyzickou architekturu a dále ukazuje komunikaci (zpravidla na síti) mezi fyzickými prvky. Jinými slovy, mapuje softwarovou architekturu, která je výsledkem návrhu, na fyzickou architekturu systému. V případě distribuovaného systému modeluje distribuci software ve fyzických uzlech. Diagram nasazení je užitečný pro komunikaci vývojářů navzájem a se zákazníkem v záležitostech týkajících se celkové (včetně hardware) architektury systému. Ukázka Obr.(4.8)

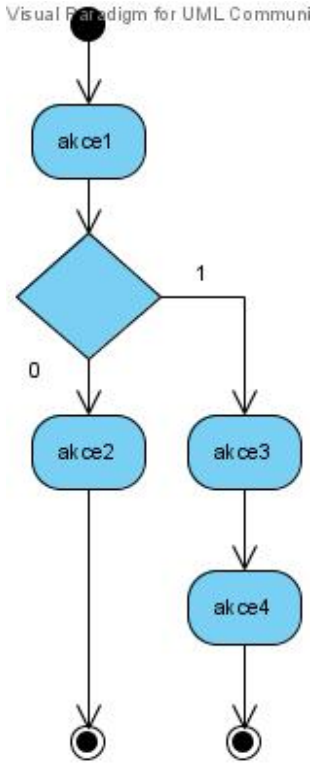
4.3 Vlastní návrh

Na základě prostudování materiálů ke tvorbě návrhu systémů, vznikly následující diagramy. Diagram aktivit Obr.(4.9), diagram tříd Obr.(4.10), sekvenční diagramy Obr.(4.11 a 4.12), diagram komunikace Obr.(4.13) a diagram nasazení Obr.(4.14).

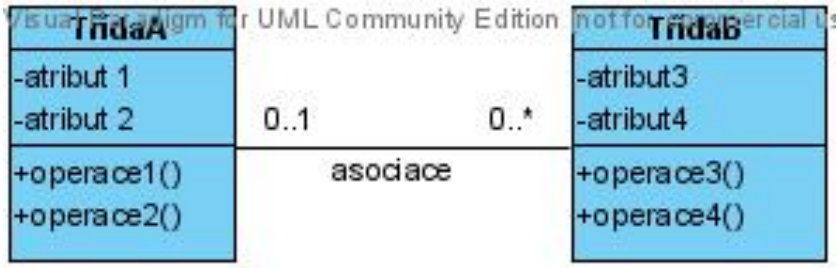
- Diagram aktivit – Zde je navržena představa fungování *SpravaZapasu*. Možnosti co vše je možné dělat se Zápasem a akcemi v něm.
- Diagram tříd – Tady už je jasně zřetelná struktura celého systému. Rozvržení pomocí architektury MVC.
- Sekvenční diagram – Na těchto dvou diagamech je patrné volání zpráv jednotlivých tříd při přihlašování a přídávání gólu.
- Diagram komunikace – Zde je patrné chování při přihlašování.
- Diagram nasazení – Na tomto diagramu je uvedeno jak bude vypadat nasazení systému do provozu.

4.4 Literatura

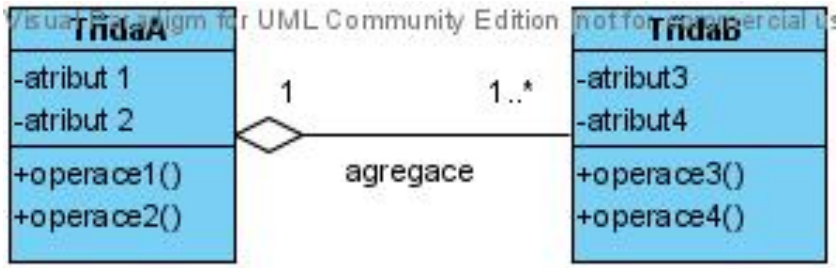
V této kapitole byly použity následující publikace [12, AIS] a webové stránky [8, Objekty].



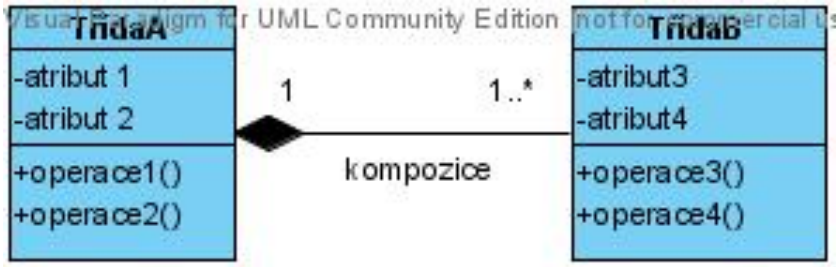
Obrázek 4.1: příklad diagramu aktivit



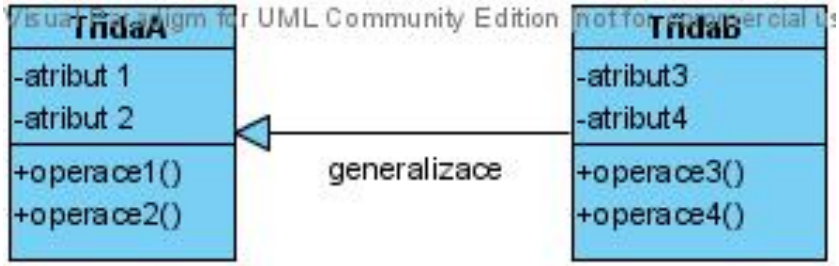
Obrázek 4.2: příklad asociace



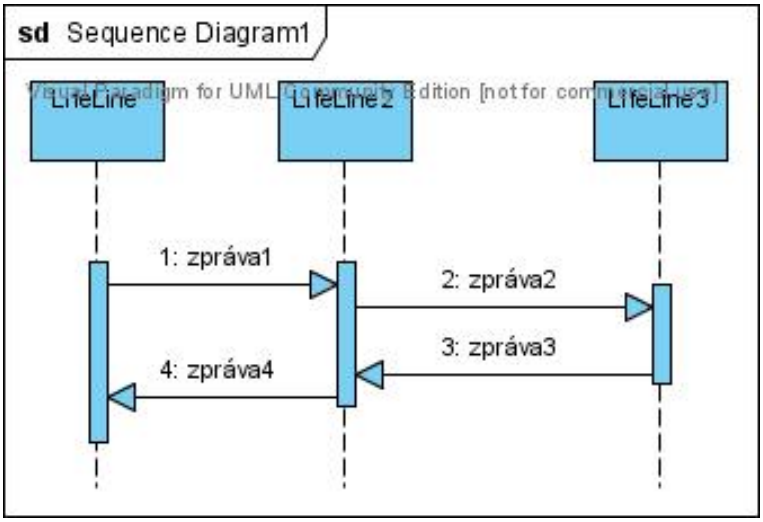
Obrázek 4.3: příklad agregace



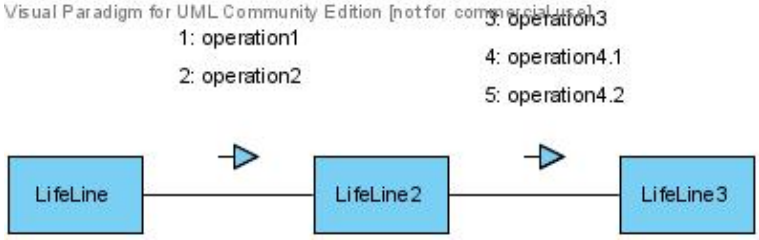
Obrázek 4.4: příklad kompozice



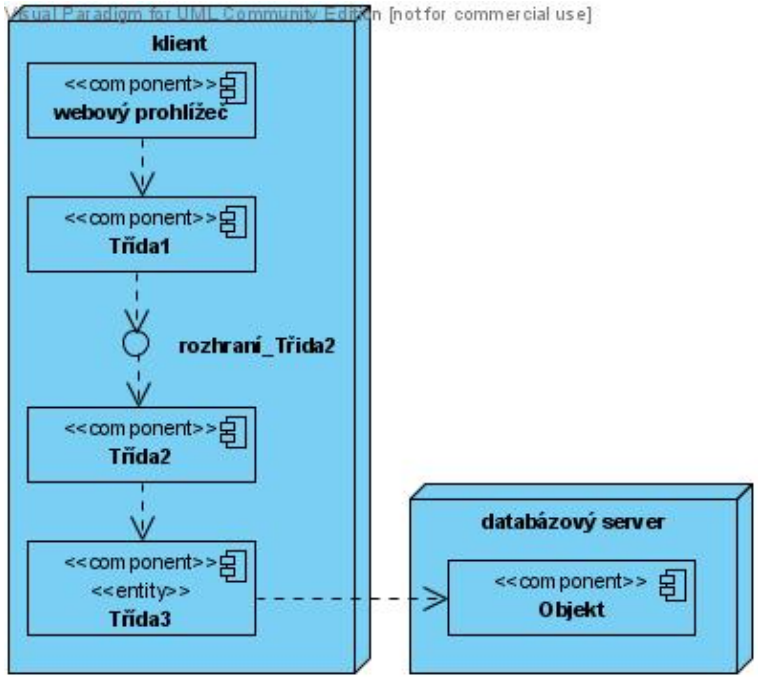
Obrázek 4.5: příklad generalizace



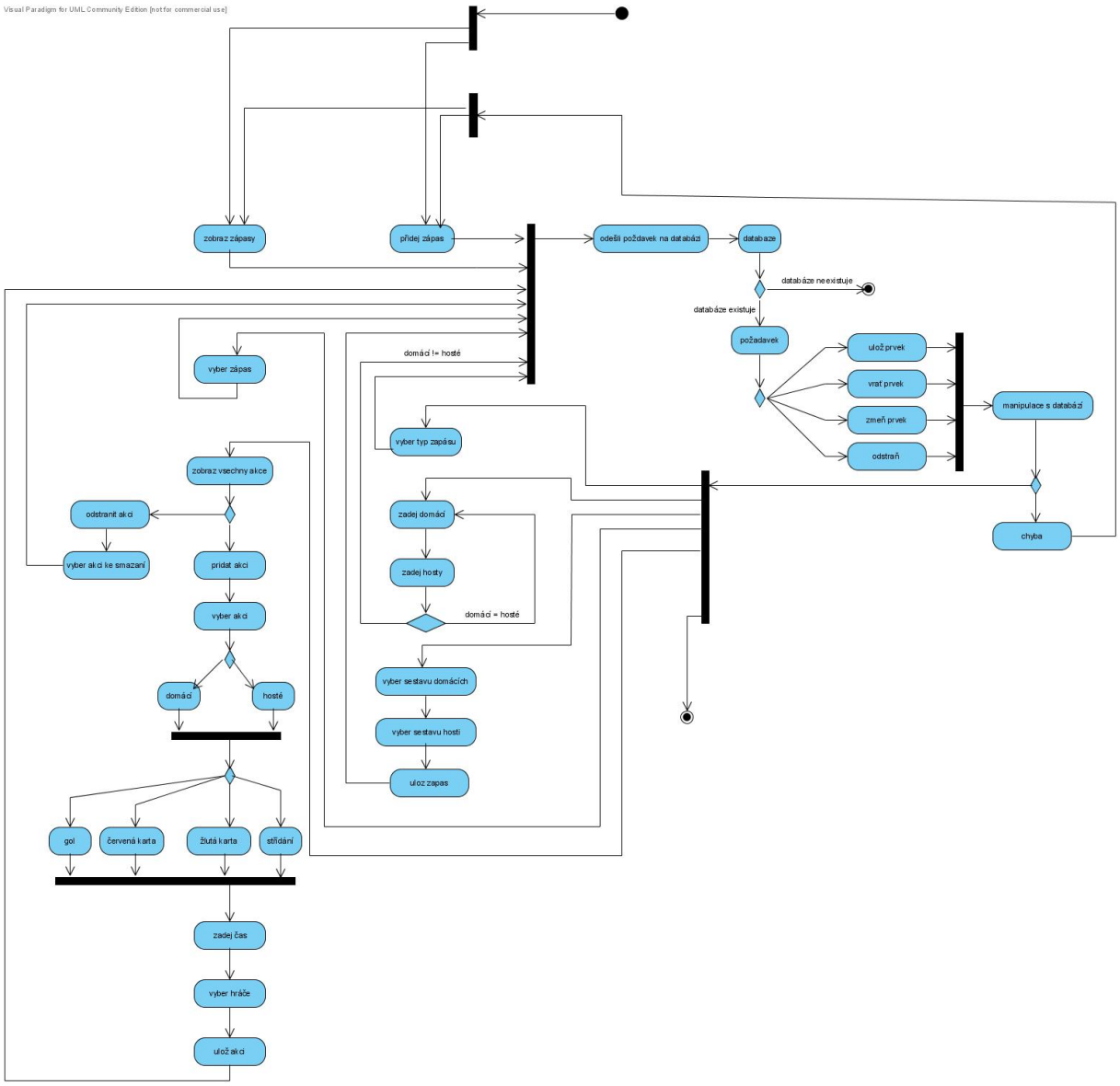
Obrázek 4.6: příklad diagramu sekvence



Obrázek 4.7: příklad diagramu komunikace

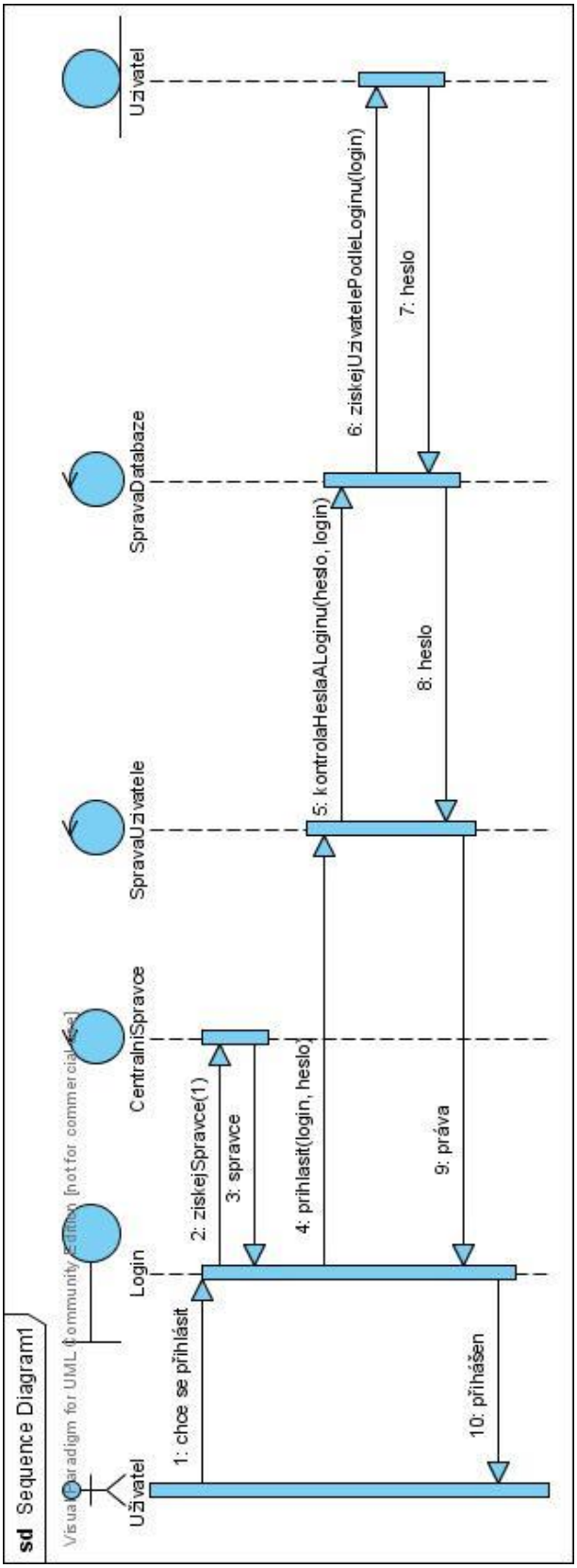


Obrázek 4.8: příklad diagramu nasazení

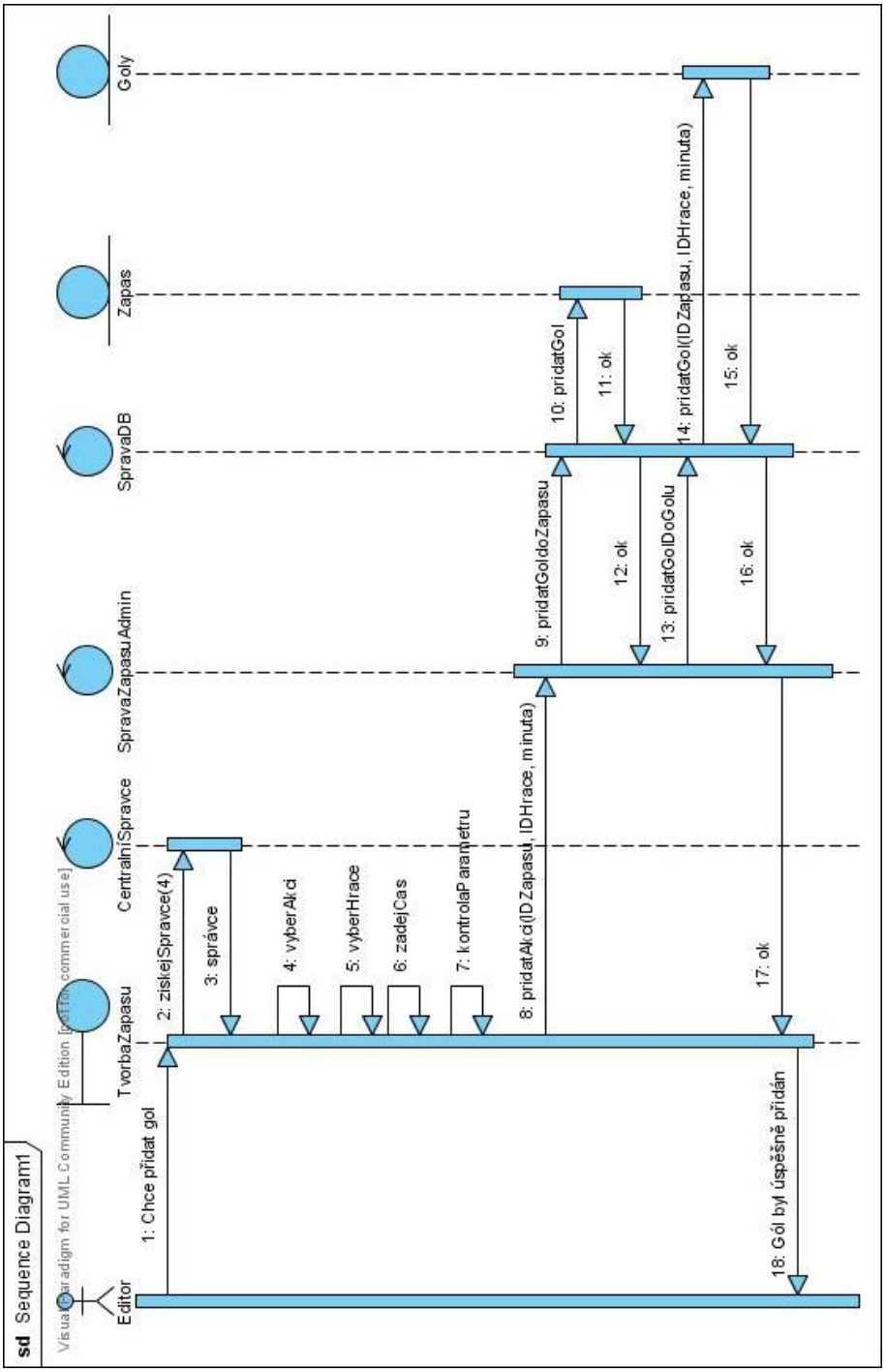


Obrázek 4.9: výsledný diagram aktivit pro přidání akce a zápasu

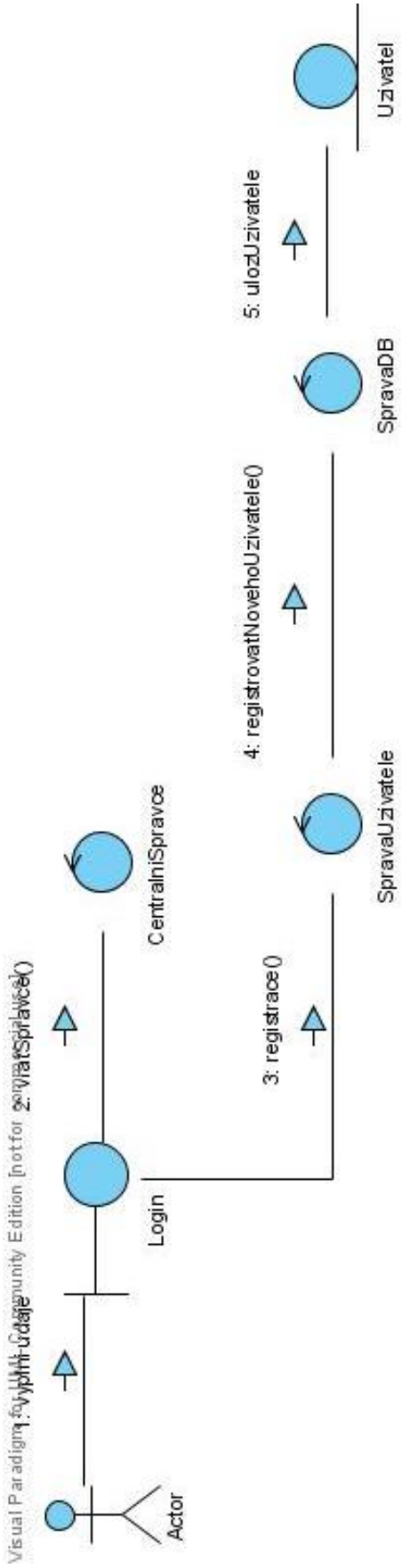
Obrázek 4.10: výsledný diagram tříd



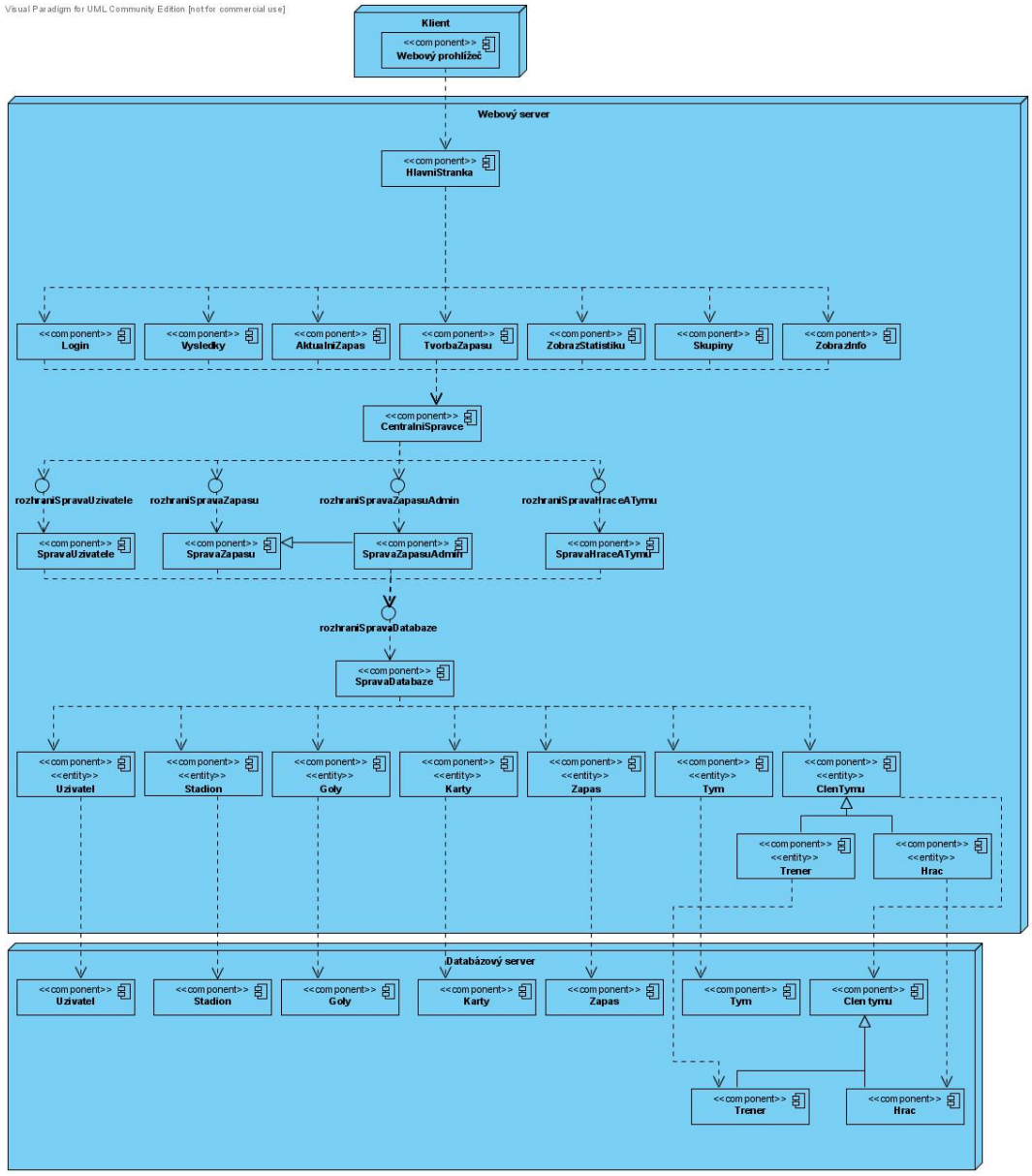
Obrázek 4.11: sekvenční diagram přihlášení



Obrázek 4.12: sekvenční diagram přidání gólu



Obrázek 4.13: diagram komunikace registrace



Obrázek 4.14: výsledný diagram nasazení

Kapitola 5

Implementace

Tato kapitola se zabývá demonstrační implementací systému. Rozebere si zde některé vlastnosti jazyka PHP. Přiblížíme si JavaScript a MySQL. Dále si zmíníme některa pravidla tvorby uživatelských rozhraních.

5.1 PHP

5.1.1 O PHP

PHP (rekurzivní zkratka PHP: Hypertext Preprocessor, „PHP: Hypertextový preprocesor“, původně Personal Home Page) je skriptovací programovací jazyk, určený především pro programování dynamických internetových stránek. Nejčastěji se začleňuje přímo do struktury jazyka HTML, XHTML či WML, což je velmi výhodné pro tvorbu webových aplikací. PHP lze ovšem také použít i k tvorbě konzolových a desktopových aplikací.

PHP skripty jsou prováděny na straně serveru, k uživateli je přenášen až výsledek jejich činnosti. Syntaxe jazyka kombinuje hned několik programovacích jazyků (Perl, C, Pascal a Java). PHP je nezávislý na platformě, skripty fungují bez úprav na mnoha různých operačních systémech. Obsahuje rozsáhlé knihovny funkcí pro zpracování textu, grafiky, práci se soubory, přístup k většině databázových serverů (mj. MySQL, ODBC, Oracle, PostgreSQL, MSSQL), podporu celé řady internetových protokolů (HTTP, SMTP, SNMP, FTP, IMAP, POP3, LDAP,. . .)

PHP se stalo velmi oblíbeným především díky jednoduchosti použití a tomu, že kombinuje vlastnosti více programovacích jazyků a nechává tak vývojáři částečnou svobodu v syntaxi. V kombinaci s databázovým serverem (především s MySQL nebo PostgreSQL) a webovým serverem Apache je často využíván k tvorbě webových aplikací. Díky velmi častému nasazení na serverech se vžila zkratka LAMP - tedy spojení Linux, Apache, MySQL a PHP nebo Perl.

S verzí PHP 5 se výrazně zlepšil přístup k objektově orientovanému programování podobný Javě.

5.1.2 Historie

Od roku 1994 je PHP jedním z nejpoužívanějších způsobů tvorby dynamicky generovaných WWW stránek. Jeho tvůrce (Rasmus Lerdorf) jej vytvořil pro svou osobní potřebu přepsáním z Perlu do jazyka C. Sada skriptů byla vydána ještě v témže roce pod názvem Personal Home Page Tools, zkráceně PHP.

V polovině roku se systém PHP spojil s programem Form Interpreter stejného autora. Tak vzniklo PHP/FI 2.0. Zeev Suraski a Andi Gutmans v roce 1997 přepsali parser a zformovali tak základ PHP3. Současně byl název změněn na dnešní podobu PHP Hypertext Preprocessor. PHP3 vyšlo v roce 1998, bylo rychlejší, obsahovalo více funkcí. Také běželo i pod operačním systémem Windows.

V roce 2000 vychází PHP verze 4, o čtyři roky později pak verze 5 s vylepšeným objektovým přístupem, podobným jazyku Java.

5.1.3 Výhody a Nevýhody

Výhody:

- PHP je relativně jednoduché na pochopení...
- PHP má syntaxi velmi podobnou jazyku C a je tedy většině vývojářů dost blízký...
- PHP podporuje širokou řadu souvisejících technologií, formátů a standardů...
- je to otevřený projekt s rozsáhlou podporou komunity...
- dají se najít kvanta již hotového kódu k okamžitému použití nebo funkční PHP aplikace. Podstatná část z hotového kódu je šířena pod nějakou svobodnou licencí a dá se použít ve vlastních projektech...
- PHP si dobře rozumí s webovým serverem Apache...
- PHP snadno komunikuje s databázemi, jako je MySQL, PostgreSQL a řada dalších...
- PHP je multiplatformní a lze jej provozovat s většinou webových serverů a na většině dnes existujících operačních systémů...
- PHP podporuje mnoho existujících poskytovatelů webhostingových služeb...

Nevýhody:

- PHP je interpretovaný, ne kompilovaný jazyk...
- kdokoli má přímý přístup k serveru, může nahlédnout do vašich PHP skriptů...
- protože je PHP aktivně vyvíjen, v budoucích verzích jazyka se mohou některé funkce změnit nebo se mohou chovat jinak než dosud...

5.2 JavaScript

Jde o multiplatformní, objektově orientovaný skriptovací jazyk, jehož autorem je Brendan Eich z tehdejší společnosti Netscape.

Nyní se zpravidla používá jako interpretovaný programovací jazyk pro WWW stránky, často vkládaný přímo do HTML kódu stránky. Jsou jím obvykle ovládány různé interaktivní prvky GUI (tlačítka, textová políčka) nebo tvořeny animace a efekty obrázků.

Jeho syntaxe patří do rodiny jazyků C/C++/Java. Slovo Java je však součástí jeho názvu pouze s marketingových důvodů a s programovacím jazykem Java jej vedle názvu spojuje jen podobná syntaxe. JavaScript byl v červenci 1997 standardizován asociací ECMA

(Europen Computer Manufacturers Association) a v srpnu 1998 ISO (International Organization for Standardisation). Standardizovaná verze JavaScriptu je pojmenována jako ECMAScript a z ní byly odvozeny i další implementace, jako je například ActionScript. Používaný při vývovoji Flash aplikací v naástrojích od společnosti Adobe.

JavaScript byl původně obchodní název implementace společnosti Netscape, kde byl vyvíjen nejprve pod názvem Mocha, později LiveScript, ohlášen byl společně se společností Sun Microsystems v prosinci 1995 jako doplněk k jazykům HTML a Java. Pro verzi firmy Microsoft je použit název JScript.

Program v JavaScriptu se obvykle spouští až po stažení WWW stránky z Internetu (tzv. na straně klienta), na rozdíl od ostatních jiných interpretovaných programovacích jazyků (např. PHP a ASP), které se spouštějí na straně serveru ještě před stažením z Internetu. Z toho plynou jistá bezpečností omezení, JavaScript např. nemůže pracovat se soubory, aby tím neohrozil soukromí uživatele.

5.3 MySQL

S databázemi se samozřejmě setkáváme denně. V nejširším slova smyslu je databáze i seznam věcí, které si můžeme nakoupit k obědu, výpis z účtu nebo třebas seznam uskutečněných telefonních hovorů. V počítačovém slova smyslu se pod výrazem „databáze“ obvykle rozumí software, který spravuje určitý „balík“ dat a umožňuje uživatelům tento „balík“ dat nějak rozumně měnit a spravovat.

Databáze není jen prosté shromaždiště, úložiště dat, ale že většinou slouží zároveň k jejich organizaci, třídění, prohledávání, seskupování a podobně. K typické dnešní databázi patří rovněž to, že zároveň chce s daty pracovat více uživatelů.

S tím souvisí jiná důležitá věc: S rozvojem sítí a zejména internetu se stalo obvyklé, že databáze mohou být přístupné vzdáleně. Taková databáze umístěná centrálně může poskytovat data mnoha víceméně nezávislým odběratelům. Z celého toho popisu bude pravděpodobně všem zřejmé, že databázové programy obecně zažívají zlaté časy a že databází bude existovat celá řada.

Rozdělit databáze je kupodivu čím dál tím složitější, protože jednotlivá kritéria se v poslední době vzájemně překrývají a mnoho databází umí hodně podobných věcí. Nicméně, pokusíme se alespoň nastínit, jak různě se dají databáze dělit.

- *Objektové* a *relační* – tam se databáze liší podle způsobu, jak ukládají data. Dle datového modelu. Relačni schéma dat a objektový pohled na data. Ve světě převládá relační model...

- *Jednouživatelské* a *víceuživatelské* – Podle toho, kolik uživatelů se k databázi může připojit. Pochopitelně, v komerční sféře jednouživatelským databázím prakticky odzvonilo. Je ale užitečné vědět, že i víceuživatelskou databázi lze většinou nakonfigurovat jako jednouživatelskou a že to může mít své opodstatnění. MySQL je víceuživatelská databáze...

- *Souborové* a *systémové* – liší se tím, zda použijí pro uložení dat jeden soubor, nebo zda je úložiště dat nějak zabudováno do systému. U souborových databází typicky stačí příslušný soubor přenést na jiný stroj a může být ihned používán, u systémových databází se musí záloha a obnova dělat nějak jinak. V poslední době toto dělení poněkud ztratilo význam, protože většina databází je systémových. MySQL je systémová databáze...

- Podle *licence* a *ceny* – Kód databáze může být uzavřený nebo otevřený, šíření software může být svobodné nebo může podléhat nějakým podmínkám, databáze se může využívat bez poplatků nebo může jít o placené software. Licence MySQL je duální, je trochu složitá a nebudeme se jí v této práci zabývat...
- Podle toho, kde je provozována – na *jednoplatformní* a *multiplatformní* – Jednoplatformní poběží jen na některém systému (třeba na Windows), multiplatformní na více systémech. MySQL je multiplatformní databáze a běží na systémech GNU/Linux, Microsoft Windows, FreeBSD, Sun Solaris, IBM's AIX, Mac OS X, HP-UX, AIX, QNX, Novell NetWare, SCO OpenUnix, SGI Irix, and Dec OSF...
- Podle *velikosti*, *výkonu* a *vhodnosti nasazení* – Databáze mívají limity ve velikosti, počtu současně přihlášených uživatelů, počtu současně probíhajících procesů a podobně. Je těžké někam zařadit MySQL. Obecně je totiž obtížné nějak rovnoprávně posoudit databáze. Diplomaticky můžeme říci, že MySQL je někde uprostřed pomyslného žebříčku vhodnosti nasazení a že v malých až středních projektech ji rozhodně použít můžete...

Databáze lze samozřejmě dělit i podle celé řady dalších kritérií. Nejdůležitější přitom je, co všechno daná databáze „umí“ s daty provádět.

MySQL je velmi populární databáze. Podle mnoha dostupných studií je to rovněž velmi rychlá databáze. Nemá však tolik funkcí a možností jako některé konkurenční databázové systémy. Vybrat si vhodnou databázi je tedy klasický kompromis mezi rychlostí softwaru a jeho schopnostmi. MySQL má samozřejmě své zastánce a odpůrce. Odpůrci například často tvrdí, že MySQL je tak rozšířená proto, že webhostingové společnosti ji často nabízejí pro hostované weby jako součást portfolia svých služeb. Kdyby se webhostingové společnosti rozhodly upřednostnit jiný software, popularita MySQL by podle jejich odpůrců podstatně klesla.

Naproti tomu zastánci MySQL tvrdí, že webhostingové společnosti nabízejí MySQL proto, že je pro nabízené služby vhodná, a kdyby existovala lepší databáze než MySQL, byla by nabízena. Zastánci MySQL prostě tvrdí, že trh si MySQL vybral - a to podle nich jasně poukazuje na její kvality.

5.4 CSS

HTML je značkovací jazyk, z toho vyplývá, že jednotlivé značky mají vyznačovat význam jednotlivých částí textu. Přesto se v HTML v průběhu let objevilo několik atributů a elementů, které ovlivňují pouze grafický vzhled, a některé z původních tagů určených pro logiské členění textu, je často účelově využíváno k dosažení určitých grafických efektů. Aby se předešlo těmto nešvarům, a bylo možné jednoduše oddělit strukturální formátování textu od grafického formátování, definovalo v roce 1997 koncorsium W3C soubor metod pro grafickou úpravu webových stránek. Tento soubor metod se jmenuje Cascading Style Sheets (CSS), česky „kaskádové styly“.

Kaskádové styly umožňují definovat způsob zobrazení (velikost a druh písma, barvu, zarovnání, orámování, pozadí apod.) u každého html elementu. Tento předpis vzhledu není přímo součástí textu stránky, a díky tomu je zápis stránky přehlednější a dobře strukturovaný. Navíc styly umožňují definovat jednotný vzhled konkrétního elementu v celém dokumentu jedním zápisem tzn. nemusí se opakovat u každého elementu. Pokud styl uložíme do externího souboru, může ho využívat více stránek najednou. Definice vzhledu všech stránek

je uložena na jednom místě. Při požadavku na změnu vzhledu stránek, stačí upravit styl na jednom místě, a změny se automaticky promítnou do všech stránek.

Jak je běžné ve světě webdesignu, i podpora kaskádových stylů v jednotlivých prohlížečích webových stránek je různá. Jednotlivé prohlížeče mají vlastní výklad či pravidla CSS.

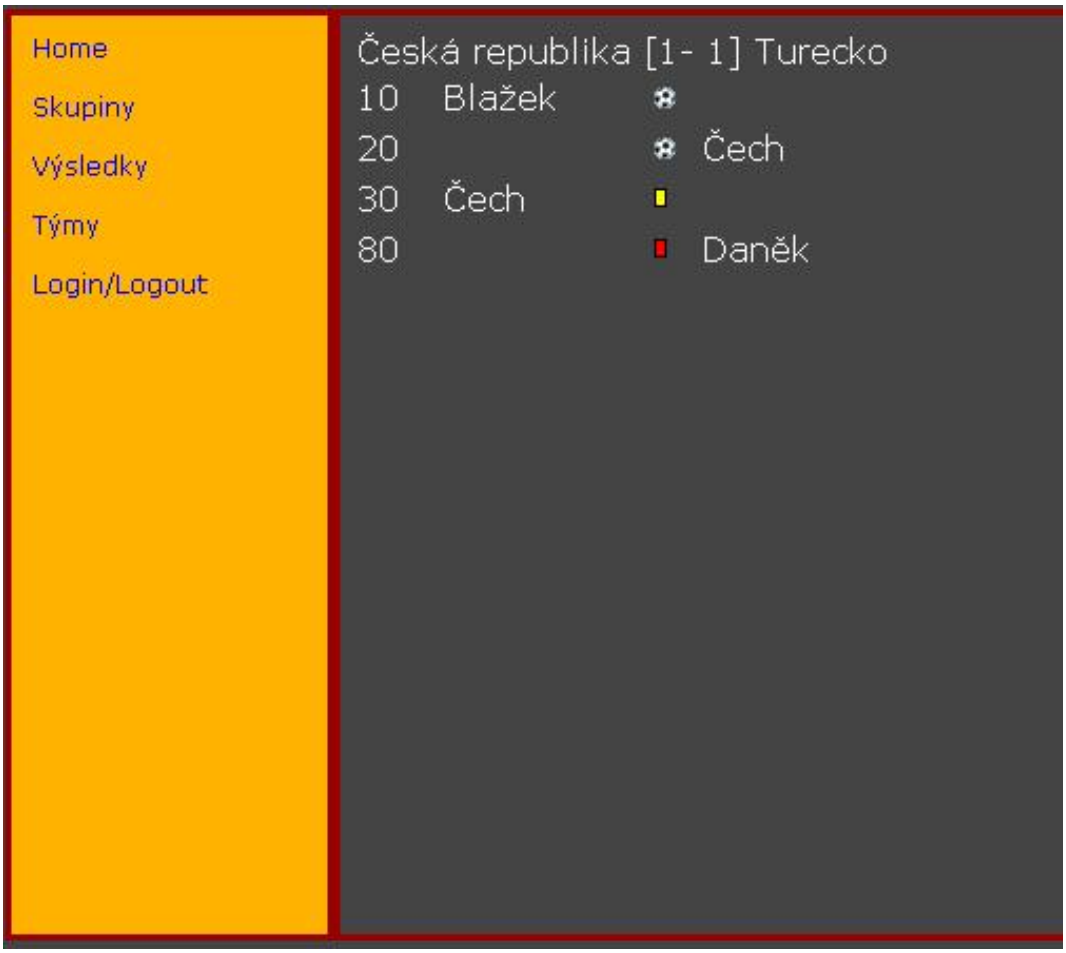
5.5 Přehled systému

Zde si na několika screenshotech ukážeme vzhled systému Obr.(5.1, 5.2, 5.3). A nektěrou z naplněných tabulek Obr.(5.4).

Obrázek 5.1: vzhled systému

5.6 Literatura

V této kapitole byla použity tyto knihy [3, PHP] a webové stránky [7, Linuxsoft], [11, interval], [9, phpNet].



Obrázek 5.2: vzhled systému

ID	Druh	Domaci	Hoste	Skore		Divaci	Stadion		
14	SK	Turecko	Česká republika	-	-	30000	Stade de Suisse Wankdorf	upravit	smazat
13	CT	Česká republika	Turecko	-	-	25000	Stadion Salzburg Wals-Siezenheim	upravit	smazat
12	SK	Česká republika	Turecko	-	-	29000	Stadion Tivoli	upravit	smazat
11	SK	Česká republika	Turecko	-	-	29000	Stadion Tivoli	upravit	smazat
10	CT	Česká republika	Turecko	-	-	25000	Stadion Salzburg Wals-Siezenheim	upravit	smazat
9	CT	Česká republika	Chorvatsko	-	-	25000	Stade de Suisse Wankdorf	upravit	smazat
8	SK	Česká republika	Švýcarsko	-	-	25000	Ernst Happel Stadion	upravit	smazat
7	SK	Česká republika	Turecko	-	-	29000	Stadion Tivoli	upravit	smazat
6	SK	Turecko	Česká republika	-	-	30000	Stade de Suisse Wankdorf	upravit	smazat
5	SK	Turecko	Česká republika	-	-	30000	Stade de Suisse Wankdorf	upravit	smazat

Obrázek 5.3: vzhled systému

←T→			logovaciJmeno	heslo	id	jmeno	prijmeni	datum	cas	prava
☐			uzivatel	uzivatel	151360	uzivatel	uzivatel	0	0	U
☐			admin	admin	151358	admin	admin	100508	1626	A
☐			editor	editor	151359	editor	editor	0	0	E

Obrázek 5.4: tabulka User

Kapitola 6

Závěr

V této kapitole zhodnotíme dosažené výsledky a budeme zde diskutovat možnosti rozšíření.

6.1 Splnění požadavků zadání

Na základě zadání byl proveden detailní návrh systému. Ten je popsán pomocí diagramů v kapitole 4.3. Diagramy byly zvoleny vhodně na základě prostudování metod tvorby návrhu.

Diagramy byly vytvořeny v programu Visual Paradigm (Community Edition). Program v mnohém převyšuje prostředky Rational Rose. Který jsme používali v 1.ročníku bakalářského studia. Visual Paradigm ma velmi prehledné uživatelské rozhraní. A dovoluje tvorbu široké škály UML diagramů. Pokud se vytvoří všechny diagramy, tak dovoluje vygenerovat část kódu.

Cílem práce bylo vytvořit pouze demonstrační implementaci, která bude demonstrovat pouze základní funkčnost navrženého systému. Hlavní implementované funkce jsou přihlašování, odhlašování, registrace uživatele, prohlížení týmů a hráčů, přidávání hráčů, vytvoření tabulky skupin, výsledky zápasů, správa účtů a nastavení.

Celý IS se bez problémů korektně zobrazuje v dnešní době v nejvíce rozšířených webových prohlížečích, jimiž jsou Internet Explorer 6, Opera 9.0 a vyšších, Mozilla Firefox 2.0 a vyšších.

6.2 Srovnání s ostatnímy systémy

Celkové srovnání s jinými webovými systémy je prakticky nemožné, protože webových systému existuje příliš velký počet, přičemž jejich kvalita se výrazně liší systém od systému. Ovšem podstatné klady vytvořeného systému jsou v tom, že je validní vzhledem k webovým standardům, snaží se udržovat dobrou přehlednost a ovladatelnost, atd.

6.3 Rozšíření

Systém umožňuje mnoho rozšíření. A to skoro v každé části.

Jednou z možných rozšíření je vytvoření jednoducheho diskuzního fóra k jednotlivým zápasům, kde by mohli registrovaní uživatelé připisovat sve názory, postřehy apod.

Další možností je odesílaní SMS o průběhu aktuálního zápasu a výsledném stavu zápasu. Samozřejme by zde musela být určitá forma zabezpečení, aby důvěrná data nebyly volně přístupné pro ostatní uživatele tzn. šifrovaná komunikace mezi klientem a serverem.

Kromě možností úpravy funkčnosti, lze také upravit pomoc CSS vzhled systému. Barvy, rozložení na stránce apod.

6.4 Literatura

K tvorbě práce bylo postupováno podle pokynu na školních stránkách viz. [10, fitWeb].

Literatura

[1] Jim Arlow. *UML a unifikovaný proces vývoje aplikací.* Computer Press, 2003.

[2] Eric Freeman. *Head First Design Patterns.* 2004.

[3] Jason Gilmore. *Velká kniha PHP 5 MySQL :kompendium znalostí pro začátečníky i profesionály.* Zoner Press, 2005.

[4] Tomáš Hruška. *Studijní opora k předmětu IIS.* 2006.

[5] Brett McLaughlin. *Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOAD.* 2006.

[6] R Pecinovský. *Návrhové vzory.* Computer Press, 2007.

[7] WWW stránky. Mysql, javascript, php. `http://www.linuxsoft.cz`.

[8] WWW stránky. Objektová analýza, návrh a programování. `http://www.objekty.vse.cz`.

[9] WWW stránky. Php. `http://www.php.net/`.

[10] WWW stránky. Pokyny ke zpracování diplomových a ročníkových projektů. `http://www.fit.vutbr.cz/info/statnice/`.

[11] WWW stránky. Tvorba webových stránek. `http://www.interval.cz`.

[12] Jaroslav Zendulka. *Analýza a návrh informačních systémů.* 2006.

[13] Jaroslav Zendulka. *Studijní opora k předmětu IDS.* 2006.