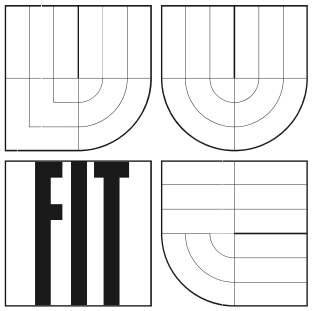
Vysoké učení technické v Brně

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Informační systém občanského sdružení Společnosti Diamantové Cesty

Ročníkový projekt

2006 Dušan Vrážel

Prohlášení

Prohlašuji, že jsem tento ročníkový projekt vypracoval samostatně pod vedením Ing. Jaroslava Švece.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....

Jméno Příjmení

Poděkování

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

Abstrakt

Tento ročníkový projekt se zabývá analýzou, návrhem a implementací Informačního systému pro občanské sdružení Společnost Diamantové cesty. Účelem tohoto informačního systému je zjednodušit práci členům sdružení a ulehčit a zrychlit komunikaci mezi jednotlivými pracovními týmy společnosti. Poskytnout dostatečné a aktuální informace všem zájemcům o problematiku, kterou se sdružení zabývá.

Systém je vyvíjen za pomocí objektově orientovaného přístupu v programovacím jazyku PHP, relačního databázového serveru MySQL s podporou uložených procedur a funkcí. Uživatelské rozhraní je implementováno pomocí webových technologii XHTML a CSS. Pro analýzu a návrh systému bylo použito modelovacího jazyka UML.

V práci je nastíněna teorie tvorby Informačních systému za použití daných prostředků a popsán vývoj a implementace jádra Informačního systému občanského sdružení Společnosti Diamantové cesty.

Klíčová slova

Společnost Diamantové cesty, SDC, Informační systém, PHP, OOP, objektově orientované programování, MySQL, XHTML, CSS, třívrstvá architektura systému, UML, SQL, uložené procedury, rozhraní

Obsah

1 Úvod

Tato práce dokumentuje vznik Informačního systému občanského sdružení Společnosti Diamantové cesty.

Občanské sdružení Společnosti Diamantové cesty vzniklo v roce 1998 s cílem sdružovat převážně mladé lidi z celé České Republiky, kteří se snaží a učí navzájem spolupracovat a komunikovat v různých oblastech lidského života.

Má motivace pro výběr tohoto tématu ročníkového projektu byla zejména potřeba tohoto informačního systému v praxi. Nově vzniklý systém bezesporu urychlí a zjednoduší práci a komunikaci jednotlivých týmů sdružení na projektech a akcích. Zároveň systém poskytne dostatečné zázemí pro prezentaci aktuálních informací a dat širší veřejnosti. Volbu tématu také ovlivnil můj kladný vztah k novým webovým technologiím jako je PHP s použitím objektově orientovaného programování, MySQL s využitím uložených procedur a funkcí a požití třívrstvé architektury systému. Mé rozhodnutí také ovlivnila zvědavost, jaké nové poznatky při využití těchto novějších postupů získáme.

V teoretické části je nastíněna problematika vývoje informačních systémů a popis nástrojů pro vývoj a implementaci Informačního systému Společnosti Diamantové cesty (IS SDC).

Praktická část se věnuje popisu všech etap vývoje IS SDC. Popsán je způsob implementace, způsob komunikace jednotlivých části, některé problémy, které se během práce vyskytly a jejich řešení.

2 Teorie

2.1 Informační systémy

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.

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

Je jasná výhoda spojení takovéhoto druhu systému s informačními technologiemi, které přinášejí možnost automatizace při zpracovávání velkých objemů dat a při vnitřní i vnější komunikaci organizace. Významným kladem je rovněž možnost přizpůsobit formu prezentovaných informací charakteru koncových uživatelů systému.

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 – customizovat. Výhodou customizovaného 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. Problémem může být i nedostatečná lokalizace a neschopnost spolupracovat s cizími aplikacemi. 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 customizovaných systému však může přinést zákazníkovi významnou konkurenční výhodu.

Z hlediska účelu rozdělujeme IS na systémy určené pro řízení každodenních operací (operativní IS) a systémy pro podporu rozhodování (manažerské IS). 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 vlastností operativních a manažerských IS vzniká exekutivní IS.

Proces vývoje IS předpokládá spoluprácí se zákazníkem a to již od samého počátku. Pro úspěch IS jsou kritické zejména první fáze jeho vývoje – formulace potřeb zákazníka a analýza. Důležitou podmínkou vzniku úspěšného IS je, aby tvůrce zvládnul alespoň základy oboru jehož se připravovaný IS týká. Míra spoluúčasti zákazníka, důraz na úvodní fáze vývoje a nutnost zvládnutí znalostí oboru aplikace IS jsou jedněmi z hlavních rozdílů oproti tvorbě jiných softwarových produktů.

2.1.1 Životní cyklus IS

Proces vývoje softwarového vybavení definuje při vývoji softwaru otázky *kdo, co, kdy a jak.*Metodika unifikovaného vývoje software (zkráceně UP) je průmyslovým standardem procesu tvorby softwarového vybavení pocházejícím od autorů jazyka UML.

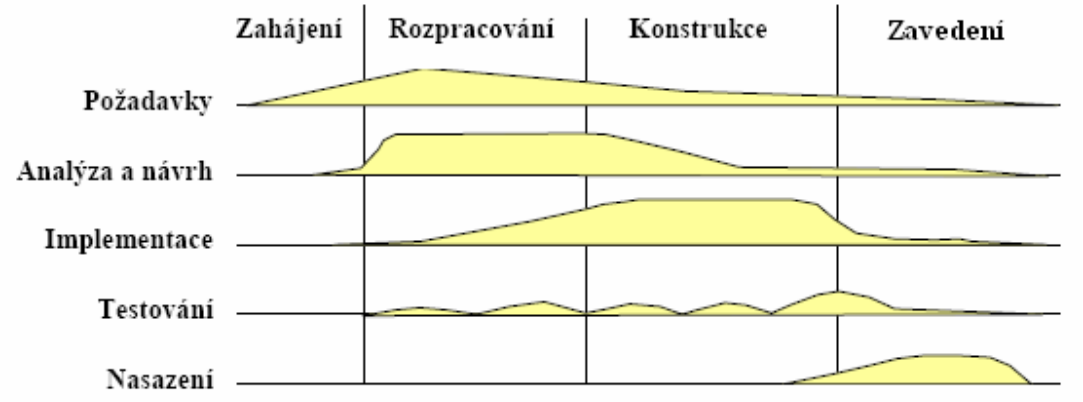
UP popisuje několik základních pracovních postupů, jež určují, co je třeba udělat a způsob jakým toho dosáhnout:

- ***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.

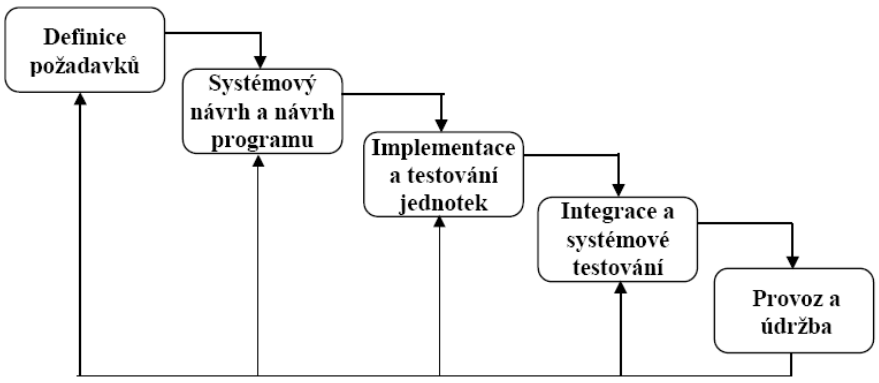
- ***Zahájení.*** Období plánování.
- ***Rozpracování.*** Období architektury.
- ***Konstrukce.*** Počátky provozuschopnosti.
- ***Zavedení.*** Nasazení produktu do uživatelského prostředí.

Na obrázku 2.1 je možno si všimnout rozložení objemu prací vykonaných na každém z základních pracovních postupů.



Obr. 2.1 Rozložení objemu vykonané práce na fáze životního cyklu IS

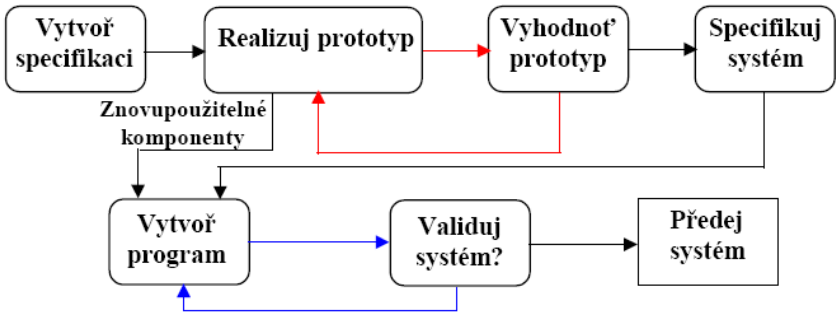
Nejznámější metoda návrhu programového díla je metoda vodopádu (viz obr.2.2). Ta představuje posloupnost: úplná specifikace požadavků, celkový a podrobný návrh, implementace, testování a uvedení do provozu. Výhodou je, že pokrok projektu je stále viditelný. Nevýhodou metody je nutnost návratu k předchozím 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.



Obr. 2.2 metoda vodopádu vývojového cyklu

Z metody vodopádu vychází spirálový model. Ten obsahuje navíc prvky plánování, postupné zpřesňování požadavků, hodnocení rizik, atd. Používaný je i iterační a inkrementální model. Stejně jako u předchozích metod, 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 metoda znamená postupné přidávání samostatné části v každém cyklu.

Jednou z dalších technik odstraňujících nedostatky metody vodopádu je tvorba softwarových prototypů (viz obr. 2.3). 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.



Obr. 2.3 metoda softwarových prototypů vývojového cyklu

2.1.2 Fáze zahájení

Cílem fáze zahájení je odstartování projektu. Hlavní důraz je kladen na specifikaci požadavků a jejich analýzou. Výstupem fáze zahájení jsou popis zainteresovaných osob,

slovník projektu, vyčíslení nákladů, potvrzení proveditelnosti., funkční požadavky a nefunkční požadavky.

2.1.3 Fáze rozpracování

Cílem fáze rozpracování je vytvořit první pokus o systém. 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.

2.1.4 Fáze konstrukce

Cílem fáze rozpracování je vytvořit první pokus o systém. Výstupy fáze konstrukce:

- Odhalit požadavky, které byly přehlédnuty.
- Dokončit analytický model.
- Implementovat počáteční funkční variantu.
- Testovat počáteční funkční požadavky.

2.1.5 Fáze zavedení

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.

2.1.6 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 stanovit, zda je IS vyvíjen pro hromadný prodej nebo je vytvářen na míru podle konkrétních požadavků. Dále 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.

Velmi obtížná bývá specifikace požadavků. Existuje několik studií, které dokazují, že neúspěch v procesu inženýrství požadavků je hlavní příčinou konečného neúspěchu celého

projektu.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.

K dosažení co nejpřesnější specifikace požadavků se při spolupráci s uživatelem používá mnoho ověřených metod. Nejčastěji používanou technikou je dobře připravený rozhovor, ve kterém se tazatel snaží zjistit zejména maximum souvislostí. Průběh rozhovoru se vhodnou formou zaznamenává a následně vyhodnocuje. Méně pracnou variantou je strukturovaný rozhovor – společné vyplňování dotazníku respondentem a moderátorem rozhovoru. Pro oslovení většího množství respondentů se osvědčilo použití klasických dotazníků. Dalšími metodami je například studium dokumentů kolujících v dané organizaci či podniku nebo přímé pozorování provozu na místě. Pokud zákazník dříve používal nějaký IS, vyplácí se vycházet ze zákazníkových zkušeností s tímto systémem.

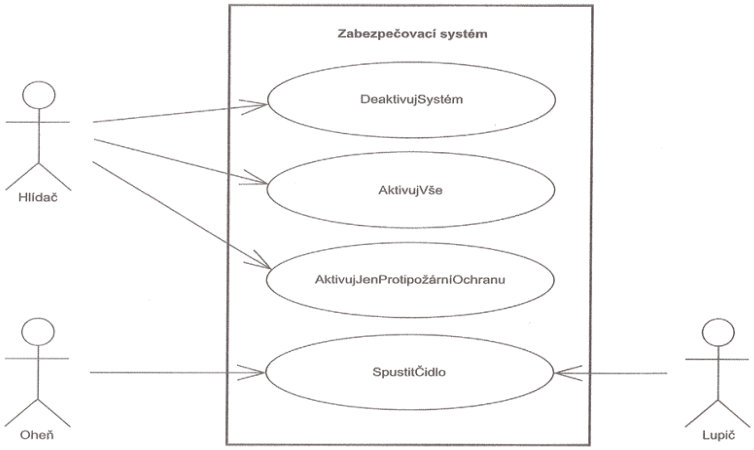
Požadavek lzde definovat jako „specifikaci toho, co mělo být implementováno“. Rozlišují se dva druhy požadavků:

- 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 stal nástroj jazyka UML - modelování případů užití (use case diagram). Případy užití vychází z ucelených činností, které mají v systému probíhat a snaží se zachytit vazby mezi činnostmi a mezi obsluhou a činnostmi. Ukázkový Use Case diagram je zobrazen na obrázku 2.4.

Modelování případů užití je vhodným prostředkem k nalezení:

- Hranic systému (oddělení systému od zbytku světa),
- jeho účastníku (osoby nebo předměty používající daný systém),
- případů užití a jejich specifikace (činnosti, které mohou účastníci vykonávat),
- tvorbě scénářů (specifická cesta případem užití).



Obr. 2.4 Ukázkový use case diagram

2.1.7 Modelování IS

IS bývají často velmi komplexními. Takto složité systémy je obtížné plně pochopit jako celek. K lepšímu porozumění se v praxi vyplácí tvorba různých druhů modelů, jako prostředků pro zjednodušení reality. Při tvorbě modelu dochází k odfiltrování nepodstatných detailů a oddělení různých pohledů na problém. Výhodami jež modely přináší jsou:

- Lepší pochopení požadavků,
- vizualizace systému,
- specifikace jeho struktury,
- vznik základu pro jeho konstrukci,
- zefektivnění činnosti návrhu.

Důležitou technikou zvyšující kvalitu a produktivitu softwarového vývoje je vizuální modelování. Vzniklé modely jsou užitečné pro pochopení problému a současně vhodné pro komunikaci se všemi účastníky vývoje.

2.1.8 Rozhraní systému

Podstata rozhraní spočívá v oddělení funkčnosti části od implementace prostřednictvím třídy nebo funkcí. Rozhraní definuje komunikaci mezi jednotlivými podsystémy.

2.1.9 Modularita IS

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

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

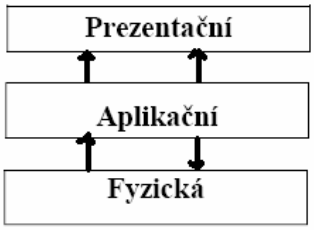
2.1.10 Strukturovaný návrh IS

Cílem strukturovaného návrhu IS je rozdělit celý systém na jednotlivé části - podsystémy, tak aby tyto podsystémy byly na sobě funkčně nezávislé. Výhodou takovéhoto rozdělení systému je jednoduší údržba, redukce šíření chyb.

Podsystém je například kolekce tříd s dobře definovaným rozhraní pro ostatní podsystémy, nebo fyzická část systému (databáze). Vztahy podsystému mohou být realizovány jako vrstvy, nebo sekce. Architektury s vrstvami mohou být buď otevřené, nebo uzavřené.

2.1.10.1 Třívrstvá architektura IS

Třívrstvá architektura IS poskytuje oddělení zobrazovací vrstvy (**presentation layer**) od aplikační vrstvy (**business layer)** a fyzického uložení dat (**data layer**). Vrstvy přistupují navzájem a komunikují mezi sebou pomocí rozhraní dané vrstvy.



Obr. 2.5 Třívrstvá architektura systému

2.2 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 90tých 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.

UML je formován jako standard jazyka pro analýzu a design objektově orientovaných systémů, založených na vydatné zkušenosti a nejlepších zvycích. V důsledku toho je tento standard rychle přijímaný (existuje řada školení, počítačové nástroje (CASE), knihy, apod.). Důležitou vlastností UML je jeho snaha býti toliko jazykem, nikoli 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.

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 (8 základních typů diagramů):
  1. diagram tříd (class diagram)
  2. diagram stavového stroje (state machina diagram)
  3. diagram spolupráce (collaboration diagram)
  4. diagram případu užití (use class diagram)
  5. diagram posloupnosti (sequence diagram)
  6. diagram činnosti (activity diagram)
  7. diagram komponent (komponent diagram)
  8. diagram zavedení (deployment diagram)
- **formální pravidla pro správnost vytvářených modelů**, vyjádřena formou omezení v tzv. „Object Constrain Language“ (OCL**)**

- předdefinovaná **doménová rozšíření** základních nástrojů, umožňujících použití UML v mnoha různých doménách k popisu tam specifikovaných pojmů a myšlenek. V současnosti jsou ve standardu jazyka 2 předdefinovaná doménová rozšíření a to:
  1. Pro návrh systémů, pracujících v reálném čase (real-time extension),
  2. pro popis podnikových procesů (business processes extension).

Principy UML jsou postaveny na „objektovém paradigmatu“, zahrnující výše diskutované vlastnosti objektů a tříd, jakými jsou zejména:

- Zapouzdření (encapsulation),
- dědičnost (inheritance),
- vícetvarost/zobecnitelnost (polymorfismus).

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,

2.2.1 Předměty

Předměty jsou elementy zpracovávaného modelu, jež jsou následně členěny do několika navzájem rozdílných podkategorií. Prvním typem podkategorií jsou takzvané **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. Například se jedná o obdélníky, elipsy, kružnice či jednoduché zdánlivě trojrozměrné tvary, například krychle či kvádry. Každý smysluplný UML diagram by měl obsahovat alespoň dvě strukturní abstrakce, při jedné abstrakci totiž modelování ztrácí svůj hlavní smysl - popis vztahů mezi jednotlivými objekty.

Kromě strukturních abstrakcí patří mezi předměty i takzvaná **chování**, která v UML diagramu prezentují interakce, tj. vzájemné komunikace mezi jednotlivými objekty. Pomocí chování lze také modelovat stavový stroj, u nějž se stavy specifikují pomocí přechodů, událostí a aktivit. Chování se v UML diagramu většinou vyznačuje pomocí různým způsobem konstruovaných a různě zakončených šipek či propojovacích čar.

Mezi předměty patří i takzvané **seskupení**, které podle potřeby modelu graficky seskupuje části diagramu na nižší úrovni. Většinou se jedná o takzvané balíčky, jež mají tvar stylizované kancelářské složky s popisem umístěným v levé horní části obdélníku (zobrazení seskupení se však může v různých prostředích odlišovat). Seskupení nejsou v současné době v některých dále popisovaných aplikacích použitelná (tj. nelze vytvářet hierarchické diagramy), což je pro tvorbu rozsáhlejších grafů velké omezení.

Posledním a současně velmi jednoduchým typem předmětů jsou **poznámky**, které blíže specifikují vlastnosti a chování dalších elementů UML diagramu. Graficky jsou poznámky na

prakticky všech typech UML diagramů většinou vyvedeny žlutě vyplněným obdélníkem s ohnutým rožkem.

2.2.2 Relace

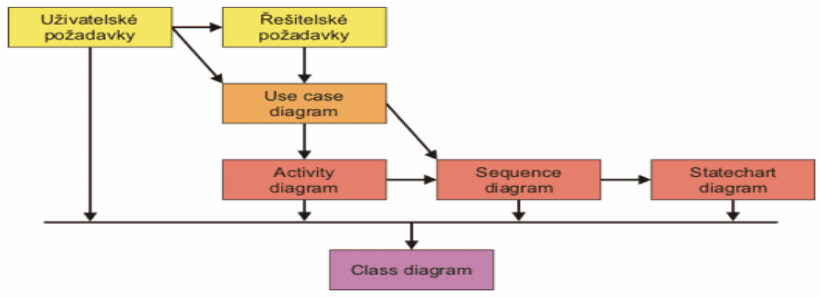
Vzhledem k tomu, že v grafech je zapotřebí předměty různým způsobem navzájem propojovat, jsou v jazyku UML specifikovány i relace, tj. vztahy mezi různými předměty. V UML jsou rozeznávány následující podtypy relací (z čistě jazykového hlediska patří relace mezi základní elementy UML):

- **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, které jsou často používány 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 (inheritance).
- **Realizace** - jedná se o druh vztahu, ve kterém jeden předmět představuje dohodu, za jejíž splnění je odpovědný jiný předmět. V objektově orientovaných jazycích se realizace vytváří pomocí rozhraní (interface) - samozřejmě za předpokladu, že daný OOP jazyk rozhraní podporuje.

2.2.3 Diagramy

Dokumentace v UML se nemusí skládat pouze z (více, či méně formálních) textů. Ze základních lexikálních elementů lze vytvářet dvourozměrné diagramy – rozmístíme na ploše určitou sadu elementů a případně je propojíme pomocí spojek. Teoreticky lze použít libovolné elementy a spojky, z hlediska unifikace je však výhodné, pokud zvolíme určitou sadu elementů, která je vhodná pro vyjádření některého smysluplného pohledu na modelovaný systém. Touto volbou jsme de facto definovali typ diagramu.

Pro přímé propojení modelů UML vytvořených v CASE a implementačního prostředí konkrétního IDE jsou obvykle využívány jen statické struktury Class diagramu, který specifikuje třídy, jejich atributy a operace, jejich vzájemné vztahy. Z těchto specifikací lze pak generovat kostry.



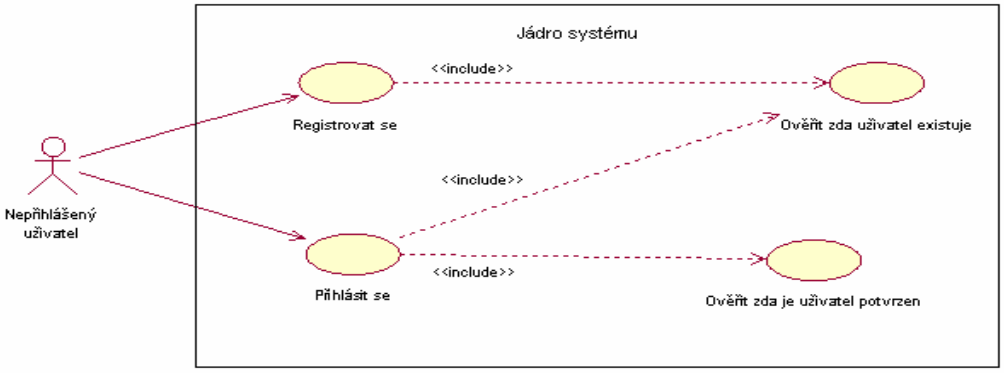
Obr. 2.6 Použití jednotlivých typů UML diagramů ve vývojovém procesu

2.2.3.1 Use Case diagram

Případy užití neboli *Use Case* se popisují jako „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. V této fázi analýzy se ještě nezabýváme technologickými aspekty řešení a používáme pouze pojmy přirozeného jazyka a termíny z problémové domény, abychom co nejvěrněji a pro zákazníka co nejsrozumitelněji načrtli funkční skeleton systému.

Případy užití jsou zachyceny ve vizuální nebo textové podobě. Diagramy případů užití (UseCase diagrams) 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).



Obr. 2.7 Příklad jednoduchého UseCase diagramu

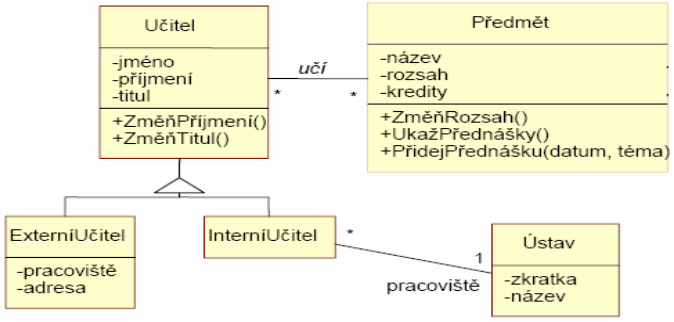
2.2.3.2 Diagram tříd

Diagram tříd (obrázek 2.7) 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 *, samotná * znamená totéž jako 0..*. Pokud je zapotřebí si o vztahu něco pamatovat, požívají se tzv. přidružené třídy (atributy vztahů). Speciální druhy vztahů představují agregace a generalizace:

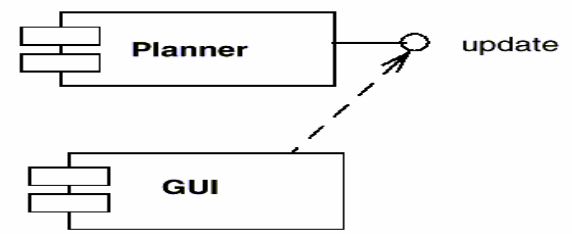
- **Agregace** (aggregation) – je druh vztahu, kdy jedna třída je součástí jiné třídy - vztah typu celek/část.
- **Kompozice** (composition) – je silnější druh agregace - u kompozice je část přímo závislá na svém celku, zaniká se smazáním celku a nemůže být součástí více než jednoho celku.
- **Generalizace**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



Obr. 2.7 Příklad jednoduchého diagramu tříd

2.2.3.3 Diagram komponent

Diagram komponent (obrázek 2.8) ukazuje rozhraní, komponenty a vztahy mezi nimi. Komponenta je fyzická vyměnitelná část systému, která realizuje určitou množinu rozhraní.



Obr. 2.8 Příklad jednoduchého diagramu komponent

2.3 Objektově orientované programování

Moderní programovací jazyky obvykle při vývoji softwaru podporují nebo dokonce vyžadují objektově orientovaný přístup. Objektově orientované programování (OOP) se snaží k usnadnění vývoje programů použít vzájemné vztahy a vlastnosti objektů, které jsou při vývoji programů zařazeny do určitého systému.

Objektově orientované programovací jazyky jsou charakterizovány těmito vlastnostmi:

- **Zapouzdření**: Základním pojmem objektového přístupu je objekt. Je to záznam (skupina proměnných) a skupina funkcí (které manipulují s proměnnými záznamu). Objekt je zapouzdřen, tj. je isolován od ostatních částí programu (i od jiných objektů).

- **Dědění**: Definice objektu (a to, jak se používá) se vytváří pomocí hierarchie dílčích definic (zvaných třídy) tak, že nižší člen hierarchie dědí všechny vlastnosti (proměnné a funkce) od svého předchůdce.
- **Polymorfismus**: Vyžadujeme-li akce téhož jména od různých objektů jedné hierarchie, může být tato akce u různých objektů realizována různě; tj. reakce objektů na tentýž podnět může být jiná.

2.3.1 Objekt

Objekty v OOP se připodobňují reálným objektům. Každý objekt obsahuje všechny své vlastnosti (tj. proměnné), některé z nich jsou veřejné (lze s nimi manipulovat z vnějšku), jiné jsou soukromé (určují stav objektu, mohou s nimi manipulovat pouze funkce objektu). Každý objekt má své funkce, které určují jeho reakce, funkce se také rozlišují na veřejné (mohou být použity z vnějšku) a na soukromé (mohou být použity pouze jinou funkcí téhož objektu).

2.3.2 Zapouzdření

Myšlenka objektů vychází z reálných objektů, příkladem může být vypínač, ten má dva stavy - "vypnuto" a "zapnuto" a dva způsoby chování - "vypnout", "zapnout". Vše je uzavřeno uvnitř a není možnost přímo měnit stav vypínače bez použití jeho metod, jeho způsobů chování. Tato vlastnost - uzavřenost, **zapouzdření** (encapsulation) je jeden ze základních pilířů OOP, zabezpečuje stavové členské proměnné proti neoprávněnému přístupu metod mimo objekt. Další výhodou je to, že skrývá vnitřní implementaci jednotlivých metod - lze např. kompletně změnit metodu "zapnout" vypínače, aniž by bylo třeba měnit cokoliv jiného, ostatní objekty nadále volají metodu "zapnout" stejným způsobem a není nutné je přizpůsobovat.

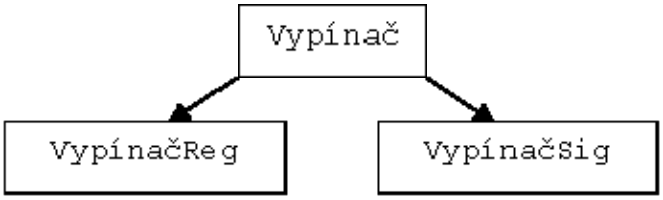
2.3.3 Třída

Budeme-li uvažovat jako příklad objektů reálného světa vypínače, můžeme objevit množství takových, lišících se např. v barvě, velikosti nebo materiálu, avšak se stejnou funkčností a principem. Tyto podobné objekty lze popsat jednotným obecným zápisem, který slouží jako vzor při vytváření jednotlivých objektů, a který se nazývá **třída** neboli objektový typ. Objekty jsou pak **instancemi** této třídy. Např. objekt s červeným vypínačem by byl instancí třídy "*Vypínač*" a měl by atribut "*barva*" nastavený na hodnotu "*červená*".

Třída jako taková, ačkoliv je pouze předpisem pro tvorbu instancí, může mít také vlastní atributy a metody, ty nazýváme statické. Vyskytují se jen jednou v celém systému a všechny instance je sdílí. Lze s nimi operovat, i když neexistuje žádná instance třídy, což může být někdy výhodné.

2.3.4 Dědičnost

Mechanismus **dědičnosti** znamená, že je možné od jakékoliv obecnější třídy odvodit třídy konkrétnější přidáním atributů a metod nebo jejich překrytím (předefinováním). Vzniká tak například hierarchická struktura (viz obr. 2.9).



Obr. 2.9 Příklad dědičnosti tříd

Následník se nazývá **potomek** a předchůdce **rodič**, každý potomek dědí, tzn. přejímá všechny atributy a metody od rodiče. Tímto postupem lze ušetřit opětovné psaní společného kódu a vyhnout se chybám - používá se ověřený kód rodiče. Další související vlastnost je vzájemná zastupitelnost objektů v dědické hierarchii. Např. ke každému vypínači lze přistupovat jako by byl instancí nejobecnější třídy "*Vypínač*", což přináší další výhody při správě objektů.

2.3.5 Polymorfismus

Polymorfismus znamená, že jméno akce (metody) je sdíleno třídami ve stromu dědičnosti a ta je implementována způsobem, který jí na dané úrovni přísluší.

2.4 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 už 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.

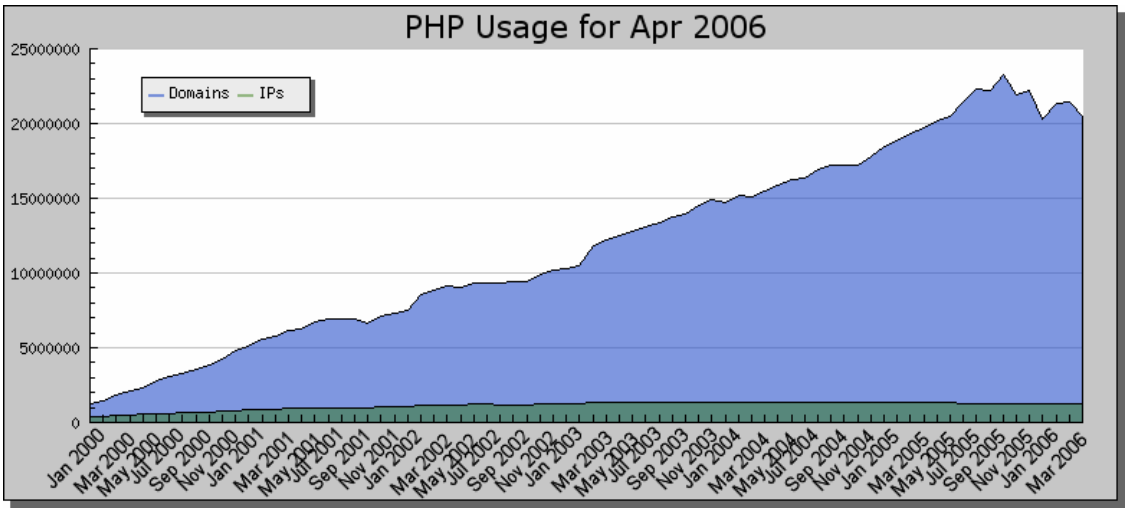
PHP skript se do HTML kódu vkládá mezi značky <*?* a ?> nebo <?php a ?>. Veškerý kód mimo tyto značky je považován za HTML a posílán na výstup. Tento princip umožňuje libovolné střídání HTML kódu s PHP skriptem, avšak značně snižuje přehlednost výsledného zdrojového

textu. Syntaxe PHP je velmi podobná jazykům jako C nebo Perl. Díky této vlastnosti nečiní programátorům problémy se jej rychle naučit.

Uživatelské relace (session) je možnost jak v PHP předávat informace mezi různými skripty, bez nutnosti odesílání dat na server metodami jako jsou GET nebo POST. Potřebná data jsou uložena v paměti na serveru, kde se nachází skripty a jsou označena určitým identifikátorem (session id). Ten je buď automaticky uložen u klienta v cookies nebo přenášen jako jedna z hodnot pole GET. Cookies jsou malé soubory u uživatele, kam mohou webové stránky zapisovat svá data.

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 lepší podpory objektové orientace). Stáhnout jej lze zdarma z www.php.net.



Obr. 2.9 obrázek převzatý ze serveru www.php.net ukazující nárůst obliby PHP v posledních pěti letech

2.4.1 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). PHP se stále hojně používá procedurální programování (data jsou uloženy v proměnných, ty se pak různě modifikují a využívají). Pro řešení komplexnějších úloh a systémů je však objektově orientovaný přístup nezbytný. Stále více programátoru již také není ochotno při přechodu z jiného vyspělého jazyka (jako Java) se vrátit zpátky k procedurálnímu programování.

Narozdíl od procedurálního programování, v OOP jsou data a funkce navzájem svázány do **objektů**. **Třída** je šablonou objektu, volání třídy se pak označuje jako **instance**.

V příkladu 2.1 objektově orientovaného kódu v jazyku PHP jsme vytvořili objekt obj třídy Myclass a voláme veřejnou metodu getText(). Ta vrátí obsah proměnné text, který se inicializoval při vytváření objektu obj.

```php
<?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 2.1 Třída v PHP

Více o OOP v PHP najdete na http://cz2.php.net/zend-engine-2.php.

2.5 MySQL

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. MySQL je stejně jako PHP Open Source. Mezi hlavní přednosti MySQL patří:

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

Aktuálně se MySQL nachází ve verzi 5.1. MySQL verze >= 5 přináší řadu vylepšení a výhod oproti starším verzím. Mezi nejdůležitější vylepšení patří podpora uložených procedur a funkcí. MySQL server je volně dostupný na www.mysql.com.

2.5.1 MySQL a uložené procedury a funkce

Implementace uložených procedur do MySQL dělá z tohoto serveru opravdu robustní a plnohodnotný databázový nástroj. Uložená procedura je sada příkazů SQL, které jsou uložené na serveru a zkompilované pro rychlejší použití. Důvody pro zavedení úložených procedur jsou:

- **Unifikace** požadavků od jednotlivých klientů,
- ulehčení **správy** databázových aplikací,
- **zabezpečení** serveru (procedury mohou samy kontrolovat počet, typ, atd),
- umožňují vyhnout se **SQL injection**,
- protože jsou zkompilovány, běží **rychleji**, než kdyby se příslušný kód vykonával příkaz po příkazu z klientské aplikace.

Zkompilování procedury znamená, že server si pro uloženou proceduru vytvoří a spravuje tzv. *prováděcí plán*. Díky tomu je subsystém serveru zvaný *optimalizátor* obvykle schopen najít nejrychlejší způsob, jak uloženou proceduru provést, a potom tento způsob opakovaně používat při jednotlivých voláních této uložené procedury.

Uložené procedury mohou dělat většinu z toho, co "běžné" příkazy. V příkladu 2.2 je jednoduchá uložená procedura, která vrátí obsah tabulky *uzivatele*:

```
create procedure vratUzivatele()
begin
  select * from uzivatele;
end
```

Uloženou proceduru zavoláme pomocí příkazu:

```
call vratUzivatele();
```

Příklad 2.2 Jednoduchá uložená procedura v MySQL

2.6 XHTML

HTML je jazyk pro publikování hypertextu na WWW. Je to nepatentovaný formát založený na SGML. HTML využívá **tagy** ke strukturování textu do nadpisů, odstavců atd. Vývoj HTML již skončil. XHTML je nástupce HTML založený na XML. Rozlišujeme 3 druhy XHTML:

- XHTML 1.0 Strict,
- XHTML 1.0 Transitional,
- XHTML 1.0 Frameset.

XHTML dokument musí 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.

Více o XHTML nalezna například na
http://www.webtvorba.cz/xhtml/uvod-do-xhtml.html.

2.7 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.

CSS se využívá k formátování obsahu HTML, XHTML a XML dokumentů. 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. Důvod pro použití CSS je oddělit data od jejich formátování.

Více o CSS například na http://www.webtvorba.cz/css/uvod-do-css.html.

3 Analýza

3.1 Stávající stav

Současný webový portál občanského sdružení obsahuje jen statické informace o společnosti a čím se zabývá. Na stránkách najdeme kalendář akcí a novinky, jenž se musí editovat ručně. Systém neposkytuje žádné služby pro registrované uživatele. Systém je postaven na univerzálním redakčním systému SAMBA, který však zdaleka nevyhovuje specifickým požadavkům sdružení.

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 bylo uskutečněno množství schůzek z různými členy sdružení, jako jsou vedoucí projektů, členové překladatelských, PR a jiných týmů, správci systému, cestující učitelé a běžní uživatelé.

Dále byly prostudovány současné informační systémy podobných společností v Evropě. Důraz byl při tom kladen na podmínky střední a západní Evropy. Při návrhu požadavků se speciální tým lidí, který byl k tomuto účelu určen, zaměřil na podobné informační systémy na Slovensku, v Polsku, Maďarsku, Rakousku a Německu.

Výsledkem analýzy stávajícího stavu a studiem obdobných systému v zahraničí byly funkční a nefunkční požadavky systému.

3.2 Požadavky na IS SDC

Na základě detailní a podrobné analýzy stávajícího stavu byly vytvořeny požadavky na Informační systém Sdružení Diamantové cesty.

3.2.1 Funkční požadavky IS SDC

- Systém bude dostupný přes síť Internet pomocí osobního počítače nebo mobilního zařízení.
- Uživatelské rozhraní bude obsahovat menu pro výběr stránky a obsahovou část.
- Systém bude poskytovat veřejnou část dostupnou ze sítě Internet nepřihlášeným uživatelům.
- Veřejná část bude obsahovat tyto části:
    - Veřejné články a novinky,
    - kalendář veřejných akcí,
    - úvodní informace o společnosti,
    - kontakty center a zástupců projektů a center,
    - odkazy na veřejnou část projektů a center,

    - veřejnou fotogalerie,
    - poradnu cestujících učitelů,
    - veřejně dostupné soubory ke stažení,
- Systém bude obsahovat neveřejnou část přístupnou registrovaným uživatelům z internetu.
- Neveřejná část bude umožňovat se zaregistrovat novým neregistrovaným uživatelům.
- Registrace bude kompletní až po potvrzení neregistrované osoby kompetentním registrovaným uživatelem.
- Neveřejná část bude umožňovat přihlášení již registrovaného uživatele.
- Pro přístup k interní části systému bude nutné se přihlásit.
- Přihlášený uživatel bude mít volně přístup ke všem údajům, ke kterým má *právo přístupu*.
- Hotový informační systém bude obsahovat tyto interní části:
    - On-line knihovna,
    - kalendář interních akcí i pro jednotlivé projekty a týmy,
    - novinky týkající se týmů, projektů i celé společnosti,
    - seznam všech center společnosti,
    - stránky projektů a center,
    - mailing list všech registrovaných uživatelů s možností filtru jen pro členy, týmů a podobně,
    - fotogalerie s možností fotogalerií pro jednotlivé týmy a projekty,
- Systém bude umožňovat fulltextové vyhledávání v článcích a novinkách.

3.2.2 Nefunkční požadavky IS SDC

Na základě analýzy stávajícího stavu a funkčních požadavků systému byly vytvořeny tyto nefunkční požadavky systému:

- Informační systém bude tvořen jádrem, které bude poskytovat rozhraní pro přístup pro moduly využívající služeb tohoto jádra.
- Předpokládaný počet registrovaných uživatelů bude 500.
- Předpokládaný maximální počet on-line uživatelů současně bude 100.
- Aplikační logika systému bude napsána v programovacím jazyku PHP5 s využitím objektově orientovaného přístupu.
- Rozhraní pro přístup k modulům bude poskytovat třída CoreInterface jádra systému.
- Databázovou vrstvu bude tvořit relační databázový server MySQL 5.
- Rozhraní k datům databázového serveru budou zajišťovat uložené procedury serveru MySQL 5.
- Znakové kódování uživatelského výstupu bude UTF-8.
- Zobrazovací vrstva bude implementována pomocí značkovacího jazyka XHTML 1.0 traditional s využitím jazyka PHP5.
- XHTML kod bude splňovat podmínky standardu W3C.
- Grafické formátování výstupu bude implementováno pomocí jazyka CSS.

- Systém bude testován a bude podporovat prohlížeče Internet Explorer 5.5 a vyšších (MS Windows), FireFox 1.0 (Linux/Windows), IE5/Mac OS X, Opera 8.
- Formátovaný výstup bude přizpůsoben prohlížečům kapesních počítačů (PDA) a telefonu s vestavěným prohlížečem webových stránek (Smartphonům).
- Systém bude napsán s důrazem na bezpečnost dat a ochranu údajů.

3.3 Požadavky na jádro informačního systému

Na základě nefunkčního požadavku na informační systém „**informační systém bude tvořen jádrem, které bude poskytovat rozhraní pro přístup pro moduly využívající služeb tohoto jádra**“, funkčních požadavků a analýzy stavu systému byla provedena analýza požadavků na jádro systému. Na základě této analýzy byly zformulovány požadavky na jádro systému.

3.3.1 Funkční požadavky jádra IS SDC

- Jádro bude poskytovat uživatelské rozraní pro:
  - Připojení / odebrání modulů systému,
  - úpravy souboru stávajících modulů,
  - administraci menu jednotlivých modulů,
  - administraci práv uživatelů,
  - administraci oprávnění k přístupu k modulům,
  - registraci uživatele,
  - editaci uživatele,
  - přihlašování uživatele.
- Jádro bude umožňovat registraci nových uživatelů.
- Registrovaný uživatel bude mít přístup do systému až po potvrzení přístupu kompetentním uživatelem.
- Jádro bude definovat tyto typy uživatelů pro přístup k modulům:
  - **Nepřihlášený uživatel**
    Nepřihlášený uživatel nebude mít práva přístupu k modulu, nebo bude mít přístup jen k veřejným stránkám modulu, pokud tyto stránky existují.
  - **Obyčejný uživatel**
    Obyčejný uživatel je úspěšně přihlášený uživatel, který bude moci v jádru:
    - měnit svůj profil,
    - odhlásit se ze systému.
  - **Kompetentní uživatel**
    Kompetentní uživatel bude mít v jádře systému všechna práva obyčejného uživatele a moci:
    - Potvrzovat, či blokovat nové, či stávající uživatele,
    - přiřazovat práva kompetentního uživatele pro obyčejného uživatele pro všechny moduly.

- **Administrátor**
  Administrátor bude mít práva kompetentního uživatele a bude moci:
  - Přidávat či odebírat moduly systému,
  - spravovat stávající moduly systému,
  - spravovat strukturu menu,
  - spravovat stránky systému, propojovat je se soubory systému a položkami menu,
  - přidávat, odebírat soubory systému,
  - spravovat práva všech uživatelů.

- Při pokusu uživatele o přístup do části modulu (stránky), ke které nemá tento uživatel právo přístupu, zobrazí se alternativní stránka, ke které má uživatel právo přístupu. Pokud tato stránka neexistuje, vypíše se chyba.

3.3.2 Nefunkční požadavky jádra IS SDC

- Informační systém bude tvořen jádrem, které bude poskytovat rozhraní pro přístup pro moduly využívající služeb jádra v podobě třídy CoreInterface.
- Jádro bude tvořeno třemi vrstvami (prezentační, business, datovou vrstvou), které oddělují data od zpracování dat a jejich uložení.
- Databázovou vrstvu bude poskytovat databázový server MySQL 5.
- Rozhraní k datům databázové vrstvy budou zprostředkovávat uložené procedury MySQL serveru, které budou volány business vrstvou jádra.
- Moduly budou mít přístup k datům jádra jen přes rozhraní business vrstvy.
- Rozhraní pro moduly bude implementováno v OOP jazyka PHP5.
- Business vrstva bude poskytovat rozhraní pro prezentační vrstvu. Toto rozhraní bude voláno funkcemi prezentační vrstvy.
- Business vrstva bude obsahovat aplikační logiku.
- Předpokládaný počet registrovaných uživatelů bude 500.
- Předpokládaný maximální počet on-line uživatelů současně bude 100.
- Business vrstva bude napsána v programovacím jazyku PHP5 s využitím objektově orientovaného přístupu.
- Znakové kódování MySQL tabulek a uložených procedur databáze bude UTF-8.
- Znakové kódování výstupu presentační vrstvy bude UTF-8.
- Zobrazovací vrstva bude implementována pomocí značkovacího jazyka XHTML 1.0 traditional s využitím jazyka PHP5.
- XHTML kód bude splňovat podmínky standardu W3C.
- Grafické formátování výstupu bude implementováno pomocí stylů CSS.
- Systém bude testován a bude podporovat prohlížeče Internet Explorer 5.5 a vyšších (MS Windows), FireFox 1.0 (Linux/Windows), IE5/Mac OS X, Opera 8.
- Formátovaný výstup bude přizpůsoben prohlížečům kapesních počítačů (PDA) a telefonu s vestavěným prohlížečem webových stránek (Smartphonům).
- Jádro bude napsáno s důrazem na bezpečnost dat a ochranu údajů.

3.4 Diagram komponent pro jádro IS SDC

Na základě požadavků na IS a jádro systému byla navržena třívrstvá architektura systému, která odděluje data od aplikační logiky a zobrazení. Vrstvy definují rozhraní pro přístup jiných vrstev k dané vrstvě (viz diagram 3.1).

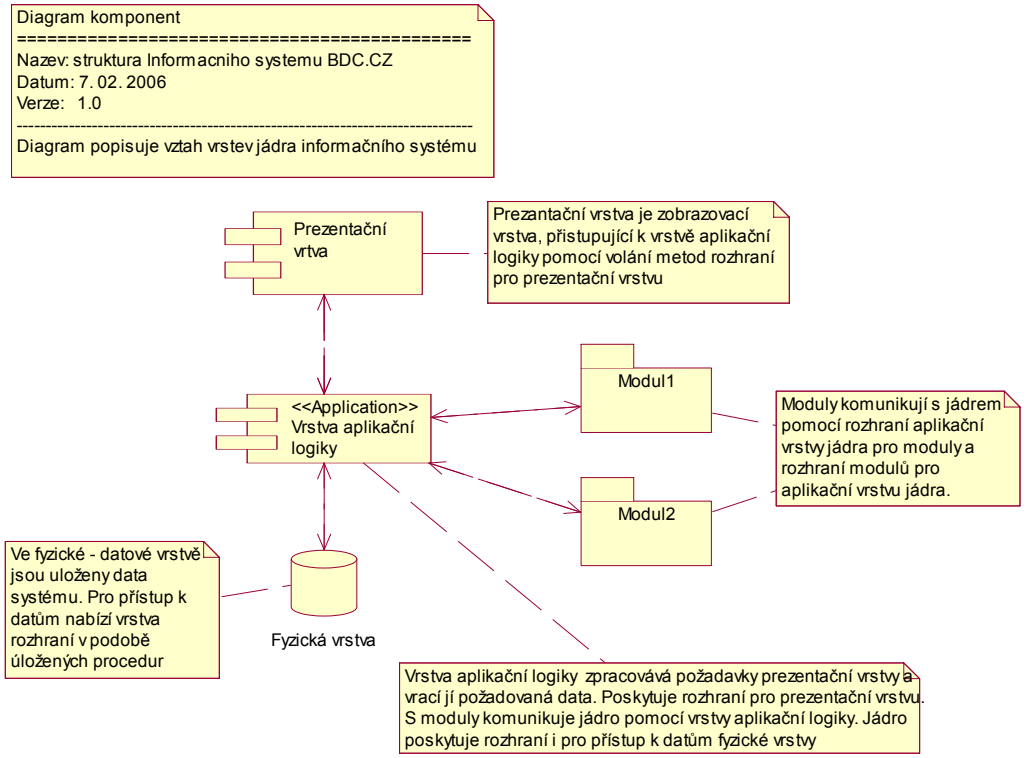


Diagram 3.1 Diagram komponent pro jádro IS

3.5 Use Case diagram pro jádro systému

Na základě analýzy funkčních požadavků jádra systému vznikl Use Case diagram jádra systému, který je díky své rozsáhlosti přidán ke zprávě jako příloha č 1.

3.6 Diagram tříd pro datovou vrstvu jádra systému

Na základě analýzy požadavků na informační systém a jádro systému byla navrhnuta struktura tříd pro datovou vrstvu systému. Diagram tříd je díky své rozsáhlosti přiložen jako příloha zprávy č. 2.

4 Implementace

4.1 Třívrstvá architektura

Systém je pro svou rozsáhlost , snadnější implementaci a pozdější údržbu navržen jako třívrstvý. Vrstvy od sebe oddělují datovou část (**datová vrstva**) od aplikační části (**business vrstva**) a zobrazení (**presentation vrstva**). Jednotlivé vrstvy mezi sebou komunikují pomocí rozhraní těchto vrstev.

4.2 Datová vrstva jádra

Všechny tabulky jádra obsahují prefix *core_* následující názvem tabulky v angličtině.. Stěžejními tabulkami jsou tabulky *core_user* a *core_modul*, nesoucí informace o uživateli a modulu.

4.2.1 Uložené procedury a funkce

Přístup k datům je možný pouze přes uložené procedury a funkce jádra. Funkce jádra jsou vždy privátní a nelze je volat z vnějšku datové vrstvy. Funkce jádra mají prefix *core_*. Procedury jádra se dělí na privátní procedury a procedury rozhraní. Privátní začínají prefixem *core_* , procedury rozhraní prefixem *intface_*. Procedury i funkce využívají funkcí systému MySQL a proto vracejí nejen data tabulky, ale i již zpracovaná data. Další vrstva se tak již nezabývá nad zpracováním dat, ale už s nimi rovnou disponuje. Například příkaz procedury *intface_getCorePresentationFile*, která vrací cestu k hlavnímu souboru presentační vrstvy (výpis 4.1)

```
select concat(cm.cesta,'/',cmf.umisteni,'/',cmf.nazev) into presFile
    from `core_modul_files` cmf, core_modul cm
    where cmf.id_stranky = INidPage and
    cm.id_modulu = idModulu;
```

Výpis 4.1 Sestavovaní výsledného dotazu z atomických hodnot

vrací již kompletní cestu k souboru, nikoliv jen její složky.

Procedury *core_addError()*, *core_learError()* a *core_getError()*, které se starají o správu chyb v procedurách a funkcích. Systém na začátku každé procedury rozhraní nejprve vyčistí tabulku chyb pomocí procedury *core_clearError()*. Pokud se při zpracování vyskytne chyba, systém ji přidá do tabulky chyb pomocí funkce *core_addError()*. Nepřidá však text chyby ale jen identifikátor chyby. Procedura *core_getError()* pak vrátí již textovou podobu všech chyb.

Výhodou tohoto řešení je, že v požadovaném okamžiku dostaneme úplný přehled všech chyb a také například pro neexistujícího uživatele bude existovat vždy jen jedno chybové hlášení, nikoliv pokaždé jiné.

4.2.2 Rozhraní pro přístup k datům datové vrstvy

Rozhraní pro komunikaci datové vrstvy s okolím zajišťují procedury rozhraní (prefix *intface_*), které nejprve ověří správnost parametrů funkce, čímž se velmi zvýší bezpečnost dat. Není možné tak využít například *SQL injection*, či podobné metody pro získávání dat z databáze a taky se zamezí nechtěným datům při špatně zvolených parametrech.

Po zpracování požadavku procedura vrací nejen daný požadavek, ale vždy i proměnnou *error*, která vrací indikuje chybu a vrací případné chybové hlášení.

4.3 Business vrstva jádra

Business vrstva obsahuje veškerou aplikační logiku jádra. Zpracovává požadavky jiných modulů, nebo zobrazovací vrstvy a vrací požadovaná data. Business vrstva se stará o veškeré zpracování a zotavení se s chyb, ošetření vstupních dat uživatele, zpracování formulářů, připojení všech aktuálních částí systému a zpřístupnění jejich rozhraní.

4.3.1 Přístup k datové vrstvě jádra

Datová vrstva jádra poskytuje rozhraní pro přístup k vrstvě v podobě uložených procedur. Pro přístup k datové vrstvě z business vrstvy za využití prostředků PHP je nutné použít funkce php balíku MySQLi (MySQL improved Extension), které rozšiřují a zdokonalují tradiční funkce pro přístup k MySQL. MySQL umožňuje i volání uložených procedur databáze.

Pro přístup k datové vrstvě byla navržena třída *Data*, která zapouzdřuje práci s datovou vrstvou. Zabezpečuje připojení k databázi, volání uložných procedur a testovací funkce.

Pro zpracování výsledků byla navržena třída *Sproc* jenž dokáže zpracovat chyby vrácené datovou vrstvou a vrátit požadovaná data.

Výpis kódu 4.2 pro přihlašování uživatele z metody *login()* třídy *User* ukazuje jeden z možných přístupu k datové vrstvě.

```
-- zavolá uloženou  proceduru intface_login a výsledky předá objektu typu Správ
$result=Data::callSP("intface_login('".$usr."','".$pass."')");
// zjistí zda se vyskytla chyba u první (tady jedinné) funkce
if (!$result->isFError()) {
   $_SESSION[self::ID_UZIVATELE] = $result->getFData();
    return '';
 }
 Else // pokud se vyskytla chyba, zavoláme ji. O zpracování se postará
třída Error
 { // zpracovat chybu
    return $result->getFError();
 }
```

Výpis 4.2 Příklad přístup k datové vrstvě

4.3.2 Zpracování Chyb

O zpracování všech chyb se stará třída Error(), ta se postará aby bylo chybové hlášení vypsáno ve správném formátu a na místě obsahu (ne například na místě menu). Třída rozhoduje, zda se zobrazí jen chybové hlášení, nebo chybová stránka. Všechny chyby business vrstvy se zpracovávají třídou *Error*.

4.3.3 Zpracování formulářů

O formuláře se stará třída *Form*. Ta poskytuje sadu elementů pro vytvoření formuláře a metody na zpracování formuláře. Vývojář jen vytvoří patřičný element a už se nestará o zpracování chyb, mezí, nesprávných údajů a podobně. Pokud formulář projde ověřením data jsou dále předána buď jádru, nebo příslušnému modulu ke zpracování.

Následující příklad ukazuje vytvoření přihlašovacího formuláře do systému. V metodě startForm() je implementována funkce pro ověřování pravosti odeslaného formuláře, která vygeneruje náhodný HASH kód pro daný formulář a ten porovnává s HASH kódem v $_SESSION. Tento mechanizmus zabraňuje nežádoucímu použití formulářů.

```
$ci->form->startForm('prihlaseni', 'core');
$ci->form->addClassFieldSet('login', 'Přihlášení uživatele');
$ci->form->addField('Uživatelské jméno',  '1010', 'text', 'login');
$ci->form->addField('Heslo','1020', 'password', 'passwd');
$ci->form->addButton('form_submit','form-submit-login','submit',
'Přihlásit se');
$ci->form->endFieldSet();
$ci->form->endForm();
```

Výpis 4.3 Formulář z elementů tohoto formuláře

Tento kus kódu zobrazí kompletní přihlašovací formulář (viz obr. 4.1). Jednotlivých polích formuláře jsou již nastaveny meze, typy vstupů a po odeslání se data automaticky ověří podle zadaných vlastností (2. parametr s textovým řetězcem obsahujícím čísla). Tyto čísla určují například viditelnost, zda ověřovat a jakým algoritmem povinnost pole a jeho velikost.

Obr. 4.1 Ukázka formuláře vytvořeného kódem 4.3

4.3.4 Rozhraní jádra pro moduly

Business vrstva poskytuje rozhraní pro další moduly systému v podobě třídy rozhraní *CoreInterface*, která je systémem připojena k modulu.

4.3.5 Rozhraní jádra pro zobrazovací vrstvu

Při vytváření aplikační vrstvy se předají požadavky prezentační vrstvy v podobě proměnných $_POST a $_GET. Tyto vstupní proměnné jsou zpracovány a analyzovány třídou *Input*. Třída, zjistí-li požadavek na novou stránku, zpracování formuláře či odkazu zavolá potřebné metody daných tříd, které požadavek zpracují.

Aplikační vrstva poskytuje prezentační vrstvě rozhraní v podobě třídy *BInterface*, která zapouzdřuje požadované metody pro přístup k business vrstvě jádra. Soubor se třídou *BInterface* je automaticky zpřístupněn zobrazovací vrstvě při vytváření zobrazovací vrstvy.

4.4 Zobrazovací vrstva jádra

Zobrazovací vrstva je napsána pomocí jazyka XHML, v případě potřeby dynamických dat, volají se funkce rozhraní pro business vrstvy. Systém určuje jaká stránka bude zobrazena. Zjistí požadované umístění stránky prezentační vrstvy a zavolá metodu shoHTLMObsah() z rozhraní modulu.

Každý modul musí obsahovat soubor s rozhraním modulu, který je definován v systému a v něm třídu *NazevModuluInterface*, která má veřejnou metodu showHTMLObsah(). Tedy například ve výpisu 4.4 je zobrazena třída *PublicInterface* modulu Public, který zprostředkovává veřejně dostupné informace. Třída se nachází v souboru *interface.php* v kořenovém adresáři modulu.

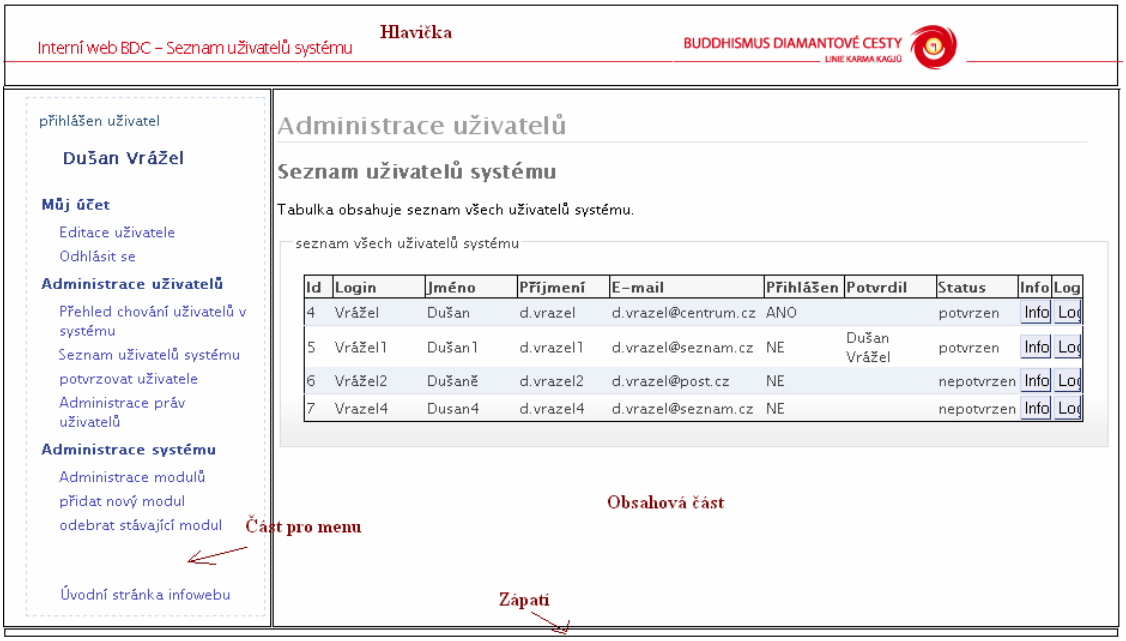
```
class PublicInterface {

  public function showHTMLObsah() {
     //zde umistit obsah, nebo odkaz na soubor
  }
}
```

Výpis 4.4 Třída PublicInterface s metodou showHTMLObsah() ze souboru interface.php

4.5 Grafické uživatelské rozhraní

Struktura grafického uživatelského rozhraní byla na základě analýzy požadavků navrhnuta jako dvojsloupcová s menu v pravé části a obsahem v levé části obrazovky. Nad obsahem se nachází úhlaví stránky, které se generuje automaticky podle obsahové částí a úpatí stránky. Menu je generováno dynamicky podle typu uživatele a jeho práv vůči modulu.



4.6 Funkce jádra

V jádru obsahuje funkce pro administraci systému, administraci uživatele a přihlašování a registrace uživatelů.

Při přechodu na interní web se nejprve ověří, zda je uživatel již přihlášen a má právo přístupu k infowebu. Pokud ano, zobrazí se ihned úvodní stránka infowebu. Pokud nikoli, zobrazí se stránka s přihlašovacím formulářem a registrací (metodu náhradních stránek je možno použít u libovolné stránky, kdy si v administraci určíme, která stránka se má zobrazit, nemáme li přístup k dané stránce).

Po přihlášení se vygeneruje menu pro uživatele podle daných práv k jednotlivým modulům a obsahová část. Pokud obsahovou část ani její alternativu nelze zobrazit, zobrazí se chybová stránka.

4.7 Testování

Testy systému probíhaly během vývoje systému. Zejména u kritických části systému, jako je rozhraní mezi moduly a grafické uživatelské rozhraní IS. Po dokončení jádra systému proběhly testy jádra na pěti uživatelích, kteří ověřili funkčnost naimplementovaných částí.

Hlavní fáze testování proběhne až po úplném dokončení všech funkcí jádra a implementaci modulů pro podporu center sdružení SDC.

4.8 Nasazení

V době psaní tohoto ročníkového projektu bylo implementováno jádro systému a modul pro veřejnou část systému. S nasazením do provozu se počítá po dokončení implementace dalších modulů systému.

4.9 Použité vývojové prostředky

Při implementaci systému byly použity tyto nástroje:

- **Elipse SDK** – Open Source vývojový nástroj s vydatnou podporou objektově orientovaného programování v PHP.
- **PHPMyAdmin** – Nástroj pro správu databáze MySQL (open source licence).
- **Rational Rose** – Komplexní CASE nástroj, pokrývající celý životní cyklus softwarových projektů.
- **Adobe Photoshop Elements 2.0** – Nástroj pro editaci a úpravu obrázků.
- **EMS SQL Manager 2005 Light for MySQL** – Nástroj pro správu databáze MySQL s podporou uložných procedur a funkcí
- **Prohlížeč FireFox s nainstalovanou součástí pro webové vývojáře.**

5 Rozšíření

V současnosti je implementováno jádro systému s hlavními administrátorskými funkcemi, přihlašováním a registrací. Dále je implementován modul pro zobrazování veřejné části systému a modul instalace systému.

Doporučuji systém dále rozvíjet a pokračovat v implantaci XML výstupu, e-mailová podpory a dalších administrativních funkcích jádra (nastavení vzhledu.). Dále je potřeba implementovat zbylé moduly systému dle požadavků kladených na systém (kalendář, poradna, novinky a články, centra, projekty, práce se soubory, on-line knihovna).

6 Závěr

Mým úkolem bylo provést analýzu a implementovat jádro Informačního systému občanského sdružení Společnosti Diamantové cesty. Během prvního semestru proběhla analýza současného stavu a formulace požadavků na systém. Na základě požadavků vznikla datová struktura jádra systému.

S příchodem MySQL 5 a podpory uložených procedur byla provedena optimalizace požadavků a přechod na třívrstvou architekturu systému s využitím OOP v PHP5 a uložených procedur MySQL. S přechodem na nový koncept bylo třeba upravit požadavky na systém a tím i datovou strukturu.

Při studiu uložených procedur MySQL se objevily problémy plynoucí s čerstvosti této nové technologie a nevyladění detailu.

Významný přínos v této práci vidím v detailním seznámení s prostředky PHP, OOP a novými vlastnostmi MySQL. Dále jsem se naučil vytvářet programové dílo za pomocí jazyka UML, v životních fázích programového díla, a za pomocí prostředků, kterými tento jazyk disponuje.

I přes úvodní problémy bylo implementováno jádro systému společně s modulem pro přístup k veřejným částem systému

Jako nedostatek v mé práci považuji pozdní nasazení nové koncepce systému a s tím plynoucí časové problémy v závěrečné fázi. Do příště bych také více využil možnosti konzultací v implementační fázi vývoje systému.

Literatura

[1] Arlow, J., Neustadt, I.: UML a unifikovaný proces vývoje aplikací. Brno, Computer Press, 2003. 408 s., ISBN 80-7226-947-X

[2] Welling, L., Thomsonová L.: PHP a MySQL rozvoj webových aplikací. Praha, Softpress, 2004. 910 s., ISBN 80-86497-60-7

[3] Zeldman, J.: Tvorba webů podle standardů. Brno, Computer Press, 2004. 410 s., ISBN 80-251-0347-1

[4] Scamray J. Shema M., Hacking bez tajemství, webové aplikace. Brno, Computer Press, 2003. 328 s., ISBN 80-7226-769-8

Přílohy

Příloha 1: Use Case diagram jádra.

Příloha 2: Diagram tříd jádra.