MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Všeobecné webové rozhranie pre databázy

BAKALÁRSKA PRÁCA

Juraj Michálek

Brno, Jar 2003

Poďakovanie

Ďakujem prof. Ing. Pavlovi Zezulovi za odbornú pomoc a pripomienky. Ďakujem Hane Prokešovej za kvalitné dátové modely a dobré nápady, ktoré mi pomohli vytváraní tejto práce. Ďakujem Michalovi Úradníkovi za kritické hodnotenia navrhnutých dátových štruktúr. Ďakujem všetkým vývojárom pracujúcich na vývoji open source softvéru, ktorý som využil pri svojej práci.

Prehlásenie

Prehlasujem, že táto bakalárska práca je mojim pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci adekvátne citujem s uvedením úplného odkazu na príslušný zdroj.

Vedúci práce: prof. Ing. Pavel Zezula, CSc.

Zhrnutie

Predkladaná práca prináša náhľad do problematiky tvorby a popisu webového rozhrania nad databázami. Tvorba webového rozhrania so sebou nesie úskalia, na ktoré musí návrhár pamätať. Pre tvorbou nového rozhrania je vhodné zistiť, aké riešenia sú v súčastnosti k dispozícii. Táto práca obsahuje popis riešenia, ktoré som navrhol a realizoval. Jedná sa o nástroj, ktorý umožňuje špecifikovať webovú aplikáciu pomocou jazyka Perl alebo XML. Implementácia tohoto riešenia sa ukázala ako prínosná a našla svoje využitie v niekoľkých projektoch.

Kľúčové slová

Perl, XML, CGI, databáza, databázové rozhranie, webová aplikácia

Obsah

Úvod 2
1 **Problematika používateľských rozhraní** 4
1.1 Podrobná formulácia problémov 4
1.2 Stručné zachytenie cieľov 7
2 **Súčasné riešenia a ich nedostatky** 8
2.1 DB Browser 8
2.2 dbMan 8
2.3 phpMyAdmin 9
2.4 Plone 9
2.5 SOFIA 10
2.6 Porovnanie 10
3 **Medvedb** 12
3.1 Konceptuálny model 13
3.2 Logický model 14
3.3 Fyzický model 15
3.4 Nástroje na popis používateľského rozhrania 16
3.4.1 Dátová štruktúra hash 17
3.4.2 Organizácia popisu do stromu 19
3.4.3 Definícia pomocou stromu 19
3.4.4 Implementácia s využitím štruktúry hash 21
3.4.5 Strom XML 21
3.4.6 Rozšírenie XML o dynamické prvky 23
3.5 Nástroje na popis typov 24
3.6 Tvorba aplikácie 26
3.7 Informácie o module 27
4 **Nasadenie v praxi** 28
4.1 Databáza počítačov FSS 28
4.2 Správa článkov časopisu Malá Fronta 28
4.3 Správa portálu Game for Linux 29
Záver 30

Úvod

Spracovanie dát vyžaduje nemalé prostriedky. V minulosti neboli k dispozícii digitálne technológie na spracovanie dát. Využívali sa rôzne kartotéky, zložky. Digitalizácia umožnila zmenu papierových archívov na digitálne. Značný prínos znamenal vznik databáz, ktoré v súčastnosti plne nahradzujú kartotéky. Zmenilo sa „rozhranie" medzi človekom a dátami. Tam, kde v minulosti stačilo pero, papier a korektor, je nutné nasadiť systém, ktorý je jednoduchý, zrozumiteľný. Používateľ takéhoto systému musí mať pocit, že má systém pod kontrolou a vie si svoju prácu v systéme spojiť s prácou na reálnej kartotéke zloženej z papierových lístkov.

Z programátorského hľadiska predstavuje tvorba používateľského rozhrania väčšinou rutinnú prácu. Pri tvorbe prostredia sa dodržiavajú tie isté postupy. Keď porovnáme viaceré používateľské prostredia, zistíme, že sa odlišujú len v určitých prvkoch (napr.: rozloženie položiek, ich formátovanie, farba). Rovnakou zostáva forma dialógu medzi používateľom a systémom. Je pomerne náročné naprogramovať dobré používateľské rozhranie a nezabudnúť pri tom na nejaký detail. Tu sa ponúka otázka, či pri tvorbe nového prostredia nutné vyvíjať všetko odznova.

Pre tvorcu systému je veľmi výhodné, pokiaľ má k dispozícii dobrý nástroj, ktorý pri tvorbe rozhrania vykoná väčšinu rutinných činností. Tvorca systému teda nie je zbytočne nútený premýšľať nad drobnými detailami. Na programátorskej úrovni to väčšinou znamená, že programátor má k dispozícii vhodné knižnice obsahujúce dopredu naprogramované funkcie. Oveľa lepšia je situácia v prípade, že autor má k dispozícii akúsi „čiernu skrinku", ktorej povie základné parametre, a tá vytvorí nejaký výstup. V prípade, že je autor nespokojný dodá ďalšie parametre a výstup spresní. Takýto prístup sa volí pri tvorbe objektových systémov.

Základné požiadavky na nástroj umožňujúci jednoduchú tvorbu

používateľského rozhrania sú: jednoduchá špecifikácia používateľského rozhrania, možnosť upresnenia detailov, prednastavené správanie, ktoré je možné modifikovať, pripravené funkcie. Nástroj by mal zo základnej znalosti informácii o štruktúre databáze byť schopný vytvoriť kompletné používateľské rozhranie. Práve vytvorenie takéhoto nástroja je cieľom tejto bakalárskej práce.

V prvej kapitole tejto práce sa zaoberám vymedzením problémovej oblasti. Druhá kapitola je venovaná prehľadu existujúcich riešení, hodnoteniu ich kladov a záporov. Tretia kapitola obsahuje konkrétne riešenie, ku ktorému som počas svojej práce dospel. Je tam zachytená problematika popisu používateľského rozhrania a spôsob riešenia tohoto popisu. Poslednú kapitolu tvorí prehľad úspešného nasadenia aplikácie v praxi.

Kapitola 1

Problematika používateľských rozhraní

V tejto kapitole sú zachytené základné problémy, s ktorými sa tvorca informačného systému môže stretnúť. Na základe kritérií, ktoré si tvorca systému stanoví, si môže následne vybrať nástroj, ktorý mu pomôže pri tvorbe systému. Každé z riešení, ktoré sú k dispozícii, má určité nedostatky. Tie je nutné zvážiť a zohľadniť.

1.1 Podrobná formulácia problémov

Na priblíženie problematiky si skúsme predstaviť, že máme navrhnúť systém, ktorého vývoj bude v prevažnej časti zameraný na používateľské rozhranie. Ak jeden takýto systém vytvoríme, môžeme si položiť otázku: „Aké množstvo práce si môžeme ušetriť pri vývoji ďalšieho, podobného systému, ak použijeme nejaké časti z predchádzajúceho systému?" Je to pomerne pragmatická otázka, pretože v prípade, že môžeme „recyklovať" to, čo sme už vytvorili, je možné ušetriť prostriedky a čas.

Na načrtnutie povahy problému uvediem malé hodnotenie, ktorá vychádza z mojej osobnej skúsenosti. Pri vyhodnotení niektorých menších systémov pracujúcich nad databázou cez webové používateľské rozhranie som dospel k nasledujúcim záverom záverom:

- 70% z celkového vývoja si vyžaduje vytvorenie programu, ktorý transformuje dáta na formuláre
- 10% vývoja predstavuje spracovanie vstupných dát
- 20% všeobecné úpravy a oprava rôznych chýb

Malé systémy sú často tak jednoduché, že ich databázová štruktúra sa dá reprezentovať používateľovi ako sústava jednoduchých tabuliek. Pri tvorbe takéhoto systému stačí programátorovi, aby zobral štruktúry tabuliek a pretransformoval ich do podoby webovej stránky, prípadne aby doplnil „cukor" — špeciálne funkcie zjednodušujúce prácu (napr. vyhľadávanie). Pri takomto systéme najväčšiu úsporu práce predstavuje využitie vhodnej knižnice, alebo modulu, ktorý sa postará o transformáciu.

U väčších systémov, využívajúcich relácie, je nutné dať používateľovi namiesto strohého výpisu identifikátoru relácie, kompletný výpis hodnôt z tabuľky, na ktorú sa vzťahuje identifikátor. Zoberme si jednoduchý adresár, kde v jednej tabuľke sú údaje o adresách a tá je spojená číselným identifikátorom s druhou tabuľkou, kde sa nachádzajú mená miest. Bolo by veľmi nepraktické, keby systém používateľovi, namiesto mena mesta v adrese, zobrazil len nič nehovoriace číslo. Pri systéme pracujúcom s reláciami by bolo možné ušetriť programátorskú prácu použitím mechanizmu, ktorý prevádza výbery z rôznych tabuliek a spája ich do jednej.

Samotnú prácu s databázou môžeme rozdeliť do dvoch kategórii. Jednak sú to operácie výlučne zamerané iba na na čítanie a prezeranie dát. A potom operácie, kde sa prevádza zmena údajov (príp. vytváranie nových položiek). Na prvý pohľad sa môže zdať, že tieto dve kategórie sú veľmi rozdielne. Na vytvorenie rozhrania pre prezeranie dát a rozhrania pre modifikáciu by sa museli teda použiť diametrálne odlišné postupy. Takéto tvrdenie je zavádzajúce a nepravdivé. Využitím správneho nástroja sa však tieto rozdiely dajú minimalizovať a na popis oboch situácii je možné využiť dokonca tie isté prostriedky.

V prevažnej väčšine prípadov vyzerá používateľské rozhranie rovnako a na jeho vytvorenie sa použijú tie isté postupy. Preto existuje množstvo pripravených funkcií, ktoré má programátor k dispozícii. Niekedy však môže nastať situácia, že žiadna z dostupných funkcií nemá požadované vlastnosti a je nutné napísať si vlastnú. Príjemným zjednodušením je nástroj, ktorý automaticky ponúka štandardné riešenie, avšak programátora neobmedzuje pri tvorbe vlastného riešenia.

Programátori pri tvorbe menších systémov nemajú príliš veľkú motiváciu na ich zabezpečovanie. Pri webových rozhraniach spočí-

va väčšina zabezpečenia na kontrole dát, ktoré používateľ do systému odoslal [**?**]. K pomerne častým programátorským prehreškom patrí napríklad priame vkladanie dát z používateľovej požiadavky do databázy. Stačí, aby sa našiel trochu skúsenejší používateľ a vložením presne naformátovaných dát môže získať plný prístup do databázy, alebo ju v horšom prípade zničiť. Zabezpečenie predstavuje teda ďalšiu oblasť, na ktorú sa dajú použiť všeobecne známe postupy.

V každom trochu rozsiahlejšom systéme je nutné pracovať nad niekoľkými tabuľkami z databáze. Nad každú tabuľku je postavený samostatný podprogram spracúvajúci a interpretujúci dáta. Vítaným zjednodušením, je možnosť tabuľky medzi sebou prepojiť, aby si používateľ mohol jednoducho vybrať, nad ktorými dátami bude pracovať.

Slabšie databázové stroje zväčša neumožňujú prevádzať s výsledkom SQL požiadavky zložitejšie operácie, ako napríklad spájanie reťazcov z viacerých požiadaviek a pod. Ak sa programátor spoľahne na funkcie databázového stroja, musí počítať s tým, že jeho systém bude na tomto stroji závislý a prechod na iný databázový stroj bude náročný. Pri malých systémoch býva prechod na iný databázový stroj „smrteľný" a často je výhodnejšie celý systém napísať nanovo. Pre programátora by bolo užitočné, aby mal k dispozícii nástroj, ktorý mu zjednodušil prechod medzi databázovými strojmi.

Každý používateľ má iné požiadavky na grafickú reprezentáciu celého systému. Tu hrajú hlavnú rolu farby, písma, rôzne ohraničenia a orámovania. Používateľove názory na to, ako by mal systém vyzerať, kam by mala prísť aká farba sa môžu postupne meniť. Dobre navrhnutý systém by malo byť jednoduché modifikovať a prispôsobiť ho používateľovým predstavám.

Vítanými vlastnosťami sú aj rôzne triedenia, vyhľadávania, záložky a rôzne ďalšie detaily, spriemňujúce používateľovi prácu. Používateľ ocení, ak bude mať celý systém rovnaký dizajn a logiku práce.

Vytvoriť nástroj, ktorý by ponúkal všetky vyššie uvedené možnosti je veľmi náročné. Na druhej strane, ak by bol k dispozícii nástroj, ktorý by mal aspoň časť z vyššie uvedených vlastností, bude znamenať prínos.

1.2 Stručné zachytenie cieľov

V predchádzajúcej kapitole boli všeobecne formulované vlastnosti, ktoré umožňujú znížiť záťaž pri tvorbe systémov. Tieto vlastnosti môžeme zhrnúť do niekoľkých bodov:

1. jednoduchá tvorba pohľadov na tabuľky
2. zobrazovanie dát z relačných tabuliek
3. podobný popis rozhrania pre prezeranie a úpravu dát
4. štandardné funkcie, avšak je možné špecifikovať aj vlastné
5. zabezpečenie systému
6. prepojenie práce s viacerými tabuľkami
7. jednoduchá možnosť prechodu na iný databázový stroj
8. možnosť flexibilnej zmeny grafického rozhrania
9. možnosť automatického triedenia, vyhľadávania

Kapitola 2

Súčasné riešenia a ich nedostatky

V súčasnej dobe je k dispozícii niekoľko projektov zameraných na webové rozhranie a prácu s databázou. Projekty sú často viazané na jeden konkrétny účel a ich nasadenie v rôznych je situáciách komplikované a niekedy až nemožné. Na porovnanie som zobral niektoré open source riešenia, ktoré umožňujú vytváranie rozhrania nad databázou.

2.1 DB Browser

Vývojárovi umožňuje vytvoriť jednoduché rozhranie nad relačnou databázou. Obsahuje funkcie zjednodušujúce manipuláciu s dátami (napr. vyhľadávanie). Informácie o výstupnom formulári získava z databáze. Pri zložitejšej aplikácii je možné ušetriť veľa práce využitím zabudovaných funkcií. V prípade zložitejších relácií je definovanie výstupu náročné. Špecifikácia aplikácie je zrozumiteľná len programátorom.

DB Browser je na rozhraní medzi nástrojom pre obyčajného používateľa a správcu dát.

Zdroj: http://www.summersault.com/software/db_browser

2.2 dbMan

Projekt určený pre správcov dát. Jedná sa konzolový typ aplikácie. Výsledky z požiadaviek na databázu je možné použitím zásuvných modulov rôzne transformovať (napr. HTML). Svojou konštrukciou je vhodný skôr pre používateľov pracujúcich s konzolovými aplikáciami a pre dátových manažérov. Má funkcie na zrýchlenú tvorbu

databázových požiadaviek a umožňuje využívanie filtrov na transformáciu výsledkov. Je dobre navrhnutý, takže je možné ho rozširovať o doplňujúce zásuvné moduly. Nie je závislý na konkrétnom databázovom stroji.

Zdroj: http://dbman.linux.cz

2.3 phpMyAdmin

Projekt je zameraný na prístup k databáze cez web. Automaticky vytvára webové rozhranie vo forme tabuliek a pomocou formulárov umožňuje editovanie hodnôt. Je určený hlavne pre správcu dát, pretože umožňuje prístupy do databázy na najnižšej úrovni. Na tvorbu systému pre bežných používateľov nie je tento projekt príliš vhodný, pretože nie je možná jeho jednoduchá adaptácia. Jedná o monolitický systém na prácu s dátami.

Nástroj pracuje nad jediných databázovým strojom a to MySQL, ktorý bohužiaľ nemá plne implementované všetky databázové funkcie a je vhodný prevažne na malé aplikácie.

K programu phpMyAdmin existuje niekoľko aplikácii, umožňujúcich vkladať dáta z rôznych tabuľkových procesorov.

Zdroj: http://www.phpmyadmin.net

2.4 Plone

Plone je systém na prácu s obsahom — content managemet systém. Jedná sa o aplikáciu postavenú nad aplikačným serverom Zope. Pomocou tohoto nástroja je možné veľmi rýchlo vytvárať používateľské rozhranie. Hlavnou výhodou Plone je veľmi prepracovaný systém tvorby workflow — pracovných tokov. Je možné veľmi presne špecifikovať od akého používateľa sa údaje dostanú inému používateľovi. To umožňuje využiť tento nástroj na tvorbu rozsiahlych systémov.

Jeho hlavnou nevýhodou je jeho veľkosť. Vzhľadom na to, že sa jedná o komplexný softvér, tak zaberá netriviálne množstvo priestoru a nie je ho dosť dobre možné nasadiť tam, kde sú potrebné len malé riešenia. Nie je závislý na konkrétnom databázovom stroji.

Zdroj: http://www.plone.org

2.5 SOFIA

The Salmon Open Framework for Internet Applications je kompletný framework na tvorbu aplikácii. Je kvalitne prepracovaný. Na svoj beh vyžaduje J2EE server (napr. Apache Tomcat). Navrhovanie jednotlivých komponent je možné robiť vizuálne cez grafické rozhranie. Je vhodný na tvorbu veľkých systémov postavených na webe. Nevýhodou je vysoká výpočtová náročnosť serveru s podporou Javy.

Zdroj: http://www.salmonllc.com

2.6 Porovnanie

Údaje z predchádzajúcich projektov som zhrnul do stručnej tabuľky, z ktorej je vidieť rozdiely medzi jednotlivými riešeniami. Ako význačné vlastnosti som vybral: jazyk v ktorom je prevažná časť aplikácie napísaná; cieľovú skupinu používateľov, pre ktorých je aplikácia najvhodnejšia; typ rozhrania poskytované používateľovi; možnosť tvorby rozsiahlych aplikácii (napr. informačných systémov); veľkosť aplikácie.

V poslednom riadku tabuľky s označením Medvedb sú zachytené vlastnosti aplikácie, ktorá je výsledkom mojej práce.

	jazyk	cieľ skupina	rozhr.	apl.	veľkosť
DB Browser	Perl	používateľ	web	áno	< 1MB
dbMan	Perl	správca	konzola	nie	< 1MB
phpMyAdmin	PHP	správca	web	nie	∼1 MB
Plone	Python	používateľ	web	áno	> 10 MB
SOFIA	Java	používateľ	web	áno	> 10 MB
Medvedb	Perl	používateľ	web	áno	< 1MB

Každé z riešení dostupných v súčasnosti má svoje nedostatky, ktoré znemožňujú jeho nasadenie v určitých situáciách. Projekty, ktoré sú určené pre správcu dát (teda sú adekvátne zložité) nie sú vhodné pre bežného používateľa. Zároveň projekty určené priamo pre bežného používateľa sú veľmi rozsiahle a nie sú vhodné pri nasadení v jednoduchých systémoch.

Vyskytla sa potreba mať k dispozícii nástroj, ktorý by bol nezávislý na databázovom stroji, bol by zrozumiteľný aj pre laického používateľa, pracoval by cez webové rozhranie, umožňoval by tvorbu aplikácií a mal by vhodnú veľkosť. Vzhľadom na to, že takýto nástroj nebol k dispozícii rozhodol som sa ho vytvoriť. Mojím cieľom bolo vytvoriť aplikáciu, ktorá by pokryla čo najlepšie miesta, kde sa súčasné riešenia nedajú použiť.

Kapitola 3

Medvedb

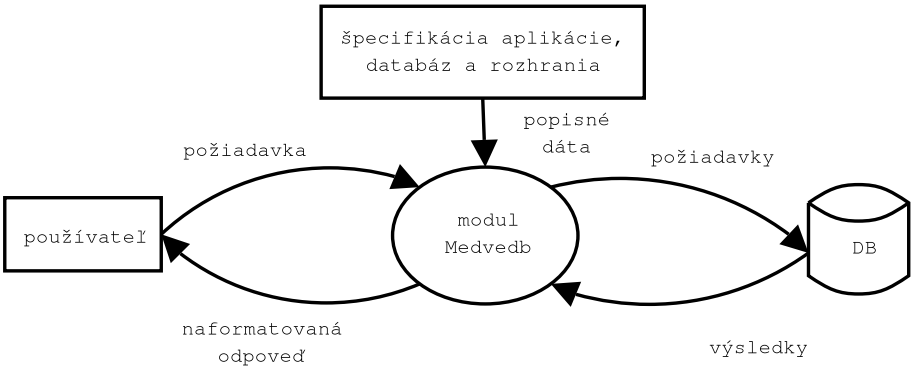
K vzniku mojej práce ma doviedla úloha vytvoriť niekoľko jednoduchých aplikácii. Jednalo sa o vytvorenie rozhrania nad databázou, cez ktoré by bolo možné dáta čítať a modifikovať.

Prvým nápadom ako riešiť problém, bolo to, že každú aplikáciu napíšem samostatne. Toto riešenie som okamžite zavrhol vzhľadom na nedostatok času, ktorý mal som na vytvorenie aplikácií.

Druhý nápad spočíval vo využití nástroja, ktorý už niekto napísal. Po neúspešnom niekoľkodňovom hľadaní som sa rozhodol pre vytvorenie vlastného, ktorý by spĺňal moje požiadavky.

Program bol zo začiatku veľmi neprehľadný. Po určitom vývoji, sa začal javiť ako pomerne rozumný a užitočný. Ako to však býva, keď si niekto napíše svoj program, tak mu väčšinou rozumie len on sám. Zdalo sa, že všetka práca vyjde na zmar. Následne po rozsiahlych úpravách, som dospel k programu, ktorý bol zrozumiteľný pre programátorov. Na napísanie aplikácie využívajúcej môj modul, boli vtedy potrebné pomerne rozsiahle programátorské znalosti.

S touto situáciou som sa neuspokojil. Chcel som navrhnúť program tak, aby pomocou neho mohli tvoriť jednoduché webové aplikácie aj ľudia, ktorí nemajú príliš veľké programátorské skúsenosti. Po mnohých zjednodušujúcich krokoch som dospel k programovému modulu. Ten bol schopný na seba prebrať väčšinu záťaže pri vytváraní rozhrani. Najčastejšie spôsoby tvorby výstupov som zakomponoval do modulu. Tieto algoritmy sa použijú implicitne (pokiaľ používateľ nepovie inak). Takýto prístup umožnil zredukovať záťaž pri tvorbe aplikácie s webovým rozhraním na minimum a pritom ponechával používateľovi otvorenú možnosť tvorby vlastných funkcií.



Obrázok 3.1: konceptuálny model

3.1 Konceptuálny model

Konceptuálny model zachytáva základné princípy fungovania modulu Medvedb.

Medvedb predstavuje medzivrstvu oddeľujúcu používateľa od databázy. Používateľ zasiela svoje požiadavky, tie sú interpretované modulom Medvedb. Modul na základe informácií, ktoré mu poskytol programátor, o štruktúre databáze, vytvorí požiadavku na databázový stroj. Výsledky získané z databáze transformuje na základe popisu používateľského rozhrania na naformátovanú odpoveď a tá je zaslaná používateľovi. Zapojenie modulu do komunikácie medzi používateľom a databázou zachytáva obrázok **??**.

Špecifikácia databáze umožňuje programátorovi povedať, aké dáta má modul od databáze požadovať a akým spôsobom sú tieto dáta prepojené.

Pomocou modulu je možné vytvoriť viac pohľadov (spôsobov zobrazenia dát) nad jednou tabuľkou. Špecifikácia aplikácie umožňuje prepojenie pohľadov do aplikácie. Používateľ má potom k dispozícii tieto pohľady a tým pádom si môže vybrať pohľad, ktorý mu vyhovuje.

Špecifikácia rozhrania zachytáva spôsob, akým sú formátované dáta získané z databáze a od používateľa. To umožní meniť formu výstupu na základe používateľových požiadaviek na rozhranie, formulovaných pri tvorbe rozhrania.

3.2 Logický model

Logický model vystihuje všeobecné princípy, ktoré nie sú implementačne závislé. Logický model predstavuje upresnenie princípov fungovania modulu Medvedb.

Základom modulu Medvedb je výkonné jadro, ktoré riadi toky vstupných a výstupných dáta.

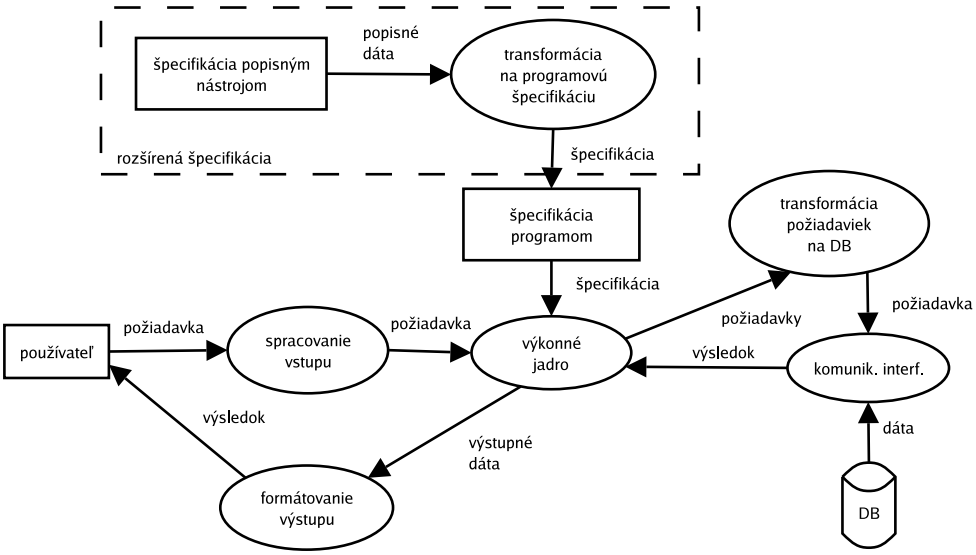
Používateľove požiadavky sú zaslané procesu spracovanie vstupu. Tento proces vyberie relevantné dáta a pretransformuje ich do požiadavky a zašle ju výkonnému jadru. Výkonné jadro na základe obdržaných infromácií načíta príslušnú špecifikáciu aplikácie a podľa nej riadi ďalší chod udalostí.

Základným spôsobom špecifikácie je špecifikácia programom. Jedná sa o kód napísaný v programovacom jazyku. Tento prístup umožňuje programátorovi rýchlu a flexibilnú špecifikáciu vstupov a výstupov. Naviac programátor môže plne využiť silu príslušného jazyka. Tento spôsob práce vyžaduje pomerne dobrú znalosť programovacieho jazyka. Pre bežných používateľov je to ale veľmi nepriehľadné a zavádzajúce. Na odstránenie tohoto nedostatku je pridaná nadstavba umožňujúca alternatívnu špecifikáciu „ľudsky zrozumiteľným" spôsobom.

Rozšírená špecifikácia využíva popisný nástroj, ktorý má dostatočnú vyjadrovaciu silu. Nástrojom môže byť napríklad zjednodušený jazyk. Vzhľadom na to, že sa jedná o zjednodušený popisný nástroj, tak má menšiu vyjadrovaciu schopnosť ako programovací jazyk. Popisné dáta zapísané v príslušnom formáte sú následne transformované priamo na špecifikáciu programom, čo umožňuje využiť ľubovolný vhodný popis. Inými slovami popis je možné napísať pomocou rôznych nástrojov a pri doprogramovaní patričného podprogramu, bude možné tento popis použiť.

Po načítaní príslušnej špecifikácie výkonné jadro sformuluje požiadavku na dáta, ktoré je nutné získať od databázy. Požiadavka je položené cez komunikačné rozhranie databázovému stroju. Výsledok z databázy je zaslaný späť výkonnému jadru.

Jadro môže vyslať na databázu viac požiadaviek, tie potom spojí a odošle spolu s formátovacími informáciami procesu, ktorý má na starosti formátovanie výstupu. Po ukončení je výsledok odoslaný používateľovi. Tým pádom je jeden životný cyklus uzavretý a po-



Obrázok 3.2: logický model

užívateľ môže formulovať ďalšie požiadavky.

Logický model je zachytený na obrázku **??**.

3.3 Fyzický model

Fyzický model zachytáva konkrétne nástroje, ktoré boli využité pri implementácii modulu Medvedb.

Výkonné jadro, spracovanie vstupov a formátovanie výstupov je napísané v Perle. K voľbe tohoto jazyka ma doviedla jeho schopnosť pracovať s regulárnymi výrazmi, platformová nezávislosť a veľké množstvo už existujúcich modulov. Perl je jazyk široko využívaný pri tvorbe webových aplikácii [**?**]. Pre webové servery (napr. Apache) existujú zásuvné moduly, ktoré umožňujú rýchle spúšťanie skriptov, čo je v prípade väčšieho nasadenia dôležitá vlastnosť. Inými kandidátmi na programovací jazyk boli PHP a Python. PHP bolo vylúčené vzhľadom na pomerne nekonzistentný spôsob vývoja. Ďalej PHP nemalo implementovanú jednotnú komunikáciu s databázovým strojom na dostatočnej úrovni. Python bol zhruba na rovnakej vyjadrovacej úrovni ako Perl. Niekoľko objektívnych dôvodov viedlo k voľbe Perlu.

Špecifikácia vstupov a výstupov je uložená v štruktúre hash. Hash sa ukázal počas vývoja ako veľmi vhodný na ukladanie stromov a rýchle vyhľadávanie.

Na ukladanie rozšírenej špecifikácie som zvolil XML strom. Pri voľbe vhodného XML stromu je možné zachytiť takmer všetky informácie, ktoré je možné zachytiť do hashu. Naviac pri zvolení vhodného pomenovania značiek je XML dokument zrozumiteľný aj menej skúsený používateľ ho dokáže prečítať. Ďalším dôvodom pre voľbu XML bola existencia nástrojov schopných overovať štruktúru dokumentu. Štruktúra XML dokumentu nebola zvolená náhodne, inšpiroval som sa štruktúrou, ktorú používa nástroj QT Designer na tvorbu rozhraní pre aplikácií [**?**].

Na interpretáciu XML stromu sa využíva XML::Twig parser [**?**]. Je optimalizovaný na spracovanie vetiev stromu a dokáže pracovať aj nad samotnými úsekmi XML kódu. Jedná sa o nadstavbu nad XML parsera, ktorý je štandardným parserom pre jazyk Perl.

Vzhľadom na to, že modul komunikuje s používateľom cez webové rozhranie, tak na spracovanie vstupných a výstupných formulárov je použitý modul CGI. Jedná sa o štandardné riešenie používané pri tvorbe CGI skriptov. Modul Medvedb pracuje tak, že programátor nie je nútený písať HTML kód. Abstrakcia od HTML kódu je riešená interne v module, ten si sám vyhodnocuje a vytvára HTML kód na základe popisov. Ako pomocný nástroj pri formátovaní využíva modul CGI.

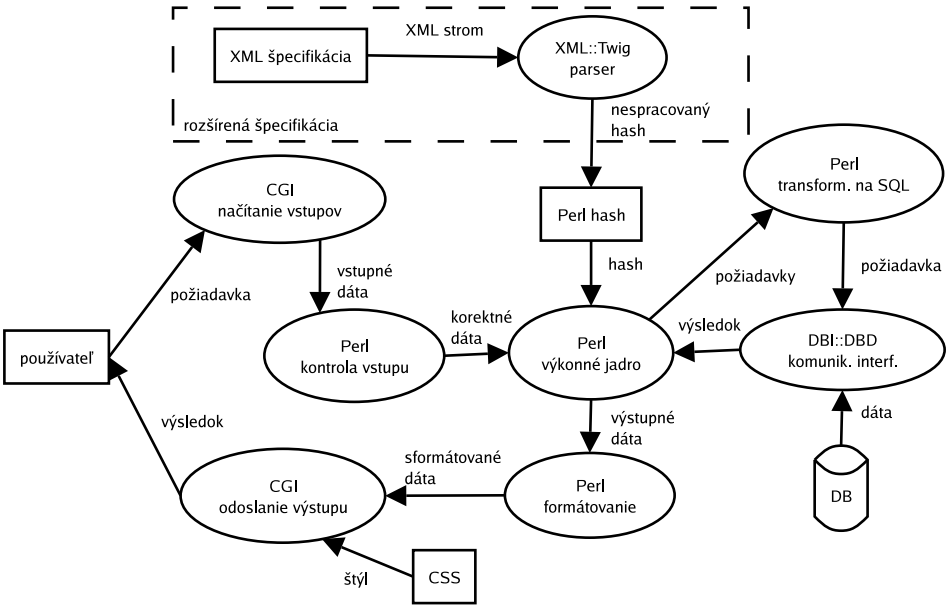
Komunikačný interface s databázou je riešený pomocou modulu DBI::DBD, ktorý umožňuje vytvárať aplikáciu nezávislú na koncovom databázovom stroji [**?**].

Všetky moduly, ktoré som v práci použil patria k štandardný riešeniam postavených na jazyku Perl [**?**].

Fyzický model je zachytený na obrázku **??**.

3.4 Nástroje na popis používateľského rozhrania

Na prácu s popisom používateľského rozhrania je zameraná väčšia časť modulu Medvedb. Jednou z hlavných myšlienok pri vzniku Medvedb bolo vytvorenie nástroja, ktorý by umožňoval jednoduchú tvorbu webových aplikácií. Tento cieľ som sa pokúsil dosiahnuť



Obrázok 3.3: fyzický model

práve cez návrh popisu, akým používateľ (resp. programátor) môže modulu povedať, ako sa má voči dátam v databáze správať.

Keď dáme používateľovi do rúk jednoduchý nástroj, ktorý nebude príliš komplikovaný, používateľ si ho obľúbi. Nie je nič horšie ako dať bežnému používateľovi aplikáciu, ktorá má obrovské množstvo funkcii, avšak je natoľko komplikovaná, že bez prečítania rozsiahleho manuálu, sa nedá ovládať. V prípade, že takýto nástroj vezme do ruky programátor, väčšinou investuje množstvo času do pochopenia princípov práce nástroja. Programátor musí byť schopný na prvý pohľad rozlíšiť, či sa jedná o nástroj vhodný pre jeho prácu alebo nie. Ak nie je možné nástroj jednoducho pochopiť, býva programátor veľmi nepríjemne prekvapený, pretože investoval svoj čas do niečoho, čo nemalo zmysel. Tieto základné fakty som sa snažil mať na pamäti počas celej tvorby modulu Medvedb.

3.4.1 Dátová štruktúra hash

Hash je dátová štruktúra využívaná prevažne v skriptovacích jazykoch. Hash je v podstate pole, ktorého index tvoria reťazce. Jeden

prvok v poli je jednoznačne identifikovaný textovým reťazcom. Textový reťazec je označovaný ako kľúč. Dáta uložené na príslušnom mieste v poli sú označované ako hodnoty. Mechanizmus správy hashov je riešený na úrovni programovacieho jazyka. Programátor sa nemusí zaoberať alokáciou a dealokáciou pamäťového priestoru, to zaňho rieši mechanizmus správy nevyužitej pamäte — garbage collection management. Okrem priameho prístupu ku konkrétnym dátam adresovaným pomocou kľúča, je možné využiť sekvenčný prístup k poľu. Sekvenčný prístup je vhodný v prípade, že je potrebné previesť globálne operácie nad všetkými dátami.

Veľmi užitočnou vlastnosťou hashov (napr. v Perle) je typová polymorfonosť. Do jednej hash štruktúry je možné vedľa seba uložiť čísla, textové reťazce, príp. binárne záznamy. Všetky tieto údaje sú programátorovi dostupné v jednom poli a nemusí pre každý typ deklarovať vlastné pole. Tejto vlastnosti sa dlhú dobu pripisovala pomalosť skriptovacích jazykov. Pri vykonávaní programu sa vždy musí previesť typová konverzia závislá od kontextu v akej sú dané dáta použité. Pri podrobnej analýze časovej náročnosti prevodu na iný typ sa ukázalo, že rýchlostný rozdiel medzi striktne typovanými a voľne typovanými jazykmi, je minimálny.

Vzhľadom na to, že v hash štruktúre je možné udržiavať ľubovolný typ dát, tak je do nej možné uložiť aj dáta typu hash. Tým pádom je možné niekoľkonásobné vnorenie, čo umožňuje budovanie stromov. Položky v listoch takto vybudovaného stromu je možné jednoznačne popísať cestou k týmto položkám (napr. sériou textových reťazcov, ktoré sa vyskytujú v uzloch). Táto vlastnosť robí zo štruktúry hash veľmi silný nástroj.

V high–end nástrojoch určených na tvorbu grafického používateľského rozhrania (GUI), býva každý objekt použitý v aplikácii, popísaný sériou vlastností. Napr. textové pole má vlastnosti ako dĺžka, výška a pod. Pri využití vlastnosti dedičnosti objektov sa tieto vlastnosti prenášajú na potomkov. Inteligentnejšie nástroje dokážu takéto objekty zobraziť v podobe stromu, ktorého listy sú príslušné inštancie objektov. Pri vhodnej voľbe pomenovania uzlov je možné takýto nástroj uložiť do štruktúry hash.

3.4.2 Organizácia popisu do stromu

Strom predstavuje ideálnu štruktúru na uloženie rôznych dát a vlastností. Jednak pri voľbe správneho stromu je možné dosiahnuť veľmi dobré výsledky v oblasti vyhľadávania. Ďalej ako listy stromu môžu byť použité rôzne dátové štruktúry. Strom s rôznymi dátovými štruktúrami v listoch (heterogénny strom) je presne tou štruktúrou, do ktorej je možné vložiť dáta a pritom získať množstvo užitočných vlastností.

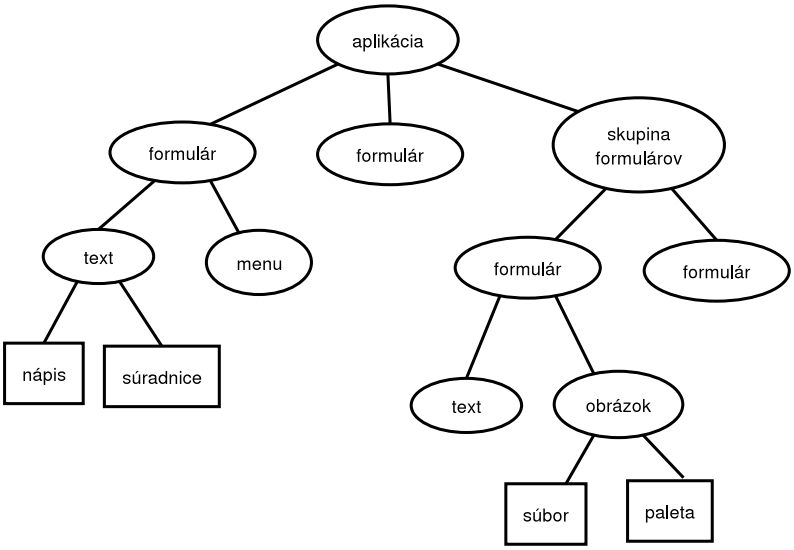
Vlastnosť dedičnosť. V prípade, že nejaký z predkov má určitú vlastnosť, je možné preniesť ju na všetkých potomkov. Zároveň však potomok môže obsahovať inú vlastnosť a dedičnosť tým pádom potlačiť. To umožňuje vytvárať štruktúry, ktoré dokážu pracovať s implicitnou konfiguráciou. Príklad: rodič má implicitne zadefinovanú veľkosť, tú dedia všetcia potomkovia, pokiaľ nie je povedané inak. Na prvý pohľad možno banálna vlastnosť umožní práve zjednodušené vytváranie aplikácie, ako to bolo popísané v predchádzajúcich kapitolách.

Manipulácia s vetvami. Sem môžeme zahrnúť presúvanie kompletných podstromov, ich kopírovanie, prípadne mazanie. Asi najčastejšie využívaná vlastnosť je kopírovanie podstromov. Pri tvorbe aplikácii sa veľmi často využíva možnosť skopírovať nakonfigurovaný objekt do nového. Príkladom je formulár, kde sa nachádza niekoľko textových polí. Ak programátor raz nastavil toto textové pole a všetky ostatné textové polia budú veľmi podobné, môže túto štruktúru jednoducho skopírovať. Pri stromovej interpretácii to znamená iba skopírovanie časti podstromu.

Z uvedených vlastností vyplýva, že strom je vhodnou štruktúrou pre uloženie popisu aplikácie. Ak naviac dátovou štruktúrou na uloženie stromu bude hash, získame ďalšie nezanedbateľné výhody (popísané v predchádzajúcej kapitole).

3.4.3 Definícia pomocou stromu

Hlavný problém pri popise používateľského rozhrania aplikácie vo forme stromu je namapovať realitu na príslušný strom. Vo väčšine prípadov si vystačíme so stromovou štruktúrou. Avšak takýto prístup nie je vždy plne použiteľný a existujú situácie, kedy strom nie je



Obrázok 3.4: strom aplikácie

možné nasadiť vôbec. Jedná sa hlavne o veľmi ťažko algoritmicky riešiteľné problémy. S nimi sa nebudeme zaoberať, pretože sú mimo rámec tejto práce (je však dobré na túto skutočnosť pamätať).

High–end nástroje pre tvorbu GUI využívajú jednu stromovú štruktúru na uloženie informácií o aplikácii (medzi rôznymi nástrojmi sa odlišuje len v malých modifikáciách). Koreň stromu predstavuje aplikácia, jej potomkovia sú formuláre. Ďalšiu úroveň tvoria textové menu, zaškrtávacie položky a pod. Listy takéhoto stromu tvoria vlastnosti objektov. Strom môže byť pomerne hlboký, pretože formuláre prípadne objekty, môžu byť previazané dedičnosťou. Tento typ stromu je zachytený na obrázku **??**.

S využitím štruktúry hash je možné takýto strom pomerne jednoducho zostrojiť, uložiť a udržiavať.

Pri podrobnejšom pohľade zistíme, že je možné pridať ešte jednu úroveň nad koreňovú úroveň aplikácie. Nazvime ju súbor aplikácií. Inými slovami, pri zvýšení úrovne stromu sme schopní do stromu uložiť nie len jednu aplikáciu, ale hneď niekoľko. To môže byť užitočné v prípade, že potrebujeme združiť väčšie množstvo aplikácií. Nie je nutné zostavovať každú z aplikácií samostatne, ale všetky sú pohromade. Taktiež to umožní vytvoriť globálnu konfiguráciu pre

všetky aplikácie. Napr. meno databázového stroja bude špecifikované v koreni stromu a všetky aplikácie a ich potomkovia túto informáciu zdedia. To predstavuje značnú úsporu pri zápise.

3.4.4 Implementácia s využitím štruktúry hash

Programátor uloží špecifikáciu aplikácie do štruktúry hash, ktorú následne predloží modulu na spracovanie. Programátor, pri tvorbe aplikácie využívajúcej Medvedb, si môže vybrať úroveň v akej bude špecifikovať aplikáciu a používateľské rozhranie.

Ako spôsob špecifikácie na najnižšej úrovni je možné použiť špecifikáciu hashom, kde koreňom stromu je konkrétna inštancia aplikácie. Zjednodušene povedané programátor do stromu vloží, všetko čo sa má z databázy načítať a v akej forme sa má výsledok zobraziť. V tomto prípade nedefinuje v strome žiadne
vzťahy s ďalšími aplikáciami. To mu zároveň necháva voľné ruky a vzťahy medzi jednotlivými inštanciami aplikácií si môže definovať sám. Takýto prístup je náročný a vyžaduje si podrobnú znalosť fungovania modulu.

Na vyššej úrovni programátor môže špecifikovať a previazať medzi sebou súvisiace inštancie aplikácie. V jednom strome je schopný spojiť inštanciu prezerania obsahu databáze, s inštanciou tvorba nových položiek a inštanciou skrátený výpis (príp. iné). Modulu sa dá na vstup celá štruktúra a pri spracovaní sa vyberie príslušná vetva zodpovedajúca aktuálne vyvolanej inštancii.

Na najvyššej úrovni je možné špecifikovať a previazať príbuzné aplikácie pracujúce nad rôznymi tabuľkami. Čo umožňuje integrovať viac aplikácií do jednej. Jedná sa o veľmi rýchly a konzistentný prístup tvorby aplikácie, pretože modulu sa predáva kompletný strom so všetkými detailami. Na druhú stranu nie je možné využiť špeciálne funkcie, ktoré by sa dali v programovacom jazyku napísať.

Programátor si musí vybrať, ktorá z úrovní mu vyhovuje a zvážiť jej klady a zápory.

3.4.5 Strom XML

Formát XML má množstvo kladov, pre ktoré je značne obľúbený. Hlavnou výhodou je jeho jednoduchá čitateľnosť, ktorá je užitočná

v prípade, že nie sú k dispozícii aplikácie pracujúce nad XML. Obsah dokumentu XML je možné modifikovať aj jednoduchým textovým editorom. Ďalšou vlastnosťou je platformová nezávislosť, ktorá z neho robí ideálny štandard na výmenu dát. Pre moju prácu však bola najdôležitejšia štruktúra. XML dokument je vytvorený formou stromu.

V predchádzajúcich kapitolách sme si povedali, že strom je vhodná štruktúra na ukladanie informácií. Strom je možné transformovať veľmi jednoducho na XML a späť.

Štruktúry hash a XML je možné namapovať na seba takmer v pomere 1:1. To umožňuje vytvárať rovnocenné typy popisov aplikácii s využitím rôznych popisovacích nástrojov. Problematické sú niektoré dáta, ktoré sa dynamicky vyhodnocujú počas behu programu priamo v jeho tele, tie nie je možné dosť dobre zachytiť do XML.

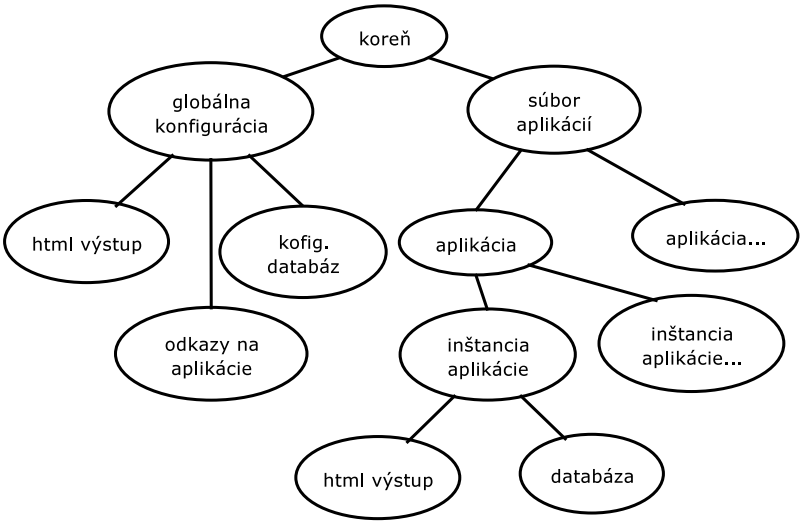
Pri špecifikácii pomocou XML je možné využiť tri úrovne prístupu k tvorbe aplikácie (rovnako ako v prípade implementácie pomocou štruktúry hash). Najzaujímavejšia je najvyššia úroveň. Jedná kompletnú špecifikáciu navzájom previazaných aplikácií. Takýto postup umožňuje zredukovať programátorskú prácu na minimum. Všetka záťaž sa prenáša na vytvorenie XML dokumentu.

Modul Medvedb má tento model dôsledne implementovaný. Pri maximálnom využití popisovacej sily XML dokumentu je na spustenie aplikácie nutný jeden skript obsahujúci jediný príkaz, ktorý povie modulu, aký XML dokument sa má načítať [**?**]. Tento model prístupu k tvorbe aplikácie zbavuje tvorcu aplikácie programátorskej záťaže. Celú aplikáciu je možné napísať do jediného dokumentu, v ktorom sú parametre označkované príslušnými značkami.

XML dokument je navrhnutý tak, aby nedochádzalo k zbytočnej redundancii informácií a zároveň, aby používateľ mal voľné ruky pri návrhu rozhrania aplikácie. Pri pohľade na konceptuálny model zistíme, že takýto systém fungovania presne zodpovedá špecifikácii.

Ukážka XML dokumentu je uvedená v prílohe B. XML dokument bol navrhnutý podľa aktuálnych potrieb. Plánujem spresniť jeho špecifikáciu, tak aby sa čo najviac priblížil všeobecným štandardom na popis rozhraní aplikácie.

Zjednodušená štruktúra XML dokumentu je nakreslená na obrázku .



Obrázok 3.5: zjednodušná štruktúra XML dokumentu

3.4.6 Rozšírenie XML o dynamické prvky

Popis pomocou XML je o niečo slabší než priamy popis pomocou programového kódu. V programovacom jazyku je možné priamo počas behu programu modifikovať premenné, načítavať z nich dáta a riadiť podľa nich beh celého programu. Ak by sme odstránili možnosť využívania premenných na popis rozhrania, zistili by sme, že máme slabý popisný nástroj. Nebolo by možné robiť dopĺňanie hodnôt za premenné a mohli by sme maximálne tak špecifikovať, čo požadujeme na výstupe. Dynamické vlastnosti s využitím mechanizmu premenných, umožňujú vytvárať požiadavky na databázu na základe predchádzajúcich požiadaviek a pod. To sa ukázalo ako veľmi podstatná vlastnosť. XML štruktúru popisu som obohatil o značky označujúce reťazce, ktoré majú byť interpretované v rámci výkonného jadra ako premenné.

Parser načítavajúci XML dokument označí reťazce, ktoré majú byť interpretované ako premenné, a posunie ich výkonnému jadru. Keď jadro narazí na označený reťazec, expanduje ho. Expanzia prebieha tak, že sa zoberie premenná s príslušným menom a jej hodnotou sa nahradí textový reťazec. Medvedb rozlišuje dva typy premenných.

Prvým typom sú premenné, v ktorých je uložený aktuálny výsledok požiadavky na databázu. Tento typ premenných slúži na tvorbu ďalších požiadaviek na databázu. Umožňuje nahradzovať číselné identifikátory textovými menami z inej tabuľky, prípadne spojiť viac výsledkov do jedného.

Druhým typom sú premenné, ktoré boli získané z informácií odoslaných používateľom. Jedná sa o požadovaný typ triedenia, hodnotu URL odkazu (odkiaľ bola stránka vyžiadaná), prípadne identifikátor položky, ktorú si praje používateľ zobraziť.

Príklad dynamickej expanzie:

```
<query>SELECT name FROM category WHERE id =
<expand>book.categoty</expand></query>
```

V príklade je ukážka SQL požiadavku, kde výsledkom bude meno kategórie, ktorá je priradená danej knihe. Vzhľadom na to, že informácia o knihe je dynamická hodnota (používateľ si zakaždým môže zvoliť inú), je označená značkou <expand>. Modul Medvedb dostane požiadavku na získanie informácií o knihe. Jednou z informácií je aj meno kategórie. V tabuľke, kde sa udržiavajú informácie o knihe, je v položke kategória iba číselná hodnota identifikujúca danú kategóriu. Modul rozexpanduje premennú book.category a položí databáze príslušnú SQL požiadavku. (Jedná sa o veľmi jednoduchý príklad, expanzia sa väčšinou využíva v prípadoch, kedy nestačia funkčné vlastnosti databázového stroja).

3.5 Nástroje na popis typov

So samotným popisom rozhrania si nevystačíme. Pomerne často je nutné špecifikovať zložitejšie rozhranie. Časť výsledku požiadavky na databázu má byť zobrazená napríklad ako text a druhá časť ako editovateľná položka. Takáto situácia je riešená tým, že používateľ môže špecifikovať o aký typ položky sa jedná.

Typy môžeme rozdeliť do dvoch kategórii: jednoduché a databázové. Pri jednoduchom type položky je jej hodnota priamo dosadená (napr. do textového poľa). Pri databázovom type sa prevádzajú na základe dostupných hodnôt ďalšie expanzie a požiadavky na databázu. Až na základe ich výsledkov je hodnota dosadená do

príslušnej položky.

Jednoduché typy:

- text — obsah položky je vypísaný ako obyčajný text
- link — výsledok je intepretovaný ako URL ukazujúce mimo aktuálnu aplikáciu (v databáze je napríklad zoznam odkazov na webové stránky, je vhodné, aby používateľ mohol cez tento odkaz priamo pristúpiť na cieľový web)
- href — výsledok je interpretovaný ako URL ukazujúce do aktuálnej aplikácie (udržiavajú sa informácie o aktuálnom stave aplikácie)
- image — výsledok je interpretovaný ako obrázok
- edit — výsledok je editovateľná položka (používa sa pri modifikácií hodnôt v databáze)
- textarea — výsledok je editovateľný blok textu (používa sa pri modifikácií hodnôt v databáze)
- select — výsledok je interpretovaný ako menu, kde si používateľ môže vybrať položku (napr.: v databáze uložené len číselné identifikátory, ktoré je nutné nahradiť zoznamom slov)

Databázové typy:

- textdb — je vykonaná dodatočná požiadavka na databázu a výsledok je interpretovaný ako text
- imagedb — je vykonaná dodatočná požiadavka na databázu a výsledok je interpretovaný ako obrázok (ak sa v databáze nachádzajú mená obrázkov umiestnené na serveri, je možné ich použitím takéhoto typu zobraziť)
- selectdb — na základe výsledku dodatočnej požiadavky je vytvorené menu (používa sa v prípade, že identifikátory a mená sú prepojené cez relácie v databáze)

- listdb — dodatočným výsledkom je zoznam hodnôt, ktoré sa majú zobraziť ako textový zoznam (používa sa pri väzbe 1:m, príp. m:n, kedy je nutné vypísať všetky položky)

Každá položka v závislosti na jej type má doplňujúce parametre, ktoré spresňujú jej chovanie.

3.6 Tvorba aplikácie

Modul Medvedb je navrhnutý tak, aby na vytvorenie reálnej aplikácie (web rozhrania nad databázou) bola nutná čo najmenšia námaha. Pri tvorbe aplikácie odporúčam dodržať niekoľko krokov. Ich postupnosť je v praxi odskúšaná a priniesla reálne výsledky.

Postupnosť krokov pri tvorbe aplikácie:

1. Návrh aplikácie. Pri návrhu je dobré držať sa vhodnej metodológie na tvorbu modelu aplikácie. Je nutné vybrať databázový stroj a implementovať dátové štruktúry. Vhodné je načrtnúť si, ako by malo používateľské rozhranie vyzerať a aké hlavné funkcie má spĺňať. V prípade, že máme príbuzné funkcie, tak ich spojiť do jednej skupiny.

2. Vytvorenie skriptov. Tieto skripty je vhodné pomenovať tak, aby niesli meno skupiny funkcii, ktoré majú vykonávať. Do tela skriptu vložiť príkaz na načítanie príslušnej konfigurácie aplikácie z XML súboru. V prípade, že XML nepredstavuje dostatočne silný nástroj na popis aplikácie, je vhodné doplniť skripty o pomocné funkcie.

3. Vytvorenie popisu aplikácie. Podľa návrhu z bodu 1. vytvoríme popis. Jedná sa o vymenovanie toho, čo používateľ vidí. Daným položkám, ktoré sú v popise zachytené priradíme typy a doplňujúce parametre. V prípade, že je nutné previesť zložitú operáciu s výstupnými dátami, prenecháva sa táto operácia databázovému stroju a do popisu sa zachytí príslušný databázová požiadavka.

4. Spustenie a testovanie aplikácie. Je možné, že nie všetky špecifikácie sú popísané správne. Preto je vhodné pred nasadením

otestovať funkčnosť pomocou ladiacich modifikátorov (tie sa vkladajú do popisu).

Presný postup implementácie aplikácie je závislý na funkciách implementovaných v module Medvedb. Pre získanie informácii o implementačných detailoch odporúčam navštíviť stránku projektu Medvedb.

Na ilustráciu uvediem príklad tvorby aplikácie pre verziu modulu 2.0.

1. Vytvoriť databázu, štruktúru tabuliek.
2. Vytvoriť CGI skripty obsahujúce meno XML dokumentu a meno aplikácie, ktorá sa má vykonávať.
3. Napísať XML dokument popisujúci aplikáciu.

V prílohe A je príklad jednoduchej knižnice. Vytvorenie samotnej aplikácie s popisom v XML bola otázka niekoľkých málo hodín.

3.7 Informácie o module

Modul Medvedb je zverejnený pod licenciou GNU/GPL na stránkach http://medvedb.sourceforge.net. Na stránkach je možné nájsť podrobnejšie informácie o implementácii. Rovnako je dostupný aj príklad jednoduchej knižnice, na ktorom je možné vidieť funkčné vlastnosti modulu.

Kapitola 4

Nasadenie v praxi

Modul Medvedb je vyvíjaný už dlhšiu dobu. Nevznikal náhodne, jeho vývoj bol usmerňovaný požiadavkami používateľov využívajúcich aplikácie postavené na tomto module. Vzniklo niekoľko rôznych aplikácií, s ktorých uvediem aspoň tie najzaujímavejšie.

4.1 Databáza počítačov FSS

Projekt Medvedb začal vznikať vďaka projektu databázy počítačov na Fakulte Sociálnich studií MU v Brne. V tej dobe som pracoval ešte na niekoľkých projektoch, ktorých prevažná časť spočívala vo vytvorení webového rozhrania . Povedal som si, že by nebol zlý nápad opakujúce sa časti skriptov zjednotiť do jedného modulu.

Bolo nutné vytvoriť rozhranie na evidenciu údajov o počítačoch (miestnosť, zásuvka, OS, atď). Tu vznikla veľká časť jadra Medvedb. Na základe požiadaviek na funckcionalitu som postupne pridával nové funkcie.

Projekt sa ujal a okrem evidencie počítačov, je evidovaný aj softvér inštalovaný je na počítačoch.

Ukážka je v prílohe C.

4.2 Správa článkov časopisu Malá Fronta

Ďalšie využitie našiel projekt Medvedb pri tvorbe jednoduchého redakčného systému na zadávanie článkov do študentského časopisu Malá Fronta. Rozhranie umožňuje redaktorovi kontrolovať články, upravovať ich, prípadne ich posunúť korektorke na opravu.

Šéfredaktorovi umožňuje systém vytvoriť kompletné číslo časopisu a publikovať ho.

4.3 Správa portálu Game for Linux

Všetky údaje webového portálu Games for Linux sú uložené v databáze. Dlhú dobu nebol k dispozícii žiadny systém na správu portálu a tak na úpravu databáz bola používaná mysql konzola. S rastúcim množstvom dát a s potrebou umiestňovania súborov cez webový prehliadač vyvstala požiadavka po správcovskom systéme.

Pri vytváraní systému bola prevzatá pôvodná štruktúra databáze a bolo sformulovaných niekoľko doplňujúcich požiadaviek. Aplikácia bola behom jedného dňa vytvorená a nasadená.

Ukážka v je prílohe C.

Záver

Predložená práca priniesla riešenie umožňujúce vytvárať webové rozhranie nad databázou podľa predstáv používateľa. Riešenie je postavené tak, aby aplikáciu mohol vytvoriť aj človek s minimom programátorských znalostí.

Modul Medvedb, ktorý je výsledkom tejto práce, umožňuje špecifikovať aplikáciu pomocou jazyka Perl, prípadne pomocou XML. Je nenáročný na systémové prostriedky a je vhodný na tvorbu malých webových aplikácií.

Momentálnym nedostatkom je štruktúra XML dokumentu, ktorá bola zvolená podľa aktuálnych potrieb a striktne nesleduje svetový trend, ktorý bol nasadený pri špecifikovaní webových aplikácií. Tento nedostatok plánujem v budúcnosti odstrániť úpravou parseru. Druhou nevýhodou pri použití modulu Medvedb, je absencia nástroja umožňujúceho jednoduché vytváranie XML pre potreby. Momentálna situácia je taká, že XML dokument si musí používateľ napísať sám s použitím textového editoru. Vhodným riešením by bol vznik aplikácie schopnej vytvárať tento popis. Problém však nie je veľmi závažný, pretože momentálna štruktúra XML je čitateľná a zrozumiteľná aj laikovi.

Dlhodobý vývoj ukázal, že Medvedb je kvalitne navrhnutý. Rozširovanie o nové vlastnosti nepredstavovalo vážnejší problém a bolo len otázkou správneho špecifikovania požiadaviek. Do budúcnosti sa počíta aj s ďalšími rozšíreniami. Napr.: Práca s autentifikovanými používateľmi a vytváranie výstupných správ.

Medvedb predstavuje open source riešenie, ktoré sa bude v budúcnosti naďalej vyvíjať. Je možné ním rozumne nahradiť frameworkové riešenia, ktoré sú často príliš veľké a hardvérovo náročné. Medvedb je dostatočným riešením web aplikácií pre menšie projekty.

Literatúra

[1] Luboslav Lacko: Web a databáze. Computer Press, Praha, 2001.

[2] Simon St. Laurent: Tvorba internetových aplikácií v XML. Computer Press, Brno, 1999

[3] Sihir Gundavaram: CGI programování webových stránek a aplikací. Computer Press, Praha, 1998

[4] QT library — Trolltech: http://www.trolltech.com

[5] Perl: http://www.perl.org

[6] XML::Twig: http://www.xmltwig.com

[7] Database interface module for Perl: http://dbi.perl.org

[8] Comprehensive Perl Archive Network: http://www.cpan.org