VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

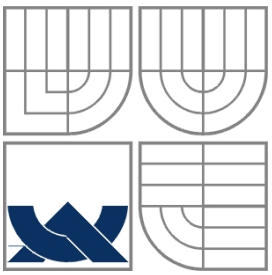
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ROZŠIŘITELNÝ INFORMAČNÍ SYSTÉM SDRUŽENÍ SDC S VÍCEVRSTVOU ARCHITEKTUROU

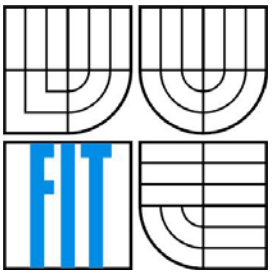
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Dušan Vrážel
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ROZŠIŘITELNÝ INFORMAČNÍ SYSTÉM SDRUŽENÍ SDC S VÍCEVRSTVOU ARCHITEKTUROU

Extendable Information System of SDC with a Multi-Tier Architecture

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Dušan Vrážel
AUTHOR

VEDOUCÍ PRÁCE — doc., Ing. Jaroslav Zendulka, CSc.
SUPERVISOR

BRNO 2007

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2006/2007

Zadání diplomové práce

Řešitel: **Vrážel Dušan**
Obor: Výpočetní technika a informatika
Téma: **Rozšiřitený informační systém sdružení SDC s vícevrstvou architekturou**
Kategorie: Databáze

Pokyny:

1. Prostudujte problematiku vícevrstvých softwarových architektur a architektonických vzorů.
2. Seznamte se s požadavky kladenými na informační systém sdružení SDC (IS SDC).
3. Navrhněte vícevrstvou architekturu IS SDC, která bude umožňovat snadné rozšiřiřování formou přídavných modulů.
4. Proveďte detailní návrh a systém implementujte použitím programovacích prostředků XHTM, PHP5 a MySQL5.
5. Při vývoji využijte modelovacích technik jazyka UML.
6. Ověřte funkčnost systému, včetně vlastností vícevrstvé architektury a rozšiřitelnosti. Rozsah ověření konzultujte s vedoucím DP.
7. Zhodnoťte výsledky a možnosti dalšího vylepšení systému.

Literatura:

- Software Architecture. Wikipedia. Dostupné na adrese http://en.wikipedia.org/wiki/Software_architecture.
- Larman, C.: Applying UML and Patterns. Third edition. Prentice Hall PTR. 2005.
- Naramore, E. et al.: PHP5, MySQL, Apache: vytváříme webové aplikace. Computer Press. 2006.
- Dokumentace k MySQL. Dostupné na adrese http://www.mysql.org/doc/.

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zendulka Jaroslav, doc. Ing., CSc.**, UIFS FIT VUT
Konzultant: Švec Jaroslav, Ing., UIFS FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 22. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Dušan Vrážel**
Id studenta: 20758
Bytem: Raťkovská 497, 756 05 Karolinka
Narozen: 07. 08. 1981, Vsetín
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
diplomová práce

Název VŠKP: Rozšiřitený informační systém sdružení SDC s vícevrstvou architekturou
Vedoucí/školitel VŠKP: Zendulka Jaroslav, doc. Ing., CSc.
Ústav: Ústav informačních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Tato diplomová práce se zabývá analýzou, návrhem a implementací informačního systému se snadnou rozšiřitelností jednotlivých části systému s využitím třívrstvé architektury a návrhového vzoru Model-view-controller.

Systém je vyvíjen za pomocí objektově orientovaného přístupu v programovacím jazyku PHP, relačního databázového serveru MySQL a webových služeb s využitím protokolu SOAP. Při implementaci zobrazovací vrstvy bylo využito webových technologii XHTML a CSS. Pro analýzu a návrh systému byl použit modelovací jazyk UML.

V práci je popsána problematika vícevrstvých softwarových architektur, teorie tvorby Informačních systému, teorie webových služeb, architektonické vzory informačních systémů, dále je popsána analýza, návrh a implementace rozšiřitelného informačního systému s třívrstvou architekturou a jeho aplikace v prostředí sdružení SDC.

Klíčová slova

Informační systém, PHP, OOP, objektově orientované programování, XHTML, CSS, vícevrstvé architektury informačních systémů, třívrstvá architektura systému, architektonické vzory, vzor Model-view-controller, UML, SQL, MySQL, rozhraní, webové služby, WSDL, SOAP, SSL.

Abstract

This master's thesis deals with analysis, design and implementation of an information system that allows easy extensibility with three-tier architecture.

System is developed using object oriented approach by programming language PHP, relational database server MySQL and web services and protocol SOAP. The View tier is implemented by web technologies XHTML and CSS. Analysis and design of this system is done with using the modeling language UML.

In this thesis is described the problem of multi-tier software architecture, theory of development information's systems and described design and implementation of an information system with a three-tier architecture and it's application in the society SDC.

Keywords

Information system, PHP, OOP, object oriented programming, XHTML, CSS, multi-tier architectures in information's systems, three-tired architecture of information's system, architectural patterns, pattern Model-View-Controller, UML, SQL, MySQL, interface, web services, WSDL, SOAP, SSL.

Citace

Dušan Vrážel: Rozšiřitelný informační systém sdružení SDC s vícevrstvou architekturou, diplomová práce, Brno, FIT VUT v Brně, 2007

Rozšiřitelný informační systém sdružení SDC s vícevrstvou architekturou

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc., Ing. Jaroslava Zendulky, CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Dušan Vrážel
20. května 2007

Poděkování

Děkuji doc. Ing. Jaroslavu Zendulkovi, CSc. za odborné vedení, rady a podněty, které mi během práce poskytoval.

Obsah

1 Úvod

Tato práce dokumentuje vznik informačního systému, založeném na třívrstvé architektuře, architektonickém vzoru Model-View-Controller a webových službách.

Má motivace pro výběr tohoto tématu byla nejen potřeba tohoto informačního systému v praxi sdružením SDC, ale také vyzkoušet použití moderních technologií jako webové služby, architektonické vzory a třívrstvá architektura v kombinaci s jazykem PHP a porovnat případné výhody a nedostatky. Volbu tématu také ovlivnil můj kladný vztah k technologiím PHP, objektově orientovanému programování a MySQL.

Občanské sdružení Společnosti Diamantové cesty vzniklo v roce 1998 a zabývá se kulturním a společenským odkazem Tibetského buddhismu. Mezi hlavní aktivity sdružení patří překlad knih, vydávání vlastních knih a časopisu zabývající se tématikou buddhismu, pořádání přednášek a kulturních akcí a to i na mezinárodní úrovni ve spolupráci s podobnými organizacemi v Evropě.

Sdružení spojuje více než tři sta mladých lidí ze 40 center po celé České republice, kteří se učí navzájem komunikovat a spolupracovat na různých projektech a aktivitách. Kvalitní informační systém proto velmi ulehčí komunikaci mezi jednotlivými týmy i členy sdružení a umožní přiblížit výsledky jejich činností široké veřejnosti.

V následujících kapitolách jsou popsány cíle práce a charakteristika současného stavu. Dále je práce zaměřena na teoretické pozadí jednotlivých aspektů a částí vývoje tohoto informačního systému. V dalších kapitolách této práce je detailně popsán návrh, vznik a implementace systému a v závěrečné kapitole jsou shrnuty výsledky práce, popsány získané poznatky a navrženo rozšíření.

Tato práce přímo navazuje na výsledky semestrálního projektu, jehož cílem bylo seznámit se s problematikou informačních systémů, problematikou vícevrstvých architektur a obeznámit se s teoretickými základy práce. Dále byla v semestrální práci provedena analýza současného stavu a seznámení se s požadavky kladenými na informační systém sdružení SDC.

2 Cíle práce

Tato kapitola se zabývá popisem cílů práce. Cíle práce přímo vycházejí ze zadání práce. Cílem kapitoly je tedy šíře rozepsat důvody, které vedly k formulování zadání této práce.

V úvodní kapitole je pozornost soustředěna na obecný popis základních cílů, v následujících kapitolách jsou pak podrobněji popsány vybrané cíle práce.

2.1 Hlavní cíle práce

Cílem práce je navrhnout, vytvořit a popsat informační systém, který by byl snadno rozšiřitelný o nové prvky formou přídavných modulů. Systém, který by umožňoval snadné sdílení dat těchto modulů mezi sebou. Systém snadno udržovatelný a nezávislý na použitých technologiích. Systém, jenž poskytuje komfort programátorovi, administrátorovi i uživateli. A v neposlední řadě systém nezávislý na jeho konkrétním nasazení a aplikaci.

Jedním z cílů této práce je reflexe, zda je za daných podmínek možné dosáhnout cílů vytyčených v zadání, jakým způsobem, či s jakými omezeními je možné daného cíle dosáhnout a zhodnotit, jaké výhody či problémy přinášejí jednotlivá řešení.

Dalším (pro autora neméně důležitým) cílem bylo naučit se vytvářet komplexnější programové dílo, naučit se procházet jednotlivými fázemi vývoje informačního systému a programového díla, nastudovat nové témata a získat tak širší pohled na problematiku informačních systému a jejich použití v prostředí internetu. V neposlední řadě to je osvojení si práce s textem v mnohdy zahraniční literatuře.

V následujících podkapitolách jsou podrobněji popsány formální cíle definované v prvním odstavci.

2.2 Snadná rozšiřitelnost systému

V současné době existuje více typů informačních systému z pohledu rozšiřitelnosti.

První skupinu tvoří systémy, které nejsou připravené na rozšiřitelnost. Pokud se rozhodneme přidat nový prvek k těmto systémům, vede to mnohdy k předělání celého systému, nebo různým náročným slepováním.

Dále jsou to systémy, které umožňují přidávat další části za zachování určitých podmínek, rozhraní apod. Těchto systému je celá řada s různou možností rozšiřitelnosti.

Poslední skupinou jsou systémy, které se snaží být co nejuniverzálnější, dosáhnout co největší flexibility za cenu velkých nároků na systémové prostředky, údržbu a administraci.

Cílem tohoto informačního systému je poskytnout rozšiřitelnost na více úrovních.

Pokusit se vytvořit systém, do kterého bude možnost **snadno přidávat nové moduly**, čili funkce, jako fotogalerie, redakční systém, mailing listy apod. Tím rozšiřovat jeho model a funkčnost (více o modulech v kapitole 5.1.6).

Další rozšiřitelnost získáme v možnosti **snadno přidávat nové sekce**, jako jsou například stránky nové pobočky a prezentace nového výrobku. Nerozšiřujeme tak funkčnost systému, ale jeho informační prostor (více o sekcích v kapitole 5.1.7). To však nikoli na úkor jednoduchosti a srozumitelnosti systému.

2.3 Snadné sdílení obsahu

V praxi se mnohokrát setkáme s častou duplicitou dat. Organizace má například vlastní systém pro správu překladů, systém pro správu faktur, každá součást organizace má vlastní webové prezentace ty mají interní sekce apod. Zde všude je potřeba adresy, telefonního čísla na určitou osobu, čísla účtů apod. Již v čase vzniku takovéhoto systému je mnohonásobná redundance dat. Problémy však nastanou např. při změně účtu v jedné takové části. Ve zbylých částech se již změny nepromítnou.

Cílem tohoto nově vznikajícího systému bude pokusit se **odstranit** takovouto **redundanci dat** a poskytnou všem částem systému **jednoduchý a sjednocený přístup k datům**, které lze sdílet vícero podsystémy (více o přístupu k datům v kapitole 6.8.1).

2.4 Snadná udržovatelnost systému

Snadná udržovatelnost systému se zdá být jako přirozený požadavek informačních systému, ne vždy je však tato podmínka splněna. Mnohdy se setkáme s aplikacemi s příliš složitě strukturovaným, či naopak nestrukturovaným kódem, či se slabým administračním rozhraním.

Snadnou udržovatelností v kontextu tohoto systému je myšleno **udržovatelnost na úrovni kódu** i snadná udržovatelnost na úrovni aplikace systému (administrativní rozhraní).

2.5 Nezávislost na použitých technologiích

Navrhnout systém, který by nebyl závislý na použitých technologiích jednotlivých částí takovéhoto systému, přináší mnoho výhod. Rozhodneme-li se po čase pro novou technologii např. ukládání dat, či zobrazování výstupu, neměla by se změna těchto části vůbec, či jen minimálně dotknout částí ostatních.

Takovéhoto požadavku se dnes již běžně dosahuje na úrovní datové vrstvy (pojem vysvětlen v kapitole 3.4.4.3), větší problém je však v zobrazovací vrstvě (pojem vysvětlen v kapitole 3.4.4.1). Tato vrstva bývá většinou pevně spjata s internetovým prohlížečem a technologiemi (X)HTML a

CSS. Zobrazovací vrstva se tak většinou stává součástí vrstvy aplikační (pojem vysvětlen v kapitole 3.4.4.2).

Navrhovaný informační systém se pokusí dosáhnout nezávislosti na všech třech úrovních, včetně **nezávislosti zobrazovací vrstvy na použité technologii a na webovém prohlížeči**.

2.6 Komfort programování a obsluhy systému

Jeden z důležitých cílů je usnadnit co nejvíce práci programátorovi a hlavně uživateli.

Na úrovni programování to pak znamená **vytvoření hotových kusů kódu**, které ulehčí programátorovi vytváření nových části. Programování by se pak dalo přirovnat např. ke stavbě budovy s předem hotových dílů. Základní kusy kódu dostane k dispozici, detaily doladí ručně.

Na úrovní uživatele to je snaha o **jasnou a přehlednou administraci**, dostatečný popis jednotlivých položek a termínů tak, aby se administrátor v systému vyznal i bez detailní znalosti struktury systému.

2.7 Systém nezávislý na prostředí a aplikaci

Jedním z cílů je vytvořit **systém nezávislý na jeho aplikaci**, čili zhotovit systém, který není vázaný na jedno konkrétní použití, ale systém, který by bez nutnosti změny kódu bylo možno použit i v jiné firmě, či podmínkách. Což znamená navrhnout systém dostatečně obecně, ale při zachování přehlednosti a udržovatelnosti.

2.8 Důraz na bezpečnost

S rostoucím rozvojem internetu a zvětšujícím se objemem sdílených, mnohdy důvěrných, dat po internetu, je důležité vynaložit úsilí na **zabezpečení a ochranu dat klienta**. Je nutné se pokusit o vytvoření co nejbezpečnějšího systému a to i z pohledu možnosti cíleného útoku na tento systém, což znamená vyvíjet systém se stálým vědomím možných bezpečnostních rizik.

2.9 Novější technologie v praxi

Jedním s cílů této práce je vyzkoušení a **ověření funkčnosti a spolupráce moderních technologií jako** třívrstvá architektura (viz kapitola 3.4.4), architektonické vzory (viz kapitola 3.5), webové služby (viz kapitola 3.6), objektový přístup v PHP (viz kapitola 3.10.1), atd. v praxi.

3 Teoretická část

Tato kapitola se zabývá teoretickým pozadím jednotlivých aspektů vyvíjeného systému. Nejprve bude pozornost zaměřena na teorii informačních systémů (kapitola 3.1), životní cyklus informačních systémů (kapitola 3.2), bezpečnost informačních systému (kapitola 3.3), dále na architektury informačních systémů (kapitola 3.4), architektonické vzory (kapitola 3.5), webové služby (kapitola 3.6), dále na protokol SOAP (kapitola 3.7), WSDL soubory (kapitola 3.8) a nakonec na jazyky UML, PHP, MySQL, XHTML a CSS (kapitoly 3.9, 3.10, 3.11, 3.12, 3.13).

3.1 Informační systém

V dnešní době stále více sílí vliv informačních technologií na lidskou činnost. Základním prvkem využívání informačních technologií jsou softwarové produkty zvané informační systémy (IS). IS se tak často stávají nástrojem změny organizace práce a významným prostředkem ovlivňujícím chod organizací a podniků.

V následujících podkapitolách budou nejprve popsány obecné systémy (kapitola 3.1.1) a poté podrobněji informační systémy (kapitola 3.1.2). Informace k této kapitole jsou čerpány z literatury [2], [4], [37].

3.1.1 Obecné systémy

Informační systém je speciální typ systému**. Systém** lze obecně chápat jako množinu prvků a vazeb mezi nimi, které jsou účelově definovány na nějakém objektu, splňuji nějaký cíl.

3.1.1.1 Hranice systému

Hranice systému vymezuje samotný systém nebo odděluje více systémů. Logická hranice je pomyslnou hranicí a vymezuje podsystémy v rámci systému, ovšem okolí systému je již „viditelnou“ hranicí. Prvky vně hranice pak ovlivňují chování systému.

3.1.1.2 Rozdělení systému

Systémy dělíme (dle [4]) na:

- **Uzavřené** × **otevřené** - podle toho, zda nastává interakce s okolím,
- **deterministické** × **stochastické** - tzn. jednoznačné nebo statistické chování,
- **statické** × **dynamické** - tzn. lineární nebo diferenciální (systém si pamatuje vnitřní stav),
- **spojité** × **diskrétní** - podle časových událostí (případně také kombinované).

3.1.2 Informační systémy

Informační systém je **konceptuální** (modeluje fyzický, reálný systém a užívá konceptuální zdroje, jako jsou informace a data), **otevřený** (provázán s okolím tokem dat) systém, sloužící ke sběru, zpracovávání, uchovávání, analýze a prezentaci dat a zabezpečující komunikaci mezi těmito daty.

3.1.2.1 Univerzální a specifické informační systémy

Vzhledem ke specifičnosti požadavků kladených na každý IS nelze vytvořit dostatečně univerzální systém aplikovatelný ve všech podmínkách a případech. Existují i velmi obecné systémy, které je však před instalací nutné přizpůsobit požadavkům zákazníka.

Výhodou takového systému je záruka, že systém bude správně pracovat, rychlejší nasazení a nízké náklady na údržbu. Nevýhodou je, že systém nemusí zcela přesně odpovídat představám zákazníka a riziko, že je založen na zastaralých technologiích.

Oproti tomu u systémů, které jsou vytvářeny již od začátku podle potřeb zákazníka, je větší pravděpodobnost, že budou obsahovat pouze vyžadované funkce. Tento způsob však znamená výrazně delší dobu vývoje, větší riziko neúspěchu nebo dokonce nedokončení, vyšší nároky na kvalitu vývojářů a celkově vyšší náklady (zejména v etapách kódování, testování a provozu). Na rozdíl od univerzálního systému však může přinést zákazníkovi významnou konkurenční výhodu.

3.1.2.2 Rozdělení informačních systému

Z hlediska účelu rozdělujeme IS (dle [4]) na systémy určené pro řízení každodenních operací (**operativní IS**), systémy pro podporu rozhodování (**manažerské IS**), **kancelářské IS, expertní IS**, **taktické IS** a **IS vedení**.

Operativní IS (OIS) pracují s nepříliš rozsáhlými daty a ovlivňují spíše vnitřní chování organizace či podniku, v němž jsou nasazeny.

Manažerské IS (MIS) zpracovávají množství dat a vytváří tak podmínky pro kvalitní a efektivní práci managementu. Spojením operativních a manažerských IS vzniká **exekutivní IS**.

3.1.2.3 Modularita informačního systému

Pro snadnější údržbu, vývoj a implementaci je dobré rozdělit systém na několik funkčně nezávislých modulů, které mezi sebou komunikují. Modul v sobě zapouzdřuje datové třídy typy a proměnné.

K snadnější implementaci modulů lze využít tzv. vzory, které poskytují jakýsi návod, či popis efektivního a účinného propojení subsystémů a jejich vztahů.

Moduly lze snadno použít i pro jiný systém, či nahradit jinými moduly se stejným rozhraním ale odlišnou implementací.

3.2 Životní cyklus informačních systému

Životní cyklus je obdobím, které začíná prvotní představou o programu a končí okamžikem, kdy se program přestane používat. Proces vývoje softwarového vybavení definuje při vývoji softwaru otázky *kdo, co, kdy a jak.*

V následujících podkapitolách se budeme nejprve zabývat unifikovaným vývojem software (kapitola 3.2.1), dále si popíšeme jednotlivé fáze vývoje (kapitoly 3.2.2, 3.2.3, 3.2.4, 3.2.5) a na závěr detailněji rozebereme jednu z nejdůležitějších etap životního cyklu software – analýzu požadavků systému (kapitola 3.2.6).

Informace k této kapitole byly čerpány z literatury [2], [37].

3.2.1 Unifikovaný vývoj software

Metodika unifikovaného vývoje software (zkráceně UP) je průmyslovým standardem procesu tvorby softwarového vybavení pocházejícím od autorů jazyka UML. Více o jazyku UML v kapitole 3.9.

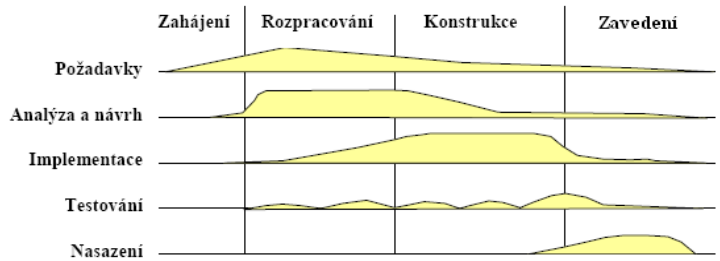
UP popisuje (dle [2]) několik základních pracovních postupů, jež určují, co je třeba udělat a způsob jakým toho dosáhnout:

- **Požadavky**. Zachycují to, co by měl systém dělat.
- **Analýza.** Vybroušení požadavků a jejich strukturování.
- **Návrh.** Realizace požadavků v architektuře systému.
- **Implementace.** Tvorba systému.
- **Testování.** Ověření, zda implementace funguje, jak se od ní očekává.
- **Nasazení.** Nasazení systému do ostrého provozu.

UP dále definuje čtyři po sobě následující fáze životního cyklu systému, z nichž každá končí hlavním milníkem.

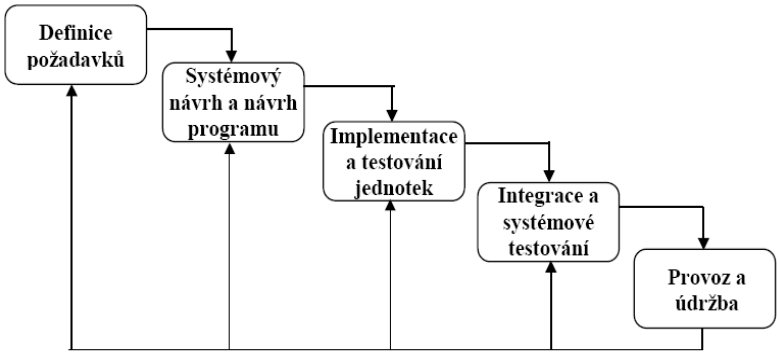
- **Zahájení.** Období plánování.
- **Rozpracování.** Období architektury.
- **Konstrukce.** Počátky provozuschopnosti.
- **Zavedení.** Nasazení produktu do uživatelského prostředí.

Obrázek 3.1 názorně zobrazuje rozložení objemu prací vykonaných na každém ze základních pracovních postupů.



Obrázek 3.1. Rozložení objemu vykonané práce na fáze životního cyklu IS. Převzato z [2].

Nejznámější model návrhu programového díla je **model životního cyklu vodopád** (viz Obrázek 3.2). Ten představuje posloupnost: úplná specifikace požadavků, celkový a podrobný návrh, implementace, testování a uvedení do provozu. Výhodou je, že pokrok projektu je stále viditelný. Nevýhodou metody je nutnost návratu k předchozím etapám vývoje při odhalení každé chyby. Negativní je i skutečnost, že zákazníkovi je možné systém předvést až po jeho dokončení, což téměř vylučuje zpracování změny některých požadavků.

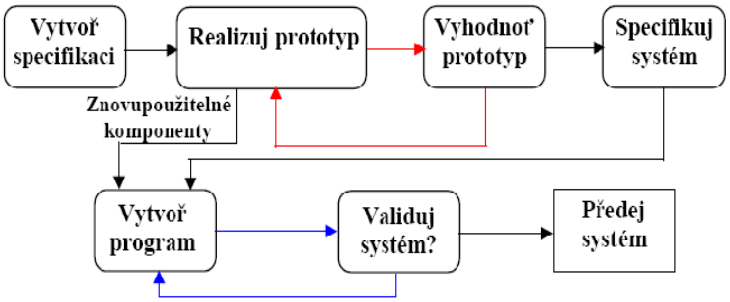


Obrázek 3.2. Model životního cyklu vodopád. Převzato z [37].

Z modelu vodopád vychází **spirálový model**. Spirálový model je založen na kombinaci prototypování a analýzy rizik. Jednotlivé kroky se při vývoji systému ve spirále opakují, ale pokud možno na vyšším stupni zvládnuté problematiky. Ten obsahuje navíc prvky plánování, postupné zpřesňování požadavků, hodnocení rizik, atd.

Používaný je i **iterační** a **inkrementální model** (dle [2]). Stejně jako u předchozích modelů, jsou oba realizovány v cyklech. V případě iteračního modelu se v každém cyklu systém zpřesňuje a zvětšuje o další část. Tento postup vede ke snáze modifikovatelnému a rozšiřitelnému systému s možností integrace produktů třetích stran. Realizace však trvá déle. Inkrementální model znamená postupné přidávání samostatné části v každém cyklu.

Jednou z dalších technik odstraňujících nedostatky modelu vodopádu je **tvorba softwarových prototypů** (Obrázek 3.3), [2], [37]. Jde však jen o částečně funkční systémy. Omezeno je množství funkcí nebo jde jen o uživatelské rozhraní imitující činnost, popřípadě je systém implementován na jiném hardwaru či v jiném jazyce než je požadováno. Účelem prototypu je pouhé zpřesnění požadavků. Nebezpečím je dopracování prototypu.



Obrázek 3.3. Metoda softwarových prototypů. Převzato z [37].

3.2.2 Fáze zahájení

Cílem fáze zahájení je odstartování projektu. Hlavní důraz je kladen na specifikaci požadavků a jejich analýzou. Do této fáze však mohou spadat i určité návrhářské práce. Výstupem fáze zahájení jsou:

- Popis zainteresovaných osob,
- slovník projektu.
- vyčíslení nákladů,
- funkční a nefunkční požadavky,
- náčrt projektu.

3.2.3 Fáze rozpracování

Cílem fáze rozpracování je vytvořit první pokus o systém, tvorba spustitelného architektonického základu, ne jen prototypu. Vzhledem k tomu, že další fáze vycházejí z výsledku fáze rozpracování, lze tuto fázi považovat za kritickou. Výstupy fáze rozpracování:

- Návrh případu užití (Use Case diagram),
- návrh struktur a tříd (Diagram tříd),
- návrh ostatních diagramů, jsou-li potřeba,
- tvorba spustitelného architektonického základu,
- testování architektonického základu.

3.2.4 Fáze konstrukce

Cílem fáze konstrukce je splnit všechny požadavky analýzy a návrhu a vyvinout ze spustitelného základu vytvořeného ve fázi rozpracování konečnou verzi systému. Klíčovým zadáním konstrukční fáze je zachování integrity architektury vytvářeného systému. Výstupy fáze konstrukce:

- Odhalit požadavky, které byly přehlédnuty,
- dokončit analytický model,
- implementovat počáteční funkční variantu,
- testovat počáteční funkční variantu.

3.2.5 Fáze zavedení

Fáze zavedení začíná v okamžiku, kdy je dokončeno testování a konečné nasazení systému. Cílem fáze je zavedení systému do ostrého provozu. Výstupy fáze:

- Dokončeny beta-testy na pracovišti (v systému zákazníka),
- uživatele produkt aktivně využívají,
- plán uživatelské podpory pro produkt,
- tvorba manuálu a další dokumentace,
- konzultace s uživateli,
- koncová revize.

3.2.6 Stanovení cílů a specifikace požadavků

Stanovení cílů projektu patří k nejvýznamnějším úkolům počátku vývoje softwaru a jeho nezvládnutí může mít velký podíl na případném neúspěchu. Je nutné omezit cíle jen na opravdu nutné a veškeré nepodstatné a zbytečné požadavky vypustit. Zvážit by se měla i složitost úkolu a míra organizačních změn vzniklých zavedením IS. Vzhledem k životnosti IS bývá tradičním cílem snadná modifikovatelnost a otevřenost systému.

Existuje několik studií, které dokazují, že neúspěch v procesu inženýrství požadavků je hlavní příčinou konečného neúspěchu celého projektu [2]. Inženýrství požadavků představuje úzkou a intenzivní spolupráci s uživatelem, který o výsledném systému nemusí mít zcela jasné představy, popřípadě jsou jeho představy nesprávné. Cílem specifikace požadavků je získat přesné informace o tom, co má systém dělat (stanovení poskytovaných služeb a omezení, s nimiž se musí počítat) a formulovat toto v jazyku zákazníka.

Požadavek lze definovat jako „specifikaci toho, co má být implementováno“. Rozlišují se dva druhy požadavků:

- **Funkční požadavky**, jež určují, jaké chování bude systém nabízet a
- **nefunkční požadavky**, které určují vlastnosti a omezení kladená na systém a jeho vývoj.

Požadavky by měly být základem všech systémů. Úspěšným nástrojem k zjištění a vyjádření funkčních požadavků se staly nástroje jazyka UML. O jazyku UML je psáno podrobněji v kapitole 3.9.

3.3 Bezpečnost informačních systému

Informační systémy nakládají mnohdy s velmi citlivými osobními údaji. Je kladen velký důraz na bezpečnost těchto dat. Je snaha zabránit potenciálním útočníkům k získání těchto citlivých informací.

Bezpečnost dat lze zajistit na více úrovních. Dobře navržený systém by měl kontrolovat a ověřovat data zadaná uživatelem a zamezit tak nechtěným, čí chtěným chybám při práci s daty.

Kromě ošetření na úrovní aplikační vrstvy je potřeba zajistit bezpečnost dat přenášených mezi účastníky komunikace (klient–server, server–server). Tuto bezpečnost lze zajistit opět mnoha způsoby, např. implementace vlastního rozhraní a protokol pro bezpečnou komunikaci nebo přidání bezpečnostních vrstev k systému. Pokud vyvíjený informační systém využívá transportního protokolu TCP/IP, byla pro účel zabezpečení komunikace mezi jednotlivými účastníky navržena bezpečnostní vrstva mezi transportní vrstvou TCP/IP a vrstvou aplikační – SSL.

Tato kapitola čerpá z [22], [24].

3.3.1 SSL (Secure socket layer)

SSL (SECURE SOCKET LAYER) vyvinula v roce 1996 firma Netscape jako nekomerční otevřený protokol. Používat ho může kdokoli pro soukromé i komerční účely.

SSL je vrstva/protokol zabezpečující data na přechodu mezi aplikační (např. HTTP) a transportní vrstvou (např. protokolem TCP/IP). Lze zajistit šifrování přenášených dat a autentizaci serveru pomocí digitálních certifikátů. SSL není omezeno pouze na protokol HTTP. SSL je možno použít i pro bezpečné připojení prostřednictvím FTP, NNTP ale i k poštovním službám přes SMTP, POP3, IMAP4 a řadu dalších protokolů. Oproti klasickým protokolům se budou zabezpečené lišit jen o písmeno "s" (například FTPS).

Pro použití SSL je třeba mít na straně serveru nainstalovanou podporu SSL a také jej musí podporovat náš prohlížeč (nebo obecně klientská vrstva), což v současnosti podporují téměř všechny.

To, že jsme se pomocí webového prohlížeče připojili na webové stránky zabezpečené pomocí SSL, poznáme podle adresy (obsahuje navíc písmeno s, např. *https://server.cz*) nebo podle indikace prohlížeče. Obyčejně je zabezpečení přenosu indikováno ikonkou zamčeného zámku ve stavovém řádku okna prohlížeče (Obrázek 3.4). V závislosti na nastavení nás prohlížeč informuje o přechodu mezi zabezpečeným a nezabezpečeným režimem.

Obrázek 3.4. Indikace zabezpečeného přenosu.

3.4 Architektury informačních systémů

S postupným nasazováním informačních systémů ve velkém měřítku nabývá na důležitosti vhodné navržení architektury, která bude i při velkém počtu požadavků zajišťovat dostatečnou dostupnost služeb poskytovaných systémem. Při návrhu systému se klade důraz zejména na výběr takových

prostředků, které vedou k efektivní realizaci všech úrovní aplikace, její údržby a možné rozšiřitelnosti.

V následující kapitole budou vysvětleny některé architektury informačních systémů. Popíšeme centralizované systémy (kapitola 3.4.1), systémy klient-server (kapitola 3.4.2), dvouvrstvé a třívrstvé architektury IS (kapitoly 3.4.3, 3.4.4).

V této kapitole je použito pramenů z literatury [20], [21].

3.4.1 Centralizované systémy

Pro tyto systémy je typická silná a těsná vazba mezi aplikací a daty. Konkrétní aplikace, programové aplikační rozhraní databáze a data využívají stejné zdroje (procesor, paměť a disk), tj. běží na jednom počítači. Takový systém neumožňuje oddělit data od logiky a jakýkoli zásah do architektury systému (změna systému řízení dat, apod.), či jeho údržba je velice náročná.

Centralizovaný přístup lze použít u malých aplikací nevyžadujících větší komunikaci s daty, či uživatelem. S centralizovanými systémy se však v praxi setkáme i v prostředích, kde je tento koncept naprosto nevhodný.

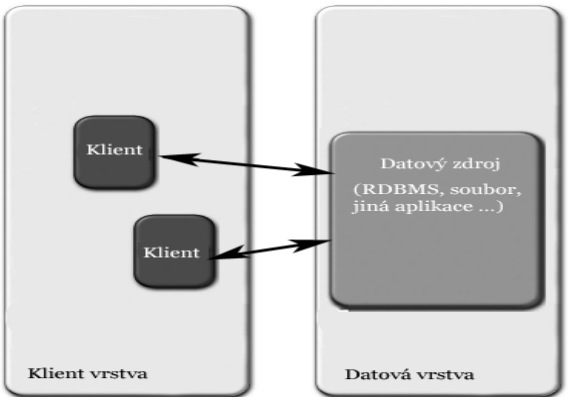
3.4.2 Klient/Server

Podstatně se liší oproti centralizovaným systémům. Aplikace a systém řízení báze dat (SŘBD) jsou odděleny (většinou i na oddělených počítačích) a pro připojení k SŘBD se využívá rozhraní či komunikačního protokolu (jazyka). V mnoha systémech tuto komunikaci zajišťuje jazyk SQL, který komunikuje s SQL Serverem (např. MSSQL, MySQL, Oracle, apod.). Klient vyšle SQL požadavek a server na tento požadavek odpoví.

Počítač, na kterém aplikace běží a poskytuje uživatelské rozhraní, se obvykle označuje jako **klient** (client). Počítač, na kterém běží SŘBD se nazývá **server**. Příklad grafického znázornění struktury klient/server viz Obrázek 3.5.

3.4.3 Dvouvrstvá architektura

Dvouvrstvá architektura (Two-tier architecture) představuje vlastně klasický model klient-server. Jednu vrstvu představuje klient, tj. ten, který požaduje nějaké služby a druhou vrstvu pak server, tj. ten, kdo tyto služby poskytuje. Uživatelské rozhraní je umístěno výlučně na straně klienta, správa databáze je umístěna na straně serveru (Obrázek 3.5).



Obrázek 3.5. Dvouvrstvá architektura systému. Převzato z [20].

Aplikační logiku lze rozprostřít mezi klienta i server (uložené procedury na straně serveru), ale nejčastěji většinu aplikační logiky má na starosti právě klientská vrstva. Server jednoho klienta může fungovat jako klient jiného serveru v hierarchické klient-server architektuře. Pak hovoříme o tzv. zřetězené dvouvrstvé architektuře. Použití dvouvrstvé architektury je dobrým řešením pro distribuované zpracování zejména pro menší sítě.

Výhody dvouvrstvé architektury oproti centralizované architektuře:

- Sdílení jednoho datového prostoru více klienty,
- snadnější údržba systému,
- větší bezpečnost dat,
- data zpracována na straně serveru,
- malý objem přenášených dat ke klientovi.

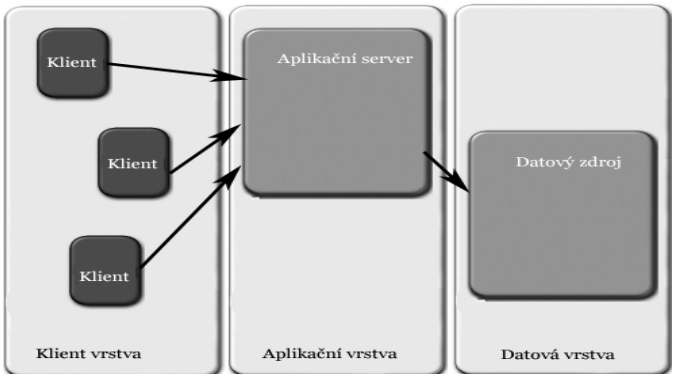
Nedostatky dvouvrstvé architektury:

- Problémy s výkonem při větším počtu klientů,
- stále nesnadná údržba,
- málo efektivní dělba práce,
- nelze sdílet data více aplikacemi.

3.4.4 Třívrstvá architektura

Při masivním rozšíření aplikací, byly definovány nové požadavky na aplikace, například nutnost sdílení zdrojů, omezení datového přenosu atd., díky kterým se začalo uvažovat o architektuře, která by tyto požadavky pokryla.

Řešení nedostatků s dvojvrstvou architekturou se našlo v podobě přidání třetí - střední vrstvy (middle tier). Třívrstvou architekturu znázorňuje Obrázek 3.6.



Obrázek 3.6. Třívrstvá architektura systému. Převzato z [20].

S přechodem na třívrstvou architekturu se změnil i význam jednotlivých vrstev, nejmarkantněji se to projevilo u vrstvy klientské. Díky nově definované aplikační vrstvě bylo možné aplikační logiku, která do té doby ležela na klientu, přesunout na aplikační server. Díky tomuto se klientům značně ulevilo, neboť veškerý výpočetní výkon byl přesunut na výkonné servery.

Zavedení střední vrstvy ulehčuje administraci a správu změn protože případné změny se zapisují pouze jednou na server střední vrstvy, a tak se stávají automaticky dostupné ve všech systémech. U jiných modelů by změna funkce musela být zapsána do každé aplikace. Podařilo se překonat a odstranit výkonnostní omezení týkající se dvouvrstvé architektury. Je zajištěn výkon i pro velké skupiny současně pracujících klientů.

Významnou měrou se podařilo redukovat datový přenos, jehož těžiště se přesunulo na trasu mezi aplikačním serverem a datovým zdrojem.

Výhody třívrstvé architektury oproti dvojvrstvé architektuře:

- Snadná správa změn,
- snadná administrace,
- využití jednoho zdroje dat pro více aplikací,
- redukce datového přenosu ke klientovi,
- největší datový přenos mezi aplikačním a datovým serverem (vrstvou),
- větší rychlost při více připojených klientech,
- vylepšená bezpečnost dat,
- lepší dělba práce,
- aplikační logika přesunuta na aplikační server.

3.4.4.1 Klientská vrstva

Klientská vrstva poskytuje prezentačních služby (příjem vstupu, zobrazování výsledků) a prezentační logiku (řízení interakce – hierarchie menu, obrazovek). Neobsahuje žádnou aplikační logiku.

Klientu z třívrstvé architektury, jehož účelem je poskytování prezentačních služeb a prezentační logiky se také říká **tenký klient** (thin client). Tenký klient je takový klient, který neobsahuje žádnou aplikační logiku. Aplikační logiku mu zprostředkuje aplikační server, ke kterému tenký klient přistupuje.

Tlustý klient, v dvouvrstvém modelu pouze klient, obsahuje většinu aplikační logiky. Ve skutečnosti i tenký klient musí nějakou aplikační logiku obsahovat, ale oproti tlustému klientu je to naprosto zanedbatelná část.

Webový prohlížeč spadá do skupiny tenkého klienta. Je dobré ale rozlišovat mezi prohlížečem a tenkým klientem. Prohlížeč postavený na standardech webu používá pro popis grafického uživatelského rozhraní jazyk HTML a CSS. Jazyk HTML v kombinaci s CSS nenabízí takové možnosti oproti tenkým klientům, aby bylo možné popsat složitější rozhraní. Další nevýhodou prohlížeče je svázání s modelem žádost/odpověď (request/response) nastaveným HTTP protokolem.

Tyto odlišnosti rozlišují mezi pojmy webový prohlížeč a tenký klient v kontextu třívrstvé architektury. Mezi častý omyl patří rozlišování tenkých a tlustých klientů podle toho jestli se jedná o webový prohlížeč. Klíčovým faktorem zda se jedná o tenkého nebo tlustého klienta je rozložení aplikační logiky.

3.4.4.2 Aplikační vrstva

Aplikační vrstva obsahuje aplikační logiku, tj. operace realizující algoritmus aplikace a zajišťuje přístup k datům, práci s daty a z jejich vystavení ve vhodném formátu pro klientskou vrstvu.

Aplikační vrstva zaznamenala nejbouřlivější vývoj. Jako nejjednodušší realizaci aplikační vrstvy můžeme považovat například webový server, jehož pomocí se volají výkonné rutiny prezentované spustitelnými soubory a serverovými skripty.

3.4.4.3 Datová vrstva

Tato vrstva slouží jako datová základna implementovaná např. pomocí relační databáze, souborovým systémem, webovou službou či jinou aplikací. Datová vrstva má na starosti logiku dat (podporu datových operací) a datové služby (manipulace s daty).

3.5 Architektonické vzory

Vzory popisují problém při návrhu a implementaci software. Pomáhají definovat, jakým způsobem mají jednotlivé podsystémy a části systému spolupracovat. Vzory nabízí řešení, které lze použít v různých kontextech.

V této kapitole bude nejprve představeno rozdělení vzorů do kategorií (kapitola 3.5.1), dále podrobněji vysvětleny vzory Model-View-Controller (kapitola 3.5.2), Layers pattern (kapitola 3.5.3),

Pipes and Filters(kapitola 3.5.4), Broker Pattern (kapitola 3.5.5) a Presentation-Abstraction-Control (kapitola 3.5.6).

Text a informace v této kapitole vychází z literatury [6], [11], [13], [14], [17], [19], [26], [28], [29], [35].

3.5.1 Rozdělení vzorů

Vzory lze rozdělit na:

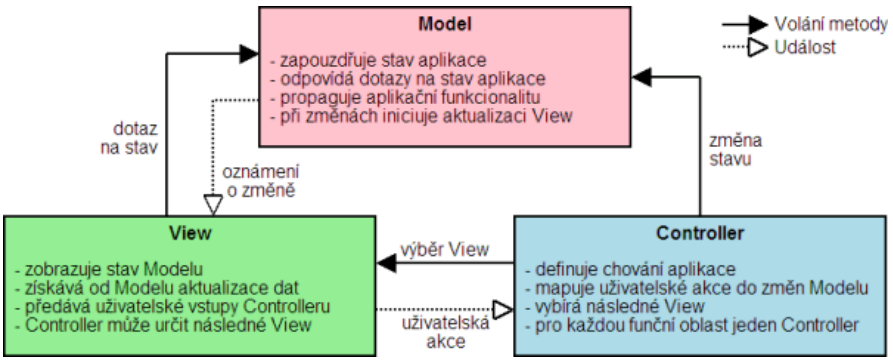
- **Analytické vzory.** Uplatní se při specifikování požadavků a vytváření analytického modelu.
- **Architektonické vzory.** Vyjadřují podstatné strukturální prvky softwarových systémů a množinu vztahů mezi nimi. Architektonické vzory můžeme považovat za jakousi obdobu architektonických stylů ve stavebnictví.
- **Návrhové vzory.** Návrhové vzory představují obecný popis řešených problémů. Zachycují možnosti řešení problémů, které se opakovaně objevují při návrzích informačních systémů. Může se jednat jak o část zdrojového kódu, nebo knihovnu s přeloženými částmi programu, tak například o přístup ke konfiguraci různých softwarových a hardwarových komponent.
- **Idiomy.** Vzor spojený s konkrétním programovacím jazykem. Popisuje, jak daný problém implementovat v konkrétním programovacím jazyku.

Dále nás budou zajímat jen *architektonické vzory*, které popisují strukturu a vztahy subsystémů.

3.5.2 Model-View-Controller

Model-view-controller (MVC) (někdy také označován jako Model-2) je architektonický vzor, který rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent tak, že modifikace některé z nich má minimální vliv na ostatní [13]. Tento vzor poprvé popsal Trygve Reenskaug v roce 1979 a poprvé byl použit v jazyce Smalltalk [26].

Vytváření aplikací s využitím architektury MVC vyžaduje vytvoření komponent Model, View (pohled) a Controller (řadič), viz Obrázek 3.7.



Obrázek 3.7. Schéma Model-View-Controller. Převzato z [29].

Komponenta pohled (View) náleží do prezentační vrstvy. Controller v závislosti na implementaci, do prezentační, či aplikační vrstvy. Model je jen jiný název pro aplikační vrstvu.

3.5.2.1 Model

Model je doménově specifická reprezentace informací, s nimiž aplikace pracuje. Je to softwarový obraz procesů reálného světa. Měl by být jako celek zapouzdřený a pro View nebo Controller nabízet přesně definované rozhraní.

3.5.2.2 View (pohled)

Převádí data reprezentovaná Modelem do podoby vhodné k interaktivní prezentaci uživateli. Přes model přistupuje k datům a stavům aplikace a specifikuje, jak mají být prezentovány. Při změně stavu Modelu aktualizuje zobrazení. U webových aplikací generuje View příslušný HTML kód, který je odeslán prohlížeči jako odpověď na požadavek. Jestliže slouží zobrazený výstup zároveň jako oblast pro zachytávání událostí od uživatele (například kliknutí myší do vykresleného okna), předává View tyto události Controlleru.

3.5.2.3 Controller (řadič)

Controller reaguje na události (typicky pocházející od uživatele) a zajišťuje změny v Modelu nebo v Pohledu. Zpracovává veškeré vstupy a události pocházející od uživatele. Na jejich základě volá příslušné procesy Modelu, mění jeho stav apod. Podle událostí přijatých od uživatele i podle výsledků akcí v modelu pak vybírá Controller vhodné View pro další zobrazení. Pro webové aplikace jsou hlavním vstupem HTTP GET nebo POST požadavky odeslané uživatelovým prohlížečem.

3.5.2.4 Princip vzoru

Ačkoliv může být koncept MVC realizován různým způsobem, obecně platí tento princip:

1. Uživatel provede nějakou akci v uživatelském rozhraní (např. stiskne tlačítko).
2. Řadič obdrží oznámení o této akci z objektu uživatelského rozhraní.
3. Řadič přistoupí k modelu a v případě potřeby ho zaktualizuje na základě provedené uživatelské akce (např. zaktualizuje nákupní košík uživatele).
4. Model je pouze jiný název pro aplikační vrstvu. Aplikační logika zpracuje změněná data (např. přepočítá celkovou cenu, daně a expediční poplatky pro položky v košíku, atd.). Některé aplikace užívají mechanizmus pro perzistentní uložení dat (např. databázi). To je však otázka vztahu mezi doménovou a datovou vrstvou, která není architekturou MVC pokryta.
5. Komponenta pohled použije zaktualizovaný model pro zobrazení zaktualizovaných dat uživateli (např. vypíše obsah košíku). Komponenta pohled získává data přímo z modelu, zatímco model nepotřebuje žádné informace o komponentě View (je na ní nezávislý). Nicméně je možné použít návrhový vzor pozorovatel, umožňující modelu informovat

jakoukoliv komponentu o případných změnách dat. V tom případě se komponenta view zaregistruje u modelu jako příjemce těchto informací. Je důležité podotknout, že řadič nepředává doménové objekty (Model) komponentě pohledu, nicméně jí může poslat příkaz, aby svůj obsah podle modelu zaktualizovala.

6. Uživatelské rozhraní čeká na další akci uživatele, která celý cyklus zahájí znovu.

3.5.2.5 Výhody vzoru

- Snadné zpřístupnění pro různé druhy klientů. Pro zavedení podpory dalšího klienta je nutné nadefinovat pouze nové View, v případě zcela rozdílných vstupních událostí i speciální Controller. Nicméně Model jako klíčová část aplikace zůstává stále stejný.
- Minimalizace duplicitního kódu. Souvisí částečně s předchozím bodem. Například bez oddělení a zapouzdření Modelu by se pro každé nové view musela znovu programovat celá aplikační logika. Ale i v rámci jednoho typu klienta je často jedna akce volána z několika různých míst aplikace.
- Rozdělení vývojářských rolí. Jednotlivé části mohou být vyvíjeny samostatně, jen se znalostí předem určených rozhraní mezi nimi. Minimalizuje se dopad modifikací, změny jsou většinou prováděny jen v dané vrstvě.
- Znovupoužitelnost kódu. Jako Model lze ve vlastní aplikaci použít standardní knihovny či třídy. Existují také univerzálně pojaté controllery.
- Vysoká komplexnost návrhu a snadná budoucí rozšiřitelnost aplikace, efektivní modularita.
- V případě centralizovaného Front Controlleru, který zpracovává všechny přicházející požadavky, výhody vyplývající z centralizace, například možnost jednotné kontroly přístupových práv, logování apod.
- A v neposlední řadě i čistota designu, způsob přemýšlení programátora nad aplikací, jejím návrhem a architekturou.

V rámci MVC se dále rozlišuje mnoho strategií, jak výslednou aplikaci koncipovat. Liší se například počtem controllerů a rozdělením úkolů mezi nimi, použitím či nepoužitím šablon v rámci View, jednotným zapouzdřeným voláním datových zdrojů [39], apod.

3.5.2.6 Model-View-Controller v prostředí internetu

Existuje několik způsobů jak MVC v prostředí webu implementovat. V dnešní době jsou dva hlavní přístupy:

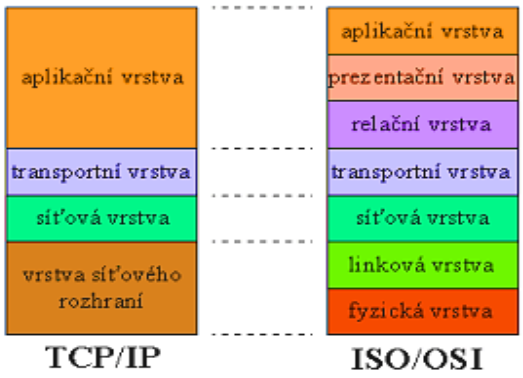
- **Front Controller**. Aplikační logika se volá před předáním zpracování do view.
- **Dispatcher view**. Aplikační logika se volá zprostředkovaně uvnitř view.

Klasickými představiteli implementace Front Controller jsou frameworky *Struts 1,2* [1], *Spring MVC* v prostředí JAVA [35]. Naopak na vzoru *Dispatcher view* jsou postaveny všechny implementace standardu *JSF* [28].

3.5.3 Vzor layers (vrstvy)

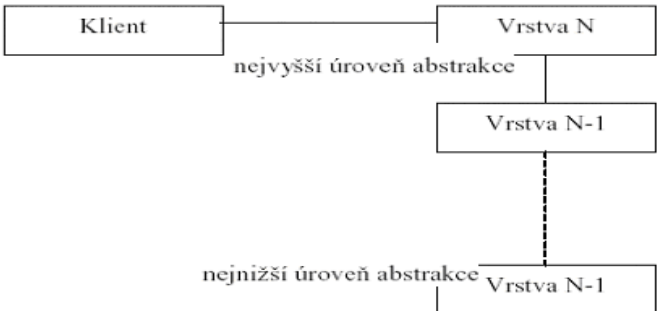
Vzor Vrstvy (Layers) dle [6] pomáhá strukturovat aplikace, které mohou být dekomponovány na skupiny podúloh, ve kterých každá skupina podúloh je v dané úrovni abstrakce.

Nejznámějším příkladem architektury vrstev jsou síťové protokoly. Protokol se skládá z množiny pravidel a konvencí, které popisují, jak počítačové programy komunikují přes hranice počítačů. Je definován formát, obsah a význam zpráv. Protokol specifikuje dohody v množství abstraktních úrovní, začínající od přenosu bitů až po nejvyšší úroveň aplikační logiky viz Obrázek 3.8.



Obrázek 3.8. Příklad použití vzoru vrstvy na síťovém protokolu TCP/IP. Převzato z [6].

Architekturu vrstev lze použít i při tvorbě informačních systémů. Systém je strukturován na jednotlivé vrstvy, tak aby změnou kódu jedné vrstvy nebyl ovlivňován celý systém, ale jen jedna vrstva. Hlavní zásadou tohoto vzoru je, že služby Vrstvy J jsou pouze použity Vrstvou J+1. Neexistuje žádná další závislost mezi vrstvami. Tato struktura může být srovnatelná se zásobníkem. Každá konkrétní vrstva chrání nižší vrstvy od přímého přístupu některé z vyšších vrstev (viz Obrázek 3.9).



Obrázek 3.9. Rozdělení systému na vrstvy. Převzato z [6].

Výhody vzoru:

- Znovupoužitelnou vrstev, dá se využít v případě dobře definované abstrakce a dobře definovaného a dokumentovaného rozhraní.
- Podpora standardizace. Jasně definované a všeobecně akceptované úrovně abstrakce umožňují rozvoj standardizovaných úloh a rozhraní.
- Závislosti jsou lokální. Standardizované rozhraní mezi vrstvami obyčejně omezuje změnu kódu pouze na danou vrstvu.
- Zaměnitelnost. Daná implementovaná vrstva může být zaměněna sémanticky ekvivalentní implementací bez velkého úsilí.

Nevýhody:

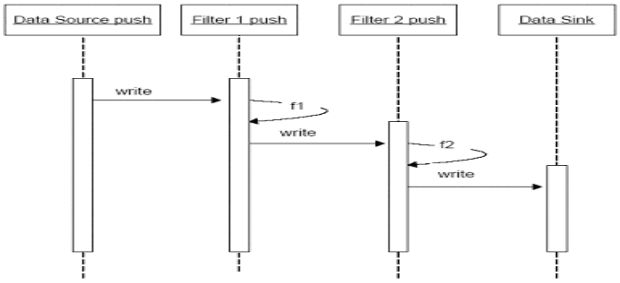
- Kaskádní změny chování. Může nastat při změně chování dané vrstvy.
- Nízká efektivnost. Architektura založená na vrstvách je obyčejně méně efektivní než monolitická.
- Problémy se stanovením správné granularity vrstev.

3.5.4 Vzor Pipes and Filters

Architektonický vzor Pipes and filters (roury a filtry) dle [7] poskytuje strukturu pro systémy, které zpracovávají tok dat. Každý krok zpracování je zapouzdřen v komponentě nazývané filtr. Data jsou posílána prostřednictvím rour (pipes) mezi sousedními filtry. Kombinace filtrů a rour dovoluje vytvořit řadu podobných systémů.

Vzor dělí řešenou úlohu do několika sekvenčních, procesních kroků. Tyto kroky jsou spojeny pomocí datových toků (rour) do systému, kde výstupní data daného procesního kroku jsou vstupem do následujícího procesního kroku. Každý procesní krok je implementován jako komponenta filtr. Filtr zpracovává a dodává data inkrementálně – nejdříve spotřebuje celý vstup dat, než vytvoří jakýkoli výstup. Každá roura implementuje datový tok mezi přilehlými procesními kroky (viz Obrázek 3.10).

Nejznámějším případem tohoto vzoru jsou programy jádra Unixu. Procesy Unixu (filtry) jsou propojeny run-time mechanizmem. Jiným příkladem je většina překladačů, z nichž řada pracuje inkrementálně.



Obrázek 3.10. Sekvenční diagram vzoru roury a filtry. Převzato z [7].

Výhody:

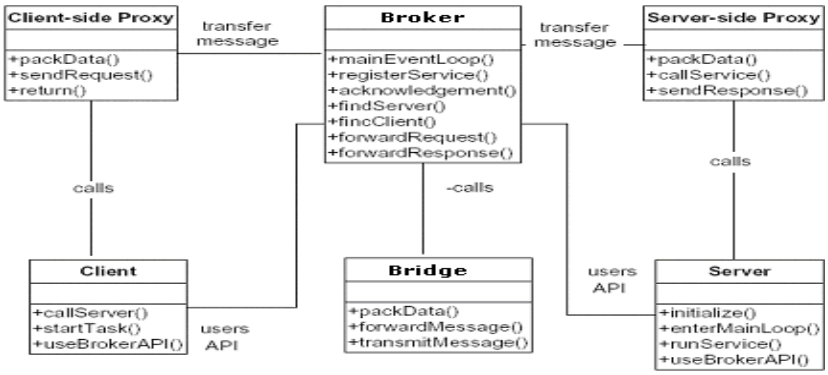
- Nejsou třeba žádné „mezi soubory“, ale jsou možné.
- Flexibilita je umožněna výměnou filtrů.
- Flexibilita rekombinaci. To umožňuje vytvořit novou zpracovávající sekvenci změnou filtrů, nebo přidáním dalších filtrů.
- Opakované využití komponenty filtru.
- Rychlé prototypování.

Nevýhody:

- Sdílení stavových (globálních) informací je nákladné a nepružné.
- Poněkud složité zpracování chyb.

3.5.5 Vzor Broker

Architektonický vzor Broker může být (dle [11]) použit na strukturování distribuovaných programových systémů s oddělenými komponentami, které jsou v interakci prostřednictvím vzdáleného volání služeb (remote service invocations). Komponenta broker je odpovědna za koordinaci komunikace, jako je odesílání požadavků, rozesílání výsledků a výjimek. Celý vzor broker se sestává z šesti typů komponent (clients, servers, brokers, bridges, client-side proxies, server-side proxies) viz Obrázek 3.11.



Obrázek 3.11. Struktura vzoru broker. Převzato z [11].

Výhody:

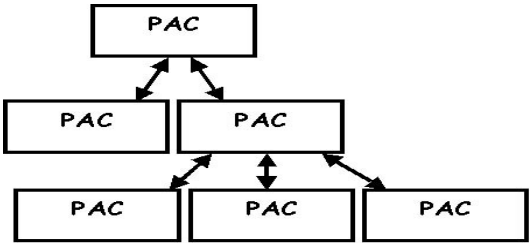
- Transparentnost umístění je umožněna tím, že komponenta broker je zodpovědná za vyhledání serveru podle jedinečného identifikátoru.
- Zaměnitelnost a rozšiřitelnost komponent.
- Portabilita systému s komponentou broker.
- Interoperabilita mezi různými systémy s komponentou broker.
- Znovupoužitelnou.

Nevýhody:

- Omezená výkonnost.
- Nízká odolnost vůči chybám.
- Testování a ladění.

3.5.6 Vzor Presentation-Abstraction-Control

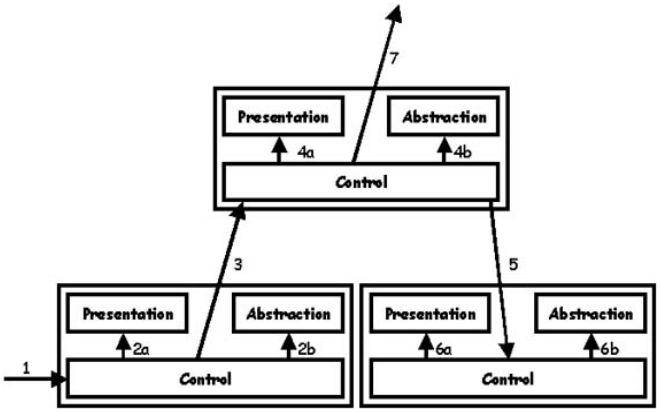
Presentation-Abstraction-Control (PAC) je (dle [3]) podobný vzoru Model-View-Controller (MVC). MVC je ale omezen na jednoduché uživatelské rozhraní (GUI) s jedním nebo více pohledy na jeden a tentýž model. Pokud máme systém složen z více podsystémů, které potřebují své vlastní specifické uživatelské rozhraní, nebo potřebujeme více komplexnější GUI pro systém, můžeme použít architekturu PAC. PAC se skládá z hierarchické struktury PAC komponent (Obrázek 3.12).



Obrázek 3.12. Schéma architektury Presentation-Abstraction-Control. Převzato z [3].

Každá PAC komponenta se stává ze tří prvků, které navzájem komunikují obdobně jako v MVC architektuře (Obrázek 3.13):

- **Presentation.** Prvek Presentation je totožný z prvkem View z MVC. Zobrazuje informace z Abstraction.
- **Abstraction.** Obsahuje data jako v MVC. Nicméně může být jen částí kompletní struktury dat aplikace. Nehraje aktivní roli v oznamování změn.
- **Control.** Control je totožný s prvkem controller z MVC. Zpracovává vnější události a aktualizuje model. Přímo tedy upravuje prvek Presentation. Rozdíl oproti controlleru z MVC a prvku control je v tom, že změny provedené prvkem control přejdou na jeho rodičovskou PAC komponentu.



Obrázek 3.13. Spolupráce komponent u architektury Presentation-Abstraction-Control. Převzato z [3].

3.6 Webové služby

Webová služba je souhrnné označení pro sadu technologií umožňujících komunikaci mezi aplikacemi na internetu. Je to jakási komponenta nabízející určitou službu vzdáleným modulům, či klientům. Takovou službou může být např. převod měn, zjištění kurzu akcie, zpracování objednávky, překlad textu, nebo data z aplikační logiky zaslaná prezentační vrstvě (pohledu ve VMC vzoru).

Webové služby využívají toho, že jazyk HTML byl od samého počátku vývoje nezávislý na používané platformě. Proto byl vybrán jako nejvhodnější základ pro univerzální řešení, které by umožnilo naplnit filozofii webových služeb, tj. propojit běžně nepropojitelné aplikace.

Webové služby umožňují uživatelům přistupovat k potřebným informacím, a to kdykoliv a kdekoliv - a navíc z libovolného zařízení: nejen z běžného osobního počítače či notebooku, ale také z kapesního počítače anebo mobilního telefonu. A to nejen z internetového prohlížeče, ale obecně z jakéhokoli tomu určeného klienta. Webové služby proto musí fungovat jako dokonale univerzální rozhraní mezi zdrojem dat a uživatelem.

V následujících podkapitolách budou zmíněny webové služby v informačních systémech (kapitola 3.6.1) a princip webových služeb (kapitola 3.6.2).

Informace k této kapitole byly čerpány z [38], [8], [9].

3.6.1 Webové služby v informačních systémech

Při tvorbě informačních systémů a rozsáhlých webových aplikací se čím dál častěji setkáváme s požadavky na oddělení zobrazovaných dat od logického zpracování (třívrstvá architektura, architektonický vzor Model-View-Controller). Tento požadavek však nemusí znamenat jen přenesení aplikační logiky na jiný počítač, než klientský, ale obecně potřebu přistupovat k jednomu modelu více klienty (např. využívat jednu službu systému dalšími systémy, či použití více druhů klientů). Při tomto požadavku je nutné, aby si jednotlivé části systému (provozované často na jiných počítačích) vyměňovaly data a zasílaly odpovědi na určité dotazy a to nezávisle na platformě či použitém jazyku.

Dříve bylo také velice obtížné integrovat různé aplikace firmy do jednoho informačního systému, které jsou často naprogramovány v odlišných programovacích jazycích a navíc jsou provozovány na nejrůznějších platformách. Jejich vzájemné propojení proto nefunguje vždy zcela spolehlivě.

Zmíněné požadavky a problémy lze zjednodušeně formulovat do požadavku integrovat libovolné aplikace, nebo různé moduly/části téhož systému, provozované na různých platformách a ovládat je prostřednictvím jednotného rozhraní. Tento požadavek splňují právě webové služby.

3.6.2 Princip webových služeb

Webové služby jsou založeny na vzájemné interakci tří základních částí:

- Poskytovateli služeb,
- registru služeb a
- klienta.

3.6.2.1 Poskytovatel webových služeb

Service Provider, tj. subjekt, který službu poskytuje. Jde o softwarovou či hardwarovou platformu, která zajišťuje provoz vlastních webových služeb. V kontextu informačních systému může jít o aplikační vrstvu, která poskytuje služby klientské vrstvě, či o datovou vrstvu, která poskytuje své služba aplikační vrstvě.

3.6.2.2 Registr služeb

Service Registry, tj. místo, kde jsou uloženy informace o webových službách a jejich poskytovatelích. Místo, na kterém uživatelé mohou vyhledávat poskytovatele a které současně poskytuje informace potřebné pro navázání komunikace mezi uživatelem a poskytovatelem služeb. Tuto službu využijeme, potřebujeme-li naše služby poskytovat veřejně.

3.6.2.3 Klient

Service Requestor, je z obchodního hlediska ten, kdo vyhledává požadovanou funkci. Z hlediska architektury se jedná o aplikaci, která funkci volá, tedy zpravidla prezentační vrstva aplikace, popřípadě aplikace, která nemá vlastní rozhraní, které nahrazuje právě prostřednictvím webových služeb.

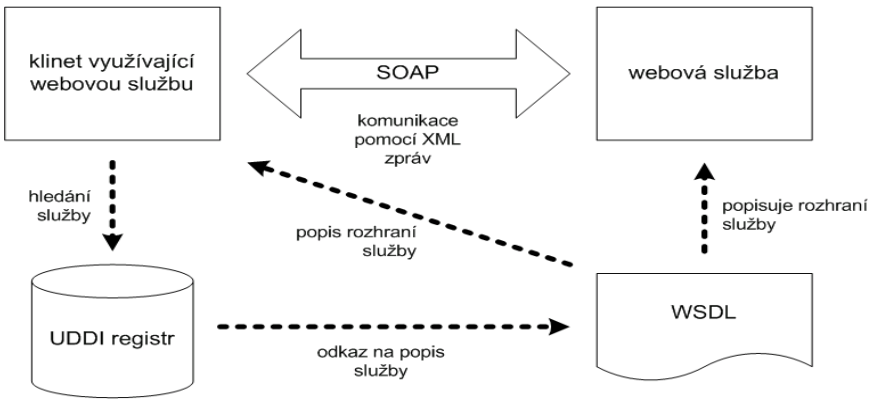
3.6.2.4 Technologie webových služeb

Webové služby umožňují jednoduchou komunikaci mezi aplikacemi, či částmi systému ve velmi heterogenním prostředí, protože komunikace je založena na platformě nezávislých standardech, především na jazyce XML a protokolu HTTP. Aplikace si mezi sebou posílají XML zprávy, které přenášejí dotazy a odpovědi jednotlivých aplikací. Celá infrastruktura webových služeb je založena na třech základních technologiích:

- **SOAP** (Simple Object Access Protocol) – protokol používaný pro komunikaci,
- **WSDL** (Web Services Description Language) – standardní formát pro popis rozhraní webové služby,
- **UDDI** (Universal Description, Discovery and Integration) – standardní mechanismus umožňující registraci a vyhledávání webových služeb.

Vzájemné vztahy mezi těmito třemi technologiemi jsou zachycené na Obrázek 3.14. Ke každé webové službě by měl být k dispozici její formální popis v jazyce WSDL. Z tohoto popisu již jde automaticky vygenerovat SOAPový požadavek. Ve větších systémech nebo přímo v otevřeném prostředí Internetu se popis služby může zaregistrovat do UDDI registru (registru služeb). Ten slouží jako jakýsi telefonní seznam („zlaté stránky“), který umožňuje vyhledávání služeb s určitými parametry.

Klient, který chce využít webovou službu, získá buď přes UDDI, nebo přímo její popis. Z něj je jasné, jakou strukturu má mít SOAPová zpráva a kam se má webové službě poslat, aby ji rozpoznala.



Obrázek 3.14. Vztah tří základních technologií webových služeb (SOAP, WSDL, UDDI). Převzato z [9].

3.7 SOAP

SOAP (SIMPLE OBJECT ACCESS PROTOCOL) je základní protokol webových služeb, který definuje strukturu zpráv a způsob, jakým se do XML kódují běžné datové typy. Nejčastěji se dnes v SOAPu používá synchronní komunikace. Klientská aplikace pošle v XML požadavek webové službě (serveru), ta dekóduje požadavek v XML, zavolá příslušný kód, a výsledek opět ve formátu XML zašle zpět klientovi.

Nejčastěji se SOAP používá ve webových službách v modelu požadavek/odpověď. Jedna aplikace pošle v XML zprávě požadavek druhé aplikaci, tak požadavek obslouží a výsledek zašle jako druhou zprávu zpět původnímu iniciátorovi komunikace. V tomto případě bývá webová služba vyvolána webovým serverem, který čeká na požadavky klientů a v okamžiku, kdy přes HTTP přijde SOAPová zpráva, spustí webovou službu a předá jí požadavek. Výsledek služby je pak předán zpět klientovi jako odpověď.

Nejprve bude zmíněna struktura zprávy SOAP (kapitola 3.7.1) a poté transportní mechanizmy zprávy SOAP (kapitola 3.7.2).

Informace k této kapitole byly čerpány z [8], [10], [25], [33].

3.7.1 Struktura zprávy SOAP

Zpráva v SOAP je jednoduchý XML dokument, který má kořenový element `Envelope`. V této obálce jsou pak uzavřeny dva elementy `Header` (hlavička) a `Body` (tělo). Hlavička je přitom nepovinná a používá se pro přenos pomocných informací pro zpracování zprávy – například identifikaci uživatele, autentizační informace (jméno, heslo) apod.

O to nejdůležitější se stará tělo zprávy, v němž se přenášejí informace identifikující volanou službu a předávané parametry, resp. návratové hodnoty služby. SOAP používá jmenné prostory pro identifikování jednotlivých částí XML zprávy. Obálka, hlavičky a tělo zprávy patří do jmenného prostoru http://schemas.xmlsoap.org/soap/envelope/. V Příklad 3.1 je ukázka struktury jednoduché zprávy SOAP.

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="urn:x-example:services:StockQuote">
      <symbol>MOT</symbol>
    </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Příklad 3.1. Struktura jednoduché zprávy SOAP.

Ukázková zpráva pro jednoduchost neobsahuje hlavičku, ale pouze tělo. V něm je požadavek na vyvolání vzdálené funkce `GetLastTradePrice` s parametrem pojmenovaným `symbol` s hodnotou `MOT` (kód akcií firmy Motorola).

3.7.2 Transportní mechanismy

Jelikož se dnes SOAP typicky používá pro volání vzdálených procedur typu dotaz odpověď, je celkem přirozené, že se pro přenos požadavku/odpovědi nejčastěji používá protokol HTTP. Důvodem je zejména široká podpora HTTP v různých aplikacích. Navíc webovou službu lze nahrát přímo na běžný webový server, jenž slouží jako „dispečer“, který jednotlivé požadavky předává odpovídající webové službě ke zpracování. Výhoda použití HTTP také spočívá v tom, že stávající síťová infrastruktura, zvláště ve firemní sféře, dovoluje v podstatě neomezenou komunikaci na portu vyhrazeném pro HTTP (TCP port 80). Webové služby je možné používat bez nutnosti zásahu do konfigurace aktivních síťových prvků, jako jsou firewally. Při použití technologií DCOM nebo CORBA je potřeba povolit komunikaci na portech, které používají příslušné přenosové protokoly (např. IIOP pro CORBA).

SOAP požadavek se zasílá v těle HTTP požadavku. Používá se přitom metoda POST, která dovoluje posílat data v těle HTTP požadavku. Požadavek musí obsahovat HTTP hlavičku SOAPAction, která identifikuje SOAP požadavek. Tuto hlavičku mohou používat jednak firewally k filtrování požadavků a jednak může obsahovat URI s identifikací služby, která se má vyvolat. Pokud je obsahem hlavičky prázdný řetězec, služba ke spuštění je identifikována přímo adresou, na kterou směřuje požadavek.

3.8 WSDL

Jazyk WSDL (WEB SERVICES DESCRIPTION LANGUAGE) patří do rodiny značkovacích jazyků (jako XML). Pomocí jazyka WSDL lze formálně specifikovat rozhraní webové služby (kde je služba k dispozici, jaké má vstupní/výstupní parametry). Pro mnoho jazyků existují knihovny, které jsou z WSDL popisu schopné automaticky generovat klientský kód, který umožní pohodlné volání webové služby stejně jako by to byla lokálně dostupná funkce nebo metoda.

WSDL vzniklo jako společná iniciativa firem Microsoft a IBM, které si uvědomily potřebu sjednocení jazyka používaného pro popis rozhraní webových služeb. Navazuje tak na předchozí aktivity, zejména na jazyky NASSL (Network Accessable Service Specification Language), SCL (SOAP Contract Language) a SDL (Service Description Language).

Tato kapitola čerpá informace z literatury [25], [9], [10], [34].

3.8.1 Základní části WSDL

WSDL soubor s definicí rozhraní služby je XML dokument. Skládá se zejména z následujících elementů, které tvoří základní části každého WSDL popisu:

- **Types.** Obsahuje definici datových struktur používaných ve zprávách. K definici lze použít teoreticky libovolný typový systém, ale nejčastěji se používají XML schémata. Nástroje pro webové služby se starají o mapování datových typů podle XML schémat na nativní datové typy použitého jazyka.
- **Message.** Definuje formát předávaných zpráv pomocí dříve definovaných datových typů. Zprávy fungují jako vstupní anebo výstupní struktury pro operace. Každá zpráva se může skládat z několika logických částí s vlastním datovým typem. Při použití SOAP pro RPC odpovídá jedna část zprávy jednomu parametru vzdálené metody.
- **Operation.** Abstraktní definice operací, které jsou službou podporovány. U operace se definuje jaké má vstupy a výstupy. Vstup a výstup je popsán již existující zprávou (message). V SOAP RPC modelu odpovídá operace metodě.
- **PortType.** Sdružuje dohromady několik operací.
- **Binding.** Slouží pro navázání určitého typu portu (portType) na konkrétní protokol a formát přenosu zpráv.
- **Port.** Jeden koncový bod služby definovaný jako kombinace síťové adresy a dříve definované vazby (binding).
- **Service.** Sdružuje několik koncových bodů (portů) do jedné služby.

3.9 UML

UML (UNIFIED MODELING LANGUAGE) je obecný a universální objektově orientovaný modelovací jazyk, vzniklý sloučením mnoha populárních metod objektově orientované analýzy a návrhu, které vznikly od počátku 90. let (Booch, OMT, ROOM, atd.). Důležitý vliv na jeho vývoj měl též vznik programovacího jazyka Java.

V roce 1997 vznikl první průmyslový standard tohoto jazyka, pod názvem „The Unified Modeling Language (UML)“, a to v rámci působení společnosti „Object Management Group“ (OMG), které vzniklo za účelem definování a prosazování standardu UML.

Od roku 2001 OMG připravuje verzi 2.0, která přináší podstatná rozšíření. Text první části (SuperStructure) byl schválen na podzim 2004, ale ještě není dokončena formální úprava dokumentu (duben 2007). Další části specifikace UML jsou připraveny k závěrečnému hlasování. Začala se již připravovat revize 2.1.

UML je grafický jazyk pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů.

Důležitou vlastností UML je jeho snaha být jazykem, nejen metodikou. Interpretace modelů, postupy a techniky jejich tvorby a ostatní metodické záležitosti jsou vědomě odsunuty mimo oblast UML – do samostatných metodik, které budou UML využívat jako základní jazykové instrumentarium. Tato snaha má především podporovat deklarovanou universálnost UML.

V dalším textu se zaměříme na jednotlivé součásti standardu UML.

Informace k této kapitole byly čerpány z [2], [30], [31], [37].

3.9.1 Součásti standardu UML

Standard UML zahrnuje:

- **Sémantický meta-model**, který je základní platformou pro definice sémantiky (významu použití) jednotlivých nástrojů (diagramů a jazyků) UML.
- Množinu **základních modelovacích konceptů** (objekt, třída, asociace, atd.), které charakterizují základní přístup a principy požívání nástrojů UML.
- **Grafickou notaci** pro použití základních modelovacích konceptů. (základních typů diagramů). Následuje přehled diagramů v UML 2.0 včetně jejich rozčlenění do skupin:
    - **strukturní diagramy:**
        - diagram tříd (class diagram),
        - diagram komponent (component diagram),
        - composite structure diagram,
        - diagram nasazení (deployment diagram),
        - diagram balíčků (package diagram),
        - diagram objektů (object diagram), též se nazývá diagram instancí,
    - **diagramy chování:**
        - diagram aktivit (activity diagram),
        - diagram užití (use case diagram),
        - stavový diagram (state machine diagram),
        - diagramy interakce:
            - sekvenční diagram (sequence diagram),
            - diagram komunikace (communication diagram, dříve collaboration diagram),
            - interaction overview diagram,
            - diagram časování (timing diagram).
- **Formální pravidla pro správnost vytvářených modelů**, vyjádřena formou omezení v tzv. „Object Constrain Language“ (OCL**).**

Celý jazyk UML je založený na třech elementech, které ale nejsou z uživatelského hlediska reprezentovány v textové podobě, ale grafickými značkami v plošném (tj. dvourozměrném) grafu. Tyto tři základní elementy jazyka UML se dle své funkce nazývají:

- **Předměty,**
- **relace,**
- **diagramy.**

3.9.1.1 Předměty

Předměty jsou elementy zpracovávaného modelu. Jsou dále členěny do podkategorií:

- **Strukturní abstrakce** UML modelu, například programové třídy, aplikační či objektové rozhraní, případy užití, komponenty či uzly. V diagramu UML jsou strukturní abstrakce zobrazeny jako různé převážně plošné tvary, které jsou však vždy uzavřené.
- **Chování**, která v UML diagramu prezentují interakce, tj. vzájemné komunikace mezi jednotlivými objekty. Chování se v UML diagramu většinou vyznačuje pomocí různým způsobem konstruovaných a různě zakončených šipek či propojovacích čar.
- **Seskupení**, které podle potřeby modelu graficky seskupuje části diagramu na nižší úrovni. Většinou se jedná o balíčky.
- **Poznámky** blíže specifikují vlastnosti a chování dalších elementů UML diagramu.

3.9.1.2 Relace

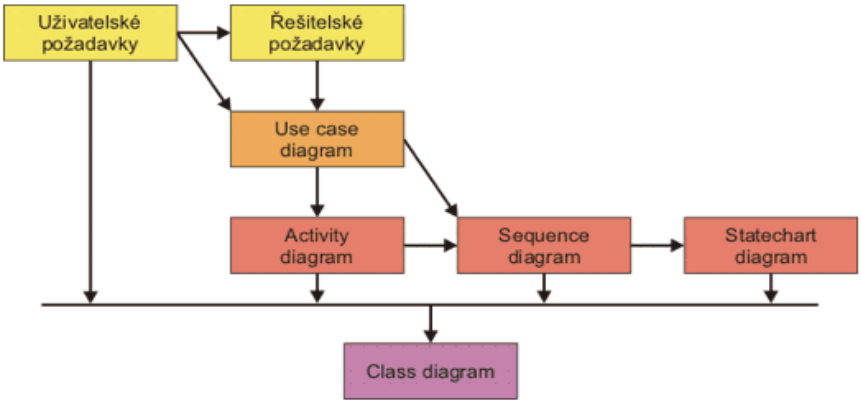
Relace zajišťují vztahy mezi různými předměty. V UML jsou rozeznávány následující podtypy relací:

- **Asociace.** Pomocí asociací se modeluje obecná souvislost předmětů, která je však v diagramu UML přesným způsobem definovaná. Speciální variantou asociace jsou takzvané **kompozice** a **agregace**, využívané často v objektově orientovaných jazycích a návrzích databází.
- **Závislost.** Použije se, pokud změna v jednom předmětu způsobí změnu v předmětu jiném, nebo mu známým způsobem poskytne požadovanou informaci.
- **Generalizace.** Pomocí generalizace se modeluje stav, kdy je jeden předmět specializací jiného předmětu. Tato relace je velmi často používána v objektově orientovaných jazycích, implementuje se většinou pomocí dědičnosti.
- **Realizace.** Jedná se o druh vztahu, ve kterém jeden předmět představuje dohodu, za jejíž splnění je odpovědný jiný předmět. V objektově orientovaných jazycích se realizace vytváří pomocí rozhraní (interface).

3.9.1.3 Diagramy

Diagramy jsou nejznámější a nejpoužívanější částí standardu. Jsou to dvourozměrné objekty, zachycující různé aspekty modelovaného systému. Diagramů existuje celá řada (diagram případů použití**,** diagram tříd**,** diagram objektů**,** diagram komponent**,** diagram nasazení**,** diagram aktivit**,**

stavový diagram**,** diagram spolupráce a sekvenční diagram). Obrázek 3.15 ukazuje použití jednotlivých diagramů ve vývojovém procesu:

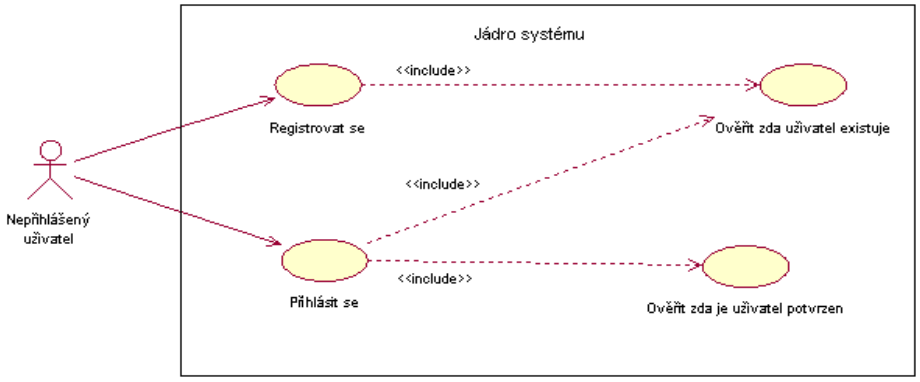


Obrázek 3.15. Použití jednotlivých typů UML diagramů ve vývojovém procesu. Převzato z [37].

Use Case diagram. Případy užití, neboli *Use Case* jsou specifikace posloupnosti činností, včetně proměnných posloupností a chybových posloupností, které systém, podsystém nebo třída může vykonat prostřednictvím interakce s vnějšími účastníky. Případy užití jsou psány z pohledu zákazníka a podávají první představu o rozsahu projektu.

Diagramy případů užití (Obrázek 3.16) nám poskytnou rychlou představu o jednotlivých funkcích systému, ale přesné postupy, rozšiřující a alternativní scénáře musejí být zachyceny v textové formě. V jazyce UML je kodifikován pouze diagram případů užití, strukturu dokumentu s textem případů užití se navrhuje zvlášť.

Prvky diagramu užití jsou **případ užití** a **aktéři** (role objektu vně systému, které se systémem přímo interaguje).



Obrázek 3.16. Příklad jednoduchého UseCase diagramu.

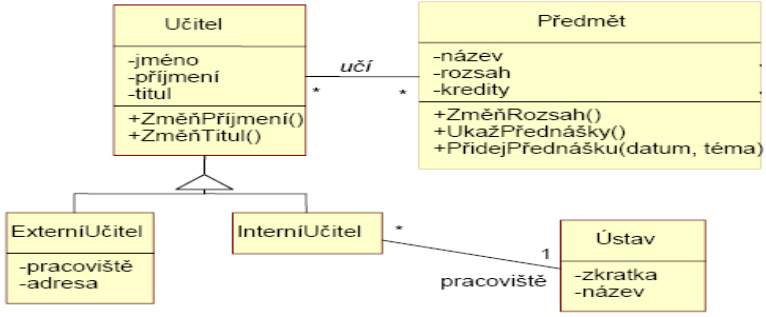
Diagram tříd (Obrázek 3.17) zachycuje typy objektů a statické vztahy mezi nimi. Třídy zachycují společné vlastnosti sady objektů - atributy a operace (metody). Stereotypem lze zavést nové druhy (skupiny) tříd. Nejčastějšími skupinami (stereotypy) tříd jsou:

- **entitní třídy** (stereotyp je <<entity>>),

- **třídy rozhraní** (stereotyp je <<boundary>>),
- **třídy řídicí** (stereotyp je <<control>>).

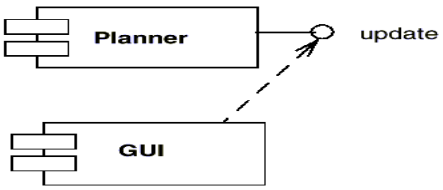
Vztahy mezi třídami (asociace) vyznačují možné vazby mezi objekty. Konce vztahů mohou být ohodnoceny rolí – role označují, jakou roli objekt ve vztahu hraje. Pro zachycení kardinality a volitelnosti vztahů se používá notace N..M, kde N a M může být číslo nebo *. Pokud je zapotřebí si o vztahu něco pamatovat, požívají se tzv. přidružené třídy (atributy vztahů). Speciální druhy vztahů představují jsou:

- **Agregace** (aggregation) . Jedna třída je součástí jiné třídy - vztah typu celek/část.
- **Kompozice** (composition) . Silnější druh agregace - u kompozice je část přímo závislá na svém celku, zaniká se smazáním celku a nemůže být součástí více než jednoho celku.
- **Generalizace** (generalization). Jedna třída je zobecněním vlastností jiné třídy (jiných tříd) - vztah typu nadtyp/podtyp, generalizace/specializace.



Obrázek 3.17. Příklad jednoduchého diagramu tříd.

Diagram komponent (Obrázek 3.18) ukazuje rozhraní, komponenty a vztahy mezi nimi. Komponenta je fyzická vyměnitelná část systému, která realizuje určitou množinu rozhraní.



Obrázek 3.18. Příklad jednoduchého diagramu komponent.

3.10 PHP

PHP byl vytvořen v roce 1994 (původní význam: Personal Home Pages, nyní: PHP Hypertext Preprocesor). V zápětí prošel jazyk velkým vývojem a v současnosti jej na svých stránkách používá přes 20 milionu serverů. PHP je skriptovací jazyk speciálně navržený pro potřeby webových stránek. Spolu s ASP, ASP.NET, JSP, CGI a dalšími patří mezi skriptovací programovací jazyky vykonávané

na straně serveru, což znamená, že PHP skripty provádí interpret na serveru a klientovi posílá až zpracované HTML stránky vytvořené či modifikované těmito skripty.

PHP tedy obsahuje všechny výhody této skupiny dynamických webových prezentací, jako je reakce na vstupní parametry nebo snazší údržba a aktualizace vytvořených prezentací. Oproti jiným technologiím dynamických stránek pak nabízí:

- Uživatelské relace, cookies,
- těsnou integraci se všemi dostupnými databázovými systémy,
- přenositelnost na nejrozšířenější platformy,
- nízkou cenu (Open Source - zdarma),
- možnosti rozšiřování,
- krátkou zaučovací dobu,
- vysoký výkon,
- k dispozici zdrojový kód PHP,
- kvalitní on-line manuál.

PHP skript se do HTML kódu vkládá mezi značky `<?php` a `?>`. Veškerý kód mimo tyto značky je považován za HTML a posílán na výstup. Tento princip umožňuje libovolné střídání HTML kódu s PHP skriptem, avšak značně snižuje přehlednost výsledného zdrojového textu. Syntaxe PHP je velmi podobná jazykům jako C nebo Perl. Díky této vlastnosti nečiní programátorům problémy se jej rychle naučit.

Díky své všestrannosti se PHP stalo oblíbeným nástrojem pro tvorbu internetových informačních systémů, portálů a elektronických obchodů.

PHP je aktuálně dostupné ve verzi 5.1.2 (verze 5 sebou přináší významná rozšíření, zejména v podobě podpory objektově orientovaného přístupu).

Stáhnout jej lze zdarma z http://www.php.net.

Informace k této kapitole byly čerpány z [8], [12].

3.10.1 Objektově orientovaný přístup v PHP

PHP verze >=5.0 umožňuji plně objektově orientovaný přístup se všemi jeho základními vlastnostmi (zapouzdření, dědičnost, polymorfismus, konstruktory, výjimky, abstrakce). V aplikacích využívajících PHP se stále hojně používá procedurální programování. Pro řešení komplexnějších úloh a systémů je však objektově orientovaný přístup nezbytný. Stále více programátoru již také není ochotno se vrátit k procedurálnímu programování.

Není cílem této práce zabývat se detaily a principy objektově orientovaného programování. V současné době existuje mnoho knížek, včetně on-line publikací, kde se lze o objektově orientovaném programování dozvědět více. Např [12].

3.10.1.1 Třídy a objekty v PHP

Příklad 3.2 ukazuje objekt `$obj` třídy `Myclass` s veřejnou metodu `getText()`. Ta vrátí obsah proměnné `$text`, který se inicializoval při vytváření objektu `$obj`.

```
<?php
class Myclass {
  private $text;
  public function __construct() {
    $this->text = "ahoj svete";
  }
  public function getText() {
    return $this->text;
  }
}
$obj = new Myclass();
echo ($objekt->getText()); // ahoj svete
?>
```

Příklad 3.2. OOP v PHP.

3.10.2 Více o PHP

Jazyk PHP je již dostatečně znám a dokumentován v mnoha knihách i on-line publikacích a není cílem této práce se detailně zabývat vlastnostmi jazyka PHP. Více o PHP i OOP v PHP například najdete na [12].

3.11 MySQL

MySQL je databázový systém, vytvořený švédskou firmou MySQL AB. Je považován za úspěšného průkopníka dvojího licencování – je k dispozici jak pod bezplatnou licencí GPL, tak pod komerční placenou licencí. Aktuální verze MySQL (duben 2007) je 5.1.17.

MySQL je oblíbený multiplatformní víceuživatelský robustní relační databázový systém (RDBMS) vhodný pro použití ve webových aplikacích. Pro dotazování dat používá standardní dotazovací jazyk SQL. Důvodem pro použití při vývoji v prostředí webových aplikací je silná podpora v PHP.

Informace k této kapitole jsou čerpány z (2007), (2007).

Mezi hlavní přednosti MySQL patří:

- Vysoká výkonnost,
- nízké náklady,
- snadná konfigurace a výuka,
- přenositelnost,
- k dispozici zdrojový kód.

MySQL bylo od počátku optimalizováno především na rychlost, a to i za cenu některých zjednodušení: má jen jednoduché způsoby zálohování, a až donedávna nepodporovalo pohledy, triggery, a uložené procedury. Tyto vlastnosti jsou doplňovány teprve v posledních letech.

Přehled aktuálně podporovaných vlastností:

- **Cizí klíče** (od verze 3.23 podporovány v tabulkách typu InnoDB),
- **transakce** (od verze 3.23 podporovány v tabulkách typu InnoDB)
- **podpora různých znakových sad** a časových pásem v datech (od verze 4.1)
- **poddotazy** (od verze 4.1),
- **uložené procedury** (od verze 5.0),
- **triggery** (od verze 5.0),
- **pohledy** (od verze 5.0).

3.11.1 Více o MySQL

Není cílem této práce podrobně dokumentovat všechny vlastnosti a principy serveru MySQL. Více informací, včetně možnosti stažení aktuální verze zdarma lze nalézt na http://www.mysql.com.

3.12 XHTML

HTML (HYPERTEXT MARKUP LANGUAGE) je jazyk pro publikování hypertextu na WWW. Je to nepatentovaný formát založený na SGML. HTML využívá tágy ke strukturování textu do nadpisů, odstavců atd. Vývoj HTML již skončil, XHTML(EXTENSIBLE HYPERTEXT MARKUP LANGUAGE) je nástupce HTML založený na XML vyvinutý konsorciem W3C.
Informace o této kapitole byly čerpány z [32], [36].

3.12.1 Rozdíly oproti HTML

XHTML dokument musí, na rozdíl od HTML dodržovat pravidla dané XHML, které vycházejí z XML jako:

- Element `html` vždy obsahuje dva elementy, `head` (hlavičku) a `body` (tělo dokumentu). Hlavička musí obsahovat element `title` a měla by obsahovat i `metatag` pro kódování (kvůli starším prohlížečům).
- Všechny tagy i atributy musí být malými písmeny, XHTML je case-sensitive.
- Všechny hodnoty atributů musí být v XHTML v uvozovkách.
- Všechny XHTML tagy musí být párové. Při použití prázdného tagu se musí tag ukončit lomítkem, např. `<img />`.
- Tagy se nesmí nikdy křížit.

3.12.2 Více o XHTML

Jazyk HTML, potažmo XHTML je v současné době široce dokumentován jak na on-line serverch, tak v mnoha knihách. Není cílem této práce více rozvádět principy a vlastnosti jazyka XHTML. Více o XHTML naleznete například na http://www.webtvorba.cz/xhtml/uvod-do-xhtml.html.

3.13 CSS

CSS (CASCADING SYTLE SHEETS) vznikly jako souhrn metod pro úpravu vzhledu stránek. První návrh normy byl zveřejněn v roce 1994, v roce 1996 byla pak vydána specifikace CSS 1, v roce 1998 CSS 2, nyní se pracuje na verzi CSS 3.

Hlavním smyslem je umožnit návrhářům oddělit vzhled dokumentu od jeho struktury a obsahu. Původně to měl umožnit už jazyk HTML, ale v důsledku nedostatečných standardů a konkurenčního boje výrobců prohlížečů se vyvinul jinak. Starší verze HTML obsahují celou řadu elementů, které nepopisují obsah a strukturu dokumentu, ale i způsob jeho zobrazení. Z hlediska zpracování dokumentů a vyhledávání informací není takový vývoj žádoucí.

CSS styly umožňují přesně určit, jak bude který element vypadat. Stylem můžeme definovat jednotný vzhled elementu pro celý dokument nebo určit odlišné formátování třeba jen jediný výskyt určitého elementu.

Informace k této kapitole byly čerpány z [37], [5].

Výhod CSS oproti klasickému formátování je mnoho, např.:

- Rozsáhlejší možnosti,
- konzistentní styl,
- oddělení struktury a stylu,
- dynamická práce se styly,
- formátování XML dokumentů,
- větší kompatibilita alternativních webových prohlížečů,
- kratší doba načítání stránky, atd.

3.13.1 Více o CSS

Není účelem této práce rozebírat podrobně vlastnosti jazyka CSS a použití stylů v HTML kódu. Více o CSS naleznete například na [5], http://www.webtvorba.cz/css/uvod-do-css.html.

4 Analýza

V této kapitole je nejprve rozebrána charakteristika současného stavu informačních systémů obecně a to hlavně informačních systému založených na technologii PHP (kapitola 4.1.1), dále je provedena analýza současného stavu informačního systému a webových portálů sdružení SDC. Na základě těchto analýz je vytvořena formální a neformální specifikace požadavků na vyvíjený systém (kapitola 4.2) a specifikace požadavků na modul Kontakty (kapitola 4.3).

4.1 Charakteristika současného stavu

Tato kapitola je zaměřena na popis stávajícího stavu informačních obecně a současného stavu informačního systému Sdružení SDC.

U obecného popisu se zaměříme hlavně na vlastnosti a aspekty, které mají významný vztah k této práci. Cílem této podkapitoly není detailní analýza stávajícího stavu informačních systému, ale stručné nastínění přístupů menších a středních informačních systémů.

4.1.1 Stávající stav informačních systémů obecně

Informační systémy lze rozdělit dle mnoha kriterií, např. podle použitých technologií. Často jsou malé a střední informační systémy implementovány pomocí technologií PHP, ASP (Active Server Pages), JSP (Java Server Pages).

Lze se domnívat, že jedny z nejrozšířenějších jsou však systémy založené na PHP a to z mnoha důvodu:

- PHP technologie je zdarma,
- PHP je jazyk blízký C, mnoho programátorů ho zná,
- Existuje mnoho článků, tipů a manuálů na webu,
- velká dostupnost na hostingových serverech zdarma,
- malá nebo drahá dostupnost jiných technologií na hostingových serverech,
- snadno k naučení, stačí velmi málo znalostí k jednoduchým skriptům v PHP.

Díky těmto důvodům se stala (dle názoru autora) technologie PHP jedna z nejrozšiřitelnějších technologií v oblasti malých a středních informačních systémů. Protože je jedním z požadavků na implementaci tohoto systému právě použití PHP, zaměříme se blíže jen na informační systémy s touto technologii.

4.1.1.1 Objektově orientovaný přístup u informačních systémů založených na PHP

Technologie PHP byla až do své trojkové verze čistě procedurální, obdobně jako jazyk C. Verze 4 a vyšší sice přinesla určitou možnost objektového přístupu, ale zdaleka nešlo mluvit o objektově orientovaném jazyku. Plnou podporu objektového paradigmatu přinesla až PHP verze 5 a vyšší.

Protože je verze 5 poměrně nová (léto 2004), jsou starší řešení založena právě na procedurálním přístupu se všemi jeho nevýhodami. Jelikož je procedurální přístup u PHP vžitý a většina průvodců a manuálů ukazuje použití PHP právě na procedurálním přístupu, vzniká i přes verzi PHP podporující objektový přístup stále dostatek nových systémů založených na procedurálním přístupu.

Protože objektový přístup přináší řadu výhod, bude i informační systém sdružení SDC založen plně na tomto paradigmatu.

4.1.1.2 Architektury informačních systémů založených na PHP

Stále používaný procedurální přístup vedl a vede programátory k psaní nepřehledných aplikací, založených na volání funkcí přímo v obsahu dokumentu a **neoddělování aplikační logiky od zobrazení.**

Mnoho informačních systémů založených na PHP spojuje technologií PHP a relační databázový systém MySQL. Je tak oddělena vrstva datová od vrstvy aplikační a lze tak mluvit o **dvouvrstvé architektuře** (klient-server) těchto systémů.

Absence třetí vrstvy přináší i u středně rozsáhlých informačních systémů velké problémy s údržbou, rozšířením, zatěžováním serveru apod.

V současné době je snaha o vznik systémů s třívrstvou architekturou (3.4.4), čili oddělení zobrazování od aplikační logiky a dat. Pokud má být však zobrazovací vrstva opravdu oddělena, znamená to, že lze nahradit tuto vrstvu jinou technologií. **Většina systémů je** však **svázaná s internetovým prohlížečem** a jazykem HTML a neumožňuje tuto zobrazovací vrstvu nahradit jiným řešením. Není tak dosaženo striktní třívrstvé architektury, ale jen určitého pomyslného rozdělení. Pokud je informační systém určen jen na použití s internetovým prohlížečem, problém to není.

Problém nastává ve chvíli, kdy máme např. více informačních systémů v jedné firmě běžících na odlišných strojích a potřebujeme využívat funkce jednoho systému v systému jiném, či pokud požadujeme zobrazení nejen přes internetový prohlížeč, ale i jiné zařízení (např. terminál, mobilní zařízení, desktopové aplikace, apod.). Poté je potřeba použit plnou třívrstvou architekturu.

Protože je zatím v PHP implementováno jen velmi málo informačních systémů, které umožňují oddělit všechny tři vrstvy a rovněž protože aplikace našeho IS v prostředí sdružení SDC klade požadavek na zpřístupnění funkcí systému již existujícím systémům sdružení (viz dále), bude i tento systém implementován jako třívrstvý s možností nahradit všechny tři vrstvy jinými technologiemi.

4.1.1.3 Architektonické vzory a informační systémy založené na PHP

Architektonické vzory jsou většinou svázány s objektovým paradigmatem. Jazyk PHP podporoval dlouho dobu jen procedurální přístup a programování na základě architektonických vzorů nebylo využíváno.

V současné době, kdy jazyk PHP ve verzi 5 a vyšší plně podporuje objektový přístup, se stále častěji objevují řešení, které implementují principy architektonických vzorů, nejčastěji vzoru Model-View-Controller.

Protože použití vzoru Model-View-Controller pro oddělení aplikační logiky od zobrazení se přímo vybízí, bude podle tohoto vzoru navrhnut i tento systém.

4.1.2 Stávající stav informačního systému IS SDC

Současný webový portál občanského sdružení (http://www.bdc.cz) obsahuje jen statické informace o společnosti a čím se zabývá (stručně o sdružení v kapitole 1). Na stránkách najdeme kalendář akcí a novinky, ty se musí editovat ručně. Editor je tak nucen znát HTML kód a PHP. Každá tato komponenta je navíc od jiného autora a je na systém nešetrně „nalepena“.

Systém neposkytuje žádné služby pro registrované uživatele. Systém je postaven na univerzálním redakčním systému SAMBA, který však zdaleka nevyhovuje specifickým požadavkům sdružení. Tento systém zatěžuje neúměrně diskový prostor i databázi. Je zbytečně moc univerzální a pro potřeby společnosti **nepřehledný a složitý na údržbu**.

Kromě základního portálu mají jednotlivé části sdružení (překladatelé, PR, lokální organizace, nadační font, atd.) vlastní webové stránky, které popisují jejich činnost, popřípadě nabízí registrovaný přístup například pro sdílení překladatelských textů, či interní informace.

Často tak dochází k nežádoucí **duplicitě dat**. Mnohdy dochází taky k neaktuálnosti dat, kdy informace je aktualizovaná například na serveru překladatelů, ale není aktualizovaná na hlavním serveru sdružení.

Členové sdružení se musí přihlašovat do různých webových portálů. Neexistuje ani **žádný rozcestník** pro tyto servery. Jeden člověk, který je například členem lokální skupiny, překládá a je členem vydavatelského týmu, se tak musí přihlašovat do tří různých webů. Tyto weby obsahuji většinou ty samé informace o přihlášeném uživateli, jako kontakty, adresu apod. Pokud se změní např. kontaktní adresa, musí ji tak uživatel **změnit na všech serverech**.

Protože dosavadní stav zdaleka nevyhovuje podmínkám rychle se rozvíjejícího sdružení s dnes již stovkami aktivních členů a několika různých týmu, pracujících na místních i mezinárodních

projektech, jejichž členové jsou mnohdy z mnoha měst republiky, vznikl ze strany sdružení požadavek na vytvoření komplexního systému, který by přesně odpovídal specifickým požadavkům sdružení.

Pro zjištění požadavků na systém byly uskutečněny schůzky s různými členy sdružení, jako jsou vedoucí projektů, členové překladatelských, PR a jiných týmů, správci systému i běžní uživatelé (členové).

Dále byly prostudovány současné informační systémy podobných společností v Evropě. Důraz byl při tom kladen na specifické podmínky střední Evropy.

4.2 Požadavky na systém

Požadavky na vznikající systém vznikly na základě důkladné analýzy stávajícího stavu informačního systému a požadavků sdružení SDC, na základě studia obdobných systému v zahraničí, studia stavu obecných systému, ale také na základě stanovených cílů této práce.

4.2.1 Slovníček pojmů

Než budou specifikovány funkční a nefunkční požadavky a proveden návrh a implementace systému, bude vytvořen slovníček pojmů, které se dále v textu vyskytují:

- **Systém.** Systémem je myšlen v této práci vznikající informační systém, který se skládá z **jádra systému** a jeho **modulů**. Systém je nezávislý na jeho konkrétní aplikaci a nasazení.
- **Jádro systému.** Jádro systému je základ *systému*. Je to samostatně fungující kostra *systému*, která poskytuje veškerá potřebná uživatelská rozhraní, třídy, funkce a prvky k možnému rozšíření. Jádro systému lze nasadit i jako samostatnou aplikaci, která se díky možnosti přidávat nové **sekce systému,** může stát plnohodnotnou samostatnou *aplikací.*
- **Modul systému.** Modul systému je balíček, který poskytuje rozšíření funkčnosti systému o nové prvky (např. Kontakty, fotogalerie, apod.). *Jádro systému* bude podporovat snadnou integrací nových modulů. Více o modulech v kapitole 5.1.6.
- **Informační systém sdružení SDC.** IS SDC je konkrétní aplikace systému. IS SDC je nasazení *systému (jádra systému* a požadovaných *modulů)* v prostředí sdružení SDC.
- **Sekce systému.** Sekce systému umožňují rozšíření informačního prostoru *systému* při zachování jeho funkcí. Nerozšiřujeme tak funkčnost systému (jako u *modulů*), ale jeho informační schopnost (např. přidáním sekce o výrobku, sekce o části firmy apod.). Více o sekcích v kapitole 5.1.7.
- **Aplikace systému.** Aplikací systému je myšleno jeho konkrétní nasazení v určitých podmínkách. *Informační systém sdružení SDC* je aplikací systému v prostředí sdružení SDC.
- **Třívrstvá architektura.** Viz kapitola 3.4.4.

- **Klientská, aplikační a datová vrstva.** Viz kapitola 3.4.4.
- **Intraweb.** Interní část *systému*, která je dostupná po přihlášení.

4.2.2 Souhrn požadavků vyplívajících z cílů této práce

Tato kapitolka nepopisuje přímo požadavky kladené na systém, ale připomíná požadavky, které pramení ze zadání práce a z cílů této práce (viz kapitola 2). Mezi tyto požadavky patří:

- Plně objektově orientovaný přístup v PHP.
- Striktní oddělení zobrazovací vrstvy, logiky a datové vrstvy tak, aby mohlo dojít k bezproblémovému nahrazení jakékoli z těchto vrstev.
- Použití návrhového vzoru Model-View-Controller.
- Navrhnout systém nezávislý na jeho aplikaci (znovu aplikovatelný v různých prostředích).
- Snadná rozšiřitelnost systému.
- Snadná údržba a administrace systému.
- Přístup k funkcím systému s jiných systému.
- Bezpečnost dat a přístupu.

Na základě podrobné analýzy stávajícího stavu informačního systému sdružení SDC, analýzy požadavků kladených na IS SDC a na základě požadavků plynoucích z cílů této práce byly vytvořeny funkční a nefunkční požadavky systému.

4.2.3 Nefunkční požadavky

Nefunkční požadavky specifikují, či vymezují způsob a podmínky, za jakých bude systém implementován (více o nefunkčních požadavcích v kapitole 3.2.6).

Na základě analýzy stávajícího stavu a požadavků plynoucích ze zadání práce a z cílů práce, byly vytvořeny tyto nefunkční požadavky systému:

- Informační systém bude tvořen funkčním jádrem (*jádro systému*), které bude poskytovat možnosti rozšíření prostřednictvím *modulů*.
- *Jádro systému* bude plně funkční *systém* i bez přidaných modulů.
- *Jádro systému* bude nezávislé na jeho konkrétním nasazení, *aplikaci*.
- Předpokládaný počet registrovaných uživatelů bude 500.
- Předpokládaný maximální počet on-line uživatelů současně bude 100.
- *Systém* bude umožňovat oddělení zobrazování dat (*zobrazovací vrstvy*) od logické části (*aplikační vrstvy*) a *datové vrstvy*.
- Systém bude umožňovat přístup k funkcím systému i jiným informačním systémům, či klientům.

- *Jádro systému* bude napsáno v programovacím jazyku PHP5.
- Databázovou vrstvu *systému* bude tvořit relační databázový server MySQL 5.
- *Zobrazovací vrstva* bude implementována pomocí značkovacího jazyka XHTML s využitím jazyka PHP5.
- Formátování obsahu *zobrazovací vrstvy* bude implementováno pomocí jazyka CSS.
- *Systém* bude testován a bude podporovat prohlížeče Internet Explorer 6.0 a vyšších (MS Windows) a FireFox 2.0.
- *Jádro systému* bude napsán s důrazem na bezpečnost dat a ochranu údajů, zabezpečení přístupu k funkcím aplikační logiky a přístupu na zabezpečené stránky bude na straně *aplikační vrstvy*.

4.2.4 Funkční požadavky

Funkční požadavky formulují to, co by měl *systém* dělat, jaké jsou jeho funkce. Proto je potřeba rozlišit funkční požadavky *aplikace systému*, které vycházejí z potřeb sdružení SDC a požadavky kladené na tento *systém* nezávislé na jeho aplikaci.

Mezi aplikačně závislé požadavky, které vycházejí s potřeb sdružení SDC patří například fotogalerie, redakční systém, diskusní fórum apod. Toto je jen otázka přidaných *modulů* k systému.

Cílem této práce není popsat vznik celé této konkrétní *aplikace systému*, ale na příkladu aplikace ukázat funkčnost navrženého systému. Proto se dále v textu budeme zabývat jen *systémem, jádrem systému* a jeho požadavky nezávislými na konkrétní aplikaci.

Funkční požadavky systému nezávislé na *aplikaci systému*:

- Jádro systému bude umožňovat přístup nepřihlášeným uživatelům.
- Jádro systému bude umožňovat přihlašování registrovaným uživatelům.
- Jádro systému bude umožňovat registraci nových uživatelů.
- Registrovaný uživatel bude mít přístup do systému až po potvrzení přístupu již registrovaným uživatelem.
- Jádro systému bude umožňovat přístup k interním stránkám (*INTRAWEBU*) na základě práv přístupu uživatelů.
- Jádro systému bude umožňovat vytváření nových sekcí systému (např. stránky projektů, týmů apod.), které budou využívat dostupných funkcí systému (jádra systému i modulů).
- Sekce systému bude moct definovat vlastní práva přístupu každému uživateli a stránce, nebo zdědí tato práva od sekce rodičovské.
- Systém bude umožňovat definovat vlastní rozvržení stránek v každé sekci a vlastní vzhled každé sekce.
- Systém bude umožňovat přidávat moduly systému, tj. rozšiřovat funkčnost systému.
- Systém bude umožňovat zabezpečený přístup k stránkám a vybraným funkcím systému.

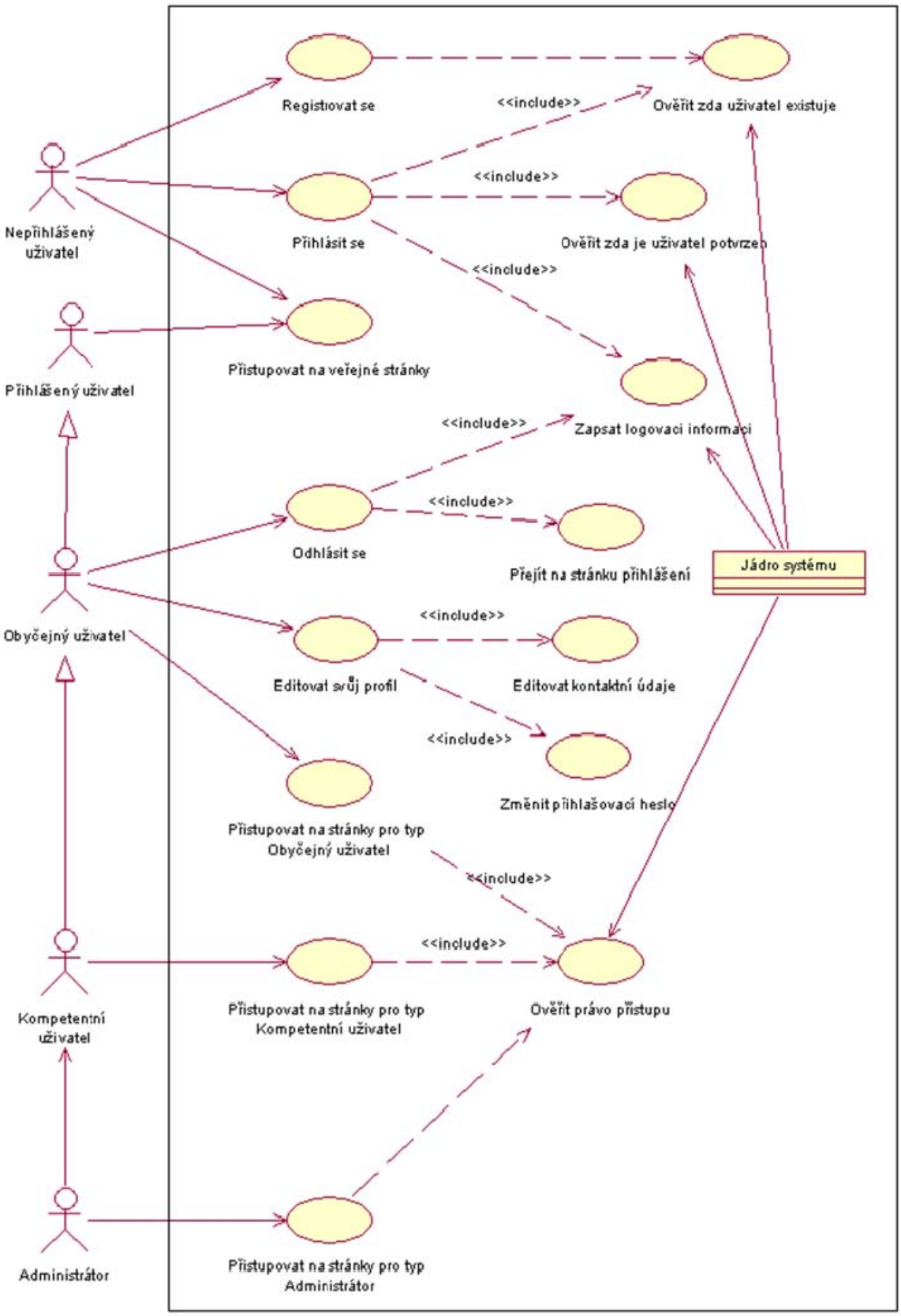
- Systém bude umožňovat vícejazyčnou mutaci textů.
- Systém bude umožňovat definovat vlastní titulek stránky, klíčová slova a popis.
- Systém bude umožňovat přidávání jazyků.
- Systém bude poskytovat tyto základní typy uživatelů:
    - **Obyčejný uživatel -** obyčejný uživatel je úspěšně přihlášený uživatel nejnižší úrovně
    - **Kompetentní uživatel -** Kompetentní uživatel bude mít v jádře systému všechna práva obyčejného uživatele a bude možnost mít některá práva navíc.
    - **Administrátor -** Administrátor bude mít nejvyšší právo v systému.
    - **Nepřihlášený uživatel –** uživatel, jenž není přihlášený.
- Každému uživateli bude možnost ke každé stránce definovat určitý typ uživatele.
- Systém bude umožňovat navigaci pomocí přehledného menu.
- Systém bude umožňovat vytváření a modifikaci menu dle potřeb v každé sekci.
- Vzhled menu, včetně jeho struktury bude možné nezávisle modifikovat v každé sekci.

4.2.5 Modelování případu užití

Dalším významným prvkem analýzy požadavků je použití diagramu případu užití. Diagram definuje na základě analýzy požadavků hranice systému, účastníky a případy užití. U některých případů užití je výhodné vytvořit specifikaci, která daný případ detailně analyzuje.

4.2.5.1 Diagram případu užití (Use Case Diagram)

Diagram případu užití na obrázkuObrázek 4.1 popisuje interakci uživatele (v rolích jednotlivých typů uživatelů) se systémem.



Obrázek 4.1. Diagram případu užití pro jádro systému.

4.2.5.2 Specifikace případu užití „Přihlásit se“

Specifikace v tabulce Tabulka 4.1 detailněji popisuje chování systému při přihlášení nepřihlášeného uživatele.

Případ užití: Registrovat se
Účastníci: Nepřihlášený uživatel
Vstupní podmínky: 1. Uživatel musí být v systému registrovaný. 2. Uživatel musí být potvrzen.
Tok událostí: 1. Uživatel vyplní uživatelské jméno a heslo. 2. Systém ověří správný formát jména a hesla 3. Je-li formát správný, pak 3.1 Systém ověří, zda je uživatel registrován a potvrzen 3.2 Je-li uživatel potvrzen a registrován, pak 3.2.1 Systém ověří správnost jména a hesla. 3.2.2 Je-li heslo správné, pak 3.2.2.1 Systém vygeneruje Hash číslo uživatele. 3.2.2.2 Systém uloží Hash číslo a ID uživatele do databáze přihlášených uživatelů. 3.2.2.3 Systém vrátí toto číslo klientské vrstvě, která ho bude používat pro identifikaci uživatele. 3.2.2.4 Systém aktualizuje model a přesměruje uživatele na úvodní stránku intrawebu. 3.2.3 Jinak: 3.2.3.1 Systém ohlásí zákazníkovi chybu. 3.3 Jinak: 3.3.1 Systém ohlásí zákazníkovi chybu. 4. Jinak: 4.1 Systém ohlásí zákazníkovi hlášení o chybném formátu.
Následné podmínky:
Alternativní tok: 1. Uživatel může kdykoli opustit stránku s přihlašováním.

Tabulka 4.1. Specifikace případu užití "Přihlásit se".

4.2.5.3 Specifikace případu užití „Přistupovat na stránky pro typ Obyčejný uživatel“

Specifikace detailněji popisuje chování systému při přístupu uživatele na stránku, která je určena pro přístup typu Obyčejný uživatel.

Případ užití: Přistupovat na stránky pro typ Obyčejný uživatel
Účastníci: Nepřihlášený uživatel
Vstupní podmínky: 1. Uživatel musí být v systému registrovaný. 2. Uživatel musí být přihlášen.
Tok událostí: 1. Uživatel se pokusí vstoupit na danou stránku. 2. Systém zkontroluje, zda daná stránka existuje. 3. Existuje-li, pak 3.1. Systém zjistí ID Stránky. 3.2. Systém zjistí ID uživatele dle Hash čísla. 3.3. Systém zjistí, jaký typ uživatele je pro danou sekci. 3.4. Systém zjistí, zda má pro danou stránku daný typ uživatele přístup. 3.5. Má-li přístup, pak 3.5.1 Vrátí informace o stránce. 3.6. Jinak: 3.6.1. Systém najde nejbližší stránku, která je definována jako stránka, kde mají všichni uživatelé přístup pro danou sekci. 3.6.2. Systém vrátí informace o stránce. 4. Jinak: 4.1. Systém najde chybovou stránku. 4.2. Systém vrátí informace o stránce.
Následné podmínky:
Alternativní tok:

Tabulka 4.2. Specifikace případu užití „Přistupovat na stránky pro typ Obyčejný uživatel“.

4.3 Požadavky na modul Kontakty

4.3.1 Nefunkční požadavky

- Modul bude implementován za pomocí technologii PHP a MySQL.
- Modul bude napojen na informační systém IS SDC.

4.3.2 Funkční požadavky

Modul bude umožňovat:

- Vytvoření a editaci kontaktů společnosti.
- Kontakty budou obsahovat tyto informace o společnosti:
    - Název,

    - IČ,
    - DIČ,
    - Odkaz na www stránky.
- Kontakty mohou obsahovat:
    - více kontaktních adres, např. poboček společnosti,
    - více bankovních účtů,
    - více kontaktních osob,
- Kontaktní osoby mohou být převzaty z uživatelů systému.
- Kontaktní adresa může být vytvořena pro každou sekci systému.

5 Návrh

Kapitola se zabývá návrhem systému, strukturou jeho jednotlivých částí a vrstev. Návrh vychází z analýzy požadavků kladených na systém. Při návrhu je použito jazyka UML, k tvorbě UML diagramů softwaru *Rational Rose Enterprise Edition 2002* (http://www.rational.com/).

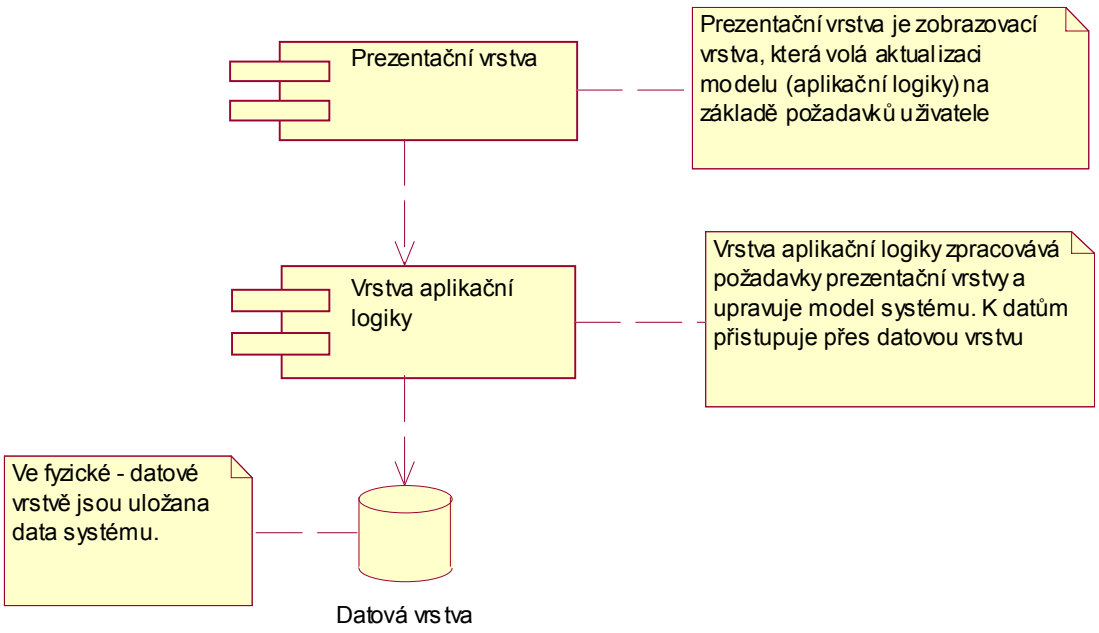
V této kapitole je popsán návrh těch části systému, které jsou nezávislé na implementaci v konkrétním jazyku, nebo technologii. V případě potřeby bude uveden odkaz na implementaci dané části.

5.1 Architektura systému

Architektura systému bude dle požadavků třívrstvá (klientská vrstva, aplikační a datová), dále musí být dodržen architektonický vzor Model-View-Controller, musí podporovat rozšiřitelnost systému, plnou nahraditelnost jednotlivých vrstev a možnost sdílení funkcí systému jinými systémy. Pro komunikaci mezi klientskou a aplikační vrstvou bude použit komunikační protokol SOAP.

5.1.1 Třívrstvá architektura

Na diagramu komponent (Obrázek 5.1) je naznačena použitá třívrstvá architektura systému.



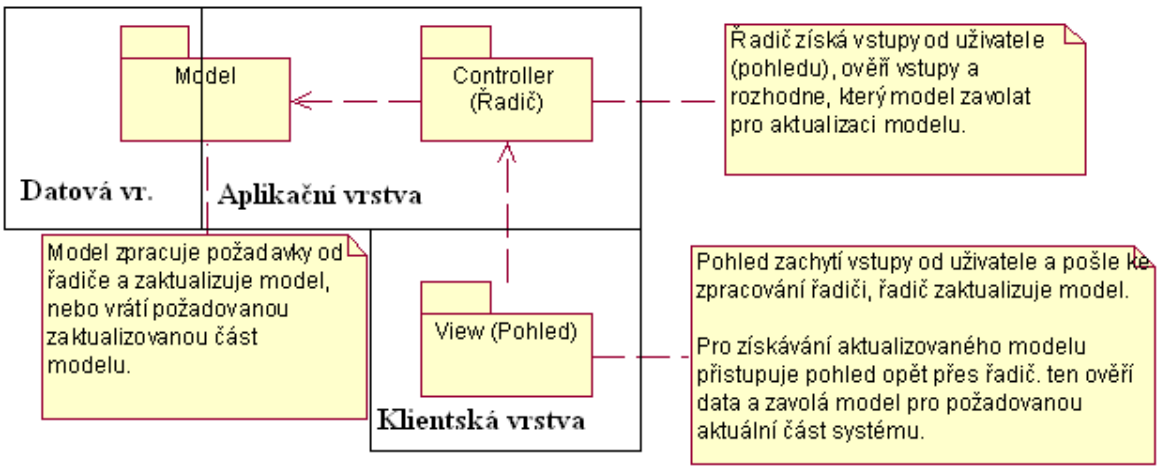
Obrázek 5.1. Třívrstvá architektura systému.

Komunikace mezi datovou vrstvou a aplikační vrstvou bude pomocí jazyka SQL, komunikace mezi aplikační vrstvou a prezentační vrstvou pomocí komunikačního rozhraní SOAP (což umožní plné oddělení zobrazovací vrstvy od vrstvy aplikační.

5.1.2 Zakomponování vzoru MVC do třívrstvé architektury

V kontextu třívrstvé architektury představuje *View*(pohled) prezentační vrstvu, *Controller* je součástí vrstvy aplikační logiky a *Model* je propojení aplikační logiky a datové vrstvy (viz kapitola 3.5.2).

Komunikace mezi Controllerem v aplikační vrstvě a komponentou View v zobrazovací vrstvě probíhá pomocí rozhraní SOAP, což umožní plné oddělení aplikační vrstvy a vrstvy klientské (komponenty View).



Obrázek 5.2. MVC v kontextu třívrstvé architektury.

5.1.2.1 Funkce View (pohledu)

View má 2 hlavní funkce:

- **Získává požadavky od uživatele**, nijak je neanalyzuje a tyto požadavky předá přes komunikační protokol SOAP řadiči. Řadič zaktualizuje model a vrátí View informaci o stavu a případné chyby.
- **Zobrazuje zaktualizovaný model.** View zobrazuje aktualizovaný Model po částech. Potřebuje-li získat zaktualizovaný stav konkrétní části systému, zavolá opět Controller, ten zanalyzuje požadavky, ověří, zda jsou požadavky správné a požádá o vrácení hodnoty modelu. Tuto hodnotu poté pošle zpět klientské vrstvě (View).

5.1.2.2 Funkce Controlleru

Controller má za úkol přijmout požadavky pohledu prostřednictvím protokolu SOAP. Analyzovat je, ověřit strukturu dat a na jejich základě zavolat požadovaný model. Data vrácena modelem potom opět předá klientské vrstvě. Pokud se objeví chyba při ověřování integrity, či při zpracování model, vrátí chybu.

5.1.2.3 Funkce Modelu

Model získá požadavek na zaktualizování modelu, provede jejich aktualizaci (např. případnou aktualizaci dat v datové vrstvě) a vrátí informaci o stavu operace controlleru. Pokud model jen vrací aktualizovaný stav, vrátí tato data controlleru. Ten pošle tato data klientské vrstvě.

5.1.3 Klientská vrstva (View)

Klientská vrstva je totožná s View(pohledem) u architektonického vzoru MVC. V klientské vrstvě bude probíhat zobrazování dat a odesílání požadavků aplikační vrstvě. Klientská vrstva může být navržena nezávisle na funkcích systému (aplikační logice) a má dvě funkce (popsané v kapitole 5.1.2.1):

- Odesílat vstupy od uživatelů aplikační vrstvě.
- Zobrazovat zaktualizovaný model systémů.

Klientská vrstva komunikuje s aplikační vrstvou (Controllerem) pomocí protokolu SOAP.

5.1.3.1 Diagram tříd pro View.

Obrázek 5.3 zachycuje diagram tříd pro komponentu View.

5.1.4 Aplikační vrstva

Aplikační vrstvu tvoří *model* a *controller* z MVC vzoru.

5.1.4.1 Controller

Controller přijímá vstupy uživatele od klientské vrstvy a rozhoduje, jaký model aktualizovat. Vstupy, či požadavky přebírá od klientské vrstvy pomocí protokolu SOAP.

V Controlleru probíhá kontrola správnosti tvarů parametrů a volá se požadovaný model. Model vrátí data, které jsou opět controllerem poslány přes SOAP protokol klientské vrstvě.

Diagram tříd pro komponentu Controller

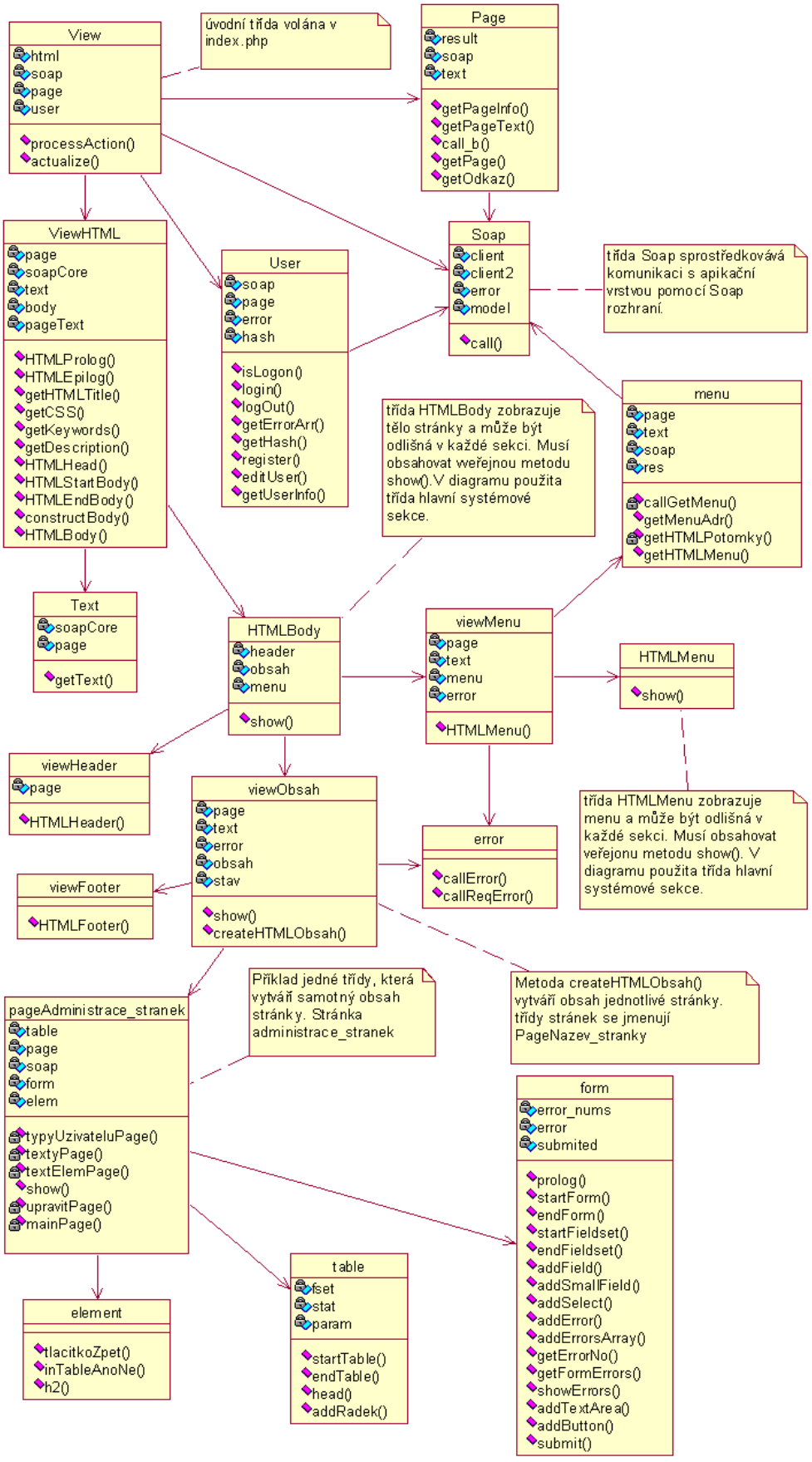
Obrázek 5.4 zachycuje diagram tříd pro komponentu Controller.

5.1.4.2 Model

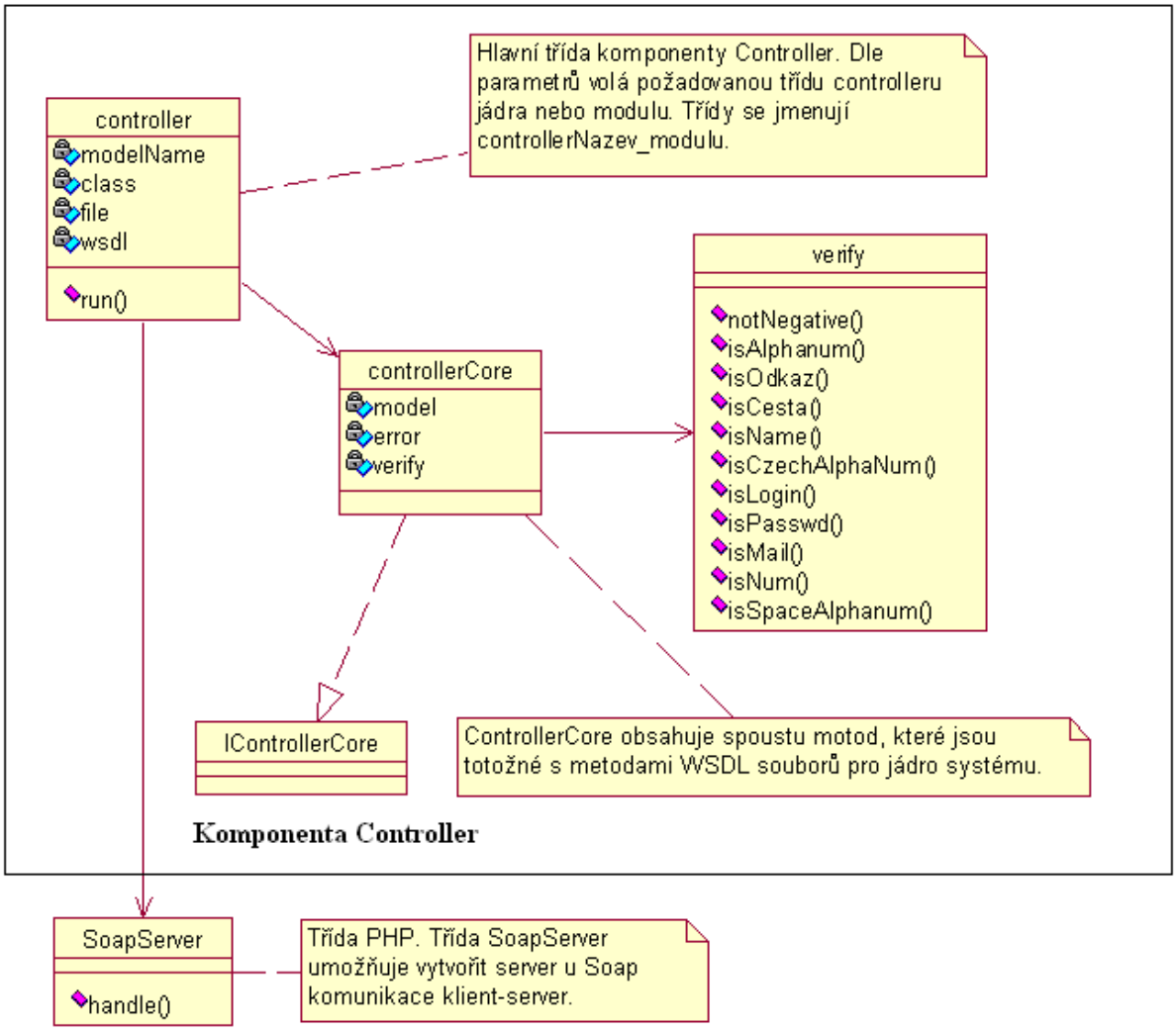
Model tvoří aplikační logiku celé aplikace. Kromě aplikační vrstvy patří do modelu i datová vrstva. Model v aplikační vrstvě získá požadavek na aktualizaci, či data od controlleru. Ověří přístupová práva a bezpečnost a aktualizuje model, popřípadě vrátí data modelu.

Diagram tříd pro komponentu Model

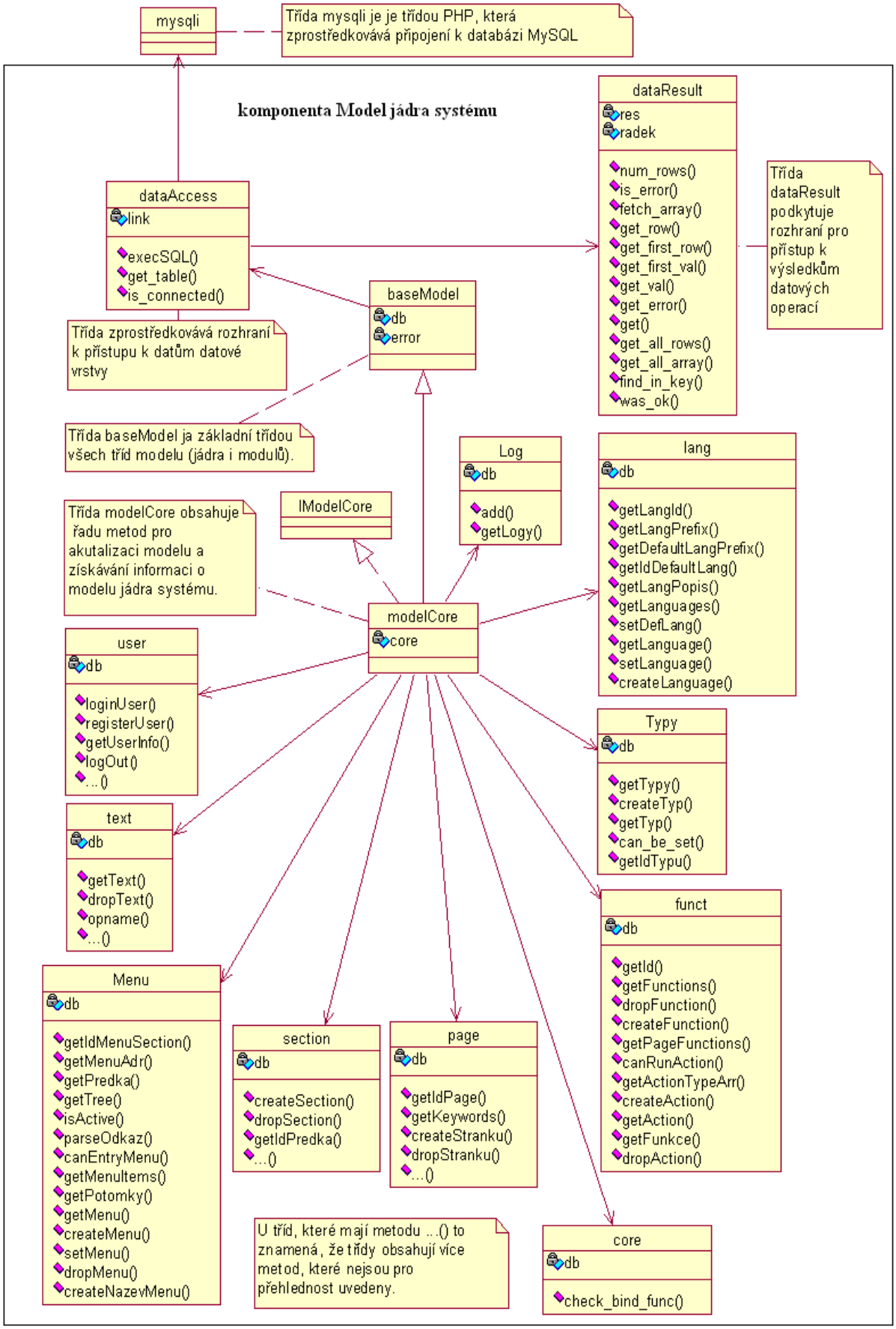
Obrázek 5.5 zachycuje diagram tříd pro komponentu Model.



Obrázek 5.3. Diagram tříd komponenty View.



Obrázek 5.4. Diagram tříd komponenty Controller.



Obrázek 5.5. Diagram tříd pro komponentu Model.

5.1.5 Datová vrstva

Datová vrstva není popsána v rámci MVC vzoru. V tomto systému je využívána jen modelem z MVC, který přistupuje k datům datové vrstvy pomocí jazyka SQL.

Návrh struktury datové vrstvy je jeden ze stěžejních úkolů návrhu architektury systému. Díky tomu, že datová vrstva patří do Modelu v MVC architektonickém vzoru, dotkne se změna datové vrstvy pouze modelu, nikoli klienta či controlleru.

Pro návrh datové vrstvy je použit Diagram tříd. Diagram tříd pro datovou vrstvu je přiložen jako Příloha 1. Diagram tříd pro datovou vrstvu.

5.1.6 Moduly

Moduly rozšiřují funkčnost systému, jeho model. Pomocí modulů lze přidávat do systému nové prvky, jako redakční systém, fotogalerii, kontakty, atd.

Rozdíl mezí sekci a modulem je v cílové vrstvě aplikace. Pokud potřebujeme vytvořit vlastní stránky a využívat již existujících funkcí systému, rozšiřujeme tak zobrazovací vrstvu a použijeme sekci. Pokud jsme vytvořili balíček funkcí, které chceme začlenit do systému, rozšiřujeme tak model a použijeme modul.

5.1.6.1 Přidávání modulů

Aby bylo možno přidávat moduly do architektury MVC musí tyto nové moduly mít:

- **Controller**, ke kterému se bude z klientské vrstvy přistupovat přes SOAP protokol a
- **model**, jež bude aktualizovat svůj stav a vracet controlleru potřebná data.

5.1.7 Sekce

Dle požadavků, musí být systém schopen **vytvářet nové sekce systému**, což znamená rozšiřovat informační prostor systému. Sekce je ucelený balíček stránek, které mají k sobě určitý vztah. Sekce může být např. Kontakty (viz kapitola 5.3) , Pobočka Brno, Pobočka Praha a podobně. V každé z těchto sekcí můžou být další podsekce, nebo konkrétní stránky. Novou sekcí nepřidáváme nové funkce (moduly), ale využíváme stávajících funkcí, jen přidáváme nový kontext a nové informace.

Nové sekce budou vytvářeny hierarchicky, kdy základní sekci tvoří **systémová sekce** nazvaná *systém*, od ní lze vytvořit potomky (např. *intraweb*) a té opět potomky(např. administrace stránek, sekcí, atd.). Tento jádro systému bude umožňovat generování titulku stránky, klíčových slov apod., ale také dědičnost práv v sekcích (víz níže).

V jádru systému bude tato základní hierarchie sekcí:

- **Systém** – základní systémová sekce, předek všech ostatních.
    - **Intraweb.** Sekce určena jen přihlášeným uživatelům.

- **Administrace sekcí. S**ekce, jenž umožňuje přidávat nové sekce a administrovat sekce stávající.
- **Administrace jazyků.** Správa jazyků systému.
- **Administrace menu.** Správa menu.
- **Administrace funkcí.** Umožňuje správu funkcí, které se kvůli bezpečnosti vážou na použití na určité stránce, či k určitému typu uživatele (podrobněji vysvětleno v kapitole 5.2).
- **Administrace práv.** Administrace přístupových práv k sekcím.
- **Administrace typů.** Administrace typů uživatelů.
- **Administrace stránek.** Správa stránek systému.
- **Administrace uživatelů.** Správa uživatelů, potvrzování, nastavování typu uživatele.
- **Administrace textů.** Správa a administrace textů závislých na jazyku (vícejazyčných textů).
- **Administrace uživatele.** Editace registračních hodnot přihlášeného uživatele.
- **Administrace logů.** Administrace logů uživatelů.
- **Nápověda.** Integrovaná nápověda systému.

U administračních sekcí bude kladen důraz na **vysvětlení jednotlivých pojmů nastavení**. Je důležité i bez přítomnosti autora webu, či velkého manuálu, aby byla **možná snadná údržba a administrace systému**, což vychází z požadavků kladených na systém.

Dále je požadavek na **možnost modifikovat menu v každé sekci**, což znamená možnost vytvořit vlastní strukturu menu v každé sekci, které se v dané sekci zobrazí. Mezi další požadavky kladenými na sekce systému patří (viz kapitola 4.2):

- **Vlastní vzhled každé sekce v systému**, včetně možnosti změny návrhu stránky.
- **Vlastní CSS styl každé sekce.**
- **Vlastní Titulek, klíčová slova a popis ke každé sekci.**
- **Přístupová práva.** Každá sekce přiřazuje každému uživateli typ uživatele a tím i práva přístupu. Více v kapitole 5.2.1.

Tyto a další požadavky ovlivní hlavně návrh klientské vrstvy a datové vrstvy (podpora těchto požadavků modelem). Na úrovni klientské vrstvy je třeba zejména navrhnout:

- Adresářovou strukturu,
- formát stránek,
- formát menu.

Tyto vlastnosti jsou přímo spjaty s konkrétní implementací klientské vrstvy, budou proto probrány v kapitole 6. Implementace.

Návrh datové vrstvy je popsán v kapitole 5.1.5.

5.1.8 Stránky

Každá stránka v systému musí patřit do určité sekce.

Stránky budou mít možnost definovat vlastní titulek, popis a klíčová slova v libovolném z dostupných jazyků.

5.1.8.1 Typ stránky

Stránkám lze přidělit typ stránky, který ovlivňuje přístup systému ke stránce. Na základě požadavků a analýzy byly navrženy tyto základní typy stránek:

- **Default.** Základní stránka systému. Na tuto stránku systém přesměruje, nenajde-li požadovanou stránku.
- **Error.** Chybová stránka v rámci sekce, na tuto stránku systém přesměruje při chybě. Pro tuto stránku není potřeba nastavovat práva přístupu.
- **Noentry.** Na tuto stránku v rámci sekce či sekcí předků systém přesměruje, nemá-li uživatel právo přístupu pro danou stránku. Pro tuto stránku není potřeba nastavovat práva přístupu.
- **Multi.** Pokud je stránka typu 'multi', znamená to, že přístup na tuto stránku není vázán na její sekci, ale to, že pokud má uživatel alespoň v jedné sekci právo přístupu na stránku takové, jako je právo přístupu (stejný typ uživatele) na tuto stránku je mu přístup povolen.
- **LinkIDSTRANKY**. Pokud je stránka typu 'link', znamená to, že stránka fyzicky v systému neexistuje, ale ukazuje na stránku daného ID stránky IDSTRANKY. Link lze využít, máme-li jednu stránku, kterou chceme zobrazit ve více sekcích (např. systém a intraweb). Př. 'link24' ukazuje na stránku s ID 24.

5.2 Bezpečnost systému

Bezpečnost systému lze rozdělit (osobní rozdělení autora) na bezpečnost vnitřní a vnější.

Vnitřní bezpečnost představuje klasickou ochranu dat, omezení přístupu k funkcím systému apod. Vnitřní bezpečnost systému řeší přístupová práva k jednotlivým částem systému.

Vnější bezpečnost je bezpečnost oproti útoku z vně systému, například heckera, či neoprávněnému vstupu uživatele.

Oba druhy bezpečnosti jsou navzájem propojené a jen jako celek tvoří systém zabezpečený, bohužel, nikdy to nebude dokonalé!

5.2.1 Přístupová práva k systému

Dle požadavků kladených na systém, byl navržen systém, který umožňuje definovat právo přístupu na stránky každé sekce. Systém umožňuje pro každou sekci a každého uživatele definovat tzv. *typ uživatele*.

5.2.1.1 Typ uživatele

Typ uživatele je zařazení uživatele do role, či skupiny v rámci dané sekce. V základu systému existují tyto typy uživatelů:

- **Obyčejný uživatel.** Obyčejný uživatel je úspěšně přihlášený uživatel nejnižší úrovně.
- **Kompetentní uživatel.** Kompetentní uživatel bude mít v jádře systému všechna práva obyčejného uživatele a bude možnost mít některá práva navíc.
- **Administrátor.** Administrátor bude mít nejvyšší právo v systému.
- **Nepřihlášený uživatel.** Uživatel, jenž není přihlášený.

Systém však už nedefinuje náplň jednotlivých funkcí. Jaký typ uživatele má na danou stránku právo přístupu už je věc aplikace systému a nastavení přístupu na stránku jednotlivým typům.

Tyto základní typy se řídí těmito pravidly:

1. Má-li k dané stránce přístup typ *Obyčejný uživatel*, má k dané stránce přístup i typ *Kompetentní uživatel.*
2. Má-li k dané stránce přístup typ *Kompetentní uživatel,* má k dané stránce přístup i typ *Administrátor*
3. *Nepřihlášený uživatel* je speciální typ virtuálního uživatele, jenž nahrazuje všechny uživatele, kteří nejsou přihlášení do systému. Neplatí, že *Obyčejný uživatel* má právo přístupu všude tam, kde *Nepřihlášený uživatel.*

Do systému lze samozřejmě přidávat nové typy uživatelů a definovat tak např. speciální právo přístupu pro určité osoby.

5.2.1.2 Systém přístupu na stránku

Přístup na stránku se řídí následujícími pravidly:

1. Každá stránka náleží určité sekci (min. základní sekce *systém*)
2. V Každé sekci je dáno, jaký typ uživatele je konkrétní uživatel pro tuto sekci (jakou hraje roli v dané sekci).
3. Každé stránce je dáno, který typ uživatele má na danou stránku právo vstupu.
4. Systém při každém pokusu o přístup na stránku:
    a. Ověří, jaký typ uživatele je daný uživatel pro danou sekci,

b. ověří, jakým typům uživatelů umožňuje stránka přístup,
c. porovná výsledky a zjistí, zda má či nemá uživatel na danou stránku přístup.

5.2.1.3 Pravidla pro odvozování typů uživatelů v sekcích

Systém při potvrzení nově registrovaného uživatele automaticky přiřadí tomuto uživateli typ *Obyčejný uživatel* pro základní systémovou sekci *systém*.

Všechny ostatní sekce jsou potomci sekce *systém*, čili získávají automatický stejný typ daného uživatele pro tuto sekci, jaký má předek této sekce.

V sekci lze definovat vlastní typ uživatele (např. v sekci *Intraweb* nastavíme typ uživatele na typ *Administrátor* a všechny následníci sekce *Intraweb* již budou mít daného uživatele jako typ *Administrátor.)*.

Algoritmus zjišťování typu uživatele je navržen následovně:

1. Zjistíme, zda daný uživatel v dané sekci má definován typ uživatele.
2. Pokud má typ definován, použijeme tento typ.
3. Pokud nemá typ definován, zjistíme předka dané sekce a pokračujeme bodem 1.

5.2.2 Bezpečnostní prvky plynoucí z třívrstvé architektury

Systém je navržen jako třívrstvý. Každá jeho část je nezávislá a může běžet teoreticky na jiném serveru, či zařízení. Jednotlivé části mezi sebou komunikují dle daných protokolů. Nejmarkantnější rozdíl je v komunikaci klientské a aplikační vrstvy.

Systémy určené přímo pro webové prohlížeče používají jako klienta internetový prohlížeč (např. Firefox). Informace putující od serveru jsou již zpracované skripty v HTML podobě. Potenciální nebezpečí je tak v odesílaných datech (hesla, apod.).

Systém založený na nezávislé klientské vrstvě má však riziko navíc. Systém se v praxi připojuje přes SOAP protokol ke vzdálenému poskytovateli webových služeb. Musí tu však být určité mechanismy, které zabezpečí přístup k těmto webovým službám. Uvedeme příklad. Změna údajů o uživateli probíhá tak, že se zavolá vzdálená služba (přes controller) s názvem `editUser` s patřičnými parametry. Bez bezpečnostních mechanizmů, tuto službu by mohl zavolat kdokoli, kdo:

- zná webovou adresu služby a
- dodrží patřičné parametry (zná WSDL soubor).

Proto byly do systému implementovány mechanizmy, které se snaží riziko nepovoleného přístupu eliminovat. Mezi tyto mechanizmy patří:

- Vygenerování HASH řetězce pro přihlášeného uživatele na straně aplikační logiky.
- Vázaní určitých funkcí systému na konkrétní stránku.
- Vázaní určitých funkcí systému na konkrétní typ uživatele.

5.2.2.1 Přihlášení uživatele a HASH číslo uživatele

Nestačí identifikovat přihlášeného uživatele jen na základě jeho ID. Pak by kdokoli, kdo zná adresu přístupu, se mohl tvářit jako uživatel s daným ID a mít tak např. přístupová práva administrátora.

Tomuto lze zabráním použití Hash řetězce vygenerovaného na serveru, místo ID uživatele. Klient si drží místo ID jen Hash kód, který při každém požadavku na webovou službu, která je vázána přístupovými právy (zobrazení stránek, administrace, apod.), pošle aplikační logice místo svého ID.

Ta ověří na základě tabulky přihlášených, zda je uživatel s daným hash řetězcem přihlášen.

Řešení má nevýhodu v ověření hash řetězce a najití ID při každém požadavku na zabezpečenou funkci. Při generování rozsáhlejší např. administrátorské stránky, se tak může dít i několikrát za stránku.

5.2.2.2 Vázání funkcí na konkrétní stránku

Dalším bezpečnostním prvkem je vázání určitých funkcí na konkrétní stránku. Představme si, že máme systém, který sice ověří uživatele, ověří přístup na stránku, ale ostatní funkce již ověřovat nebude. Takový systém spoléhá na to, že klient vždy umístí správné funkce na správnou stránku.

Systém už ale není zabezpečen proti útoku. Stačí znát WSDL soubor a adresu controlleru a můžeme si napsat vlastního ‘HACK‘ klienta třeba v JAVĚ a mít tak pod kontrolou celý systém.

Potřebujeme tedy zabezpečit všechny důležité funkce. Funkci, která vykonává (nebo zpřístupňuje) určitou utajenou skutečnost (jako seznam uživatelů, úprava uživatele, administrátorské funkce, apod.), definujeme jako **vázanou na určitou stránku**.

Víme například, že funkci *přidání jazyků do systému* lze použit jen v souvislosti se stránkou, která poskytuje administrátorská rozhraní pro přidávání jazyků do systému. Vytvoříme vazbu této funkce na danou stránku. Při ověřování oprávnění užití dané funkce se pak postupuje následovně:

1. Ověří se, zda daný uživatel (dle jeho HASH řetězce) má přístup na stránku, čili:
    a. je-li pro danou sekci takový typ, který má na stránku přístup,
    b. nebo, je-li typ stránky *multi* (viz kapitola 5.1.8.1), zjistíme, zda je uživatel aspoň jedné sekci v celém systému takový typ, který má na stránku přístup.
2. Ověří se, zda daná funkce smí být spuštěna na dané stránce (je s ní svázána).
3. Pokud ověření v pořádku, vykoná model požadovanou akci (zavolá metodu), jinak nedovolí spuštění metody a nahlásí chybu.

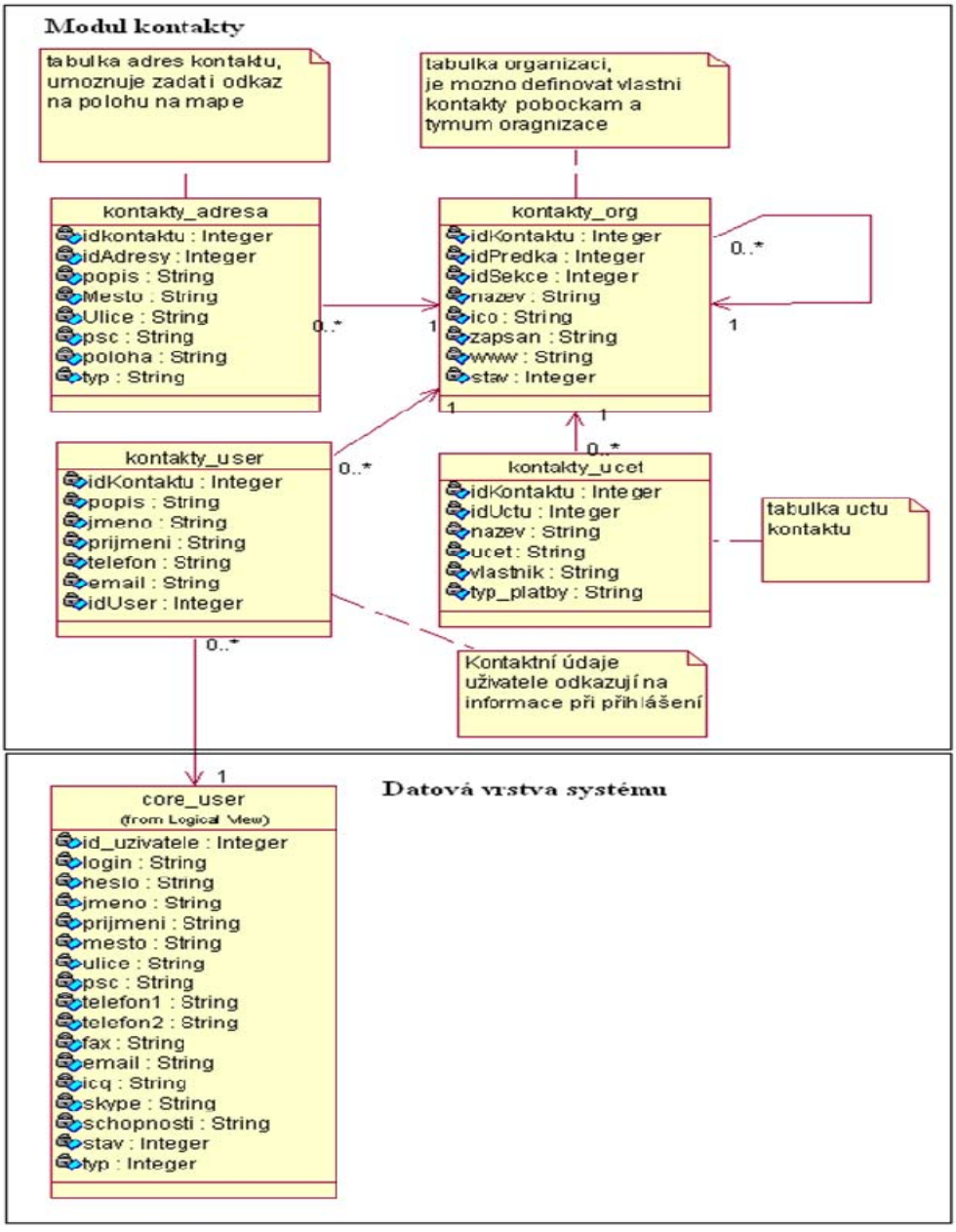
5.2.2.3 Vázání funkcí na typ uživatele

Obdobné, jako vázání funkcí na stránku, s tím rozdílem, že funkci nesvážeme s určitou stránkou ale s určitým typem uživatele. Typické použití akce *logOut* (odhlášení ze systému) není vázána na žádnou ze stránek, ale na přihlášeného uživatele, čili na typ *Obyčejný uživatel* (tím i typ *Kompetentní uživatel* a *Administrátor*).

5.3 Modul kontakty

Modul kontakty je modul určený k vytváření a zprávě kontaktů organizace, jejích poboček a týmů.

Na základě požadavků kladených na modul kontakty (viz kapitola 4.3) byla navržena datová struktura datové vrstvy modulu Kontakty. Strukturu zachycuje diagram tříd (obrázekObrázek 5.6).



Obrázek 5.6. Diagram tříd pro datovou vrstvu modulu Kontakty.

6 Implementace

Tato kapitola se zaměřuje na implementaci navrženého systému. Bude popsána implementace nejdůležitějších, nebo implementačně zajímavých částí tak, aby z popisu byla zřejmá celková implementace systému a jeho jednotlivých prvků.

U některých řešení budou zmíněny výhody či nevýhody tohoto řešení, problémy při implementaci apod.

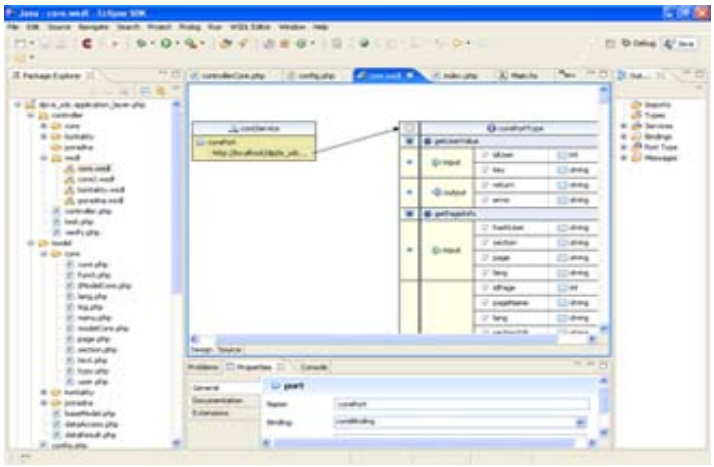
Nejprve bude zmíněno vývojové prostředí implementace, konvence použité v této kapitole, nasazení systému v praxi, princip fungování systému, princip zobrazení stránek, menu, části view, controller a model. V závěru kapitoly bude popsána implementace bezpečnostních prvků, komunikace pomocí SOAP protokolu a implementace modulu Kontakty a jeho přidání do systému.

6.1 Vývojové prostředí

Celý systém byl implementován za pomocí vývojového prostředí *Eclipse SDK 3.2.2* (Obrázek 6.1). Eclipse je vývojové prostředí určené původně pro jazyk JAVA (v JAVĚ kompletně napsáno). Je k dispozici jako Open Source projekt pro operační systémy Linux i Windows. Stáhnout jej lze zdarma ze stránek projektu (http://www.eclipse.org).

Eclipse je plně funkční vývojové prostředí s možností zvýrazňování kódu, zobrazování tříd, debuggingu, break pointu, krokování, inteligentního vyhledávání, skoky a dalšími funkcemi, srovnatelné s komerčními vývojovými prostředími.

Síla prostředí Eclipse spočívá v možnosti přidávat plug-iny a rozšiřovat tak podporu prostředí o nové jazyky a technologie. Lze tak bez problému doplnit podporu jazyků PHP, XHTML, XML, CSS, ale i poloautomatický grafický generátor souboru WSDL, nebo třeba podporu pro HASKELL, či PROLOG.



Obrázek 6.1. Vývojové prostředí Eclipse s plug-inem pro editaci souborů WSDL.

6.2 Základní konvence implementace

V této podkapitole budou vysvětleny základní konvence při implementaci systému. Nejprve jsou popsány pravidla pro tvorbu tříd, dále pravidla pro tvorbu souborů s obsahem stránek a pravidla pro tvorbu sekcí.

6.2.1 Třídy a soubory

Pro soubory a třídy jsou v systému použita následující pravidla (příklad **Chyba! Nenalezen zdroj odkazů.**):

- Každá třída je definována v samostatném souboru.
- Jméno třídy, nebo rozhraní začíná velkým písmenem, další slovo názvu opět velkým písmenem. Např. `ModelCore`.
- Znak '_' pro oddělení víceslovných názvů v názvu tříd a rozhraní není použit.
- Rozhraní i třída implementující toto rozhraní jsou v samostatném souboru.
- Název souboru odpovídá názvu třídy s přidanou příponou .php. Např. *modelCore.php.*
- Název souboru začíná malým písmenem.

Rozhraní `IControllerCore` v souboru *IcontrollerCore.php*:

```
interface IControllerCore {
…
}
```

Implementace rozhraní `IControllerCore` třídou `ControllerCore` v souboru *controllerCore.php*:

```
Soubor controllerCore.php:
//rozhrani
include_once('IControllerCore.php');
class ControllerCore implements IControllerCore {
…
}
```

Příklad 6.1. Názvy souborů a tříd.

6.2.2 Sekce systému

6.2.2.1 Vlastní menu

Systém je navržen tak, že umožňuje mít v každé sekci vlastní menu. Pokud si přejete mít v sekci vlastní menu, s vlastními odkazy a mít toto menu nezávislé na menu definované v předcích, je nutné dodržet následující:

- Mít v kořenovém adresáři sekce soubor *HTMLMenu.php*.
- V souboru *HTMLMenu.php* třídu `HTMLMenu`.

- Ve třídě *HTMLMenu* konstruktor s parametry `$page`**,** `$text` **a** `$menu`, kterým při vytváření objektu `HTMLMenu` systém přiřadí odkazy na objekty tupů `Page`, `Text`, `Menu`.
- Ve třídě `HTMLMenu` metodu `show()`, která zobrazí menu.

6.2.2.2 Vlastní design

Systém umožňuje v každé sekci definovat vlastní design stránky. Lze vytvořit sekci nejen s vlastním CSS, ale dokonce sekci s kompletně definovanou strukturou stránky, což znamená, že přestože bude sekce součástí systému, může se tvářit jako úplně jiný web a přitom mít k dispozici všechny výhody systému. Pro vlastní design je nutné:

- V kořenovém adresáři sekce soubor *HTMLBody.php*.
- V souboru *HTMLBody.php* třídu `HTMLBody`.
- Ve třídě `HTMLBody` konstruktor třídy se 2 parametry `$page`**,** `$text`, které odkazují na třídy `Page`, a `Text`.
- Ve třídě `HTMLBody` metodu `show()`, která zobrazí obsah HTML tagu `body`.

6.2.3 Soubory s obsahy stránek ve View

Soubory stránek ve View jsou tvořeny prefixem *page* dále názvem stránky a příponou *.php*. Např stránka pro administraci stránek ponese název *pageAdministrace_stranek.php*.

Obsah souborů stránek musí dodržovat následující pravidla:

- Je tvořen třídou jmenující se stejně, jako název souboru, jen s velkým počátečním písmenem. Např. `PageAdministrace_stranek`.
- Konstruktor stránky musí obsahovat dva parametry:
  - `$page` – systém vytvoří odkaz na objekt třídy `Page`,
  - `$text` – systém vytvoří odkaz na objekt třídy `Text`.
- Pokud stránka obsahuje formuláře, či prvky, které mění model systému, je potřeba tyto prvky umístit do konstruktoru stránek.
- Třída stránky musí obsahovat metodu `show()`, která je volána vždy při požadavku na zobrazení obsahu dané stránky.

6.3 Principy fungování systému

V této kapitole budou popsány principy fungování a komunikace systému. Bude popsán proces zpracování požadavku uživatele od získání vstupu uživatele až po aktualizaci modelu systému.

6.3.1 Vstupy od uživatele

Uživatel zadává své požadavky prostřednictvím internetového prohlížeče. Vstupy od uživatele se uskutečňují prostřednictvím:

- **POST požadavků.** Jde většinou **o požadavek na aktualizaci modelu**. Např. vyplnění formuláře.
- **GET požadavků.** Jde většinou **o požadavek na získání dat pro zobrazení modelu**. Např. kliknutí na odkaz, položku menu, zadání adresy do prohlížeče apod.

Všechny požadavky zadané přes formulář jsou zpracovány metodou POST (více o zpracování a vytváření formulářů v kapitole 6.6.1), ostatní požadavky jsou předávány metodou GET.

6.3.2 Zpracování požadavku

Požadavky můžou být buď formou GET, nebo POST požadavků. Výpis 6.1 ukazuje metodu `actualize()` třídy `View`, která aktualizuje model a dále aktualizovaný model zobrazuje. Aktualizace modelu probíhá v metodách `processAction()` a konstruktoru stránek v metodě `contructBody()`. V Dalších metodách třídy `HTMLView` se postupně vypisuje obsah aktualizovaného modelu (stránky).

```
//nejprve zpracujeme pripadne akce
$this->processAction($_GET['action'], $_POST['action']);
//informace o strance
$this->page->getPageInfo();
// reakce na formular drive nez se zacne zobrazovat obsah,
$this->html->constructBody();
//ziska textove informace o strance (titulek, klicova slova,popis)
$this->page->getPageText();
//vypiseme stranky
$this->html->HTMLprolog();
$this->html->HTMLhead($this->page->text);
$this->html->HTMLStartBody();
$this->html->HTMLBody();
$this->html->HTMLEndBody();
$this->html->HTMLepilog();
```

Výpis 6.1. Metoda actualize() třídy View.

Princip generování stránky:

- Nejprve z modelu získáme informace o stránce,
- vygenerujeme, menu, hlavičku,
- postupně generujeme obsah stránky, každý dotaz na stav modelu se zpracovává zvlášť,
- vygenerujeme zápatí a konec stránky.

6.3.2.1 Zpracování požadavku GET

Pokud uživatel žádá prostřednictvím GET parametrů např. o zobrazení určité stránky, není co aktualizovat. Po vytvoření objektu stránky se začne generovat požadovaná stránka.

6.3.2.2 Zpracování požadavku POST

Pokud uživatel vyplnil nějaký formulář a jeho odesláním (metoda POST) tak požaduje o změnu modelu systému, je postup následující:

- Nejprve v konstruktoru stránky odeslán požadavek na aktualizaci modelu,
- controller příjme požadavek, ověří vstupní data a odešle požadavek na aktualizaci modelu,
- model aktualizuje systém,
- model vrátí informaci o stavu a případných chybách,
- pokud existují chyby, předají se formuláři pro výpis chyb,
- pokud neexistují chyby, přejde se na generování stránky.

6.3.2.3 Zpracování požadavku na aktualizaci modelu (POST) v konstruktoru stránky

Výpis 6.2 ukazuje zpracování požadavku (formuláře) na aktualizaci textu stránky (titulek, klíčová slova, popis) dle jazyka z konstruktoru stránky *administrace_stranek*.

```
//ověřeni zda byl odeslan pozadovany formular
if (Form::submit('novy_text_stranky')) {

  $idStranky   = $_POST['id_stranky'];
  $idLang      = $_POST['id_lang'];
  $titulek     = $_POST['titulek'];
  $kl_slova    = $_POST['kl_slova'];
  $popis       = $_POST['popis'];
  //odeslani pozadavku pres SOAP
  $res = $this->page->call_b('createTextStranky',
  $idStranky, $idLang, $titulek, $kl_slova, $popis);
  if ($res['createError']) {
     //zpracovani pripadnych chyb
    Form::addErrorsArray($res['createError']);
  }
  else {
    unset($_POST);
  }
}
```

Výpis 6.2. Zpracování formuláře na přidání textu stránky.

Nejprve se ověří, zda byl odeslán požadovaný formulář, poté se volá metoda `call_b` ze třídy `Page`, která vytvoří požadavek na zavolaní potřebné vázané funkce (viz kapitola 5.2.2) pro třídu `Soap`. Ta vrátí informace do proměnné `$res`, ověří se, jestli nenastaly chyby při zpracování a případné chyby předá formuláři k výpisu.

6.3.3 Volání funkcí aplikační vrstvy z klientské vrstvy

Poté, co View získalo požadavek od uživatele na změnu modelu (byl odeslán formulář), v konstruktoru dané stránky vyl tento požadavek zpracován a byla zavolána funkce aplikační vrstvy s požadavkem na aktualizaci systému.

Systém umožňuje volání normálních funkcí (přes třídu `Soap`) a funkcí vázaných na stránku (kapitola 5.2.2), které zprostředkovává třída `Page`.

Metoda `call_b` třídy `Page` usnadňuje programátorovi práci a doplňuje povinné prvky každé vázané funkce jako:

- Přidání prefixu *b_* k vázané funkci,
- přidání *Hash řetězce* uživatele,
- přidání *id stránky*,
- zavolání metody `call` třídy `Soap`.

Detailním popisem třídy Soap a komunikace přes SOAP protokol se zabývá kapitola 6.10.

6.3.4 Zpracování požadavku controllerem

Klientská vrstva odeslala požadavek vrstvě aplikační. Zde tento požadavek odchytí controller. Jinak řečeno, view zavolalo přes SOAP protokol přímo konkrétní metodu daného controlleru. Každý modul má vlastní controller. Potřebný controller se vybírá automaticky z cesty portu definované v WSDL souboru. Více o popisu komunikace pomocí SOAP protokolu se zabývá kapitola 6.10.

Daná funkce controlleru nejprve ověří (za využití třídy `Verify`), zda dané parametry mají požadovaný tvar a zavolá metodu modelu, která aktualizuje model a vrátí informaci o stavu a případné chybové hlášení. Výpis 6.3 ukazuje metodu `b_getLanguages()`, třídy controlleru jádra `ControllerCore`. Prefix `b_` určuje, že je daná funkce vázaná na stránku (viz kapitola 5.2.2.2).

```
public function b_getLanguages($hashUser, $idPage) {
    $res['error']     = '';
    if (($this->verify->is_alphanum($hashUser, 1))
    && ($this->verify->not_negative($idPage))
    ) {
      $res = $this->model->b_getLanguages($hashUser, $idPage);
    } else {
      $res['error'] = $this->error->callError(get_class($this), "Neplatné
      vstupní parametry.");
    }
    return $res;
}
```

Výpis 6.3. Metoda b_getLanguages() třídy ControllerCore.

Více o implementaci controlleru v kapitole 6.7.

6.3.5 Zpracování požadavku modelem

Úkolem modelu je vykonat aktualizaci modelu, nebo vrátit požadovaná data. Nejprve se ve třídě modelu jádra `ModelCore` ověří případná práva přístupu, převedou názvy na ID objektů atd. a poté je volána konkrétní metoda dané třídy modelu, která požadovanou operaci vykoná.

Výpis 6.4 ukazuje metodu `b_getLanguages()`. Nejprve se ověří, zda smí daný uživatel vstoupit na danou stránku, dále, zda se smí daná funkce spustit z dané stránky a pokud ano, zavolá se metoda `getLanguages()` třídy `Lang`, která vrátí informace o dostupných jazycích systému. Tyto informace jsou předány, jako výstup metody. Pokud nastane chyba, je tato chyba popsána a vrácena zpět controlleru, který ji pošle klientovi.

```
  public function b_getLanguages($hashUser, $idPage) {
    $user = new User($this->db);
    $canEntry = $user->canEntryHashUser($hashUser, $idPage);
    if ($canEntry) {
      // overime, zda ma dana stranka zaregistrovanou pozadovanou sluzbu
      $page = new Page($this->db);
      $canRun = $page->canRun('getLanguages', $idPage);
      if ($canRun) {
        //spustime vlastni matodu
        $lang = new Lang($this->db);
        $ret['langItems'] = $lang->getLanguages();
      } else {
            $ret['error']   = $this->error->callError(get_class($this),
            $page->get_error('E_NO_RUN'));
      }
    } else {
      $ret['error']   = $this->error->callError(get_class($this),
      $user->get_error('E_NO_ENTRY'));
    }
    return $ret;
  }
```

Výpis 6.4. Metoda b_getLanguages() třídy ModelCore.

6.3.5.1 Zpracování požadavku konkrétní třídou modelu.

Posledním krokem ve vývoji požadavku je přímá aktualizace modelu, či získání dat konkrétní třídou modelu. Tyto třídy už neověřují správnost dat, ani přístupy, jen vykonají potřebnou akci. Výpis 6.5 ukazuje metodu getLanguages třídy Lang, která získá z datové vrstvy informace o jazycích a tyto inforamace vrátí jako svou návratovou hodnotu. Více o modelu a datové vrstvě v kapitole 6.8.

```
public function getLanguages() {
  $sql = 'SELECT * FROM '.$this->db->get_table('core_language')
  ." order by id_language";
  $res = $this->db->execSQL($sql);
  $ret = $res->get_all_rows();
  return $ret;
}
```

Výpis 6.5. Metoda getLanguages() třídy Lang.

6.4 Zobrazení stránek systému

V této kapitole projdeme krok za krokem procesem 'vývoje' požadavku na získání obsahu požadované stránky od zadání požadavku uživatelem až po získání informace o stránce a zobrazení stránky.

6.4.1 Získání vstupu od uživatele

Uživatel si přeje administrovat menu systému, klikne proto na odkaz *ADMINISTRACE MENU* v menu systému, nebo zadá adresu do internetového prohlížeče. Pokud náš systém bude běžet na adrese http://www.system.cz, bude to http://www.system.cz/index.php?page=administrace_menu.

Na základě této žádosti se přistoupí k souboru *index.php* na klientské vrstvě (ve view).

```
session_start();
include ('view/view.php');
$view = new View();
$view->actualize();
```

Výpis 6.6. Index.php.

Z Výpis 6.6 souboru *index.php* je vidět, že *index.php* kromě aktivace *session*, (využita na udržování hash uživatele), obsahuje jen vytvoření `View` a jeho aktualizaci metodou `actulize()`.

6.4.2 Aktualizace View

Aktualizaci `View` (Výpis 6.7) zajišťuje metoda `actualize()` která:

- Nejprve zpracuje požadavky uživatele na aktualizaci modelu, které jsou vázané na typ uživatele, metoda `processAction()`. Více viz kapitola 5.2.2.3.
- Dále získá informace o stránce pomocí metody `getPageInfo()` třídy `Page`.
- Následně se vykonají ostatní požadavky uživatele na aktualizaci systému (v konstruktoru stránek) a vytvoří se objekt těla stránky (`constructBody`).
- Poté postupně vygeneruje HTML kód obsahu celé stránky i menu metodami třídy `ViewHTML` (objekt `Huml`).

Dodržuje se tak koncept MVC vzoru, kdy nejprve aktualizujeme model a tento model se následně zobrazí.

```
public function actualize() {
    //nejprve zpracujeme pripadne akce
    $this->processAction($_GET['action'], $_POST['action']);
    //informace o strance
    $this->page->getPageInfo();
    // reakce na formular drive nez se zacne zobrazovat obsah,
```

```
    $this->html->constructBody();
    //ziska textove informace o strance (titulek, klicova slova,popis)
    $this->page->getPageText();
    //vypiseme stranky
    $this->html->HTMLprolog();
    $this->html->HTMLhead($this->page->text);
    $this->html->HTMLStartBody();
    $this->html->HTMLBody();
    $this->html->HTMLEndBody();
    $this->html->HTMLepilog();
}
```

Výpis 6.7. Aktualizace View.

6.4.3 Získávání informací o stránce

Třída `Page` implementuje metody pro práci se stránkou ve View. Metoda `getPageInfo()` zavolá aplikační vrstvu a požádá o navrácení základních informací o stránce (viz Výpis 6.8) dle těchto kritérií:

- **Hash uživatele.** Jedinečný řetězec přihlášeného uživatele.
- **Sekce stránky**. Nepovinný, pouze pokud existuje více stránek daného jména. V menu se generuje automaticky.
- **Název stránky.** Název požadované stránky.
- **Jazyk.** Prefix jazyka.

```
public function getPageInfo() {
  //vrati informace o strance
  $this->res = $this->soap->call('getPageInfo', $_SESSION['hash_user'],
$_GET['section'], $_GET['page'], $_SESSION['lang']);
}
```

Výpis 6.8. Metoda getPageInfo() třídy Page.

Soap klient ve View zavolá požadovanou metodu controlleru (viz kapitola 6.10). Ta ověří data a zavolá příslušnou metodu modelu. Metoda `getPageInfo()` z třídy `ModelCore` (třída modelu jádra systému) využívá následujícího algoritmu:

1. Získá ID uživatele dle zadaného hash řetězce.
2. Zjistí ID stránky na základě jména stránky a sekce, což znamená:
    - Pokud stránka s daným názvem a sekcí existuje, vrátí ID této stránky.
    - Pokud stránku nenajde, vrátí ID chybové stránky (typ *error*).
    - Pokud je název stránky prázdný, vrátí defaultní stránku (typ *default)* pro danou sekci.
    - Pokud defaultní stránka pro danou sekci neexistuje, vrací defaultní stránku sekce předka, kde je defaultní stránka definovaná.
3. Pokud je ID stránky shodné s ID chybové stránky, má právo přístupu každý uživatel.
4. Pokud je ID stránky různé od ID chybové stránky, ověří se právo přístupu uživatele na stránku (viz kapitola 6.9.2.2).

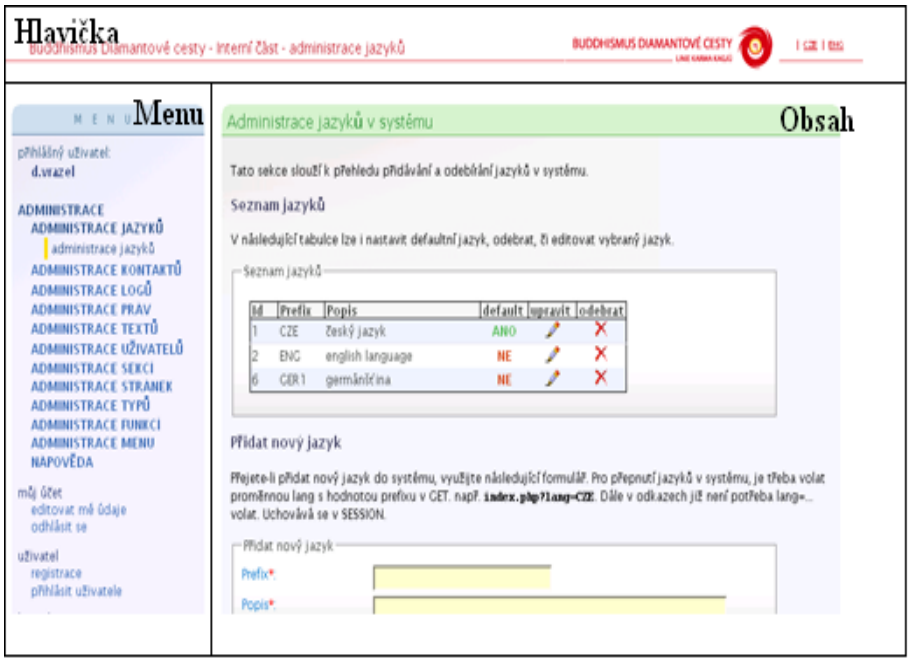
5. Pokud uživatel nemá přístup na stránku, získá se ID stránky, kde se nachází např. hlášení o omezeném přístupu, či přihlašovací formulář, atd. (typ stránky *noentry*).
6. Na základě ID stránky a jazyka vrátí metoda `getPageInfo` následující informace:
    - Id stránky,
    - název stránky,
    - jazyk,
    - cesta k adresáři sekce,
    - název vlastního souboru CSS sekce kde je definován vlastní design,
    - cesta k adresáři se sekcí s vlastním designem,
    - pokud je stránka typu *link*, odkaz na skutečnou stránku.

6.4.4 Vytvoření stránky

Pokud máme v objektu třídy `Page` základní informace o stránce získané metodou `getPageInfo()`, můžeme konstruovat jednotlivé části stránky. Nejprve se vytvoří objekt stránky, kde se v konstruktoru zpracují požadavky (metoda `ConstructBody()`), poté se vygeneruje menu a následně obsah stránek.

Generování stránky i menu se děje automaticky na základě informací o stránce získaných z metody `getPageInfo()` a na základě dodržených konvencí o obsahu souboru stránek (název souboru, název třídy stránky, metoda `show()`, viz kapitola 6.2.3).

Strukturu stránky lze definovat v každé sekci zvlášť. Systém používá v základu rozdělení stránky na hlavičku, menu obsah a patku (viz Obrázek 6.2).



Obrázek 6.2. Rozdělení stránky na hlavičku, menu a obsah.

6.5 Menu

Tato kapitola se zabývá implementací menu. Nejprve je popsáno, jak se menu zobrazuje, dále vlastnosti menu systému, jak se generují položky menu a nakonec vzhled menu.

6.5.1 Vytvoření a zobrazení menu

Systém umožňuje každé sekci vytvořit vlastní strukturu menu. K tomu, aby bylo možné menu v sekci využít, je nutné dodržet určité dohodnuté koncepty (viz kapitola 6.2.2.1). Požadavek na Menu vytváří třída `HTMLBody`, která volá ve své metodě `show()` metodu `HTMLMenu()` třídy `ViewMenu` (viz. Výpis 6.9).

```
public function show() {
  //zobrazime stranku
  echo(' <div id="main"><!-- main -->'."\n");
  $this->header->HTMLheader();
  //nejprve vykreslime menu,
  $this->menu->HTMLMenu();
  //ted zobrazime obsah
  $this->obsah->show();
  echo(" </div><!-- konec main -->\n\n");
}
```

Výpis 6.9. Metoda show() třídy HTMLBody.

Motoda `show()` třídy `HTMLMenu`:

- Vytvoří počáteční a koncové HTML tagy menu,
- zjistí adresář, kde se nachází soubor *HTMLMenu.php*,
- ověří cesty a existenci třídy `HTMLMenu` a její metody `show()`
- a zavolá funkci `show()` této třídy.

Implementace metody `show()` se může lišit v každé sekci, která chce mít definované vlastní menu. Pokud se však využívá menu systému, volá se metoda `getHTMLMenu()` třídy z view `Menu`, která vytvoří kompletní menu.

6.5.2 Prvky menu

Systém podporuje tří základní prvky menu.

6.5.2.1 Nadpis

Nadpis je položka menu, která nemá žádný odkazový charakter. Je uvedena např. před celou kapitolou menu, jako "pobočky", "uživatel", “odkazy“ a podob.

6.5.2.2 Sekce

Sekce je položka menu, která není samotný odkaz na stránku, ale složka, ve které je k dispozici jeden, nebo více odkazů. Sekce může obsahovat i odkaz a to stejný odkaz jako některý z odkazů, které jsou jeho potomkem. Potom po odkliknutí této sekce se otevře zároveň i tento odkaz. Dále může sekce obsahovat odkaz s parametrem menu, např. **menu=administrace** , což umožní rozkliknutí dané sekce.

Sekce v menu je odlišný pojem než sekce systému shlukující stránky (viz kapitola 5.1.7).

6.5.2.3 Odkaz

Odkaz je klasický odkaz na stránku.

6.5.3 Vlastnosti menu

Systém **podporuje názvy menu ve více jazycích**. Libovolné položce menu lze nastavit její název a jazyk tohoto názvu.

Menu dále **umožňuje vícenásobné vytváření předků**, což znamená, že po rozkliknutí jedné sekce menu se zobrazí seznam dalších položek, pokud obsahují sekce, ty lze opět rozkliknout, atd.

6.5.4 Generování položek menu

Položky menu se generují v modelu jádra (třída `CoreModel`) metodou `getMenu()`. V této metodě, obdobně jako u metody `getPageInfo()` se nejprve získá ID Stránky ze vstupního názvu stránky a poté se volají metody `getMenuAdr()` a `getMenuItems()` třídy `Menu`, které vrátí cestu k souborům menu a hierarchickou strukturu položek menu, včetně jejich typů, odkazů, názvů atd.

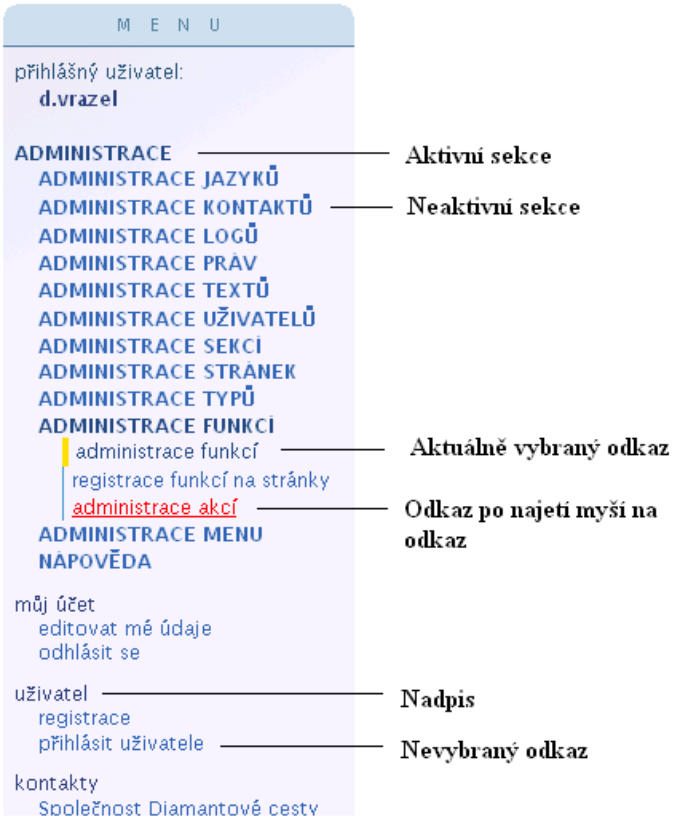
Metoda `getMenuItems()` třídy `Menu` v modelu jádra vygeneruje **položky menu na základě přístupových práv uživatele k jednotlivým stránkám systému**. Což znamená, že každému uživateli se budou v menu zobrazovat jen ty odkazy, nadpisy a sekce, ke kterým (nebo jeho potomkům) má daný uživatel přístup.

Menu je tak plně přizpůsobeno uživateli a zobrazují se tak jen ty položky, na které má daný uživatel přístup. Při analýze podobných systému bylo menu většinou tvořeno tak, že se řeklo, že administrátor má přístup tam a tam a na základě tohoto konceptu bylo vybudováno menu pro administrátora, menu pro překladatele atd. Horší variantou pak bylo vygenerování statického menu a při kliknutí na položku, kde uživatel nemá přístup, nastala chyba. Nejhorší variantou byly systémy, které umožňovaly jednouživatelský přístup po přihlášení metodou vše anebo nic.

Princip použitý v tomto systému je odlišný. Systém prochází při generování menu celý prostor položek menu a ověřuje, zda má uživatel přístup na daný odkaz. Pokud se tak změní právo uživatele (typ uživatele) v dané sekci a přestane tak mít daný právo přístupu na stránky této sekce, projeví se to okamžitě i v menu.

6.5.5 Vzhled menu

Vzhled menu značně závisí na aplikaci systému. Lze měnit vzhled ale i formu menu v každé sekci. Na obrázkuObrázek 6.3 je vidět standardní vzhled menu používaného v systému s popsanými jednotlivými prvky tohoto menu.



Obrázek 6.3. Standardní vzhled menu.

6.6 View

Komponenta view slouží k zobrazení modelu systému a k odchytávání požadavků uživatele. Z požadavků na systém, se komponenty View týká zejména:

- přidávání nových sekcí do systému, což souvisí přímo s adresářovou strukturou,
- komunikace s Controllerem pomocí protokolu SOAP,
- snadná údržba a rozšíření systému,
- možnost modifikace Menu dle sekcí,
- definice vlastního titulku, klíčových slov a popisu pro jednotlivé stránky a sekce,
- definice vlastního stylu pro jednotlivé sekce.

Dle požadavků, je klientská vrstva implementována pomocí PHP a značkovacího jazyka XHTML tak, aby byl přístup umožněn z internetových prohlížečů Firefox 2.0 a Internet Explorer 6.0.

V této kapitole budou popsány jen prvky systému vztahující se přímo ke komponentě view stránky, jako použití formulářů, tabulek a elementů. Ostatní části implementace view jsou popsány v jiných kapitolách.

- Implementace komunikace view přes SOAP protokol je popsána v kapitole 6.10.3.
- Implementace zobrazení stránek je popsána v kapitole 6.4.
- Implementace menu je popsána v kapitole 6.5.
- Základní konvence při použití souborů a tříd ve view jsou popsány kapitole 6.2.

6.6.1 Formuláře

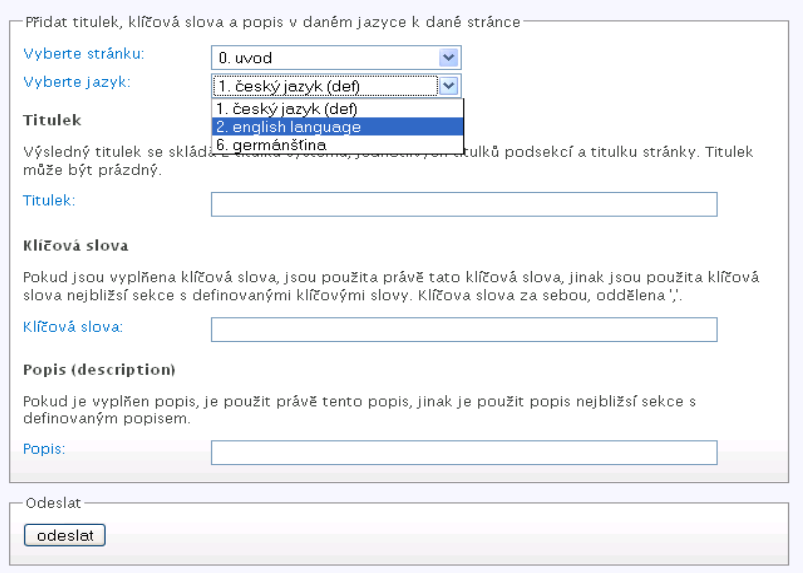
Formuláře umožňují uživateli komplexnější interakci se systémem. Umožňují vyplňovat vstupní pole, vybírat z možností, zaškrtávat volby a mnoho dalších funkcí.

Systém poskytuje výrazné usnadnění programátorovi pro tvorbu a zpracování formulářů. Pomocí třídy `Form` nabízí systém podporu pro budování formulářů, odchytávání a zobrazování chyb a odesílání formulářů v podobě předem hotových kusu kódu a metod, které vykonávají opakující se rutiny při práci s formuláři.

Kromě zjednodušení tvorby formulářů, získávají formuláře jednotný vzhled v rámci celého systému, což je výhodné pro přehlednost a uživatele a jednotnou strukturu v rámci celého systému, což je výhodné pro údržbu.

6.6.1.1 Vytváření formulářů

Obrázek 6.4 zobrazuje formulář pro přidávání a úpravu textů stránky závislých na zvoleném jazyku. Každá stránka umožňuje mít titulek stránky, popis a klíčová slova stránky v daném jazyce. Tento formulář umožňuje daný text v daném jazyce přidat, či pokud již existuje přepsat.



Obrázek 6.4. Formulář pro přidávání a editaci textů stránky.

Zdrojový kód tohoto formuláře ukazuje Výpis 6.10. Pro jednoduchost byly odstraněny vysvětlující texty k jednotlivým polím a metody pro získávání obsahů select boxů. Lze vidět, že vytvoření kompletního formuláře je otázka párů řádků.

```
$this->form->showErrors('novy_text_stranky');
$this->form->startForm('novy_text_stranky', $this->page->res['pageName']);
$this->form->startFieldSet('Přidat titulek, klíčová slova a popis v daném
jazyce k dané stránce');

$this->form->addSelect('Vyberte stránku', 'id_stranky', $param);
$this->form->addSelect('Vyberte jazyk', 'id_lang', $param);

$this->form->addField('Titulek', 'titulek', $param);
$this->form->addField('Klíčová slova', 'kl_slova', $param);
$this->form->addField('Popis', 'popis', $param);

$this->form->endFieldSet();

$this->form->startFieldSet('Odeslat');
$this->form->addButton('odeslat');
$this->form->endFieldSet();

$this->form->endForm();
```

Výpis 6.10. Formulář pro přidávání a editaci textů stránky.

Metody jako `addField()`, `addSelect()`, atd. zjišťují, zda byl formulář odeslán a pokud ano, přiřadí hodnotu dané položky z pole POST.

Dále si projdeme jednotlivé metody třídy `Form` a popíšeme funkce jednotlivých metod na zobrazeném příkladu.

6.6.1.2 Třída Form

Třída `Form` obsahuje metody pro práci s formuláři. Dále popíšeme nejdůležitější metody třídy:

- **`showErrors($nazev`**). Vypíše chybové hlášení. Formuláře se zpracovávají v konstruktorech stránek, čili v okamžiku zobrazení stránek již víme, zda aktualizace modelu proběhla v pořádku, či jaké se vyskytly chyby. Parametr `$název` určuje název formuláře (ověří se, zda byl odeslán formulář s daným jménem).
- **`startForm($nazev, $target`**). Vytvoří úvodní hlavičku formuláře s daným názvem, `$target` určuje, cíl zpracování formuláře. Většinou je to aktuální stránka (využití třídy `Page` a jejich metod a konstant pro aktuální stránku).
- **`startFieldset($legend, $class)`**. Vytvoří fieldset(rámeček) s daným textem a danou třídou (využití v konfiguraci CSS).
- **`addField($label, $name, $param`**). Přidá vstupní pole.
- **`addSmallField($label, $name, $param`**). Přidá malé vstupní pole.
- **`addSelect($label, $name, $param`**). Přidá select box.
- **`addTextArea($label, $name, $param)`**. Přidá textovou oblast.

- **`addButton($value, $param`**). Přidá tlačítko,
- **`endFieldSet()`**. Uzavře fieldset.
- **`endForm()`**. Uzavře formulář.

Důležitým parametrem všech funkcí je `$param`. `$param` obecně umožňuje detailnější nastavení daného prvku jako:

- Jaký je prvek typu (*password*, *checkbox*, *text*, atd.),
- je-li viditelný, či nikoli,
- je-li povinný, či nikoli (změna barvy pole, nutnost ověření, atd.),
- jeho velikost,
- hodnoty jednotlivých option v selectu,
- Vybranou položku v select boxu,
- další.

Pokud nejsou některé položky pole `$param` definovány, jsou použity standardní hodnoty.

Detaily pro použití `$param` lze najít v hlavičkách jednotlivých funkcí.

6.6.1.3 Zpracovávání formuláře

Metody pro zpracování formulářů se umísťují do konstruktoru stránek. Provede se tak případná aktualizace modelu ještě před samotným zobrazením obsahu.

Ve
```
if ((Form::submit('upravit_jazyk'))) {
  $res = $this->page->call_b('setLanguage', $_POST['id_language'],
  $_POST['prefix'], $_POST['popis']);
  if (!$res['stav']) {
    Form::addError($res['serError']);
  }
}
```

Výpis 6.11 je vidět zpracování formuláře na změnu jazyka v konstruktoru stránky *administrace jazyků*. Nejprve se ověří, zda byl volán požadovaný formulář, pokud ano, zavoláme patřičnou metodu controlleru pomocí SOAP protokolu. Tady jde o vázanou funkci, proto je použita nástavba třídy `Page`. Výsledek (informace o stavu je vrácena do proměnné `$res['stav']`). Pokud nastala chyba, je tato chyba přiřazena třídě `Form` a následně při zobrazení stránky vypsána.

```
if ((Form::submit('upravit_jazyk'))) {
  $res = $this->page->call_b('setLanguage', $_POST['id_language'],
  $_POST['prefix'], $_POST['popis']);
  if (!$res['stav']) {
    Form::addError($res['serError']);
  }
}
```

Výpis 6.11. Zpracování formuláře v konstruktoru stránky.

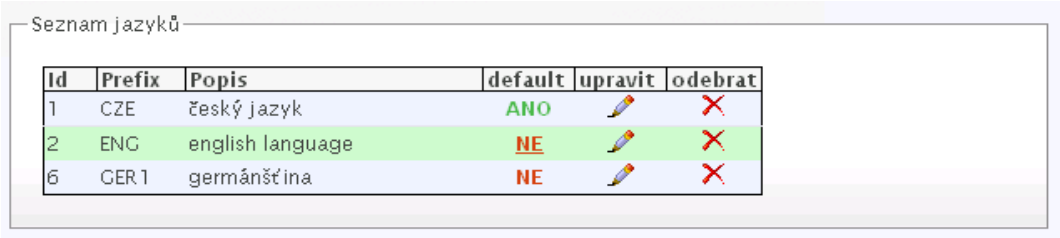
6.6.2 Tabulky

Obdobně jako u formulářů, umožňuje klientská vrstva(view) snadné vytváření tabulek z předem vytvořených bloků, prvků. Kromě výrazného usnadnění práce programátorovi, poskytuje tento přístup jednotný vzhled tabulek a zvyšuje tak přehlednost stránek a systému. Snadnou práci s tabulkami umožňuje třída `Table`.

Mezi vlastnosti tabulek vytvořených přes třídu `Table` patří:

- Jednotný vzhled,
- barevné odlišení lichých a sudých řádků,
- barevné odlišení aktuálního řádku (přes *JavaScript*),
- možnost řazení dle vybraných položek,
- možnost přidávání odkazů, obrázků atd.

Na obrázku Obrázek 6.5 je tabulka jazyků, která zobrazuje seznam dostupných jazyků na stránce *administrace jazyků* v systému. Výpis 6.12 ukazuje zdrojový kód této tabulky, včetně metody pro volání aktuálních dat modelu.



Obrázek 6.5. Tabulka jazyků.

```
$res = $this->page->call_b('getLanguages');
$lngs = $res['langItems'];

$this->table->startTable('', 'Seznam jazyků');
$this->table->head('Id', 'Prefix', 'Popis', 'default', 'upravit',
'odebrat', 30, 50, 200, 50, 50, 50);
$i = 0;
$count = sizeof($lngs);
foreach ($lngs as $val) {
  $id     = $val['id_language'];
  $prefix = $val['prefix'];
  $popis  = $val['popis'];
  $def    = ($val['def'] == 1 ?
    "<span class=\"ano\">ANO</span>" :
    "<a class=\"ne\" href=\"index.php?page=administrace_jazyku&def=$id\"
alt=\"nastavit jako default\">NE</a>") ;
  // vynachány definice $upravi a $odebrat
  $this->table->addRadek($i, $count, $id, $prefix, $popis, $def, $upravit,
  $odebrat);
  $i++;
}
$this->table->endTable();
```

Výpis 6.12. Zdrojový kód tabulky jazyků.

Zdrojový kód nejprve ukazuje volání vázané funkce `getLanguages()` pro získání seznamu jazyků do proměnné `$lngs`. Dále je volána metoda `startTable()` třídy `Table`, která vytvoří úvodní HTML tagy tabulky. Dále je vytvořená hlavička tabulky pomocí metody `head()`, kde se definují nejprve názvy sloupců hlavičky a poté jejich velikosti v pixelech.

V cyklu se dále zpracovávají postupně jednotlivé hodnoty získané z metody `b_getLanguages()` tak, aby odpovídali potřebám zobrazení v tabulce. Metoda `addRadek()` třídy `Table` poté zobrazí potřebné údaje. První parametr metody je číslo řádku, kvůli barevnému označování sudých a lichých řádků. Druhý parametr je počet řádků. Na základě tohoto parametru je správně nastaven poslední řádek. Zbylé parametry metody jsou hodnoty, které se mají zobrazit.

Poslední metoda `endTable()` je ukončení tabulky.

Do tabulek lze implementovat řazení na základě vybraných názvu prvků. Seřazení probíhá již v modelu systému. Je proto nutné, aby řazení podporovala daná funkce (parametr `orderBy` v definici funkce). Princip implementace spočívá ve vytvoření odkazu na funkci s potřebným parametrem `orderBy`. Výpis 6.13 ukazuje implementaci řazení dle určitých hodnot v tabulce stránek na stránce *administrace stránek*.

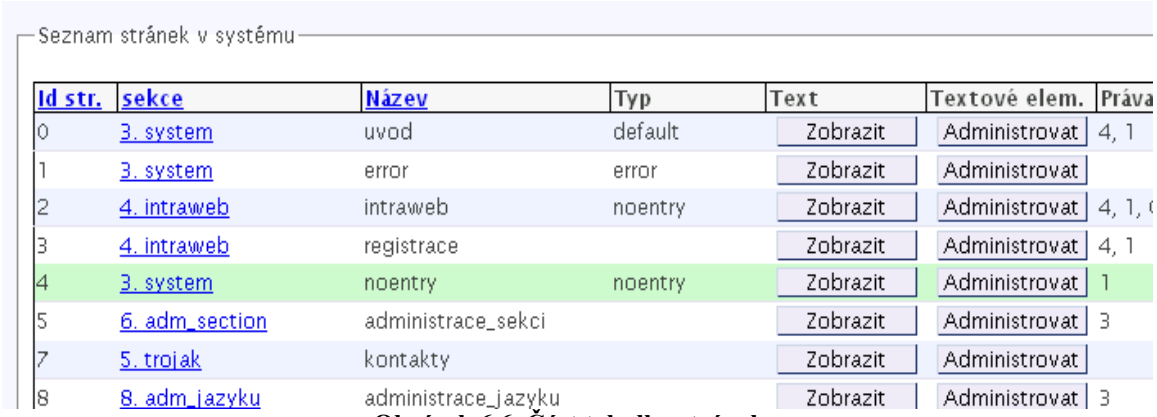
```
$res = $this->page->call_b('getStranky',  $_GET['taborder']);
$fncs = $res['pageItems'];
$param = array();
$param[0]['odkaz'] = 'page='.$this->page->
res['pageName'].'&taborder=id_stranky';
$param[1]['odkaz'] = 'page='.$this->page->
res['pageName'].'&taborder=id_section';
$param[2]['odkaz'] = 'page='.$this->page->
res['pageName'].'&taborder=nazev';
$this->table->startTable('', 'Seznam stránek v systému', $param);
```

Výpis 6.13. Zdrojový kód úvodu tabulky stránek.

V prvních řádcích se volá vázaná funkce `getStranky` s parametrem `$_GET['taborder']`, který určuje setřídění údajů podle tohoto parametru.

V dalších řádcích jsou do `pole $param` vytvořeny příslušné odkazy, které budou přiřazeny k názvům sloupců v hlavičce tabulky na dané pozici.

Voláním metody `startTable` s parametrem `$param` poté zařídí vytvoření příslušných odkazů v názvech. Ostatní části tabulky zůstávají nezměněny. Obrázek 6.6 ukazuje tabulku stránek s možností řazení dle Id stránky, sekce a názvu.



Obrázek 6.6. Část tabulky stránek.

6.6.3 Elementy

Elementy jsou předem vytvořené prvky kódu, které ulehčují programování některých, opakujících se prvků systému. Mezi tyto prvky patří:

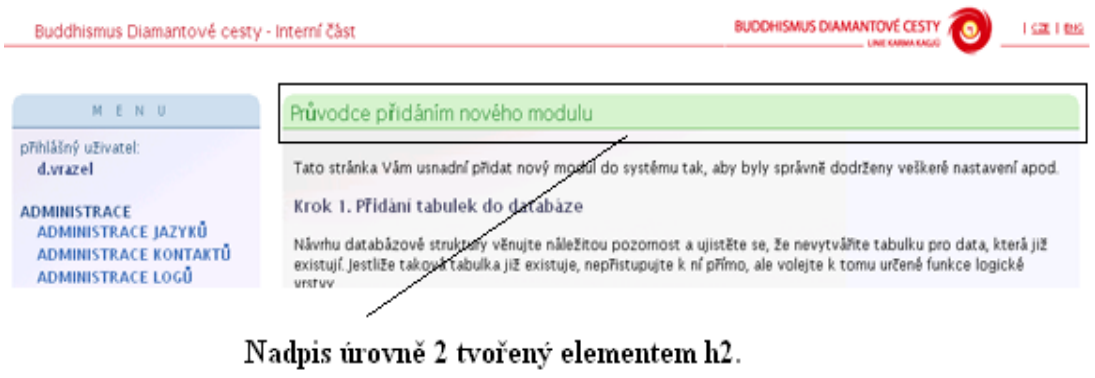
- `tlacitkoZpet($odkaz)` – Vytvoří na stránce tlačítko zpět s daným odkazem.
- `intable_AnoNE($bool)`- Vytvoří v tabulce text Ano nebo Ne a nastaví vzhled tohoto textu.
- `h2($text)` – Vytvoří nadpis H2 s požadovaným textem.

Výpis 6.14 ukazuje metodu `show()` stránky *pridat_modul*, která poskytuje podrobného průvodce přidáním nového modulu. Vidíme zde použití elementu `h2`, který vytvoří nadpis úrovně 2.

```
public function show() {
  $this->elem->h2('Průvodce přidáním nového modulu');
  $this->mainPage();
}
```

Výpis 6.14. Metoda show() stránky pridat_modul.

Obrázek 6.7 pak ukazuje výsledek tohoto zdrojového kódu.



Obrázek 6.7. Nadpis druhé úrovně.

6.7 Controller

V této kapitole je popsána implementace controlleru. Nejprve je zmíněna funkce controlleru v systému, dále se zaměříme na ověřování hodnot a třídu `Verify` a nakonec na soubory a adresářovou strukturu.

Vytvoření SOAP komunikace na controlleru je popsáno v kapitole 6.10.4.

6.7.1 Funkce controlleru

Controller je prvek, který má za úkol:

- Přijímat požadavky z View přes SOAP protokol (soubor *controller.php*).
- V dané funkci ověřit správný formát vstupních parametrů (za pomocí třídy `Verify`).
- Pokud je formát vstupních parametrů v pořádku:
  - Zavolat patřičnou metodu modelu. Požadovaný model se získá z parametru.
  - Získat výstupní hodnoty modelu.
  - Tyto hodnoty vrátit View přes SOAP protokol.
- Pokud není formát v pořádku, vrátí přes SOAP protokol informace o chybě, které se použijí jako chybové hlášení u daného formuláře.
- Pokud daný controller neexistuje, nebo neexistuje daná funkce v controlleru, vrátí controller závažnou chybu, která přeruší další vykonávání skriptu ve View vrstvě.

6.7.2 Třída ControllerCore

Třída `ControllerCore` je třída controlleru určeného pro jádro systému. Tato třída obsahuje metody definované ve WSDL souborech *core.wsdl* a *core2.wsdl,* určené pro volání modelu systému. Třída implementuje rozhraní `IControllerCore` definované v souboru *IControllerCore.php* viz výpis Výpis 6.15.

Rozhraní `IControllerCore` v souboru *IcontrollerCore.php*:

```
interface IControllerCore {
…
}
```

Implementace rozhraní `IControllerCore` třídou `ControllerCore` v souboru *controllerCore.php*:

```
Soubor controllerCore.php:
//rozhrani
include_once('IControllerCore.php');
class ControllerCore implements IControllerCore {
…
}
```

Výpis 6.15. Třída ControllerCore.

6.7.2.1 Metody třídy ControllerCore

Metody třídy jsou přímo volány pomocí SOAP protokolu z view a všechny metody mají obecně tento účel:

- Ověřit správnost vstupních údajů,
- zavolat model systému pro aktualizaci modelu, či získání dat modelu,
- pokud nastane chyba vstupního parametru, vytvořit chybovou hlášku, která bude použita při výpisu chyb formuláře.

Ve výpisuVýpis 6.16 je zobrazena typická metoda třídy `ControllerCore` s názvem `b_createLanguage`, která přidá nový jazyk do systému s daným prefixem a popisem. Bezpečnostní parametry `$hashUser` a `$idPage` slouží pro ověření, zda daný uživatel má přístup na danou stránku a zda z této stránky smí danou funkci spouštět. Více o implementaci bezpečnostních prvků systému v kapitole 6.9.

```
public function b_createLanguage($hashUser, $idPage, $prefix, $popis) {
  $res['stav']  = 0;
  $res['error'] = '';
  $res['createError'] = '';
  if (($this->verify->is_alphanum($hashUser, 1))
  && ($this->verify->not_negative($idPage))
  && ($this->verify->is_alphanum($prefix, 0))
  && ($this->verify->is_czech_alphanum($popis, 5))
  ) {
    $res = $this->model->b_createLanguage($hashUser, $idPage, $prefix,
    $popis);
  }
  else {
    $res['createError'] = $this->error->callError(get_class($this),
    "Parametry prefix a popis musí být vyplněny. Prefix musí obsahovat
    jen znaky [a-z,0-9] a popis musí být větší než 5 znaků!");
  }
  return $res;
}
```

Výpis 6.16. Metoda b_createLanguage třídy ControllerCore.

Prefix `b_` říká, že metoda je vázána na stránku (viz kapitola 5.2.2.2). Nejprve jsou nastaveny počáteční hodnoty výstupních parametrů. Za pomocí třídy `Verify` je ověřena struktura vstupních parametrů a pokud je vše v pořádku, je volána příslušná metoda modelu s ověřenými parametry.

Pokud jsou parametry chybné, vytvoří se chybová hláška (`$res['createError']`), která se použije pro výpis chyby u daného formuláře na stránce ve View.

Pokud se vyskytne závažná chyba systému, po které se má zastavit vykonávání skriptu, je do proměnné `$res['error']` přiřazena zpráva s chybovou hláškou (viz kapitola 6.10.3.2). Třída `Soap` ve `View` v klientské vrstvě pak zjistí, že tato proměnná není prázdná, vypíše hlášku a zastaví běh skriptu. Výpis 6.17 ukazuje metodu `isLogon()`, které ověřuje, zda je uživatel s daným hash

řetězcem přihlášen. Pokud hash řetězec neobsahuje jen alfanumerické znaky, nebo není prázdný, controller vygeneruje závažnou chybu, protože je podezření, že se jedná o pokus o útok. Hash přidělený aplikační vrstvou vždy splňuje podmínky ověření.

```
public function isLogon($hashUser) {
  //ovezi zda je dany uzivatel prihlasen na serveru
  if ($this->verify->is_alphanum($hashUser, 1)) {
    $res = $this->model->isLogon($hashUser);

  } else {
    $res['error'] = $this->error->callError(get_class($this), "Neplatné
    vstupní parametry.");
  }
  return $res;
}
```

Výpis 6.17. Funkce isLogon() třídy Controller.

6.7.3 Ověřování hodnot

Ke snadnému ověřování zadaných vstupních parametrů v controlleru slouží třída `Verify`. Třídu lze použít v kterémkoli controlleru, včetně controllerů modulů. Třída `Verify` implementuje tyto metody na ověřování vstupních dat:

- `not_negative($num)`. Ověří, zda je proměnná nezáporný integer.
- `is_alphanum($str, $zero = 0)`. Ověří, zda je proměnná `$str` alfanumerický řetězec + zkaky '_','-', `$zero` určuje, zda může být nulový či nikoli.
- `is_odkaz($str)`. Ověří, zda je řetězec platný odkaz.
- `is_cesta($str, $minCount)`. Ověří, zda je daný řetězec platná cesta, `$minCount` určuje min. počet znaků.
- `is_name($str)`. Ověří, zda je daný řetězec jméno.
- `is_czech_alphanum($str, $min_znaku = 0)`. Ověří, zda daný řetězec obsahuje alfanumerické znaky + znaky české abecedy.
- `is_login($login)`. Ověří, zda je daný řetězec login.
- `is_passwd($passwd)`. Ověří, zda je daný řetězec heslo.
- `is_mail($mail)`. Ověří, zda je daný řetězec e-mailová adresa.
- `is_num($str, $min, $max, $extra = '')`. Ověří, zda je `$str` číslo z rozsahu délky řetězce `<$min, $max>`, `$extra` povoluje zvláštní znaky.
- `is_space_alphanum($str, $is_zero = 0)`. Ověří, zda je proměnná alfanumerický řetězec + znak mezera.

6.7.4 Adresářová struktura a názvy

Protože hlavní třída Controller vybírá automaticky ze zadaných parametrů potřebný controller, a objekt třídy `SoapServer` poté vytvoří třídu požadovaného názvu a zavolá metodu této třídy, je potřeba dodržovat určité konvence a to zejména:

- V kořenovém adresáři controlleru jsou vytvořeny adresáře stejného názvu, jako název modulu. Např. *core*.
- Tyto adresáře povinně obsahují soubor *controllerNazev_modulu.php,* např. *controllerCore.php*.
- Tento soubor musí obsahovat třídu *ControllerNazev,* např. *ControllerCore*.
- V této třídě musí být funkce stejných názvů a parametrů jako definuje daný WSDL soubor. Právě tyto funkce jsou volány Soap klientem.
- V kořenovém adresáři controlleru se nachází adresář WSDL obsahující WSDL soubory všech modulů.
- Soubory se musí jmenovat *nazev_modulu.wsdl*. Např. *core.wsdl*.
- Pokud se WSDL soubor jmenuje jinak, než název modulu, je třeba tento název použít v parametru webové adresy portu v daném WSDL souboru. Např. http://www.system.cz/controller/controller.php?wsdl=core2 říká, že bude při vytváření SOAP komunikace v controlleru použit WSDL soubor se jménem *core2.wsdl*.

6.8 Model

Model tvoří aplikační logiku celého systému. Úkolem modelu je na základě daného požadavku buď aktualizovat systém, nebo vrátit požadované informace o systému. Každý modul implementuje vlastní model. Všechny třídy modelu jsou potomky třídy `BaseModel`.

Třída `BaseModel` vlastní objekt třídy `DataAccess` pro přístup k datové vrstvě.

Hlavní třídou modelu jádra systému je třída `ModelCore` v souboru *modelcore.php*, která implementuje rozhraní `IModelCore` a je potomkem třídy `BaseModel` (viz Výpis 6.18).

```
class ModelCore extends BaseModel implements IModelCore {
…
}
```

Výpis 6.18. Hlavička třídy ModelCore.

Metody Třídy `ModelCore` obecně nejprve ověří přístupová práva na základě vstupních parametrů, vrátí ID objektů z názvů apod. a poté volají metodu konkrétní třídy (např. `User`, `Page`, `Func`, `Lang`, atd.), která provede požadovanou aktualizaci modelu, či vrátí požadovaná data.

Protože mnoho řešení implementace modelu je zmíněno v ostatních kapitolách, bude dále popsána jen problematika univerzálního přístupu modelu k datové vrstvě a popis dodržení konvencí při názvech souborů a vytváření adresářové struktury.

6.8.1 Přístup k datové vrstvě

Dle požadavků na systém, musí být implementace datové vrstvy nezávislá na vrstvách ostatních. Je tedy potřeba vytvořit jednotné rozhraní pro model, které bude přistupovat k datové vrstvě.

Toto rozhraní zprostředkovává třída `dataAccess`. Jelikož je objekt třídy vytvořen v základní třídě `BaseModel`, od které jsou zděděny třídy jednotlivých modulů a třída systému, mají tak všechny modely ihned k dispozici přístup k datové vrstvě.

V konstruktoru třídy je na základě konstanty typu spojení vytvořeno požadované spojení s datovou vrstvou.

Dále se zaměříme na základní metody této třídy, metody `execSQL()` a `get_table()`.

6.8.1.1 Metoda execSQL

Hlavní a nejdůležitější metodou třídy je metoda `execSQL()`, která zprostředkovává ověření připojení, zavolání datové vrstvy a vrácení objektu typu `DataResult` (viz další kapitola) ke zpřístupnění dat. Metoda se nejprve otevře spojení s databází, zavolá daný SQL příkaz, na základě vrácených dat vytvoří objekt třídy `DataResult` a ukončí spojení s databázi.

Výpis 6.19 zobrazuje část metody `execSQL()` pro typ databáze MySQL. Je zde vidět, že se volá datová vrstva a ověřuje se, zda SQL dotaz proběhl korektně. Dále je vytvořen obsah vrácených dat a tento obsah vložen jako parametr konstruktoru objektu třídy `DataResult`, který je vrácen jako výsledek volání metody.

```
$db = $this->link;
if (!($result = @($db->query($sql)))) {
  $res['error'] = $error->callError('DataAccess', self::E_DATA_RES);
} else {
  $res['num_rows'] = $result->num_rows;
  $res['ok']       = true;
  $i = 0;

while (($result->num_rows > 0) &&
($row = $result->fetch_array(MYSQLI_BOTH))) {
  $res['data'][$i] = $row;
  $i++;
}
if (is_object($result))
  $result->close();
}
$dataResult = new DataResult($res);
return $dataResult;
```

Výpis 6.19. Část metody execSQL() třídy DataAccess.

6.8.1.2 Metoda get_table

Metoda `get_table()` vrátí tabulku daného názvu. Veškerý odkaz na tabulky se nezprostředkovává přímo v SQL dotazu, ale zavolá se metoda `get_table()`, která vrátí požadovaný název tabulky.

Tento přístup je vhodný, jsme-li nuceni použít jiný název jedné nebo více tabulek. Poté stačí jen změnit tento název v metodě `get_table()` a není potřeba měnit všechny SQL příkazy v celém modelu.

6.8.2 Zpracování výsledků získaných z datové vrstvy

Metoda `execSQL()` spustí požadovaný SQL dotaz a vrátí objekt třídy `DataResult`. `DataResult` je třída, která zprostředkovává jednotné rozhraní pro práci s výsledky databázových dotazů.

PHP disponuje funkcemi a třídami pro připojení k různým databázovým systémům. Ty, které nepodporuje přímo, podporuje pomocí ODBC rozhraní. Každý z balíků funkcí pro jednotlivé databázové systémy (potažmo XML soubory a jiné datové úložiště) disponuje jinými názvy funkcí, jinou množinou funkcí apod.

Proto je výhodné vytvořit jednotné rozhraní pro přístup k získaným datům z datové vrstvy. Toto rozhraní zprostředkovává třída `DataResult`.

`Třída dataResult` tedy poskytuje množinu metod pro práci s daty získanými z datové vrstvy. Jednotlivé třídy modelu tak nepřistupují k datům přímo, ale prostřednictvím třídy `DataResult` a množiny metod, která tato data zpřístupňují. Mezi tyto metody patří:

- `num_rows()`. Vrátí počet řádku dat, obsahuje test na prázdné, atd.
- `is_error ()`. Vrací TRUE, nastala-li chyba při provádění SQL dotazu.
- `fetch_array()`. Vrátí data aktuálního řádku a posune ukazatel na další řádek. Při prvním použití je nastaven první řádek, pokud je ukazatel na posledním řádku, při dalším pokusu o volání této metody je vráceno FALSE.
- `get_row($row_num)`. Vrátí data řádku daného čísla.
- `get_first_row()`. Vrátí data na prvním řádku.
- `get_first_val($key)`. Vrátí prvek na prvním řádku dle daného klíče.
- `get_val($line, $key)`. Vrátí hodnotu daného klíče na daném řádku.
- `get_error()`. Vrátí vzniklou chybu.
- `get()`. Vrátí první prvek na prvním řádku, pokud nastala chyba, vrací FALSE.
- `get_all_rows()`. Vrátí data na všech řádcích.
- `get_all_array($key)`. Vrátí data ze sloupce `$key` ze všech řádků.
- `find_in_key($key, $val)`. Vrátí TRUE, najde-li ve všech řádcích hodnotu $val v klíči $key.
- `was_ok()`. Vrátí TRUE, proběhl-li poslední dotaz v pořádku.

6.8.3 Použití přístupu k datové vrstvě

V předchozích kapitolách byl vysvětlen princip komunikace s datovou vrstvou, v této kapitole se zaměříme na ukázky implementace tohoto principu.

6.8.3.1 Vybírání hodnot z databáze.

V prvních příkladech se zaměříme na vybírání hodnot z datové vrstvy.

Ve výpisu Výpis 6.20 jsou načteny informace o všech jazycích z datové vrstvy prostřednictvím metody `execSQL()`. Výsledkem operace je objekt `$res` třídy `DataResult`. Nakonec se volá metoda `get_all_rows()` pro navrácení dat ze všech řádků výsledku a tyto informace jsou vráceny jako pole výstupních parametrů metody.

```
public function getLanguages() {
  //vrati vsechny informace o jazycich
  $sql = 'SELECT * FROM '.$this->db->get_table('core_language')
    ." order by id_language";
  $res = $this->db->execSQL($sql);
  $ret = $res->get_all_rows();
  return $ret;
}
```

Výpis 6.20. Příklad výběru všech hodnot z datové vrstvy.

Výpis 6.21 ukazuje použití metody `fetchArray()`. Nejprve je obdobně, jako v předešlém příkladu provedena metoda `execSQL()` a výsledek navrácen v `$res`. Dále se prochází postupně řádek po řádku výsledkem metodou `fetchArray()` a provádí se načtení obsahu řádku do pole, dále se spouští další dotaz a výsledek tohoto dotazu je pomocí metody `get()` (první hodnota na prvním řádku) přidán k výslednému poli.

```
$sql = 'SELECT * FROM '.$this->db->get_table('core_section')
." ORDER BY id_section";
$res = $this->db->execSQL($sql);

$i = 0;
$ret = array();
while ($row = $res->fetch_array()) {
  $ret[$i] = $row;
  $sql = 'select nazev FROM '.$this->db->get_table('core_section')
  ." WHERE id_section = ".$row['id_predka'];
  $res1 = $this->db->execSQL($sql);
  $ret[$i]['predek'] = $res1->get();
  $i++;
}
```

Výpis 6.21. Příklad použití metody fetch_array().

6.8.3.2 Vložení hodnot do databáze

Výpis 6.22 metody `createLanguage` třídy `Lang` v modelu systému ukazuje použití tříd `DataAccess` a `DataResult` při vkládání údajů do tabulky. Pro ověření zda dopadla operace dobře, či nikoli je použito metody `was_ok()` třídy `DataResult`.

Obdobně, jako u vkládání údajů lze použit metody `was_ok()` i na aktualizování tabulek a odstraňování údajů z tabulek.

```
public function createLanguage($prefix, $popis) {
  //nastavi jazyk, vraci id_jazyka, pokud v poradku, jinak FALSE
  $ret = false;
  $prefix = strtoupper($prefix);
  $sql = 'INSERT INTO '.$this->db->get_table('core_language')
  ." VALUES (NULL, '$prefix', '$popis', 0)";
  $res = $this->db->execSQL($sql);
  $ret = $res->was_ok();
  return $ret;
}
```

Výpis 6.22. Vložení hodnot do databáze.

6.8.4 Adresářová struktura

Obdobně jako u částí View či Controller, je nutné i u Modelu dodržovat adresářovou strukturu pro přidání nových modulů a to jen to, že každý modul bude obsažen v adresáři se stejným názvem, jako název modulu, umístěném v kořenovém adresáři modulu.

6.9 Bezpečnost systému

V této kapitole je zmíněna bezpečnost systému z hlediska implementace konkrétních částí. Principy bezpečnostních prvků jsou popsány v kapitole 5.2.

6.9.1 Bezpečnost komunikace přes SOAP protokol

Nejjednodušší cestou, jak ochránit data využívající služeb SOAP protokolu, je vytvořit zabezpečenou komunikaci mezi Soap klientem a serverem pomocí vrstvy SSL. To ale není otázka přímo systému, ale konfigurace serverů, či zařízení, na kterých systém poběží. Podmínkou je, aby klientské zařízení i server, kde je umístěn controller podporovali SSL protokol.

6.9.2 Bezpečnostní prvky plynoucí z třívrstvé architektury

6.9.2.1 Přihlášení uživatele a HASH číslo uživatele

O funkce spojené s uživatelem se stará třída `User` z Modelu systému. Metoda `logIn(idUser)` této třídy zajišťuje správné přihlášení uživatele do systému.

Metoda pracuje následovně:

1. Vygeneruje HASH řetězec pro uživatele, na základě aktuálního času a náhodného řetězce (viz Výpis 6.23).
2. Ověří, zda už je uživatel přihlášen, pokud ano, odstraní starý hash.
3. Přidá hash řetězec spolu s ID do tabulky přihlášených uživatelů
4. Vráti uživateli hash řetězec.

```
//vytvorime hash z aktualniho casu a nahodne vygenerovaneho retezce
$listAlpha =
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
$str1  = substr(str_shuffle($listAlpha),0,10);
$str1 .= mktime();
$hash  = sha1($str1);
```

Výpis 6.23. Vytvoření hash řetězce přihlášeného uživatele.

Ověřování hash řetězce probíhá při každém přístupu na stránku, či ověřování vázané funkce na stránku, řešení má proto nevýhodu při generování rozsáhlejších stránek, kdy ověřování se tak může dít i několikrát za stránku, podle počtu vázaných funkcí.

6.9.2.2 Ověření, zda smí uživatel vstoupit na danou stránku

Ověřování, zda smí uživatel vstoupit na danou stránku či nikoli, probíhá v modelu přímo před zavoláním dané aktualizace modelu či požadavku na zjištění stavu. Ověření se provádí na základě parametrů `hashUser` a `idPage` (viz Výpis 6.24).

Výpis ukazuje použití ověření přístupu. Volá se metoda `canEntryHashUser` z třídy modelu `User`, která ověří přístup a vrací `true/false`, podle toho, zda má daný uživatel přístup, či nikoli.

```
$user = new User($this->db);
$canEntry = $user->canEntryHashUser($hashUser, $idPage);
```

Výpis 6.24. Ověření přístupu na stránku.

Metoda `canEntryHashUser` využívá následující algoritmus:

1. Vrátí ID uživatele na základě `hashUser`.
2. Zjisti se, zda je daná stránka typu *multi*, což znamená, že se právo přístupu na stránku (jaký je typ uživatele) nevztahuje na sekci dané stránky, ale že právo přístupu na stránku má uživatel, který je daného typu aspoň v jedné sekci v systému (viz kapitola 5.1.8).
3. Pokud je typu *multi*:
    a. Zjistí typy uživatele, které mají přístup na stránku.
    b. Zjistí, jestli má daný typ uživatele uživatel s daným ID aspoň v jedné sekci systému.
    c. Vrátí výsledek předešlé operace.
4. Pokud není typu *multi*:
    a. Zjistí ID Sekce dle stránky.

b. Zjistí typ uživatele pro daného uživatele v dané sekci.
c. Ověří, zda daný typ smí na stránku,
d. Vrátí výsledek předešlé operace.

6.9.2.3 Ověření, zda smí být daná funkce spuštěna z dané stránky

Požadavek na ověřování spuštění funkce z dané stránky vychází z požadavku na bezpečnost systému(více v kapitole 5.2.2.2).

Ověřování, zda smí být daná funkce spuštěna z dané stránky, probíhá v modelu přímo před zavoláním dané aktualizace modelu či požadavku na zjištění stavu. Ověření se provádí na základě parametrů `hashUser` a `idPage`. Nejprve se provede ověření, zda smí daný uživatel na stránku pomocí metody `canEntryHashUser` třídy `User`, a poté se zavolá metoda `canRun()` třídy `Page`, která ověří, zda má funkce daného názvu (v příkladu Příklad 6.2 `addPageFunction()`) přístup na danou stránku.

Pokud ano, provede se požadovaná akce, jinak se vytvoří kritická chyba, která způsobí vypsání hlášky a zastavení provádění skriptu ve view.

```
$user = new User($this->db);
$canEntry = $user->canEntryHashUser($hashUser, $idPage);

if ($canEntry) {
  //jestlize muze vstoupit, overime, zda ma dana stranka
  //zaregistrovanou pozadovanou sluzbu
  $page = new Page($this->db);
  $canRun = $page->canRun('addPageFunction', $idPage);
  if ($canRun) {
      //akce
  } else {
   $ret['error']=$this->error->callError(get_class($this),
   $page->get_error('E_NO_RUN'));
  }
}
```

Příklad 6.2. Ověření, zda lze spustit funkci na dané stránce.

Algoritmus metody `canRun()` třídy `Page` jen ověří v datové vrstvě, zda je v tabulce `core_page_function` dvojce *název_funkce – idPage*.

6.10 Komunikace přes SOAP protokol

V této kapitole bude popsána komunikace přes SOAP protokol, přístup klienta, aplikační vrstvy a tvorba a konfigurace WSDL soboru.

6.10.1 Princip komunikace obecně

Princip spočívá v tom, že klientská vrstva (klient) se připojí k controlleru v aplikační vrstvě (serveru aplikační vrstvy). Zavolá potřebnou funkci controlleru a čeká na odpověď. K tomu, aby klientská vrstva věděla cestu ke controlleru, název funkce a formát vstupních a výstupních parametrů je potřeba znát cestu k WSDL souboru.

6.10.2 WSDL soubor

WSDL soubor popisuje oběma stranám komunikace přes SOAP protokol formát jednotlivých funkcí, jejich parametry (vstupní i výstupní) a typy parametrů.

6.10.2.1 Pravidla při vytváření WSDL souboru

Systém používá při komunikaci přes SOAP protokol určitých pravidel použití vstupních a výstupních parametrů. Tyto pravidla nejsou závazná (lze danou funkci vystavět i jinak), ale zpřehledňují následnou práci s daty. Mezi tato pravidla patří:

- Návratová hodnota jednoduchého typu se jmenuje *return.*
- Návratová hodnota složitého typu (pole parametrů) se jmenuje *nazevItems,* například funkce `getOdkaz()` vrací pole parametrů `odkazItems`.
- Je přítomna návratová hodnota `erorr`, která informuje o chybné struktuře dat, či chybných vstupních parametrech.
- Pokud je potřeba, je přítomna návratová hodnota *nazevErrors*, (typicky *createErrors*), která poskytuje chybová hlášení, která jsou použita poté jako chyby formulářů.
- Pokud funkce mění stav systému, je přítomna návratová hodnota `stav`, určující výsledek, zda proběhla změna modelu, či nikoli.
- Vázané funkce (na stránku, na typ) jsou označovány prefixem *b_*.
- Pokud je potřeba dodat funkci Hash řetězec uživatele (vázané funkce), jmenuje se tento parametr `hashUser`.
- Pokud je potřeba předat vstupní parametr id stránky, jmenuje se tento parametr `idPage`.
- Pokud je potřeba předat vstupní parametr pro seřazení údajů dle určitého kritéria, jmenuje se tento parametr `orderBy`.

6.10.2.2 Parametry portu ve WSDL souboru

Při tvorbě WSDL souboru je možné při definici portu a cesty k webové službě aplikační vrstvy použít dva parametry:

- **Controller.** Udává název controlleru a modelu,
- **Wsdl.** Určuje cestu k WSDL souboru v controlleru.

Při tvorbě WSDL souboru je nutné při definici portu a cesty k webové službě aplikační vrstvy (controlleru) uvést, pokud se jedná o WSDL soubor modulu, za webovou adresu ke controlleru parametr *controller=nazev_modulu* Například máme-li modul kontakty a controller umístěn v http://www.system.cz/controller/controller.php, vytvoříme port s cestou ke controlleru http://www.system.cz/controller/controller.php?controller=kontakty

Dále, pokud modul používá více WSDL soborů, je potřeba uvést parametr *wsdl=nazev_wsdl_souboru_bez_wsdl,* který určuje jaký WSDL soubor se má použít v controlleru. Např.

http://www.system.cz/controller/controller.php?wsdl=core2 říká, že bude při vytváření SOAP komunikace v controlleru použit WSDL soubor se jménem *core2.wsdl.*

6.10.2.3 Implementace WSDL souboru

WSDL soubor je pro člověka náročný na implementaci a údržbu. Proto se používají různé nástroje, usnadňující generování WSDL souborů. Při implementaci bylo využito plug-inu pro generování WSDL souboru do prostředí Eclipse.

Problém při využití tohoto nástroje nastal při definici více funkcí v jednom souboru. Počítači trvalo neúměrně dlouhou dobu tento soubor zpracovat a uložit. Bylo proto potřeba rozdělit definici do více WSDL souborů.

6.10.3 Použití SOAP ve View

View (klientská vrstva) využívá ke komunikaci přes SOAP třídu `Soap`. Ta má jen konstruktor a metodu `call()`.

6.10.3.1 Konstruktor třídy Soap

Konstruktor třídy Soap má za úkol:

Přiřadit WSDL soubor dle vstupního parametru `$model`. Soubory WSDL jsou automaticky přiřazeny dle hodnoty parametru `$model`. Cesty k WSDL souboru jsou dány jako konstanty v třídě `Soap`. Znamená to, že vždy při přidání nového modulu je potřeba jen vytvořit konstantu daného jména s cestou souboru a už není potřeba měnit další kód.

Dále konstruktor vytvoří objekt třídy `SoapClient($WSDLSURi)`s parametrem cestou k souboru WSDL a tím **zaregistrovat klientskou část komunikace**. Po té je možné volat funkce

controlleru. Pokud se nepodaří klienta zaregistrovat, zavoláme systémovou chybu s požadovanou hláškou. Část kódu konstruktoru lze vidět na výpisu Výpis 6.25.

```
$path = strtoupper($model).'_WSDL_URI';
 //existuje-li metoda na volani WSDL
 if (defined("self::$path")) {
    $WSDLSURi = constant("self::$path");
    try {//existuje-li wsdl a je li spravne, zaregistrujem SOAP klienta
      $this->client = new SoapClient($WSDLSURi);
    } catch  (Exception $e) {
      $this->error->callError(get_class($this),'Nelze zaslat požadavek
      Controlleru - chyba souboru '.$WSDLSURi.'!');
    }
 } else
   $this->error->callError(get_class($this),'Nelze získat WSDL soubor!');
```

Výpis 6.25. Konstruktor třídy Soap.

6.10.3.2 Metoda call

Metoda `call()` slouží k volání vzdálené funkce. Princip metody spočívá v:

- Ověření parametrů,
- vystavět požadovaný formát funkce a jejich parametrů,
- připojení k SOAP serveru (controlleru) a zavolání příslušné funkce přes SOAP,
- ověření výstupních parametrů,
- vrátit seznam výstupních hodnot volající metodě.

Pokud nastane chyba zprávy, funkce, či třídy na SOAP serveru, zavolá se kritická chyba, která vypíše hlášení a ukončí běh skriptu.

Metoda dále testuje návratovou hodnotu `error`. Pokud je tato hodnota neprázdná, znamená to, že nastala kritická chyba v controlleru nebo v modelu a bude zavolána kritická chyba, která vypíše hlášení a ukončí běh skriptu.

6.10.3.3 Volání metody call

Metoda `call()` se volá buď přímo na funkce, které se nevážou na stránku, např. funkce pro zjištění informací o stránce (viz Výpis 6.26):

```
$this->soap->call('getPageInfo',$_SESSION['hash_user'],   $_GET['section'],
$_GET['page'], $_SESSION['lang']);
```

Výpis 6.26. Volání metody call() přímo.

nebo prostřednictvím metody `call_b()` třídy `Page`, která je určena pro volání vázaných funkcí na stránku, např volání vázané funkce pro registraci textu stránky (Výpis 6.27):

```
res = $this->page->call_b('createTextStranky',
      $idStranky, $idLang, $titulek, $kl_slova, $popis);
```

Výpis 6.27. Voláni metody call_b() třídy Page.

6.10.4 Použití SOAP na Controlleru

Klient komunikace SOAP si zjistí port (webovou adresu), na kterém jsou k dispozici webové služby. V našem případě je to controller, který webové služby zprostředkovává. Adresa portu z našeho příkladu je http://www.system.cz/controller/controller.php. Služba na klientovi si tak vynutí obsah této stránky. Ve Výpis 6.28 vidíme spustitelný obsah souboru *controller.php*. Soubor dále obsahuje třídu `Controller` s jedinou metodou `run()`.

```
$controller = new Controller($_GET);
$controller->run();
```

Výpis 6.28. Soubor controller.php

6.10.4.1 Konstruktor třídy Controller

Na základě parametrů webové adresy portu se v konstruktoru třídy `Controller` (viz Výpis 6.29):

- Zjistí název modelu,
- vytvoří název třídy požadovaného controlleru daného modelu dle názvu modelu,
- vygeneruje se cesta k dané třídě,
- vygeneruje cesta k WSDL souboru.

```
if ((is_array($getvars)) && (isset($getvars['controller'])) ) {
   //jmeno modelu
   $contr = strtolower(strip_tags($getvars['controller']));
   $this->modelName = $contr;
 }
 else {
   //neni-li receno jinak, vezmeme model jadra
   $this->modelName = 'core';
 }

 $this->class     = 'controller'.ucfirst($this->modelName);
 $this->file       = $this->modelName.'/'.$this->class.'.php';

//wsdl soubor
 if (isset($getvars['wsdl'])) {
   $this->wsdl = 'wsdl/'.$getvars['wsdl'].'.wsdl';
 }
 else {
   $this->wsdl = 'wsdl/'.$this->modelName.'.wsdl';
 }
```

Výpis 6.29. Konstruktor controlleru.

Parametry portu webové služby (webové adresy controlleru) mohou být:

- **Controller.** Název controlleru i modelu. Podle tohoto názvu se:
  - Vytváří název modelu,
  - vytváří cesta k WSDL souboru,
  - vytváří cesta k souboru controlleru,
  - vytváří název třídy controlleru.
- **Wsdl.** Pokud je definován tento parametr, cesta k WSDL souboru je převzata z tohoto parametru.

6.10.4.2 Metoda run() třídy Controller

Dále se volá jediná metoda třídy `Controller`, metoda `run()`, která (viz Výpis 6.30) :

- Ověří správnost cest,
- pokud v pořádku, vytvoří objekt typu `SoapServer`,
- zaregistruje požadovanou třídu,
- a zavolá požadovanou funkci.

```
if ((file_exists($this->wsdl)) && (class_exists($this->class))) {
   $soapServer = new SoapServer($this->wsdl);
   $soapServer->setClass($this->class);
   $soapServer->handle();
}
else {
   //vyvola chybu na klientovi, ale neprozradi nic
   //o serveru (cesty)
   $soapServer = new SoapServer();
   $soapServer->handle();
}
```

Výpis 6.30. Metoda run() třídy Controller.

Při nesplnění podmínky o existenci souboru WSDL nebo třídy controlleru (např. `ControllerCore`) by za normálních okolností vznikla chyba, která by se zobrazila přímo do prohlížeče, což je pro nás nežádoucí, proto je při nesplnění podmínky vytvořen server, zahájeno spojení, ale protože není definovaná třída, je vygenerovaná chyba, ale tato **chyba** je **poslána zpět klientu, který s ní může pracovat**. Navíc není prozrazena cesta ke controlleru, což není nijak vysoké bezpečnostní riziko, ale ani běžný uživatel nemusí vědět, že ve skutečnosti se jeho požadavek zpracovává na jiném serveru, než na který je připojen.

6.11 Vzhled systému

Vzhled systému je do značné míry ovlivněn použitým CSS souborem a definicí struktury zobrazení a může se lišit v každé sekci. Systém v základu používá jednotný vzhled, kdy na bílém pozadí je

nejprve zobrazena hlavička s logem a nadpisem celé sekce (ta se generuje ze stránky, její sekce a předků této sekce), dále vlevo menu a v pravé části obsahová část (viz Obrázek 6.8).

V obsahové části je nejprve uveden nadpis stránky a obsah stránky. Vzhled formulářů i tabulek je v celém systému jednotný díky použití bloků prvků z tříd `Form` a `Table` z View.

Obrázek 6.8. Vzhled aplikace systému.

6.12 Rozšíření systému o modul Kontakty

V této kapitole bude stručně popsána implementace rozšíření systému o modul Kontakty. Ověříme tak požadavek na rozšiřitelnost systému a dále bude na tomto konkrétním příkladu vysvětlen proces přidávání nových modulů do systému.

Nejprve se zmíníme o implementaci modulu Kontakty, dále bude na tomto příkladu popsán proces rozšiřování systému o nové moduly a na konec ukážeme aplikaci modulu v praxi.

6.12.1 Implementace modulu Kontakty

Při implementaci modulu kontakty bylo využito existujících prvků systému. V části model byl použit jednotný přístup k datové vrstvě, v části View byly použity třídy `Form` a `Table` k vytvoření formulářů a tabulek. Proto byla implementace modulu poměrně snadná.

Obrázek Obrázek 6.9 ukazuje aplikaci modulu Kontakty v prostředí sdružení SDC.

Obrázek 6.9. Aplikace modulu Kontakty.

Výpis metody `show()` stránky *kontakt_sdc* (Výpis 6.31) ukazuje, že veškeré informace o daném kontaktu vygenerovala jediná metoda `showHTMLKontakt()` třídy `Kontakty`. Tato metoda získá informace o kontaktu od modelu kontakty a získané informace jen zobrazí v požadovaném formátu.

```
public function show() {
  $this->elem->h2("Kontaktní informace");
  $this->kont->showHTMLKontakt(2);
}
```

Výpis 6.31. Metoda show() stránky kontakty_sdc.

6.12.2 Postup při rozšíření systému o nový modul

Na konkrétním modulu kontakty bude popsán proces rozšíření systému o nový modul.

6.12.2.1 Přidání tabulek do databáze

Jména tabulek začínají jedinečným prefixem modulu (prvních 4 až 5 znaků z názvu modulu), např. **kont_** pro modul *Kontakty*. Název celé tabulky potom může být **kont_adresa**.

6.12.2.2 Vytvoření WSDL souboru

Při vytváření WSDL souboru je nutné v portu v odkazu směřujícím na controller se SOAP serverem uvést jako GET parametr **controller=NAZEV_MODEL**. Příklad: http://www.system.cz/controller/controller.php?controller=kontakty.

6.12.2.3 Uložení WSDL souboru

WSDL soubor se uloží do kořenového adresáře části Controller, adresáře *wsdl*. Název souboru bude **jmeno_modulu.wsdl**, např. *kontakty.wsdl*.

6.12.2.4 Ve View vrstvě

- V souboru *soap.php* v kořenovém adresáři VIEW přidáme privátní konstantu nazvanou **NAZEV_MODULU_WSDL_URI**, která obsahuje cestu k Vašemu WSDL souboru, např. `KONTAKTY_WSDL_URI`.
- Objekt třídy `Soap` bude vytvořen s parametrem udávajícím název modulu. Např. `$mySoap = new Soap('kontakty').`

6.12.2.5 Vytvoření Controlleru

V controlleru je potřeba:

- Vytvořit adresář s názvem našeho modulu v adresáři 'controller', např. *kontakty*.
- V tomto adresáři vytvořit soubor *controllerNAZEV_MODULU.php*, který obsahuje implementaci controlleru pro daný modul, např. *controllerKontakty.php*.

6.12.2.6 Vytvoření Modelu

V modelu nového modulu je potřeba:

- Vytvořit adresář s názvem našeho modulu v kořenovém adresáři modelu, např. *Kontakty*.
- V tomto adresáři vytvořit soubor *modelNAZEV_MODULU.php*, který obsahuje implementaci modelu pro daný modul, např. *modelKontakty.php*.

6.12.2.7 Vytvoření sekcí v systému

Pokud je potřeba, pomocí administračního rozhraní lze vytvořit novou sekci do systému.

6.12.2.8 Vytvoření stránek k sekci

Pomocí administračního rozhraní systému lze vytvořit novou stránku do systému. Nové stránce je potřeba přiřadit práva, která určují, jaký typ uživatele má k dané stránce přístup.

6.12.2.9 Vytvoření menu ke stránkám

Pomocí administračního rozhraní systému lze vytvořit požadované odkazy na jednotlivé stránky. Struktura menu nemusí odpovídat (a většinou neodpovídá) strukturám sekci. Lze vytvářet vlastní sekce menu, názvy a odkazy na stránky v různých jazycích.

6.13 Nasazení systému v praxi

Jelikož stále probíhá vývoj a implementace modulů systému pro jeho aplikaci v prostředí sdružení SDC, není jeho ostré nasazení v praxi zatím dostupné. Testovací verze systému běží na stránkách http://www.duelsoft.cz/dpi/.

Zde si lze v praxi vyzkoušet plnou funkčnost sytému, včetně přístupu pro jednotlivé typy uživatelů. Přihlašovací jména - hesla jsou:

- admin – admin, pro uživatelský typ *Administrátor,*
- komp – komp, pro uživatelský typ *Kompetentní uživatel,*
- obyc – obyc, pro uživatelský typ *Obyčejný uživatel.*

Adresa systému je ve skutečnosti jen adresa klientské vrstvy. Aplikační vrstva může být díky technologii webových služeb na úplně jiné doméně i serveru, což je příznivé z hlediska rozdělení zátěže serverů, ale vznikají zde však větší nároky na síť.

7 Závěr

Dle názoru autora, bylo dosaženo cílů stanovených v úvodu této práce.

Byl navržen a implementován systém, který umožňuje oddělení zobrazovací vrstvy od vrstvy aplikační a datové. Systém, který je nezávislý na prostředí a implementaci jednotlivých části. Systém, který umožňuje běh jednotlivých jeho částí na samostatných serverech a doménách. Systém, který umožňuje snadné rozšíření o další moduly systému. Systém, který umožňuje sdílet funkce tohoto systému i jiným systémům.

Programátorům a administrátorům systém umožňuje snadnou údržbu a vytváření nových částí systému. Systém dále poskytuje širokou variabilitu ve výsledném vzhledu systému jako celku i jeho jednotlivých částí. Uživatelům systém poskytuje přehledný a sjednocený vzhled a u administračních částí podrobné vysvětlení významu jednotlivých nastavení. Menu systému se přizpůsobuje uživateli a zobrazuje jen ty položky, ke kterým má uživatel přístup. Je implementována podpora vícejazyčného přístupu bez omezení počtu systémových jazyků.

Při návrhu a implementaci byla dodržena architektura návrhového vzoru Model-View-Controller. Dále je systém naprogramován za pomocí objektového přístupu, což na rozdíl od mnoha projektů založených na funkcionálním programování v PHP poskytuje výhodu snadné údržby, rozšíření, aktualizace a přehlednosti. Z hlediska bezpečnosti systém používá mechanizmus ověřování přístupu na straně Modelu, čímž snižuje riziko zneužití dat.

Implementace systému podobného rozsahu v prostředí PHP a MySQL není častá a z dostupných zdrojů nebylo možné nastudovat principy řešení podobného systému v prostředí PHP. Proto většina programátorských řešení a návrhu různých dílčích částí jsou vlastní postupy autora práce.

Bylo navrženo a implementováno jádro systému a modul Kontakty. K tomu, aby mohl být systém aplikován v praxi, např. v prostředí sdružení SDC, je ale nutné implementovat další moduly a rozšířit tak stávající systém o prvky jako fotogalerie, redakční systém, mailing listy, registrace na akce, modul pro sdílení dat a další.

V rámci koncepce systému je možné zefektivnit komunikaci mezi jednotlivými částmi systému optimalizací SQL dotazů, nebo optimalizací vstupních a výstupních parametrů funkcí webové služby aplikační vrstvy systému.

Díky použité technologií webových služeb není systém závislý na internetových prohlížečích a je dosaženo široké univerzálnosti. Lze tak napsat klienta v libovolném jazyku podporujícím SOAP protokol, či speciálního klienta pro mobilní zařízení. Dále lze funkce systému využívat v jiném systému a přidat tak do již existujícího systému funkce nové. Mezi další pozitivum patří vstupní cena použitých technologií (zdarma) a v neposlední řadě i získání nových poznatků a zkušeností autora práce s oblasti architektur informačních systémů, PHP, SOAP komunikace, bezpečnosti a dalších.

Mezi nevýhody použití daných technologií však patří to, že vznikají větší nároky na přenesená data mezi klientskou a aplikační vrstvou, díky nutnému systému ověřování přístupu k datům se zvyšuje výpočetní náročnost na straně aplikační vrstvy. Pokud aplikační a klientská vrstva běží na jednom serveru, zátěž se ještě zvyšuje. Mezi další nevýhody patří náročnost na programování takovéhoto systému, kdy je nutné pro jednu funkci systému vytvořit její definici ve view, WSDL souboru, controlleru, modelu a konkrétní třídě vykonávající danou operaci. Mezi další nevýhody patří náročnější přidávání modulů (je potřeba vytvořit model, controller, WSDL soubor, sekce, atd.). Zvyšují se tak poměrně výrazně nároky na čas potřebný k implementaci a na velikost kódu. Celý systém se tak prodražuje.

Kombinace použitých technologií má své výhody i nevýhody. Dle názoru autora srovnání výhod a nevýhod systému, je příznivější pro výhody, vzhledem k tomu, že díky rychlosti dnešních počítačů a sítí je zpomalení nevýrazné. Naproti poměrně velkému vstupnímu úsilí programátora tak stojí široké využití funkcí systému, ať už v napojení na systémy existující, nebo implementací nových klientů rozšiřujících uživatelovo pohodlí, nebo umožňujících přístup k systému z jiných prostředí jako mobilní aplikace, příkazová řádka, atd. Nezajímavá je i možnost konfigurovat různý vzhled stránek, či používat různá menu. Velké plus je také v použití objektového paradigmatu, třívrstvé architektury a návrhového vzoru Model-View-Controller, což zvyšuje přehlednost systému a jeho udržovatelnost.

Literatura

[1] **Apache Software Foundation. 2007.** Struts. *Apache.org.* [Online] 30. Březen 2007. http://struts.apache.org/.

[2] **Arlow, Jim a Neustadt, Ila. 2003.** *UML a unifikovaný proces vývoje aplikací.* Brno : Computer Press, 2003. ISBN 80-7226-947-X.

[3] **Bergen, Patrick van. 2007.** Presentation-Abstraction-Control. [Online] 2007. http://www.dossier-andreas.net/software_architecture/pac.html.

[4] **Wikipedia. 2007.** Informační systém. *Wikipedia.* [Online] 9. Květen 2007. http://cs.wikipedia.org/wiki/Informa%C4%8Dn%C3%AD_syst%C3%A9m.

[5] **Janovský, Dušan. 2007.** CSS. *Jak psát web.* [Online] Květen. 6 2007. http://www.jakpsatweb.cz/css/.

[6] **Kažula, Radovan. 2006.** vzor Layers. *radovan.bloger.cz* . [Online] 2006. http://radovan.bloger.cz/informatika/objektove-aplikace/vzor-Layers.

[7] **Kažula, Radovan. 2006.** vzor Pipes and filters. *radovan.bloger.cz* . [Online] 20. Duben 2006. http://radovan.bloger.cz/informatika/objektove-aplikace/vzor-Pipes-and-filters.

[8] **Kosek, Jiří. 1999.** *PHP tvorba interaktivních internetových aplikací.* Praha : Grada Publishing, 1999. ISBN 80-7169-373-1.

[9] **Kosek, Jiří.** Využití webových služeb a protokolu SOAP při komunikaci. [Online] http://www.kosek.cz/diplomka/html/websluzby.html.

[10] **Kozák, David. 2002.** Jak fungují webové služby. *Interval.cz.* [Online] 8. Říjen 2002. http://interval.cz/clanky/jak-funguji-webove-sluzby/.

[11] **Kužela, Radovan. 2006.** vzor Broker. *radovan.blogger.cz.* [Online] 3. Duben 2006. http://radovan.bloger.cz/informatika/objektove-aplikace/vzor-Broker.

[12] **The PHP Group. 2007.** Manuál PHP. *PHP manual.* [Online] 22. Březen 2007. http://www.php.net/manual/cs/.

[13] **Wikipedia. 2007.** Model-view-controller. *Wikipedia.* [Online] 14. Květen 2007. http://en.wikipedia.org/wiki/Model-view-controller.

[14] **Wikipedie. 2007.** Model-view-controller. *Wikipedie.* [Online] 7. Březen 2007. http://cs.wikipedia.org/wiki/Model-view-controller.

[15] **Wikipedie. 2007.** MySQL. *Wikipedie.* [Online] 6. Květen 2007. http://cs.wikipedia.org/wiki/MySQL.

[16] **MySQL. 2007.** MySQL 5.0 Reference Manual. *MySQL Documentation.* [Online] 2007. http://dev.mysql.com/doc/refman/5.0/en/select.html.

[17] **Wikipedie. 2007.** Návrhový vzor. *Wikipedie.* [Online] 23. Leden 2007. http://cs.wikipedia.org/wiki/N%C3%A1vrhov%C3%BD_vzor.

[18] **Wikipedie. 2007.** Objektově orientované programování. *Wikipedie.* [Online] 25. Březen 2007. http://cs.wikipedia.org/wiki/Objektov%C4%9B_orientovan%C3%A9_programov%C3%A1n%C3%AD.

[19] **Pavlíček, Luboš. 2005.** Návrhové vzory (design patterns). *Návrhové vzory (design patterns) - diplomová práce M. Dvořáka.* [Online] 16. Červen 2005. http://objekty.vse.cz/Objekty/Vzory.

[20] **Pichlík, Roman. 2004.** Třívrstvá architektura v kostce I. *Dagblog.* [Online] http://www.sweb.cz/pichlik/archive/2004_11_07_archive.html.

[21] **Pichlík, Roman. 2004.** Třívrstvá architektura v kostce II. *http://www.sweb.cz/pichlik/archive/2006_07_23_archive.html.* [Online] 11. Listopad 2004. http://www.sweb.cz/pichlik/archive/2006_07_23_archive.html.

[22] **Růžička, Pavel. 2002.** Bezpečnost především - použití SSL. *Interval.cz.* [Online] 6. Červen 2002. http://interval.cz/clanky/bezpecnost-predevsim-pouziti-ssl/.

[23] **Scambray, Joel a Shema, Mike. 2003.** *Hacking bez tajemství: Webové aplikace.* Brno : Computer Press, 2003. ISBN 80-7226-769-8.

[24] **Wikipedia. 2007.** Secure Sockets Layer. *Wikipedia.* [Online] 22. Duben 2007. http://cs.wikipedia.org/wiki/Secure_Sockets_Layer.

[25] **Wikipedia. 2007.** Simple Object Access Protocol. *Wikipedia.* [Online] 7. Březen 2007. http://cs.wikipedia.org/wiki/Simple_Object_Access_Protocol.

[26] **Wikipedie. 2007.** Smalltalk. *Wikipedie.* [Online] 27. Březen 2007. http://cs.wikipedia.org/wiki/Smalltalk.

[27] **Sun Microsystems. 2002.** Designing Enterprise Applications. *J2EE Architecture Approaches.* [Online] 2002. http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/app-arch/app-arch2.html.

[28] **Sun Microsystems, Inc. 2007.** JavaServer Faces Technology. *Sun Devoleper Network.* [Online] 2007. http://java.sun.com/javaee/javaserverfaces/.

[29] **Tichý, Jan. 2004.** *Programová podpora tvorby webových aplikací.* [Online] 25. Srpen 2004. http://www.jantichy.cz/diplomka/pozadavky/architektura.

[30] **Wikipedie. 2007.** Unified Modeling Language. *Wikipedie.* [Online] 8. Duben 2007. http://cs.wikipedia.org/wiki/Unified_Modeling_Language.

[31] **Wikipedie. 2007.** Unified Modeling Language. *Wikipedie.* [Online] 8. duben 2007. http://cs.wikipedia.org/wiki/Unified_Modeling_Language#Historie_UML.

[32] **Webtvorba. 2004.** Úvod do XHTML. *WEBTVORBA.* [Online] 2004. http://www.webtvorba.cz/xhtml/uvod-do-xhtml.html.

[33] **W3C. 2004.** Latest SOAP versions. *W3C.* [Online] 2004. http://www.w3.org/TR/soap/.

[34] **Wikipedia. 2007.** Web Services Description Language. *Wikipedia.* [Online] 9. Březen 2007. http://cs.wikipedia.org/wiki/Web_Services_Description_Language.

[35] **www.springframework.org . 2006.** Spring Framework. *Spring Framework.* [Online] 2006. http://www.springframework.org/.

[36] **Zeldman, Jeffrey. 2004.** *Tvorba webů podle standardů XHTML, CSS, DOM, ECMAScript.* Brno : Computer Press, 2004. ISBN 80-251-0347-1.

[37] **Zendulka, Jaroslav.** Projektování programových systémů. Poznámky k přednáškám v elektronické podobě. *Projektování programových systémů.* [Online] FIT VUT Brno. http://www.fit.vutbr.cz/study/courses/PPS/public/prednasky_e.htm.

[38] **Žaloudek, Jiří.** Webové služby. *Java web services.* [Online] http://www.volny.cz/zaloudekjiri/rp/ch03.html.

[39] **Žižka, Petr. 2002.** Možnosti přístupu k datovým zdrojům 1. *Interval.cz.* [Online] 11. Prosinec 2002. http://interval.cz/clanky/moznosti-pristupu-k-datovym-zdrojum-1/.

Seznam příloh