VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Generátor tabulek a webových formulářů

Bakalářská práce

2005 Jaroslav Kuboš

Zadání

Generátor tabulek a webových formulářů

Odevzdáno na Fakultě informačních technologií Vysokého učení technického v Brně, dne 1. června 2005.

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Radka Burgeta Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jaroslav Kuboš
21.4.2005

Abstrakt

Výsledkem tohoto bakalářského projektu je knihovna pro tvorbu webových tabulek a formulářů z databáze, která je popsaná XML souborem s definicí metadat. Podporuje vytváření, editaci, mazání a prohlížení položek v databázi. Také dokáže vypisovat položky z pohledu do tabulky. Knihovna je plně konfigurovatelná, včetně vzhledu výstupu. Knihovna je vytvořena v PHP5 a je určena pro použití s databázovým strojem MySQL, není ale problém upravit ji pro práci s jiným databázovým strojem. Metadata popisující databázi jsou uložena v XML souboru s DTD validací. Výstup je realizován jako XHTML stránka, ale je možné snadno předefinovat kreslící třídu pro výstup do jiného výstupního formátu. Políčka formuláře mohou být omezena omezením definovaným v rámci XML metadat. Knihovna dále obsahuje mnoho předem vytvořených tříd pro práci s MySQL, třídami popsanými v XML, a další obecně použitelné nástroje.

Klíčová slova

automaticky generovaný, formulář, tabulka, databáze, XML, metadata, PHP, MySQL

Poděkování

Děkuji Ing. Radku Burgetovi za konzultace a korekci práce.

Abstract

The result of this bachelor project is a library for creating web forms and tables from a database which is described by an XML file with metadata definition. The library supports creating, editing, viewing and deleting items from the database. It also can print table from view. The library is fully configurable including the visual appearance of the result. The library is written in PHP5 and should be used with MySQL database but conversion for another database engine is not a problem. The metadata for describing the database structure is stored in an XML file with DTD validation. The output is realized as an XHTML page but there it is possible to redefine the drawing class for another output format. The form fields can be constrained by definitions in XML. The library also contains many classes for the MySQL database, for describing classes in XML and more generally usable facilities.

Keywords

automatically generated, form, table, database, XML, metadata, PHP, MySQL

Obsah

1 Úvod

Tento dokument popisuje proces návrhu a tvorby knihovny pro automatické generování webových tabulek a formulářů pomocí metadat popisujících strukturu databáze, formuláře, jejich omezení a složení pohledů na tabulky. Celá knihovna vznikla po praktických zkušenostech s psaním webové aplikace. Celou práci v podstatě tvořilo neustálé tvoření výpisů z tabulek a tvorba formulářů. Při této mechanické činnosti vznikaly často chyby a nebylo se možno soustředit na to hlavní – strukturu dat. Proto jsem se snažil najít nástroj, který bych naplnil svou specifikací databáze, tabulek a formulářů a on by provedl mechanickou práci za mne. Dlouho jsem nemohl žádný najít, a když už jsem nějaký našel, nesplňoval většinu mých očekávání. Proto jsem se rozhodl pro vývoj vlastní knihovny, která by práci programátora omezila na zodpovědnost za definici metadat a navíc vyřešila zmatečné programování, které v těchto oblastech návrhu a programování webových aplikací panuje.
Dokument popisuje proces návrhu, tvorby a testování knihovny. Nezabývá se návodem na použití. Použití je popsáno v příloze. Popis kapitol:

- „Současný stav“ – Popisuje aktuální styl psaní tabulek a formulářů, problémy, chyby, ...
- „Cíle“ – Uvádí výčet vlastností a funkcí zamýšleného řešení. Zamýšlím se nad hlavními rysy celého mnou navrženého řešení. U každého rysu je popsáno proč je důležitý, případně jsou uvedeny problémy, které plynou z jeho nezačlenění. Pokud je to možné, uvádím i možnosti řešení bez použití rysu nebo rysu s jinou konfigurací.
- „Řešení“ – Popisuje návrh a implementaci knihovny.
- „Použití“ – Demonstruje kroky při návrhu systému pomocí této knihovny.
- „Shrnutí“ – Shrnuje výsledný stav, principy testování.
- „Závěr“ – Obsahuje popis současného stavu projektu a jeho budoucnost. Shrnuje poznatky získané při jeho tvorbě a snaží se vysvětlit, jak by jich šlo využít při práci na nové verzi.

2 Současný stav

V současné době se pro tvorbu formulářů a výpisů tabulek používají téměř výhradně různá ad hoc řešení. Tedy řešení přímo na míru aplikaci. S tím je spojena řada problémů.

Kód je pořád dokola přepisován do dalších formulářů a tabulek, vyskytuje se tak v mnoha kopiích v celé aplikaci (redundance). Případné změny se musí distribuovat do všech míst výskytu (nekonzistence). Tím se zvyšují rizika zanesení chyby. V lepším případě se vytvoří knihovna, která se používá například pro procházení záznamů, či vykreslování tabulek.

Nezkušení programátoři tvoří nebezpečná řešení velmi snadno napadnutelná technikami jako je „SQL Injection.“ Již několikrát jsem pozoroval, že nezkušený programátor posílal SQL dotaz pro měnění dat v databázi jako GET proměnnou. Navíc tito programátoři nedokáží správně ošetřit vstupní data a tím zvětšují riziko zanesení chybných dat do databáze.

Vytváření formulářů a tabulek tímto způsobem je navíc velmi neekonomické. Zvláště v komerční praxi. Je mnohem snazší použít hotové řešení, než vyvíjet své vlastní.

Dalším problémem je, že neexistuje vazba mezi návrhem a realizací programu. Tedy návrh ER diagramu dat pro aplikaci je jednorázově transformován do programu. Tím jsou vlastně popisy dat dva. První v návrhu a druhý zakódovaný v programu. Při změnách struktury dat v návrhu se program stává nekonzistentním a je problém jej sesouhlasit s metadaty. Ukázka typického současného kódu (pouze vylistování tabulky, velmi zjednodušené):

```
<?
 include_once("include/cDataSet.php");
 $g_UsersSet = new cDataSet;
 $g_UsersSet->SetOrderField("ID");
 $g_UsersSet->LoadTable("Users");
?>
<br>
<table border="1" bgcolor="<?=TABLE_COL?>" align="center">
 <tr bgcolor="<?=TABLE_HEAD_COL?>">
  <th>ID</th>
  <th>Login</th>
  <th>Typ</th>
 </tr>

 <?
```

```
  $User = new cUser();
  while( $g_UsersSet->Next() )
  {
   $User->LoadById($g_UsersSet->GetItem("ID"));
?>
  <tr>
   <td align="center"><?=$g_UsersSet->GetItem("ID")?></td>
   <td align="center"><?=$g_UsersSet->GetItem("Login")?></td>
   <td align="center"><?=$g_UsersSet->GetItem("Admin")?></td>
  </tr>
<?
  }
  $User->Free();
?>
</table>
```

Je vidět, že pro správnou funkci je potřeba dodržet názvy sloupců v databázi. Jakákoliv změna sloupců (přejmenování, zrušení, ...) povede k nutnosti projít kód a opravit změny. Navíc se nepoužívá knihovna pro kreslení tabulky, ale tabulka se generuje přímo z kódu.

Dalším problémem je použití přímého výstupu XHTML kódu. Pokud se rozhodne o změně vzhledu aplikace, opět bude nutno upravit všechny stránky.

Při vytváření formulářů vznikají ještě mnohem horší konstrukce. Je potřeba vytvořit SQL dotazy pro vytváření, náhled, editaci a smazání prvku v databázi. Dále se musejí nadefinovat omezení vkládaných dat, vzhled, atd. Nejspíše ani není nutné zdůrazňovat, jak je tento postup obtížný, chybový a zbytečný.

3 Cíle

3.1 Specifikace vlastností

Při vytváření ideje pro tuto knihovnu jsem vycházel z vlastních zkušeností při tvorbě komerční webové aplikace. Ze svých negativních zkušeností jsem formuloval podmínky, které by měla knihovna splňovat. V mnoha z nich lze rozpoznat základní paradigmata programování.

3.1.1 Metadaty řízená knihovna

Tento aspekt je klíčový. Kvůli němu vlastně knihovna vznikla. Ve standardních programech je databázová struktura definována při tvoření tabulek a složitě popisována při dotazech. Například při změně názvu sloupce je potřeba nejen napsat skript na změnu jména v databázi, ale i přepsat všechny odkazy na jméno ve všech dotčených dotazech. Při přidání sloupce je třeba napsat skript na změnu v databázi a přidat odkaz na sloupec do dotazů, tabulek a formulářů. Všechny tyto akce při používání knihovny odpadnou, protože stačí provést změny v metadatech. To je ovšem běžný jev v metadaty řízených programech.

Směr vývoje programování se čím dál tím více přiklání k používání dat na řízení programu. V některých případech se dokonce používají meta2data. Konkrétní se v programu nahrazuje obecným. Tento posun je možný jen díky zvyšující se rychlosti počítačů, protože metadatové programy bývají pomalejší, než ty napsané „natvrdo“. To ovšem rozhodně nemusí být pravidlem. Lépe by asi bylo říci, že metadatové programy jsou většinou pomalejší, než ty kvalitní napsané „natvrdo“. Pro většinu programů ovšem metadata znamenají zjednodušení kódu a tím i snazší možnost optimalizací.

Další výhodou metadat řízených programů je to, že se zvyšuje jejich integrita. Změny se obvykle vyřeší v datech a do programu není třeba zasahovat. Pokud jsem dříve potřeboval přidat sloupec do tabulky v databázi, znamenalo to mnoho úprav na rozličných místech kódu. Nyní se změní metadata a změna je provedena.

Data by měla být v celém výsledném IS popsána pouze v XML souboru s definicí metadat. Systém by měl minimalizovat potřebu uživatele vytvářet vlastní dotazy SQL.
Nerespektování tohoto kritéria vede v nekonzistentní programy, které se těžko programují, testují a hlavně udržují.

3.1.2 Jednoduchost

Použití knihovny musí být jednoduché. V ideálním případě vytvoření objektu tabulky/formuláře a specifikace metadatové entity a fyzické tabulky. Jednoduchost použití se dosahuje na úkor vyšší složitosti metadat. Je proto potřeba najít rozumný poměr kód/metadata. S menší složitostí metadat narůstá složitost a v závislosti na ní nekonzistence programu, protože se zvýší počet míst, která je třeba změnit při změnách ve zbytku programu. Tedy např. se zvyšuje počet metod, které je třeba zavolat před vykreslením objektu. Při změnách je často potřeba projít veškerá volání a opravit.

3.1.3 Univerzálnost

Knihovna musí být univerzální nejen co do popisu databázové struktury, ale i co do vzhledu. Vzhled musí být možno zcela změnit, případně realizovat do úplně jiného výstupu než XHTML. Univerzálnost může být vykoupena složitým kódem knihovny a omezeními v oblasti funkcionality, čehož je třeba se vyvarovat.

Pro školní účely i většinu úloh v praxi postačí výběr hlavních důležitých schopností a vlastností použitého databázového stroje.

3.1.4 Bezpečnost

Tím, že budou všechny formuláře generovány automaticky, a to relativně malým kódem, bude systém velmi bezpečný. Většina chyb totiž vzniká při bezmyšlenkovitém opisování/kopírování kódu z jiných formulářů. Ovšem skrytím bezpečnostních opatření do nitra knihovny se zvyšuje riziko, že ji nepoučený programátor špatně použije. Proto je u popisu použití knihovny důležité popsat i popis nutného zabezpečení ze strany uživatele.

3.1.5 Rozšiřitelnost

Systém musí být snadno rozšiřitelný a konfigurovatelný. Musí být schopen vytvářet dynamické (nejsou v metadatech) sloupce v pohledech. Ty se dají použít na vypisování informací, které metadata nepostihnou – tedy například sloupce, které jsou získány složitým sumačním SQL dotazem. Další podporu rozšiřitelnosti představuje možnost nastavit některá neviditelná políčka ve formuláři na dynamicky zadanou hodnotu – tedy např. při vytváření zboží se jako zakladatel zapíše aktuálně přihlášený zaměstnanec.

3.2 Specifikace funkcí

Po knihovně jsem chtěl vše, co může díky metadatovém základu zvládnout. Tedy zejména možnost editace tabulek v databázi, výpis tabulek z databáze a tvoření její struktury. Mimo to ještě existuje možnost použít metadata pro definici tříd, které reprezentují řádky tabulky.

3.2.1 Tvorba tabulek v databázi

Knihovna umožňuje nejen vytvoření tabulek podle metadat, ale také jejich úpravy podle změn v metadatech – tedy po přidání a odebrání sloupce. Po rozšíření definice metadat o podporu verzování by bylo dokonce možné konvertovat databázi o několik verzí – automaticky, bez nutnosti psaní konverzních skriptů.

3.2.2 Výpis obsahu tabulek z databáze

Knihovna umožňuje vypisování dat z pohledů. Dokáže slučovat sloupce databáze do souhrnných sloupců . Podporuje listování podle počtu řádků na stránku a podle prvního písmena zvoleného sloupce. Tyto dva režimy lze zkombinovat. Lze tedy např. vypisovat klienty s příjmením začínajícím na písmeno K a pokud bude výpis delší než 20 řádků, vypisovat po stránkách. Dále je možné řadit data kliknutím do záhlaví sloupce. Všechny sloupce lze řadit vzestupně a sestupně. Všechny tyto volby se uchovávají v rámci session a po návratu na stránku, se obnoví.

3.2.3 Generování formulářů pro editaci databáze

Formuláře podporují editaci téměř všech datových typů uložitelných do databáze. Umožňuje editovat a zobrazit:

- set – množina, výběr žádného až všech prvků
- enum – výběr jedné z variant
- password – speciální metadatový typ, do formuláře se generují 2 pole, na heslo a jeho potvrzení, do databáze se ukládá jako SHA1 hash
- text – dlouhý text
- foreign – zobrazení výběru cizího klíče
- value – všechny ostatní hodnoty jako čísla a krátké řetězce

Dále umožňují validaci dat podle definic uložených v XML, zobrazení správného formátu pole a popis pole. Pole, která byla špatně zadána, změní svůj vzhled.

3.2.4 Popis tříd reprezentujících řádek tabulky

Knihovna umožňuje definovat třídy, jejichž datové prvky jsou popsány pomocí metadat a uloženy v databázové tabulce. Jedna instance třídy popisuje jeden řádek tabulky. Je možno definovat vlastní metody a používat předdefinované. Jedná se tedy v podstatě o zautomatizování databázové vrstvy, která se obvykle programuje ručně.

3.2.5 Výsledná forma

Automatické generování formulářů se dá vyřešit dvěma způsoby:

1. generování kódu formulářů
    a. výhody
        - rychlost – spouští se rovnou kód, který generuje formulář
        - zákazník nemusí vlastnit knihovnu, stačí mu dát vygenerované stránky
    b. nevýhody
        - formuláře je třeba po každé změně metadat znovu vygenerovat (pravděpodobně ručně)
        - napsat knihovnu, která generuje kód, je horší, než napsat knihovnu generující data
        - kód je závislý na verzi PHP – po změně verze se musí opravit nejen knihovna, ale také data, která generuje
        - hůře ze začleňují dynamické změny uživatelem – musí vkládat kód místo referencí, či dat
2. generování dat popisujících formuláře – tedy univerzální kód pro generování formulářů podle metadat
    a. výhody
        - schopnost samoaktualizace vůči metadatům – změna metadat znamená automaticky změnu výstupu
        - snadný a univerzální kód pro generování výstupu
        - snadné změny uživatelem knihovny – vkládá data
    b. nevýhody
        - knihovna se musí dát i uživateli
        - nižší rychlost běhu

Rozhodl jsem se pro generování dat popisující formulář pomocí metadat. Tedy formulář je tvořen při každém načtení stránky znovu.

3.3 Specifikace nástrojů

3.3.1 Platforma, nástroje

Platforma řešení je specifikována už v zadání. Upřesnění jsem si navrhl sám. Vycházel jsem z faktu, že pokud použiji nejnovější verze nástrojů, budu moci využít jejich nejnovější rysy a navíc budou současné verze pravděpodobně dlouho podporovány, a tím se prodlouží životnost knihovny. Další výhodou je fakt, že v nejnovějších verzích jsou opraveny nově zjištěné chyby, a proto jsou mnohem bezpečnější. Použití nejnovějších nástrojů na druhou stranu znesnadňuje použití na místech, kde není možnost ovlivnit konfiguraci serveru (webhosting).

Na níže uvedených verzích nástrojů bude knihovna vyvíjena a testována. Pro nasazení s novou verzí nástrojů bude třeba prostudovat změny (change log) nástrojů a vyhodnotit možný vliv na knihovnu.

Konfigurace nástrojů:

- PHP 5
  - aktuální verze 5.03
  - bezpodmínečně verze 5.0, kvůli použití nových rysů
  - nepoužívá nepřednastavená nastavení
- MySQL
  - aktuální verze 4.1.10.a
  - bezpodmínečně verze 4.1 a vyšší
  - nastavení základní znakové sady na UTF-8
- XHTML – volitelné
  - kompatibilní prohlížeč u klienta, STRICT verze

4 Řešení

4.1 Postup prací

Nad problémem jsem přemýšlel už dlouho před zadáním bakalářské práce a své nápady si ověřoval pokusnými verzemi knihovny. Z problémů, které jsem přehlédl v návrhu, jsem se vždy snažil poučit a v další verzi je eliminovat. Vzniklo tak několik verzí, ve kterých byla metadata nejprve ve voláních funkcí, poli, objektech, až se nakonec dostala do XML souboru. Snažil jsem se dosáhnout takového stavu, abych vyhověl výše uvedeným požadovaným vlastnostem. Navíc vznikaly další, vesměs z „neohrabanosti“ předchozí verze v nějakém směru. Také jsem testoval možnosti použitých nástrojů – volání kódu uloženého v datech, práce s XML, ... Když jsem byl s prototypem spokojen a měl jsem ověřeny všechny principy, mohl jsem se pustit do finálního návrhu knihovny.

Při analýze jsem vycházel z požadavků, které jsem si stanovil a ze zadání. Při postupu bylo důležité najít a uvědomit si všechny závislosti a potencionální problémy. Používal jsem metody UP (Unified process). Na analýzu navazuje proces návrhu. Jeho úkolem je navržení struktury knihovny, podle které ji lze naprogramovat. Při návrhu jsem používal diagramy jazyka UML (Unified Modeling Language). Pak už jsem mohl začít programovat a průběžně testovat. Používal jsem metody statického i dynamického testování. Po dokončení a posledním otestování jsem začal psát technickou zprávu a dokumentaci. Pak následovalo vytvoření balíčku a odevzdání.

4.2 Teorie použitých nástrojů

4.2.1 PHP

PHP je univerzální skriptovací jazyk, s volně dostupným kódem. Je obzvláště vhodný pro dynamické generování webových stránek. Je dostupný pro mnoho platforem a webových serverů. Syntaxe vychází z jazyka C. Aktuální verze 5 značně rozšiřuje schopnosti jazyka hlavně v objektové oblasti, kde však stejně nedosahuje úrovně objektově orientovaných jazyků. Přesto je velmi oblíben, nejspíše proto, že je volně šiřitelný a tedy zadarmo. Toho využívají zejména poskytovatelé serverových kapacit (webhosting), pro které PHP neznamená další náklady.

4.2.2 MySQL

MySQL je volně šiřitelný databázový stroj, a platí pro něj téměř všechny vlastnosti řečené u PHP. Neoplývá množstvím funkcí, zato je velmi rychlý. V těchto dnech vychází verze 5.0, která by měla

nedostatky funkcionality odstranit. MySQL+PHP se stává ustáleným výrazem, neboť množství PHP skriptů používajících MySQL je zdrcující.

4.2.3 UP

Je průmyslový standard pro tvorbu programového vybavení, pocházející od tvůrců jazyka UML. Popisuje proces tvorby software a nástroje (diagramy UML) používané v jednotlivých fázích. Stejně jako se při návrhu většiny softwarových projektů používá UML, dodržuje se i postup UP.

4.2.4 UML

Jazyk UML je univerzální jazyk pro vizuální návrh a modelování obecných (tedy nejen softwarových) systémů. UML není jazyk zcela nový, ale byl vytvořen sloučením a upravením do té doby standardních jazyků. V dnešní době je spolu s UP používán při modelování a návrhu většiny softwarových projektů. Jazyk se skládá z několika typů diagramů, používaných v různých fázích UP. Důležité je, že je jazyk UML standardizován, což nemalou měrou přispělo k jeho rozšíření.

4.2.5 XHTML

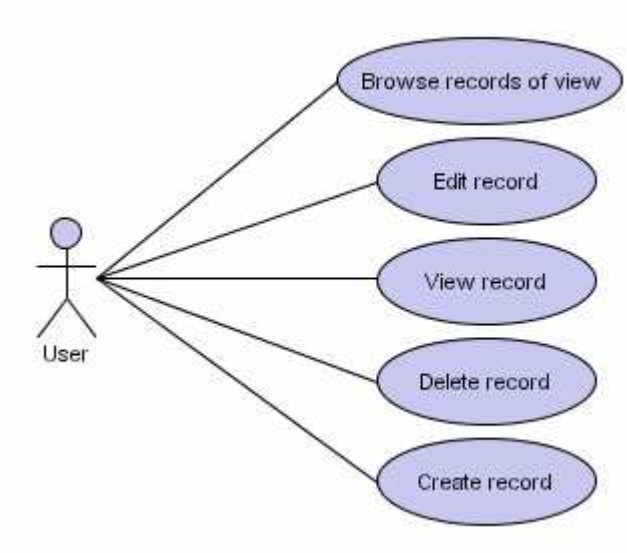
XHTML je relativně nový formát formátovacího jazyka pro internetové stránky. Vychází z HTML 4.0 a z XML.

Z XML přebírá zápis elementů, atributů a hlaviček. Dokument v XHTML je vlastně validním XML dokumentem podle XHTML DTD deklarace. Z toho také vyplývají omezení v oblasti zápisu proti HTML. Velmi často používaným nešvarem je v HTML křížení elementů. Tedy to, že element začíná v jiném, ale je uzavřen až po jeho uzavření. Při zápisu v XHTML nebude dokument vůbec zobrazen. Dále se často vyskytovaly neuzavřené elementy, používání malých i velkých písmen ve jménech elementů, či atributy bez hodnoty.

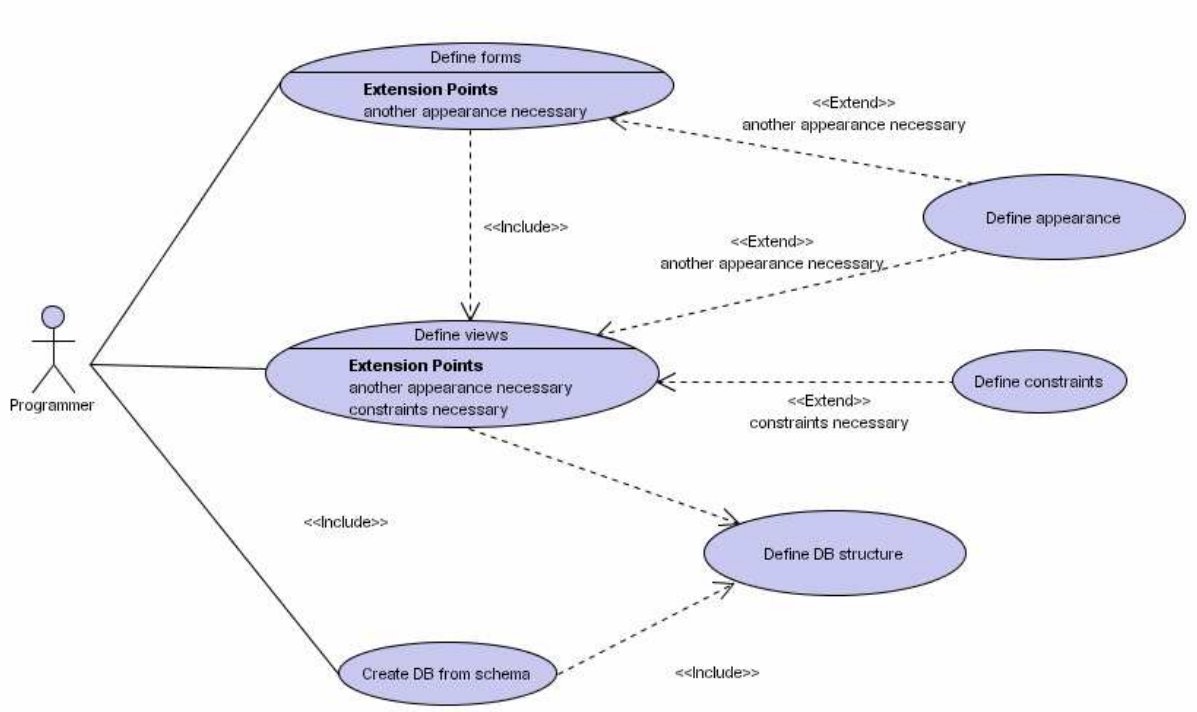
Z HTML přebírá většinu elementů kromě těch, které nahrazuje CSS. Tedy většinou elementy pro formátování vzhledu. XML kód bývá kompatibilní s HTML 4.0 kódem a lze jej tedy zobrazit i ve starších prohlížečích.

4.3 Analýza

Prvním krokem analýzy bylo vytvoření diagramů případů použití (Use Case Diagram), které zpřehlední služby, které by měla knihovna poskytovat. Nejprve z pohledu uživatele – uživatele produktů knihovny. Vycházel jsem ze specifikace.



Další diagram použití zobrazuje pohled programátora, používajícího knihovnu. Diagram relacemi <<include>> specifikuje, které funkce jsou použitelné jen v závislosti na jiných. Relace <<extend>> specifikují volitelné použití.



Velká část analýzy proběhla už před samotným zadáním projektu a je popsána v kapitole 4.1. Hlavní problémy byly odhaleny díky testovacím prototypům.

4.4 Návrh struktury metadat

Návrh struktury metadat je ovlivněn specifikací funkčnosti a rysů, ale také výsledky analýzy. Musí odrážet celou požadovanou funkcionalitu. Hlavním problémem je zde nadefinovat strukturu takovou aby:

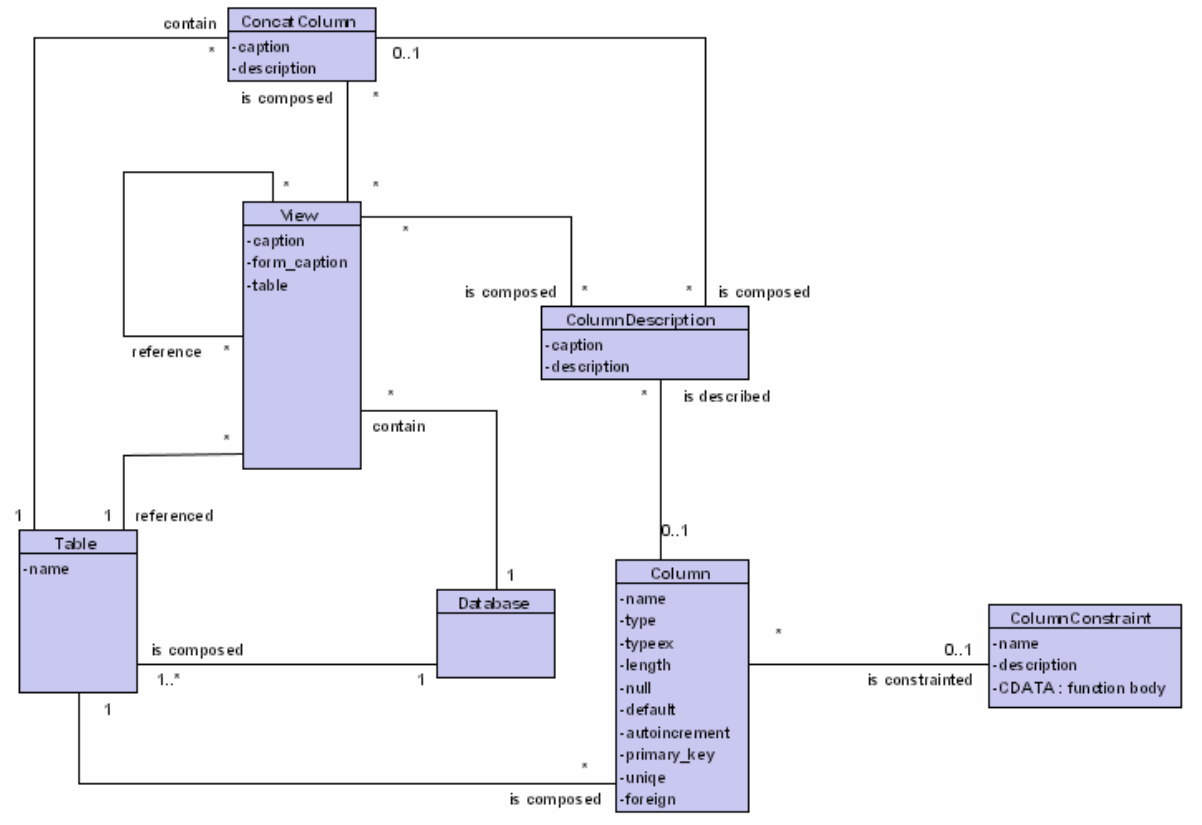
- dokázala popsat a uložit všechna potřebná data podle analýzy funkčnosti
- vyhovovala vlastnostem určeným ve specifikaci
- byla snadno implementovatelná v XML a využívala jeho výhod

4.4.1 Schéma

Vytvořený ER diagram popisuje používané entity a vztahy mezi nimi. Zajímavé jsou vazby typu M:N, ty totiž nejdou uložit do přímé stromové struktury. Uvažoval jsem o vyjádření pomocí diagramu tříd (Class diagram), který je také možno použít, ale nakonec se mi zdál subjektivně ER diagram pro popis struktury vhodnější.

Textový přepis návrhu by mohl znít asi takto:

- popis databáze se skládá z tabulek, omezení a pohledů
- tabulka se skládá ze sloupců a složených sloupců
- sloupce mohou být složeny z popisů sloupců (různá jména v jiných pohledech), ty se používají pro referování
- sloupec může být omezen omezením uloženým v XML
- složené sloupce se skládají z referencí na popisy sloupců
- pohledy se skládají referencí na popisy sloupců, složené sloupce a jiné pohledy
- omezení se skládá z bloku dat, který je vlastně kusem kódu pro PHP



4.4.2 Převod na DTD

Převod ER diagramu není tak triviální záležitostí, jak by se mohlo zdát. Je totiž třeba správně převést všechny typy použitých vazeb. Také bylo potřeba vhodně zvolit atributy a rozsahy jejich hodnot. To se dělo v souladu se zadáním.

Vazba 1:N se dá uložit přímo do stromu XML, bez nutnosti používat identifikátory. Například vazba mezi tabulkou a sloupci se dá uložit takto:

```
<Table>
        <Column/>
<Column/>
<Column/>
</Table>
```

S vazbou M:N vznikají potíže. Do klasické stromové struktury ji nelze zapsat bez pomoci identifikátorů. Při převodu, je potřeba domyslet logický význam vazby. Například vazba mezi ColumnDescription a View lze zapsat dvěmi způsoby:

```
<Table>
  <Column>
    <ColumnDescription id=’item1’ caption=’A’/>
  </Column>
  <Column>
    <ColumnDescription id=’item2’ caption=’B’/>
  </Column>
</Table>

<View>
  <ColumnRef ref=’item1’/>
  <ColumnRef ref=’item2’/>
</View>
```

nebo

```
<Table>
  <Column>
    <ColumnDescription caption=’A’>
      <ViewRef ref=’cols’/>
    </ColumnDescription>
  </Column>
  <Column>
    <ColumnDescription caption=’B’>
      <ViewRef ref=’cols’/>
    </ColumnDescription>
  </Column>
</Table>

<View id=’cols’>
</View>
```

Tedy zda bude reference obsahovat pohled, nebo budou reference obsahovat popisy sloupců. Z logiky věci vyplývá, že by bylo lépe umístit reference do pohledů, protože definice tabulky jistě nemusí obsahovat informace o tom, kdo ji používá. Toto rozlišení je ale hodně subjektivní, obecně je jedním z paradigmat návrhu. Obě verze budou funkční, záleží na výběru návrháře.

4.4.3 Vlastnosti, omezení

Celý systém je naprogramován pro práci v kódování UNICODE zapsaném pomocí UTF-8. S UTF-8 v MySQL už není problém, protože v posledních verzích se v něm vnitřně definují dokonce databázová metadata a rovněž všechny funkce jej podporují. V XML a XHTML rovněž nevznikají žádné obtíže, obě jsou na UTF-8 dobře připraveny. Problém vzniká u PHP. Tvůrci PHP (většinou jsou z národů, kterým stačí ASCII) už dlouhou dobu problém znakových sad ignorují. Díky knihovně iconv sice lze převádět řetězce mezi celou škálou znakových sad a provádět základní řetězcové operace (zjištění délky, nalezení podřetězce, a několik dalších) i pro UTF-8, ale většina funkcí řetězce bere jako prosté ASCII. Pokud se k řetězci nemusí přistupovat po znacích a identifikovat je, obvykle vše funguje. Jako hlavní problém bych identifikoval regulární výrazy. Dost možná existuje nějaká knihovna, která regulární výrazy pro UTF-8 podporuje, ovšem používání takovýchto knihoven nemusí být všude možné (webhosting). Proto je třeba vymýšlet jiné metody.

Vztah mezi schématem tabulky a počtem současně použitých tabulek je 1:1. Aby uživatel nemusel pokaždé specifikovat tabulku přidruženou k referenci. Pokud totiž pohled používá 2 reference na pohledy stejného schématu, z nichž každý používá jinou fyzickou tabulku, vznikla by nejednoznačnost nezachytitelná v XML.

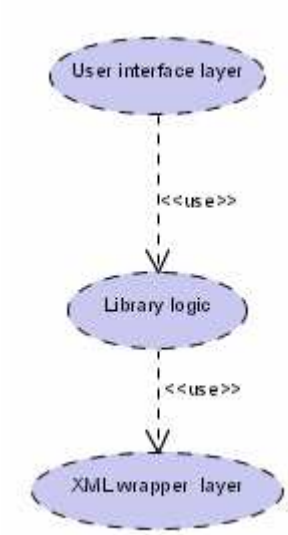
Knihovna podporuje použití právě jednoho primárního klíče v tabulce. Potřebuje jej pro identifikaci záznamu při editaci. Nepodporuje vícenásobné klíče a tabulky bez klíčů.

Vstupní soubory XML s metadaty musí být validní vzhledem k DTD souboru „db.dtd“, uloženému v kořenu knihovny. Podle DTD definice se nejen provádí validace definice metadat, ale slouží také k identifikaci identifikátorů (id), ale také referencí na ně (idref) při tvoření DOM stromu v paměti. Tuto činnost je potřeba dělat při každém použití – při každém načtení stránky. Tuto činnost bohužel nejde ušetřit metodou serializace, protože současná verze PHP nedokáže serializovat DOM model z paměti. Pokud by se toto ukázalo jako problém, šel by DOM strom načtený do paměti popsat pomocí vlastních tříd a serializovat tyto. S touto strukturou by bylo možné provádět podstatné optimalizace pomocí funkcí poskytovaných novými verzemi PHP.

4.5 Návrh struktury programu

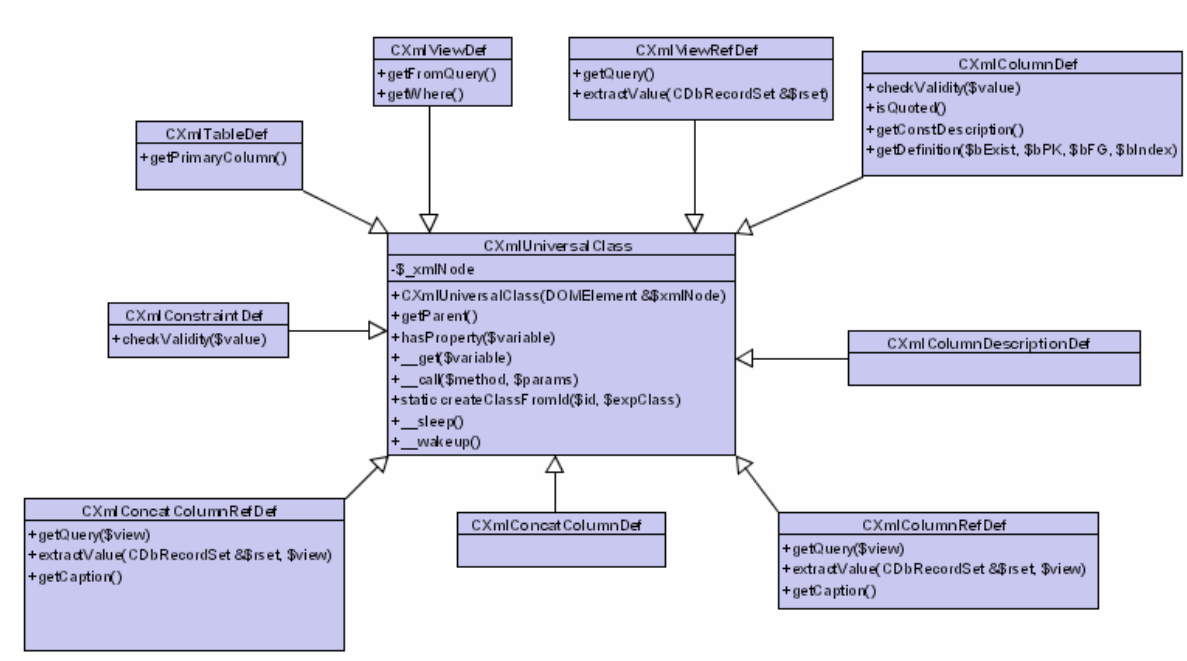
Struktura knihovny se skládá ze dvou částí, lépe řečeno vrstev. První vrstva obaluje paměťovou reprezentaci XML v DOM modelu strukturou tříd, která odstiňuje aplikační logiku od XML. Jedná se tedy o jakousi databázovou vrstvu, známou z třívrstvé architektury. Vrstva aplikační logiky využívá služeb XML vrstvy. Návrh vrstev se dá oddělit, ale je potřeba začít psát kód od XML vrstvy, protože vrstva aplikační logiky ji využívá.

Jako prezentační vrstva by se daly chápat třídy, které zajišťují výstup (kreslení) a jsou řízené aplikační logikou.



4.5.1 Struktura tříd – popis XML

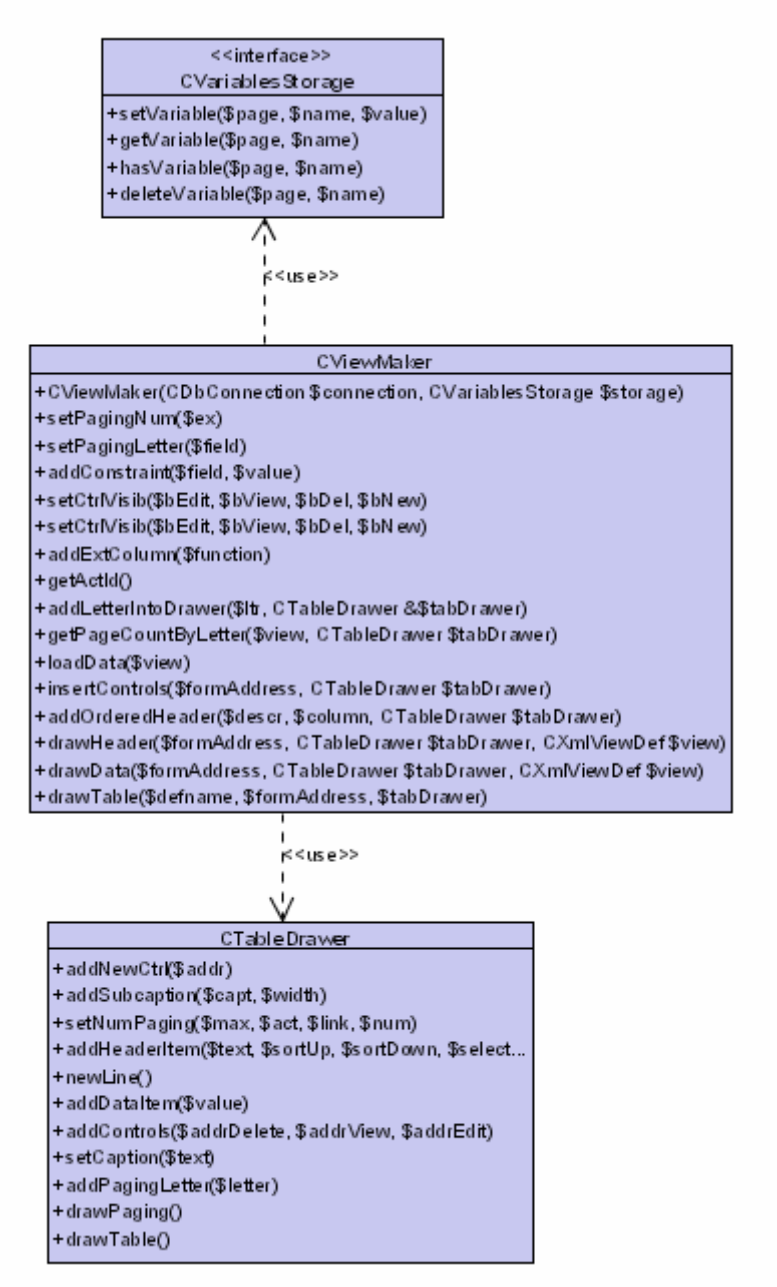
První obrázek ukazuje strukturu tříd popisujících obsah XML souboru s metadaty. Všechny třídy jsou odvozeny od bázové třídy CXmlUniversalClass. Ta definuje základní přístup k uzlu DOM modelu XML v paměti a přístup k jeho atributům. Umožňuje získat rodičovský element, jako objekt třídy náležející danému typu. Tentýž princip je užit při instancování objektů podle id. Další třídy přidávají funkčnost specifickou pro daný typ elementu v XML.



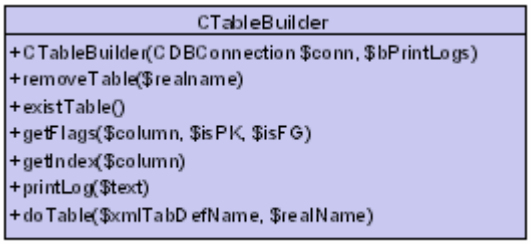
Struktura dynamického propojení těchto tříd je určena XML souborem. Tedy pokud chce některý objekt získat všechny své potomky, musí projít DOM strom, vyhledat je a instancovat. Každý instancovaný objekt má svůj odkaz na pozici v DOM stromu.

4.5.2 Struktura tříd – výkonná část

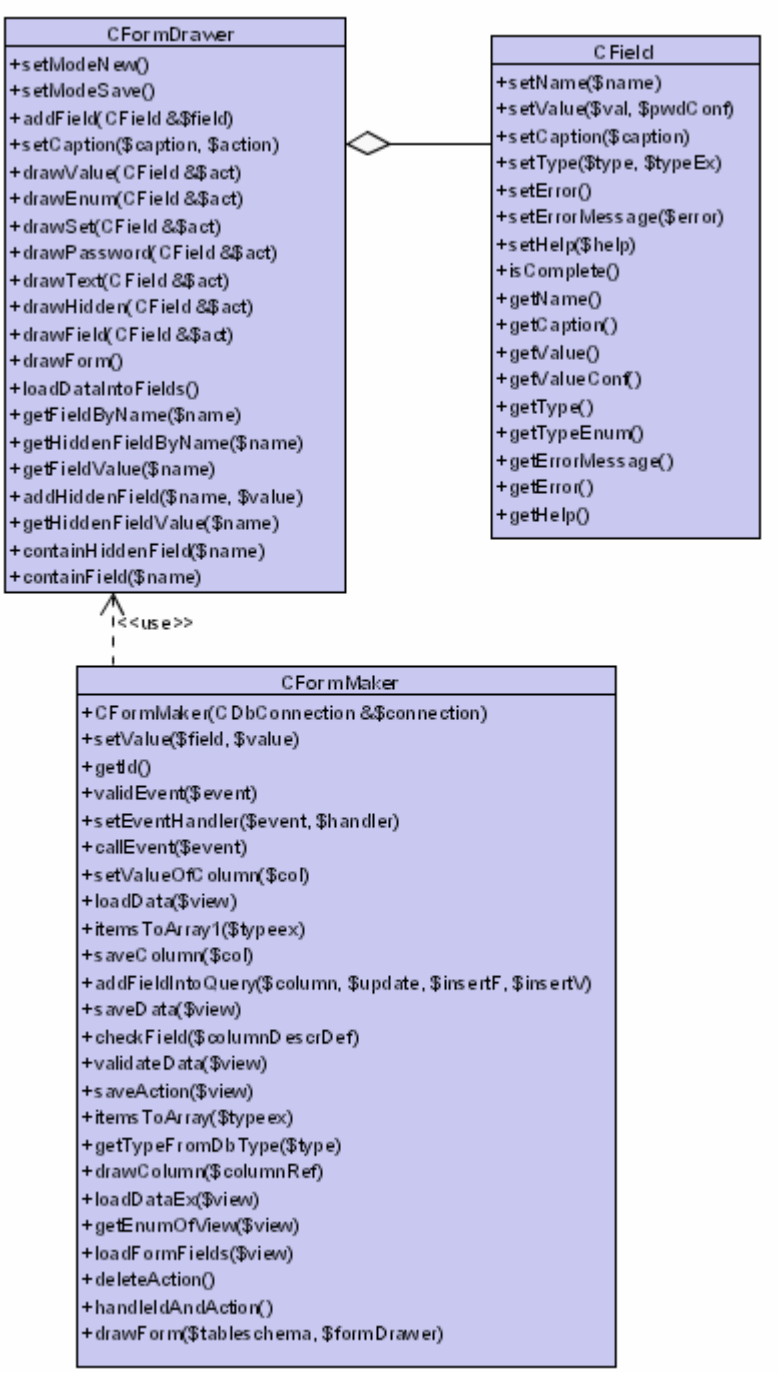
Následující struktura tříd popisuje sadu použitou pro vytváření výpisů z databáze – tabulek z pohledů. Rozhraní CVariablesStorage slouží k přístupu k objektu uschovávajícímu data mezi přechody mezi stránkami – tedy například sloupec, podle kterého se třídí data. Třída CTableDrawer slouží k vykreslení tabulky v XHTML. Jejím zděděním a přepsáním potřebných metod lze realizovat výstup do jiného formátu.



Třída CTableBuilder slouží k vytváření a rušení tabulek podle metadat. Metoda doTable slouží k vytvoření tabulky jménem $realName podle tabulky s identifikátorem $xmlTabDefName.



K vytváření formulářů slouží třída CFormMaker, která za pomoci třídy CFormDrawer vytváří formuláře v XHTML. Pro změnu výstupu platí totéž, co pro tabulky.



4.6 Programování

Po důkladném návrhu je napsání kódu knihovny formalitou. Je třeba dbát pouze na psaní komentářů a dodržování pravidel omezujících negativní vlastnosti PHP. Tyto vlastnosti jsou popsány v kapitole 6.1.

4.6.1 Klíčové aspekty

Je třeba si uvědomit, že výsledným produktem je knihovna a tomu přizpůsobit práci, používat jednotné návyky a ideje a to zejména:

- komentáře – Komentáře musí být maximálně výstižné, protože umožňují rozšíření knihovny neznalým programátorem.
- přehledná a jednoduchá rozhraní – Jednoduchá rozhraní nezatahují do programů množství tříd z knihovny. Jednoduchá musí být proto, aby se uživatelům snadno užívala.
- univerzálnost – Knihovna musí být maximálně univerzální pro použití v různých situacích.
- nezávislost – V ideálním případě je knihovna zcela nezávislá na dalších.

4.6.2 Použité vlastnosti nástrojů

Při programování byly použity určité prostředky jazyka PHP5, které nejsou součástí jazyka verze 4, proto je popíšu a odůvodním. Odůvodním také všechny nestandardní funkce dostupné i v PHP4.

- přetěžování metod – Přetěžování metod se používá ve dvou situacích:
  - volání metod se jménem obsahujícím dynamická data v objektech popsaných v XML – např. loadBySurname
  - volání metod se jménem obsahujícím dynamická data v objektech zastřešujících XML DOM model – např. getClientsColl
- přetěžování atributů – Tato schopnost PHP5 se používá ve dvou situacích:
  - čtení XML atributu zastřešující třídy – Pokud se zavolá objekt s neexistujícím atributem, zkusí se, zda v příslušném tagu neexistuje atribut stejného jména. Pokud ano, vrátí se hodnota, pokud ne, výjimka.
  - čtení a modifikace atributu objektu popsaného v XML – Pokud se zavolá objekt s neexistujícím atributem, zkusí se, zda v příslušné definici existuje objekt stejného jména. Pokud ano, provede se příslušná akce, pokud ne, vrátí se výjimka.
- generování kódu za běhu skriptu – Tato možnost se používá pro generování funkcí z omezení zapsaných v XML souboru. Omezení je vlastně kód PHP uložený v XML. Tímto má uživatel možnost přidávat omezení beze změn v knihovně.
- UTF-8 – Použití tohoto kódování je pro neznalého člověka krokem do neznáma. Autoři PHP totiž jiné kódování než ASCII nemají v lásce.
- automatické vkládání souborů – Pokud se při běhu skriptu vyskytne instancování neznámé třídy, pokusí se PHP zavolat funkci __autoload se jménem neznámé třídy jako parametrem a očekává, že tato funkce provede vložení potřebného souboru. Této vlastnosti se dá brilantně využít. Knihovna exportuje proceduru onDemandIncludeFLCX, kterou je potřeba zavolat

z __autoload procedury, pokud aplikace sezná, že nemůže potřebný soubor nahrát samostatně. Výhodou je to, že není třeba vkládat všechny potřebné soubory ručně a zároveň se pokaždé nemusí vkládat vše.

4.6.3 Struktura knihovny

Knihovna je uložena v samostatném adresáři s názvem „flcx“ a tak se také připojí k projektu. Všechna data knihovny jsou uložena v něm.V kořenu knihovny se nachází soubory:

- „db.dtd“ – definice struktury XML souborů, které je knihovna schopna zpracovat
- „flxc.php“ – hlavní „hlavičkový“ soubor knihovny, obsahuje funkci pro dynamické načítání tříd
- „readme.txt“ – soubor s informacemi o knihovně a její struktuře
- adresář „exec“ – obsahuje hlavní třídy knihovny určené k přímému spouštění
- adresář „utils“ – obsahuje pomocné třídy pro databázi, výjimky a konfiguraci knihovny
- adresář „xmlDefs“ – obsahuje třídy pro popis XML dokumentu

5 Použití

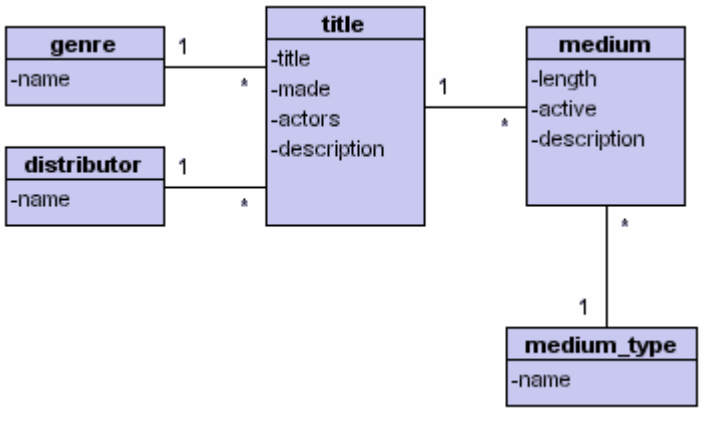
Tato kapitola popisuje typický postup při začleňování knihovny do projektu. Všímá si také nestandardních situací. Podle tohoto návodu je možné knihovnu zprovoznit. Postup je demonstrován na části testovacího informačního systému, který je jako ukázka distribuován s aplikací, takže je možné si prostudovat reálný kód. V poslední části je zobrazen výsledný vzhled.

5.1 Návrh struktury databáze

Nejprve si je třeba rozmyslet strukturu dat plánovaného projektu a tuto strukturu si nakreslit třeba ve formě ER diagramu. V našem příkladu půjde o zachycení vztahů při ukládání informací o médiích ve videopůjčovně. Zjednodušené zadání:

- uschovávají se běžné informace o médiích (titul, rok výroby, ...)
- médium má svého distributora, žánr a typ
- jeden titul může mít více médií

Po převodu vznikne následující struktura. Je třeba použít všechny doporučené postupy při návrhu struktury databáze.



5.2 Vytváření metadat

Definice metadat se velmi podobá definici struktury databáze známé z běžných projektů. Pouze je posunuta o úroveň výš, do metadat.

5.2.1 Tabulky

Při návrhu postupujeme stejně, jako při tvorbě klasické databázové struktury až do chvíle, kdy by přišlo na řadu psaní dotazů. Tedy do chvíle, kdy máme ER diagram převeden na soustavu databázových tabulek. Pak můžeme začít definovat metadata. My se zaměříme na tabulku médium, u ostatních tabulek se postupuje analogicky.

Nejprve si nadefinujeme tag „Table," do kterého později vepíšeme strukturu tabulky. Tag bude umístěn v kořenovém elementu XML souboru. Atribut „name" je typu ID a specifikuje jméno, kterým budeme tabulku adresovat.

```
<Table name="medium">

</Table>
```

Dále si vytvoříme sloupec tabulky, který bude jejím jediným primárním klíčem (viz omezení metadat v hlavním dokumentu). Navíc pokud je potřeba, aby šel tento sloupec použít v cizích klíčích, musí mít přiřazeno ID.

```
<Column name="id" type="INT" null="false" autoincrement="true" primarykey="true" index="true"
id="medium__id"/>
```

Další klíč ukazující do tabulky titulů vytvoříme takto. Atribut „foreign" je typu ID a musí referovat na sloupec jiné tabulky. Pokud chceme použít sloupec v pohledu, musí obsahovat tag(y) typu popis sloupce s jednoznačným ID. Pokud se jedná o cizí klíč, nemusí mít název (resp. název nemá smysl). Případně je možno doplnit popis sloupce.

```
  <Column name="title" foreign="title__id" index="true" null="false">
    <ColumnDescription id="medium__title"/>
  </Column>
```

Celá definice tabulky pak může vypadat například takto. Více o vytváření sloupců se dozvíte v odstavci 5.2.2.

```
<Table name="medium">
```

```
    <Column name="id" type="INT" null="false" autoincrement="true" primarykey="true"
index="true" id="medium__id"/>

    <Column name="medium_type" foreign="medium_type__id" index="true" null="false">
      <ColumnDescription id="medium__type"/>
    </Column>

    Column name="title" foreign="title__id" index="true" null="false">
      <ColumnDescription id="medium__title"/>
    </Column>

    <Column name="note" type="TEXT" null="false">
      <ColumnDescription id="medium__note" caption="poznámka" description="Poznámka ke
klientovi."/>
    </Column>

    <Column name="active" type="ENUM" typeex="'ano', 'ne'" null="false" default="ano">
      <ColumnDescription id="medium__active" caption="aktivní" description="Je klient aktivní?"/>
    </Column>

  </Table>
```

5.2.2 Zápis sloupců databáze

Tento odstavec podrobně popisuje atributy sloupce tabulky. Sloupcem lze popsat velkou část možností v SQL, ne však všechny.

- name
- type – typ sloupce, jedna položka z:
  - INT
  - TEXT
  - TINYTEXT
  - SET
  - ENUM
  - DATE
  - CHAR

    - PASSWORD – speciální typ, vytvoří se jako text, ale při ukládání se ukládá SHA1 otisk dat
- length – délka dat sloupce, závisí na typu – viz manuál k MySQL
- null – true, pokud má být možno sloupec nenastavit
- default – přednastavená hodnota sloupce
- autoincrement – true, pokud se má sloupec inkrementovat automaticky
- primarykey – true, pokud má být sloupec primární klíč
- index – true, pokud má být sloupec indexován – primární klíč musí být i index!
- unique – true, pokud mají být hodnoty ve sloupcích unikátní
- typeex – pokud je typ ENUM nebo SET, obsahuje seznam prvků oddělených čárkou
- foreign – ID primárního klíče z cizí tabulky, pokud je tento sloupec cizí klíč
- constraint – ID omezení uloženého v témž souboru

5.2.3 Omezení

Omezení dat jsou v podstatě pojmenované bloky kódu v PHP, které validují hodnotu dat při ukládání do databáze. Předpokládá se, že blok bude tvořit tělo funkce vracející hodnotu typu boolean a jako jediný parametr s názvem „$value“ dostává hodnotu sloupce. Vrací hodnotu true, pokud jsou data validní.

```
<Constraint name='constr_phone' description='\bbTelefonní číslo může být ve formátu:\be\n+420 789 987 987\n+420789987987\n789 987 987\n789987987\n-nemusí být zadáno'>
 <![CDATA[
  return strlen($value)==0||preg_match("/^(\+\d{3})? ?\d{3} ?\d{3} ?\d{3}$/", $value);
 ]]>
 </Constraint>
```

Tento příklad ukazuje omezení typu telefonní číslo. Atribut „name“ je jednoznačný identifikátor. Atribut „description“ používá jednoduché formátování:

- \bb – styl textu „nadpis“ po další změnu stylu
- \be – „normální“ styl textu po další změnu stylu
- \n – nový řádek

5.2.4 Pohledy

Pohled musí být složen pouze ze sloupců uvedené tabulky, či z referencí na jiné pohledy. Toto bohužel nejde ošetřit v metadatech, takže se chyba objeví až při použití.

Nejprve si nadefinujeme vlastní pohled. Je identifikován pomocí jednoznačného ID, obsahuje určení hlavní tabulky a tituly výpisu pohledu a formuláře. Titulek formuláře obsahuje konec názvu – před něj se doplňuje „editovat, vytvořit, uložit či prohlížet.“

```
<View id="medium_main_view" caption="Média" table="medium" form_caption="média">
 </View>
```

Reference na sloupec referuje na jeho popis. Jiné atributy nepotřebuje, vše je uloženo v popisu a definici sloupce.

```
<ColumnRef ref="medium__active"/>
```

Reference na jiný pohled se provádí pomocí určení pohledu (atribut „ref“) a cizího klíče v hlavní tabulce (atribut „foreign“).

```
<ViewRef ref="title_ref_view" foreign="medium__title"/>
```

Celá definice pohledu by se tedy dala zapsat asi takto. Je samozřejmě možno vytvořit definic několik a uživateli odkrývat tolik dat, na kolik má právo, případně kolik chce.

```
<View id="medium_main_view" caption="Média" table="medium" form_caption="média">
   <ColumnRef ref="medium__active"/>
   <ViewRef ref="title_ref_view" foreign="medium__title"/>
   <ViewRef ref="medium_type_main_view" foreign="medium__type"/>
 </View>
```

5.3 Nastavení prostředí

Požadavky na prostředí jsou uvedeny v hlavním dokumentu. Při splnění všech podmínek by neměl být problém knihovnu používat.

Prvním krokem je vložení hlavního hlavičkového souboru knihovny, který obsahuje metodu pro automatické načtení potřebných souborů v případě potřeby.

```
include_once("./include/flcx/flcx.php");
```

Dalším krokem je vytvoření funkce, která se volá v případě, že PHP objeví instancování třídy, kterou nezná. Pokud víme, že požadovaná třída pochází z knihovny, můžeme zavolat funkci „onDemandIncludeFLCX“, která zařídí načtení požadovaných souborů. Pokud třídu nezná, vrátí výjimku. Tento způsob vkládáni je mnohem více efektivní, než načítat všechny hlavičky a zároveň snadnější, než v každém souboru uvádět potřebné hlavičky.

```
function __autoload($class)
 {
  $file = '';

  switch ($class)
  {
   case 'CEmployee': $file='CEmployee.php'; break;
   ...

   default: onDemandIncludeFLCX($class, './include/flcx/'); return;
  }

  require_once('./include/'.$file);
 }
```

Dále je potřeba definovat fyzické názvy tabulek pro symbolické názvy v metadatech. A také zadat kořenový element XML souboru s metadaty. Respektive objekt z DOM stromu XML.

```
...
public function getDbXmlRoot()
{
$this->dbXmlDef = new DOMDocument();
$this->dbXmlDef->validateOnParse = TRUE;
$this->dbXmlDef->resolveExternals = TRUE;
if ( !$this->dbXmlDef->load("./xml/db.xml") )
        {
                throw new CException('XML definition isn\'t valid!',
"\$php_errormsg=$php_errormsg");
        }
        return $this->dbXmlDef;
```

```
}
...

CConfiguration::setXmlDocument($g_Sess->getDbXmlRoot());

CConfiguration::addTableSchema('medium_type', 'medium_type');
CConfiguration::addTableSchema('distributor', 'distributor');
CConfiguration::addTableSchema('genre', 'genre');
CConfiguration::addTableSchema('permission_type', 'permission_type');
CConfiguration::addTableSchema('employee', 'employee');
CConfiguration::addTableSchema('title', 'title');
CConfiguration::addTableSchema('medium', 'medium');
CConfiguration::addTableSchema('client', 'client');
CConfiguration::addTableSchema('permission_set', 'permission_set');
```

V této chvíli by knihovna měla fungovat. Pokud nebude něco v pořádku, bude vyvolána výjimka. Proto by při použití měla být definována globální procedura pro odchycení nezachycených výjimek.

```
function myExceptionHandler($exc)
{
	echo('<br/><br/><br/><br/>');
	echo('<b>Exception was thrown!</b><br/>');
	echo('<br/><br/>');
	echo(str_replace("\n", '<br/>', $exc->getMessage()));
	echo('<br/><br/>');
	echo(str_replace("\n", '<br/>', $exc->getMessageEx()));
	echo('<br/><br/>');
	echo(str_replace("\n", '<br/>', $exc->getTraceAsString()));
exit;
}
```

Dále je potřeba vytvořit objekt pro připojení k databázi. V ideálním případě se vytvoří jen jednou za běh skriptu. Připojení se provede až při prvním pokusu o přístup k identifikátoru spojení. Databáze samozřejmě musí existovat.

```
$this->_adminConn = new CDbConnection("localhost", "root", "xyz", "mojedb");
...
public function &getAdminConnection()
{
    return $this->_adminConn;
}
...
```

5.4 Tvorba tabulek podle schématu

Tvorba tabulek je velmi snadná. Zadává se i jméno fyzické tabulky pro případ tvoření více tabulek se stejnou strukturou. Hondota „true“ v konstruktoru značí, že chceme vypisovat logy o vytváření.

```
$builder = new CTableBuilder($g_Sess->getAdminConnection(), true);
$builder->doTable('medium', 'medium');
```

5.5 Výpis pohledů

Níže uvedený kód vypisuje obsah pohledu stránkovaný podle prvního písmene titulu a po 20 záznamech na stránku. Také definuje funkci, která vrací hodnoty přídavného sloupce. Hodnota „$g_Sess“ v konstruktoru je reference na objekt, který implementuje rozhraní „CVariablesStorage“. Při volání metody „drawTable“ se jako poslední parametr specifikuje formulář pro řádky tabulky.

```
function drawMediumColumn(CViewMaker &$table, $bHeader, &$rset=NULL)
{
if ( $bHeader )
        {
                return "média";
        }
        else
        {
                return rand();
        }
 }
```

```
$tab = new CViewMaker($g_Sess->getAdminConnection(), $g_Sess);
$tab->addExtColumn('drawMediumColumn');

$tab->setPagingLetter('title_simple_view.title');
$tab->setPagingNum(20);
$tab->drawTable('title_simple_view', 'titleForm.php');
```

5.6 Editace pohledů

Další ukázka vytvoří formulář a při pokusu o vytvoření záznamu zabezpečení nejprve přidá informaci o zaměstnanci, jemuž patří.

```
function onBeforeCreate(&$form)
{
   global $g_Sess;
   $empid = $g_Sess->getVariable('permissionSet.php', 'emp_id');
   $form->setValue('employee', $empid);
   return true;
 }

$form = new CFormMaker($g_Sess->getAdminConnection());
$form->setEventHandler('on_before_create', 'onBeforeCreate');
$form->drawForm("permission_set_main");
```

5.7 Bezpečnost

Knihovna poskytuje velkou míru bezpečnosti, ale programátor ji musí umět bezpečně použít. Knihovna samotná je zabezpečena proti útoku typu „SQL Injection“ a „Script Injection.“

Prvním požadavkem je, aby uživatel knihovny nadefinoval funkce pro odchycení neočekávané výjimky, případně chyby a skryl ji uživateli. Je velmi vhodné tyto chyby ukládat do logů. Standardní výpis na obrazovku může být velmi nebezpečný.

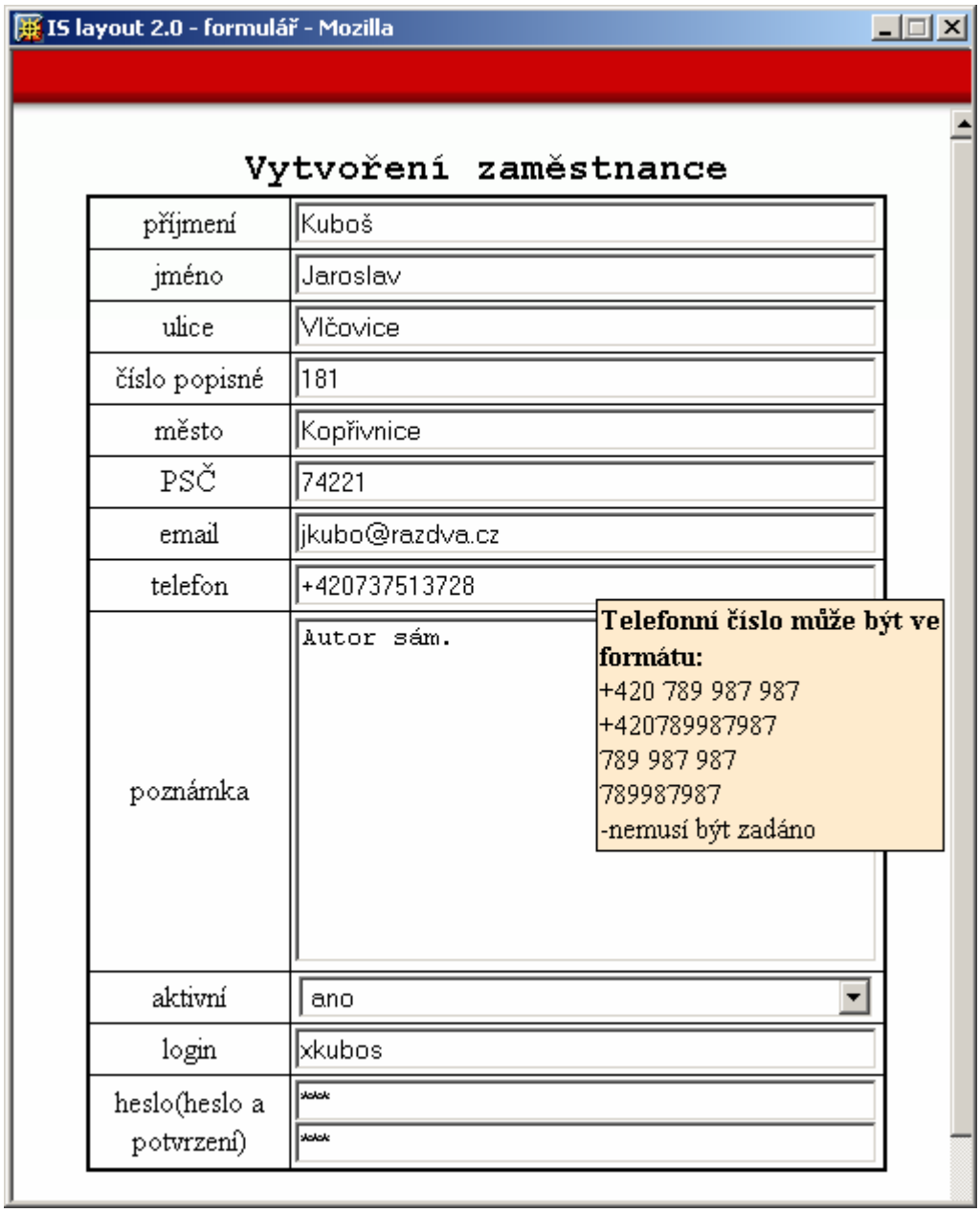
Dalším problémem jsou chyby PHP, MySQL a webového serveru. Pro tyto aplikace jsou neustále vyvíjeny opravy a je zapotřebí tyto opravy aplikovat. V opačném případě jsou veškerá opatření v knihovně či v aplikaci zbytečná.

Je potřeba mít stále na paměti, že člověk nemůže být nikdy dost paranoidní. Je nutné si uvědomit, že knihovna, byť je sama bezpečná, nedokáže ochránit celou aplikaci a veškerá zodpovědnost leží na bedrech programátora.

5.8 Použití a výsledný vzhled

Na obrázku je uvedena ukázka formuláře otevřeného z tabulky do nového okna. Jde o formulář editující zaměstnance videopůjčovny. Je zobrazen informační text o formátu pole. Stránka je vygenerována v XHTML a používá JavaScript. Kód:

```
<?
        include_once('./formLayout.php');

 function onBeforeCreate(&$form)
        {
  global $g_Sess;

  if ( $g_Sess->hasVariable('employee.php', 'bra_id') )
  {
   $braid = $g_Sess->getVariable('employee.php', 'bra_id');
   $form->setValue('branch', $braid);
  }

  return true;
 }

 $form = new CFormMaker($g_Sess->getAdminConnection());

 $form->setEventHandler('on_before_create', 'onBeforeCreate');

 $form->drawForm("employee_main_view_form");
?>
```



Na dalším obrázku je zobrazen pohled na tituly videopůjčovny. Je použito kombinované stránkové zobrazení (první písmeno názvu+20 záznamů na stránku). Sloupce „Žánry“ a „Tituly“ jsou obsahy tabulek, s nimiž má tabulka titulů vazbu. Kód:

```
<?
        include_once('./layout.php');

        function drawPermissionColumn(&$table, $bHeader, &$rset=NULL)
        {
          if ( $bHeader )
```

```
            {
     return "prava";
    }
    else
    {

     $link = "<a href='permissionSet.php?fid=";
     $link .= $rset->getValue('employee_main_view_id');
     $link .= "'><img src='./images/lock.png' alt='' style='border-style: none;'></a>";

     return $link;
    }
  }

          $tab = new CViewMaker($g_Sess->getAdminConnection(), $g_Sess);

          $tab->addExtColumn('drawPermissionColumn');

          $tab->drawTable('employee_main_view', 'employeeForm.php');
?>
```

Tituly

Tituly		@Distributoři	Tituly	@Žánry	
akce	**Titul**	**Distributor**	**Vyrobeno**	**Žánr**	**média**
	Aljaška v plamenech	Warner Bros	1994	akční	
	Agent z Hong Kongu	SPI	2001	akční	
	Aréna smrti	Hollywood Ent.	2001	akční	
	Americká krása	Bonton Home	2000	drama	
	Anakonda	Bonton Home	1998	drama	
	Anatomie	Bonton Home	2000	horor	
	Alice Cooper	Sony Music Bonton	2003	hudební DVD	
	AC DC Live at Donington	Sony Music Bonton	2003	hudební DVD	
	Audioslave	Sony Music Bonton	2003	hudební DVD	
	Anastacia	Sony Music Bonton	2000	hudební DVD	
	Alice in Chains - Music Bank T	Sony Music Bonton	2001	hudební DVD	
	Andrew - člen naší rodiny	Bonton Home	2000	komedie	
	Arachnofobie	Warner Bros	2002	komedie	
	Austin Powers - Goldmember	Warner Bros	2002	komedie	
	Agent v sukni	Bonton Home	2002	komedie	
	Ach ty ženy	Warner Bros	2002	komedie	
	Adaptace	SPI	2002	komedie	
	Agenti Dementi	Intersonic	2004	komedie	
	Amélie z Montmartu	Intersonic	2004	komedie	
	Agent Cody Banks 2	Bonton Home	2004	komedie	

6 Shrnutí

6.1 Testování

K testování byly použity oba dva hlavní typy testů. Tedy statické a dynamické testování. Jednoduchosti testování napomáhal fakt, že je celá knihovna postavena na výjimkách. Pokud se kdekoliv (zejména v nízkoúrovňových metodách) v knihovně vyskytne potencionální problém, vytvoří se výjimka. Kód je hustě proložen testy na mnoho různých rysů, které nehlídá PHP automaticky:

- datové typy, zejména typ třídy objektu – jednak se hlídá typ třídy u parametrů volně dostupných metod, ale také typy objektů vytvářených automaticky
- nulové reference
- chyby v SQL dotazech
- chyby v připojení k databázi
- chyby v datech posílaných během session mezi stránkami – uživatel se rozhodne neposlat některé prvky formuláře
- chyby použití – například nezavolané metody pro nastavení objektu

PHP je náchylné na kvalitu práce programátora a je třeba si ohlídat mnoho věcí, za které v překládaných jazycích ručí překladač – kontrola typů, výrazů, ... Proto se jakákoliv výjimka nebo hlášená chyba, či varování zobrazí jako kritické – ukončí se běh skriptu.

Další chybou, na kterou se je potřeba soustředit, jsou útoky zvenčí. V našem případě se jedná hlavně o útoky typu „SQL injection.“ Jedná se vlastně o vložení krátkého SQL skriptu do dat posílaných formuláři v naději, že hodnota z POST/GET proměnné nebude ošetřena a dostane se do dotazu. V tom případě lze smazat celou databázi tímto jediným příkazem. Mnohem horší ale je, pokud se pomocí těchto útoků útočník dozví o struktuře databáze, případně se dostane k údajům z ní. Potom má už jen 2 možnosti – data z databáze ukrást, nebo je změnit k obrazu svému. Proto jsem se pokoušel o útok tohoto typu, ovšem bezvýsledně. Veškerá data se totiž před použitím zbaví nebezpečných znaků převodem na tvar s tzv. „escape“ sekvencemi – tedy ve tvaru \x. Pak už nebezpečí tohoto druhu nehrozí.

6.1.1 Dynamické

Při tomto typu testování se produkt posuzuje jako celek, respektive jeho transformace vstupů na výstupy. U dynamického testování rozlišujeme 2 hlavní postupy:

- metoda černé skříňky - O vnitřní struktuře produktu není známo nic. Jako vstupní data lze použít reprezentativní množinu dat (ta by měla být co nejkomplexnější), které má produkt zpracovat, ale také chybové vstupy. Mezi chybové vstupy je potřeba zahrnout také známé chyby produktů stejného druhu, případně produktů vytvořených stejnými nástroji. Knihovna byla po celou dobu vývoje testována pomocí testovacího informačního systému, který demonstruje všechny dostupné rysy. Zároveň byl testovací IS navržen tak, aby pokrýval pokud možno všechny možnosti použití – tedy celou funkcionalitu. Testování by šlo velmi dobře zautomatizovat některým z tržně dostupných testovacích robotů.

- metoda bílé skříňky - Při testování metodou bílé skříňky je oproti metodě černé skříňky vnitřní struktura produktu známa. Testují se tedy problematická místa v kódu jako spolupráce tříd a komponent, funkčnost metod, atd.

6.1.2 Statické

Statické testování se provádí procházením zdrojových kódů a hledáním chyb v návrhu, či implementaci.

- Při vývoji – programátor kontroluje funkčnost a správnost kódu bezprostředně po jeho vytvoření, či při integraci do systému.
- Při revizi kódu – programátor prochází kód, hledá podezřelé a nebezpečné úseky a ty testuje. Tuto fázi je vhodné spojit s komentováním a refaktorizací kódu. Je možné ji provádět ve skupině, případně při akci zvané „code review.“ Tedy při procházení kódu na poradě.

7 Závěr

Projekt v současné době zcela splňuje zadání a specifikaci. Všechny činnosti na projektu byly dovedeny do úspěšného cíle, knihovna je připravena k použití. Při jejím vytváření vyšlo najevo několik skutečností, které by měly být reflektovány při práci na budoucích verzích. Jedná se především o vlastnosti a funkčnost usnadňující použití či snižující podíl napsaného kódu nutného k použití. Mezi takovéto myšlenky patří:

- správa verzí databáze – možnost popsat stav databáze různých verzí a změny nutné k převodu
- správa změn v databázi – možnost nadefinovat akce typu přejmenování sloupce v databázi, ...
- podpora jiných databázových strojů
- program s grafickým rozhraním pro definování metadat (ročníkový projekt?)
- automatická tvorba databázové vrstvy

Vlastní přínos vidím v realizaci knihovny, která má velmi dobré vlastnosti a nenutí programátora znát je nebo se jimi řídit. Vnáší do zmatečného programování formulářů a tabulek načítaných z databáze v PHP nový rozměr. Použitím knihovny se snižuje počet chyb, zlepšuje udržovatelnost programů, zvyšuje konzistence. Při korektním používání se metadata v programu vyskytují jen jednou, není třeba psát složité SQL dotazy. Knihovnu je možno definovanými postupy přizpůsobit jak graficky, tak funkčně.

Projekt bych chtěl rozšířit prostřednictvím své diplomové práce, případně realizovat v jiném jazyce, tentokrát kompilovaném. Dále bych ho chtěl zaregistrovat na www.sourceforge.com jako projekt šířený pod licencí GPL. Tímto přístupem se umožní používání knihovny v praxi. Žádost o povolení FIT VUT ke zveřejnění jsem už podal, odpověď mi nebyla v době tisku práce známa.

Literatura

[1] Page-Jones, M. Základy objektově orientovaného návrhu v UML. Praha, Grada 2001.

[2] Arlow, J., Neustadt I. UML a unifikovaný proces vývoje aplikací. Brno, Computer Press 2002.

[3] Achour, M., aj. PHP Manual. PHP Documentation Group, 2005. Dokument dostupný na URL http://www.php.net/manual/en/ (duben 2005).

[4] MySQL Reference Manual. MySQL AB, 2005. Dokument dostupný na URL http://dev.mysql.com/doc/mysql/en/index.html (duben 2005).

[5] Yergeau F, aj. Extensible Markup Language (XML) 1.0 (Third Edition) W3C Recommendation, W3C, 4. prosince 2004. Dokument dostupný na URL http://www.w3.org/TR/2004/REC-xml-20040204 (duben 2005).

[6] W3C, XHTML 1.0 The Extensible HyperText Markup Language (Second Edition) W3C Recommendation, 26. ledna 2000. Dokument dostupný na URL http://www.w3.org/TR/xhtml1/ (duben 2005).

[7] Interval.cz: webdesign a vývoj aplikací denně, Zoner Software, 2005, ISSN 1212-8651

[8] Kosek, J. Seriál o XML pro Softwarové noviny, 2000. Dokument dostupný na URL http://www.kosek.cz/clanky/swn-xml/index.html (duben 2005).

Příloha 1 – DTD pro validaci metadat

Převodem v dokumentu uvedených pravidel vznikl následující DTD soubor popisující strukturu XML souboru pro popis metadat. Nejprve vznikla jeho struktura a pak byly doplňovány atributy podle potřeby. Některé byly známy už při návrhu, jiné bylo třeba přidat při programování.

```
<!--
  Definition of DTD for XML definition file of FLCX library
  *Author: Jaroslav Kubos, xkubos00@stud.fit.vutbr.cz
  *Date: 2005/03/29
  *Copyright: FIT, VUT Brno
  *Description: part of bachelor project
-->

<!-- definition of root element -->
<!ELEMENT Database (Table+, View*, Constraint*)>

<!-- table structure -->
<!ELEMENT Table (Column+, ConcatColumn*)>
<!ATTLIST Table
        name ID #REQUIRED
>

<!-- column of table -->
<!ELEMENT Column (ColumnDescription*)>
<!ATTLIST Column
              id ID #IMPLIED
        name CDATA #REQUIRED
        type (INT|TEXT|TINYTEXT|SET|ENUM|DATE|CHAR|PASSWORD) #IMPLIED
        length CDATA #IMPLIED
        null CDATA #REQUIRED
        default CDATA #IMPLIED
        autoincrement CDATA #IMPLIED
        primarykey CDATA #IMPLIED
        index CDATA #IMPLIED
        unique CDATA #IMPLIED
```

```
        typeex CDATA #IMPLIED
        foreign IDREF #IMPLIED
        constraint IDREF #IMPLIED
>

<!-- concatenated column -->
<!ELEMENT ConcatColumn (ColumnRef+)>
<!ATTLIST ConcatColumn
            id ID #REQUIRED
        caption CDATA #REQUIRED
        description CDATA #REQUIRED
>

<!-- description of column -->
<!ELEMENT ColumnDescription EMPTY>
<!ATTLIST ColumnDescription
            id ID #REQUIRED
        caption CDATA ""
        description CDATA ""
>

<!-- reference to column -->
<!ELEMENT ColumnRef EMPTY>
<!ATTLIST ColumnRef
        ref IDREF #REQUIRED
>

<!-- reference to concatenated column -->
<!ELEMENT ConcatColumnRef EMPTY>
<!ATTLIST ConcatColumnRef
        ref IDREF #REQUIRED
>

<!-- reference to view -->
<!ELEMENT ViewRef EMPTY>
<!ATTLIST ViewRef
        ref IDREF #REQUIRED
```

```
        foreign IDREF #REQUIRED
    >

    <!-- view definition -->
    <!ELEMENT View (ConcatColumnRef|ColumnRef|ViewRef)+>
    <!ATTLIST View
        id ID #REQUIRED
        caption CDATA #REQUIRED
        form_caption CDATA #REQUIRED
        table IDREF #REQUIRED
    >

    <!-- constraint of field definition -->
    <!ELEMENT Constraint (#PCDATA)>
    <!ATTLIST Constraint
        name ID #REQUIRED
        description CDATA #REQUIRED
    >
```