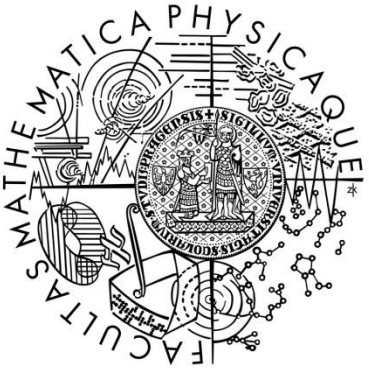
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Radovan Jankovič

Webmin – generátor GUI pre správu relačních databáz

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Pavel Ježek, Ph.D.

Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2014

Ďakujem vedúcemu mojej práce, pánovi Mgr. Pavlovi Ježkovi, Ph.D., za mnoho cenných pripomienok a usmernení a za čas, ktorý mi venoval.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 16.5.2014 Radovan Jankovič

Název práce: Webmin – generátor GUI pre správu relačných databáz

Autor: Radovan Jankovič

Katedra / Ústav: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Pavel Ježek, Ph.D, Katedra distribuovaných a spolehlivých systémů

Abstrakt: Tato práce analyzuje možnosti automatizovaného návrhu prostředí pro správu obsahu relačních databází při zachování vysoké míry personalizace dosahované u obdobných prostředí navržených lidmi. Práce kombinuje vlastnosti takovýchto řešení s generickými alternativami.

Na základě této analýzy je v rámci práce provedena implementace funkčního základu odpovídajícího programu. Při tom je kladen důraz na rozšířitelnost řešení a jeho nezávislost na typu operačního či databázového systému.

Jsou také identifikovány některé klíčové podmínky praktické nasaditelnosti tohoto druhu systému a možné směry dalšího rozšiřování aplikace, k čemuž je poskytnut stručný návod.

Klíčová slova: databáze, reverzní datové inženýrství, CMS, editor

Title: Webmin – generator of relational database management tools

Author: Radovan Jankovič

Department: Department of Distributed and Dependable Systems

Supervisor: Mgr. Pavel Ježek, Ph.D., Department of Distributed and Dependable Systems

Abstract: This thesis analyses the options of automated composition of a relational database content management environment preserving the high degree of flexibility and personalization found in such environments composed manuály. The thesis combines some of the properties of such systems with their generic counterparts.

Based on this analysis, the thesis includes an implementation of a working prototype of a basic version of such a system. While doing so, the implementation focuses on operation system and database management system type independence and the extensibility of the resulting solution.

Some of the aspects of such a system critical to its practical usability are identied. Directions of future extension of the application are proposed as well. A brief tutorial for such an extension is provided.

Keywords: databases, reverse data engineering, CMS, editor

Názov práce: Webmin – generátor GUI pre správu relačných databáz

Autor: Radovan Jankovič
Katedra / Ústav: Katedra distribuovaných a spolehlivých systémů

Vedúci bakalárskej práce: Mgr. Pavel Ježek, Ph.D, Katedra distribuovaných a spolehlivých systémů

Abstrakt: Táto práca analyzuje možnosti automatizovaného návrhu prostredia pre správu obsahu relačných databáz pri zachovaní vysokej miery personalizácie dosahovanej u obdobných prostredí navrhnutých ľuďmi. Práca kombinuje vlastnosti takýchto riešení s generickými alternatívami.

Na základe tejto analýzy je v rámci práce vykonaná implementácia funkčného základu zodpovedajúceho programu. Pri tom je kladený dôraz na rozšíriteľnosť riešenia a jeho nezávislosť na type operačného či databázového systému.

Sú tiež identifikované niektoré kľúčové podmienky praktickej nasaditeľnosti tohto druhu systému a možné smery ďalšieho rozširovania aplikácie, k čomu je poskytnutý stručný návod.

Kľúčové slová: databázy, reverzné dátové inžinierstvo, CMS, editor

Obsah

Úvod ... 11
Štruktúra práce ... 12
1 Východzí stav a ciele ... 13
1.1 Existujúce riešenia ... 13
1.1.1 CMS ... 13
1.1.2 Prostredia databázových vývojárov ... 13
1.1.3 Generátory kódu ... 14
1.2 Hlavné problémy ... 14
1.2.1 Generátor prostredia ... 14
1.2.2 Manuálne úpravy ... 15
1.2.3 Rozšíriteľnosť ... 15
1.3 Použiteľnosť ako CMS ... 15
1.4 Zhrnutie cieľov ... 16
2 Teoretické minimum ... 17
2.1 Skupiny entít a vzťahy medzi nimi ... 17
2.2 Relačný model ... 18
3 Architektúra a použité technológie ... 20
3.1 Použitie aplikácie ... 20
3.1.1 Užívateľské skupiny ... 20
3.1.2 Časť návrhára ... 21
3.1.3 Prostredie koncového užívateľa ... 21
3.2 Druh aplikácie ... 21
3.2.2 Webové riešenie ... 22
3.2.2 Možnosti desktopovej alternatívy ... 22
3.3 Celková štruktúra ... 23
3.3.1 Skladba užívateľského prostredia ... 23
3.3.2 Proces spracovania stránky ... 25
3.3.3 Prehľad dátovej vrstvy ... 27
3.3.4 Centrálna riadiaca jednotka ... 28
3.4 Reverzné dátové inžinierstvo ... 30
3.4.1 Zdroje metadát ... 30
3.4.2 Upresnenie metadát užívateľom ... 31
3.4.3 Dodatočné zdroje ... 31
3.4.4 Vzťahy medzi entitami ... 32
3.4.5 Komplexnejšie riešenia ... 33
3.4.6 Kompromisné riešenie ... 33
3.5 Základné vrstvy systému ... 34
3.5.1 Oddelenie časti závislej na type RDBMS ... 34
3.5.2 Oddelenie častí návrhára a administrátora ... 35
3.6 Výber programovacieho jazyka a frameworku ... 37

3.6.1 Prístup k dátam ... 37
3.6.2 Multiplatformovosť ... 38
3.6.3 Nepoužitie ASP.NET MVC ... 38
3.7 Databázová vrstva ... 39
3.7.1 Systémová databáza ... 40
3.7.1 Základná databázová vrstva ... 41
3.7.2 Vrstva nad INFORMATION_SCHEMA ... 42
3.7.3 Vrstva nad systémovou databázou ... 43
3.7.4 Vrstva nad produkčnou databázou ... 44
3.8 Objektová vrstva ... 45
3.9 Centrálna riadiaca jednotka a navigácia ... 47
3.10 Návrh prostredia ... 48
3.10.1 Automatizovaný návrh ... 48
3.10.2 Editácia zo strany návrhára ... 50
3.11 Bezpečnosť ... 51
3.11.1 Prístupové práva ... 51
3.11.2 Paralelný prístup ... 51
3.12 Modulárnosť ... 52
4 Vývojová dokumentácia ... 53
4.1 Prehľad procesu ... 54
4.2 Štruktúra databázy ... 55
4.3 Triedy databázovej vrstvy ... 57
4.4 Triedy objektovej vrstvy ... 58
4.5 Prezentačná vrstva a návrhár prostredia ... 59
4.6 Užívateľsky definované polia ... 60
4.6.1 `IColumnField` ... 61
4.6.2 `IColumnFieldFactory` ... 64
4.6.3 `ICustomizableColumnFieldFactory` ... 65
4.7 Rozšírenie o podporu ďalších RDBMS ... 66
4.7.1 `IDbDeployableFactory` ... 67
4.7.2 `IBaseDbLayer` ... 68
4.7.3 Získavanie metadát ... 69
4.7.4 „Registrácia“ novej databázovej vrstvy ... 70
5 Problémy pri implementácii ... 71
5.1 Dynamické vytváranie GUI ... 71
5.2 Ovládací prvok správy vzťahu relácií M:N ... 71
5.3 Podpora Linuxu operačných systémov: ModMono ... 72
5.4 Inštalátor ... 73
6 Záver ... 74
6.1 Zhodnotenie naplnenia cieľov ... 74
6.2 Porovnanie výsledku práce s podobnými produktmi ... 75
6.3 Ďalší vývoj ... 76
A Užívateľská dokumentácia ... 78
A.1 Inštalácia ... 78
A.1.1 IIS + MS SQL ... 79

A.1.2 Linux + MySQL 79
A.2 Správa projektov 80
A.3 Rozhranie návrhára prostredia 80
A.3.1 Detekované problémy 81
A.3.2 Pohyb v návrhu 82
A.3.3 Úprava menu 84
A.3.4 Pridanie a odstránenie tabuliek 85
A.3.5 Úprava prehľadových obrazoviek 85
A.3.6 Úprava editačných obrazoviek 86
A.4 Rozhranie administrátora 89
A.5 Prideľovanie užívateľských práv 90
A.6 Nastavenie užívateľského účtu 91
B Obsah priloženého CD 92
Zdroje 93

Úvod

Slová ako informatika či informačné technológie, ktoré v súčasnosti patria k najskloňovanejším nielen v odborných kruhoch, ale de facto naprieč celým spoločenským spektrom, vyvolávajú nepochybne mnohé asociácie. Nemožno ale zabúdať na základ týchto termínov – pojem informácie. Skutočne, od počiatkov odboru do dnešných dní bolo a je základnou úlohou väčšiny aplikácií spravovať a uchovávať dáta. I keď túto základnú úlohu už rieši veľké množstvo systémov, je stále výzvou. Stačí spomenúť dynamicky sa rozvíjajúcu oblasť dátových skladov a vyťažovania dát, ktorá vyžaduje efektívne spracovanie neustále narastajúcich objemov dát. (Objem dát uložených vo svetových databázach sa zdvojnásobí približne každých 20 mesiacov [DM].)

Samotné uloženie dát však pre väčšinu aplikácií nepostačuje. Pokiaľ je aplikácia určená pre priame použitie koncovým užívateľom alebo správcom, musí poskytovať vhodné užívateľské rozhranie, v rámci ktorého možno dáta vyhľadávať, prehliadať a upravovať a to, pokiaľ možno, intuitívne, bez potreby rozsiahleho štúdia systému. Mnohé moderné aplikácie, zameriavajúce sa na užívateľov netechnikov, vytvárajú nad prístupom k dátam ešte jednu vrstvu, pretvárajúcu tieto dáta do graficky príťažlivej formy s množstvom grafiky a iných efektov. Tento druh aplikácií sa zvykne deliť na užívateľskú časť (frontend), zameranú na prezentáciu dát, a správcovskú časť, ktorej primárnou úlohou je správa dát a požiadavky na vizuálnu stránku sú tu zväčša skromné. Toto je nepochybne prípad mnohých webových stránok, ale aj ďalších dátovo zameraných systémov.

Správcovská časť (backend) má často štandardnú podobu, ktorá sa opakuje vo viacerých aplikáciách vyvíjaných jednou spoločnosťou, pričom hlavné rozdiely sú viazané na databázovú štruktúru, ktorá sa má spravovať. Preto je pri vývoji týchto aplikácií kladený dôraz na znovupoužiteľnosť častí aplikácie, ktoré budú vyžadovať len minimálne modifikácie, prípadne bude systém natoľko všeobecný, že umožní administrovať ľubovoľnú databázu.

Webové aplikácie v súčasnosti tvoria podstatnú časť aplikačného software. Ich podiel na trhu bude pravdepodobne ďalej rásť, vychádzajúc zo súčasného trendu cloudových riešení a presunu už existujúceho software na distribuované úložiská, pri ktorých je webové rozhranie pohodlnou možnosťou prístupu pre správcov systému, ale najmä pre koncových užívateľov. Prílišná všeobecnosť však so sebou často prináša obmedzenie na základné funkcie alebo tvorbu nástrojov, ktoré sú tak tenkou obálkou nad databázovým systémom, že ich užívateľ sa nezaobíde bez znalosti tohto systému.

Cieľom tejto práce je vytvoriť grafický systém správy databázy (Webmin), ktorý kombinuje prístup bez iniciačnej konfigurácie a úpravy zdrojového kódu, ktorý šetrí vývojárom čas, s možnosťou následnej úpravy prostredia podľa špecifických potrieb logiky za databázovým modelom a požiadaviek užívateľa, vďaka čomu môže takýto systém používať aj zákazník, ktorý nemá záujem o znalosti z oblasti databáz.

Štruktúra práce

Prvá kapitola popisuje niektoré z existujúcich riešení, ciele projektu, teda to, čím by sa predkladané riešenie malo od predošlých odlíšiť, a na všeobecnej úrovni zhodnocuje ich realizovateľnosť.

Druhá kapitola poskytuje stručný úvod do problematiky návrhu databázového modelu. Na ňu nadväzuje tretia kapitola zaoberajúca sa analýzou architektonického návrhu riešenia a voľbou použitých technológií. Zachytáva kľúčové body riešenia z hľadiská prínosu pre užívateľa, výkonu aplikácie a jej ďalšej rozšíriteľnosti. Štvrtá a piata kapitola následne popisujú konkrétnu implementáciu v jazyku C#, realizáciu návrhu z predchádzajúcej kapitoly a problémy, ktoré sa pri implementácii vyskytli.

Šiesta, záverečná kapitola zhodnocuje výsledky práce, mieru naplnenia výchdzích cieľov a možnosti ďalšieho vývoja projektu.

1 Východzí stav a ciele

Výber prostredia pre správu dát závisí primárne od potrieb a znalostí užívateľa. Modelovou situáciou, z ktorej táto práca vychádza, je IT firma ponúkajúca tvorbu informačných systémov (IS) na kľúč. Táto firma od zákazníka – podnikateľského subjektu – dostane popis reality, ktorú má jej systém reprezentovať. Obsah IS chce zákazník spravovať sám, preto musí byť prostredie dostatočne intuitívne a jednoduché. Aplikácia ako celok môže a nemusí mať klientskú časť, ktorej vizuálnu podobu a funkcie môže zákazník bližšie špecifikovať. Klientská časť má byť zrejme pútavá a má odlíšiť firmu zákazníka od konkurencie, preto by postrádalo zmysel vytvárať pre ňu uniformné generické prostredie.

Administrátorovi, naopak, postačí sada jednoduchých formulárov, prostredníctvom ktorých môže spravovať obsah informačného systému. Vytváranie týchto formulárov a logických väzieb medzi nimi je z pohľadu vývojára rutinná povinnosť pri každom novom projekte. Preto naša IT firma hľadá systém, ktorý by zákazníkovi poskytol potrebné funkcie a bol zároveň použiteľný pri väčšine projektov s minimálnymi úpravami.

1.1 Existujúce riešenia

Dostupné systémy správy dát cez užívateľské prostredie možno rozdeliť do niekoľkých kategórií.

1.1.1 CMS

Systémy CMS, reprezentované napríklad voľne dostupnou aplikáciou WordPress, zväčša obsahujú predpripravené administračné prostredie s jednoduchým a príjemným používaním. Nevýhodou mnohých týchto systémov je pevne daná štruktúra databázy a z toho vyplývajúce limitácie v správcovskej aj klientskej časti aplikácie. Vyspelejšie z týchto systémov síce ponúkajú množstvo pluginov, ktoré túto štruktúru môžu modifikovať a tým poskytnúť podporu pre rôznorodý obsah a logiku jeho spracovania, avšak pre zložitejšie aplikácie, ktoré vyžadujú neštandartné užívateľské rozhranie alebo špecifický, na výkon optimalizovaný databázový model opatrený SQL procedúrami a pod., tieto systémy nemusia stačiť.

1.1.2 Prostredia databázových vývojárov

Systémy ako MS SQL Server Management Studio, ORACLE SQL Developer alebo MySQL Workbench tiež umožňujú spravovať uložené dáta, pre koncového užívateľa však nie sú vhodné. Nielen že sú tieto nástroje na úpravu dát zbytočne komplexné, ale pre užívateľa neznalého databáz sú neprehľadné a riziko neúmyselného poškodenia dát je aj pri striktných nastaveniach prístupových oprávnení vysoké. Pre daný účel sú teda prakticky nepoužiteľné. Dôvodom ich zahrnutia je fakt, že umožňujú efektívne manipulovať s ľubovoľnou databázou s obmedzeniami danými len oprávneniami užívateľa a možnosťami príslušného RDBMS. Takisto spĺňajú podmienku okamžitej pripravenosti – s

dátami možno manipulovať okamžite po pripojení k serveru. Sú teda akousi krízovou možnosťou pre komplexné aplikácie v prípade zlyhania redakčného systému.

1.1.3 Generátory kódu

Existuje množstvo jednoduchých nástrojov, ktoré sú schopné pre danú databázovú tabuľku vygenerovať segment kódu v príslušnom programovacom jazyku, ktorý vytvorí webovú stránku s panelom, v ktorom možno vykonávať CRUD (create, read, update, delete) operácie voči tejto tabuľke, tie ale nie sú komplexným riešením danej úlohy. Pokročilejší príklad tohto druhu riešenia poskytuje systém ASP.NET Dynamic Data, ktorý na základe databázovej štruktúry vytvorí formuláre prepojené navigáciou, v ktorých sa dá táto databáza ľahko spravovať. Generovanie prostredia je rýchle a toto prostredie môže vývojár následne upraviť. Dynamic Data navyše dokážu pracovať s každou databázou, pre ktorú existuje implementácia základnej vrstvy technológie ADO.NET.

Zmenu v takto vygenerovanom kóde však môže vykonať jedine programátor. Keďže ide o zmenu kódu, musí sa tento kód po úprave znovu preložiť, čo znemožňuje vykonať takúto úpravu z webového prehliadača, i keď by táto možnosť bola pre menšie úpravy žiaduca.

1.2 Hlavné problémy

1.2.1 Generátor prostredia

Prvoradým cieľom projektu je urýchliť prácu vývojára pri implementácii administračného prostredia. Systém musí vygenerovať prvotný návrh prostredia na základe skúmania databázy a to, pokiaľ možno, hneď v podobe vhodnej pre užívateľa, v ktorej bude musieť programátor vykonať len menšie úpravy, ako je nastavenie popiskov vo formulároch a v navigácii alebo pridanie validačných pravidiel polí. Generovaný návrh by mal okamžite umožňovať prehliadať dáta z tabuliek, pohybovať sa medzi nimi a upravovať dáta v nich a to v grafickom prostredí.

Po vygenerovaní návrhu by sa používanie systému malo charakterovo blížiť štandardným CMS, kde užívateľ nepracuje s objektmi relačnej vrstvy databázy, ale s logickými entitami reprezentujúcimi relevantné objekty reálneho života. Preto je potrebné previesť relačný model na konceptuálny, a to za použitia metód reverzného dátového inžinierstva (RDE) [Kz]. Tento prevod ale nie je jednoznačne určený – napríklad cudzí kľúč môže byť interpretovaný ako vzťah 1:N medzi entitami alebo ako vzťah 1:1, čo zas môže predstavovať jednoznačné priradenie medzi dvoma rôznymi objektmi alebo realizáciu ISA hierarchie. Z relačného modelu tiež nie je možné jednoznačne poznať, pre ktoré tabuľky sa majú generovať editačné formuláre a ktoré, naopak, predstavujú pomocné dátové úložisko, ktoré je plne v správe aplikačnej logiky a s entitami modelovanej reality nemá priamy súvis. Ako príklad možno uviesť logovacie a archivačné tabuľky.

Z tohto dôvodu bude potrebné časť rozhodnutí pri generovaní iniciačného návrhu ponechať na užívateľa. Nič menej, návrh musí zostať automatizovaný do tej miery, aby tvorbu administračného rozhrania oproti klasickému postupu, pri ktorom dizajnér explicitne určí podobu všetkých častí

rozhrania, stále citeľne urýchlil. Preto je vhodné pred vygenerovaním prvej verzii prostredia vyžadovať od užívateľa čo najmenej informácií a následné doladenie vybraných častí návrhu ponechať na ručné zásahy, kde už návrhár opravuje len to, čo systém odhadol chybne. Konkrétne pôjde o odstránenie alebo pridanie vybraných entít alebo formulárových polí či nastavenie dodatočných validačných pravidiel. Návrh ale nesmie kolidovať s podkladovou relačnou databázou a jej integritnými obmedzeniami.

1.2.2 Manuálne úpravy

Úpravu prostredia bude návrhár vykonávať v grafickom prostredí. Toto prostredie by malo byť podobné tomu, s ktorým bude pracovať koncový užívateľ, aby mohol okamžite posúdiť vhodnosť zvolenej formy. Základné úpravy prostredia by mali byť jednoduché a rýchle, nesmú vyžadovať zásah do kódu alebo konfigurácie aplikácie. Po úprave sa nová verzia prostredia automaticky rozdistribuuje k užívateľom. Tu nutno, samozrejme, predísť konfliktným zmenám. Návrhár prostredia nemusí byť programátor a nemusí poznať detaily implementácie systému. Stačí mu znalosť štruktúry databázy a logiky za ňou. Do konfigurácie by mal zasahovať len výnimočne. Všetky úpravy vykonané návrhárom musia byť vratné. Naviac platí podobná podmienka ako pre automaticky generovaný návrh a síce, že žiadna z týchto úprav nesmie viesť na validačný model, ktorý by odporoval štruktúre tabuliek, typom stĺpcov alebo integritným obmedzeniam daným podkladovou databázou.

1.2.3 Rozšíriteľnosť

Riešenie by tiež malo byť dobre rozšíriteľné – minimálne o podporu pre ďalšie typy RDBMS, prípadne o iné užívateľské prvky. Tieto rozšírenia by mali byť schopní vytvárať užívatelia bez hĺbkového štúdia systému na základe poskytnutých manuálov a strednej znalosti príslušného programovacieho jazyka. Predpokladá sa, že prvá verzia systému bude obsahovať len základnú sadu formulárových prvkov s jednoduchým jednotným designom. Základné úpravy by mali spočívať v úprave grafickej podoby editačného poľa a jeho validačných pravidiel.

Aplikácia by mala byť schopná komunikovať s rôznymi databázovými systémami a malo by ju byť možné prevádzkovať tak v prostredí Windows, ako aj na Linuxe. Medzi podporovanými databázovými systémami by malo figurovať okrem iných MySQL. Dôvod je čisto pragmatický – vyvinutý software pravdepodobne nebude dosahovať kvalít požadovaných veľkými komerčnými informačnými systémami. Preto je vhodné zamerať sa na menšie vývojové tímy, medzi ktorými sa MySQL, súc najpoužívanejším open source databázovým systémom, teší veľkej obľube.

1.3 Použiteľnosť ako CMS

Po vygenerovaní návrhu a jeho doladení návrhárom by mal sa mal systém koncovému užívateľovi javiť ako štandardný CMS. Extenzívny zoznam nárokov kladených na súčasné CMS možno nájsť v zozname portálu CMS Review [Cr]. Z týchto možností bolo, vzhľadom na povahu projektu, zvolené minimum nutné k praktickej použiteľnosti softvéru. Z pohľadu koncového užívateľa musí systém umožňovať nasledovné.

- Správu štruktúry webstránok, produktov a kategórií
- Pohodlnú editáciu textových dát prostredníctvom WYSIWYG editoru s bohatou funkcionalitou.
- Správu multimédií – nevyhnutná je podpora nahrávania a prehliadania obrázkov, vhodná je aj podpora pre video, prípadne pre nahrávanie ľubovoľného typu súborov, pokiaľ možno s podporou prehliadania obsahu bez nutnosti sťahovať dokument.
- Vyhľadávanie v obsahu.
- Správu užívateľských oprávnení.
- Personalizáciu – nastavenie CMS na mieru konkrétneho produktu.

Nutným predpokladom k splneniu prvých štyroch bodov je poskytnutie dostatočne širokej palety formulárových a navigačných prvkov návrhárovi, ktorého zodpovednosťou je ich následné zostavenie do vhodnej výslednej štruktúry. Z toho potom vyplynie splnenie poslednej požiadavky, keďže návrhár je pri vytváraní personalizovaného prostredia limitovaný len sadou implementovaných ovládacích prvkov a konzistenciou s podkladovým databázovým modelom.

Správa užívateľských oprávnení je potrebná tak na úrovni administrácie ako aj návrhu prostredia, teda je potrebné naviac spravovať užívateľov majúcich možnosť meniť užívateľské prostredie. Súčasťou zabezpečenia systému bude zamedzenie konfliktným úpravám vykonaným viacerými užívateľmi, či už ide o administrátorov obsahu alebo návrhárov prostredia.

1.4 Zhrnutie cieľov

Zosumarizujme požiadavky na aplikáciu plynúce z predchádzajúceho.

1) Schopnosť odvodiť konceptuálny model z relačného použitím RDE.
2) Schopnosť vytvoriť formuláre a navigáciu na základe výsledku 1).
3) Možnosť vygenerované formuláre modifikovať za behu aplikácie z GUI.
4) Použitie výsledného návrhu na správu reálnych dát ako CMS.
5) Rozšíriteľnosť o pokročilé editačné prvky a podporu ďalších RDBMS.
6) Podpora MS Windows aj Linuxu

2 Teoretické minimum

Vývoj software predstavuje postupnú konkretizáciu užívateľských požiadaviek a ich prevedenie do fyzickej implementácie. Ak je v riešení použitá databáza, prechádza táto obdobnými vývojovými fázami ako celé dielo. V rámci tejto práce sa obmedzíme na relačné databázy, ktoré majú v praxi stále dominantné postavenie, hoci pre mnohé požiadavky súčasných trendov existujú efektívnejšie riešenia (časť vymožeností týchto nových systémov klasické databázové systémy postupne integrujú a vznikajú tak objektovo-relačné databázové systémy). Vývoj relačnej databázy možno rozdeliť do piatich štádií [DS]:

- Špecifikácia užívateľských požiadaviek je založená na konzultáciách so zákazníkom, pri ktorých sa vývojový tím zoznámi s povahou reality, ktorú bude databáza reprezentovať, a bežnými požiadavkami, ktoré na ňu budú kladené.
- Pri koncepčnom návrhu sa definujú entity zastupujúce objekty danej reality, ktoré budú v databáze uložené. Určia sa atribúty týchto entít a vzťahy medzi nimi. Tento model je vysokoúrovňový a zrozumiteľný aj pre užívateľa netechnika, vďaka čomu možno dosiahnuť súlad medzi predstavou zákazníka a dodávateľa o dátovom modele.
- Pri logickom (relačnom) návrhu sa koncepčný podklad namodeluje prostriedkami konkrétneho relačného databázového systému, čím vznikne relačná schéma. Určia sa integritné obmedzenia, čiže podmienky, ktoré musia platiť o reláciách modelu a vzťahoch medzi nimi, aby model zostal konzistentný.
- Fyzický návrh je zameraný na zvýšenie výkonu logického modelu. V tomto kroku sa určí spôsob fyzického uloženia dát a pomocné štruktúry, ako sú indexy a materializované pohľady, ktoré bude databáza využívať pri plnení náročných operácií.
- V poslednom kroku sa návrhár databázy zaoberá okolnosťami mimo databázy samotnej. Určí sa vzťah databázových entít k funkčným celkom vyvíjanej aplikácie, k jej užívateľom a vonkajšiemu svetu. Dôraz je pritom kladený na zabezpečenie systému.

2.1 Skupiny entít a vzťahy medzi nimi

Databáza typicky slúži k uloženiu veľkého množstva entít, ktoré ale možno rozdeliť na niekoľko skupín s rovnakými vlastnosťami a vzťahmi k ostatným skupinám. Tým vznikajú skupiny entít [DS]. Napríklad databáza leteckej spoločnosti môže obsahovať stovky entít lietadiel, letísk a milióny entít zákazníkov, ale k zákazníkom sa pristupuje jednotne – o každom sa uchováva rovnaká množina informácií a, až na výnimky, každý zákazník si môže kúpiť lístok na ktorýkoľvek let. Skupina entít je vo fáze konceptuálneho návrhu charakterizovaná svojimi atribútmi a vzťahmi k ostatným entitným skupinám [DS]. Vzťahom medzi entitami môže byť napríklad fakt, že pasažier P nastúpil na palubu letu L alebo to, že let L štartuje z dráhy D na letisku A. Z pohľadu modelu entitných skupín je potom dôležité určiť kardinalitu týchto vzťahov, t.j. koľko entít z entitnej skupiny A môže figurovať vo vzťahu k jednej entite zo skupiny B a naopak.

Vzťahy medzi entitnými skupinami môžu byť troch druhov:

- **Vzťah 1:1**, kde entita z A je priradená práve jednej entite z B a naopak. V našom leteckom príklade je takýmto vzťahom priradenie miesta v lietadle pasažierovi – pasažier môže sedieť len na jednom mieste a miesto môže byť naopak obsadené len jedným pasažierom.
- **Vzťah 1:N**, kde jedna entita z B môže byť priradená viacerým entitám z A, ale každej entite A je priradená len jedna entita z B Príkladom je vzťah „štartuje z“ medzi letom a letiskom. Let štartuje práve z jedného letiska, ale z tohto letiska môže štartovať mnoho iných letov.
- **Vzťah M:N** vyjadruje obecné mapovanie medzi entitami skupín A a B. Každý prvok z A môže byť vo vzťahu k ľubovoľnému počtu entít z B. Tak môže každé lietadlo (v rámci viacerých letov) pristávať vo viacerých letiskách a na každom letisku pristáva viac lietadiel.

Aby sa spresnila informácia daná kardinalitou vzťahu, zvykne sa uvádzať dolná a horná hranica na počet entít v danom vzťahu. Napríklad pre vzťah medzi A a B zápis 0..1:1..N vyjadruje, každej entite A je priradená naviac jedna entita z B, ale entity A môžu existovať aj bez vzťahu k žiadnej entite B. Na druhej strane, každá entita B je priradená aspoň jednej entite z A.

Entity môžu byť tiež v istom vzťahu k iným entitám tej istej skupiny. Môže to byť napríklad vzťah medzi zamestnancom leteckej spoločnosti a jeho priamym nadriadeným, ktorý je typu 0..N:0..1 (každý zamestnanec má aspoň 0 podriadených a špeciálne generálny riaditeľ nemá nadriadeného). Vzťah medzi entitami náležiacimi jednej skupine je častým spôsobom modelovania hierarchických dát.

2.2 Relačný model

Pri vytváraní relačného modelu sa entity z konceptuálneho návrhu prevedú na tabuľky konkrétneho databázového systému. Stĺpce tabuliek obsahujú atribúty entít, môžu sa tiež pridať ďalšie pomocné stĺpce. Každý záznam (riadok) v tabuľke by mal byť unikátne identifikovaný hodnotami svojich stĺpcov (aj keď viaceré súčasné databázové systémy podporujú duplicitné riadky) a preto možno o týchto záznamoch hovoriť ako o prvkoch relácie v priestore danom prípustnými rozsahmi hodnôt jednotlivých stĺpcov ako dimenziami a o celej tabuľke ako o relácii. Logika mnohých databázových operácií vychádza práve z tohto relačného náhľadu.

Podmnožina stĺpcov tabuľky taká, že každý záznam tabuľky je unikátne identifikovaný svojimi hodnotami v týchto stĺpcoch, sa nazýva nadkľúč tabuľky. Ak naviac platí, že žiadna vlastná podmnožina stĺpcov tohto nadkľúča nie je nadkľúčom, ide o kľúč [DS].

Kľúč tabuľky je vhodným prostriedkom uloženia vzťahu medzi entitami. Pri vzťahu 0..N:1 medzi reláciami A a B sa do tabuľky B začlenia dodatočné stĺpce, ktoré budú obsahovať kľúč príslušného záznamu v A. (Z pohľadu fyzického návrhu je vhodné, aby tento kľúč mal čo najmenšie nároky na pamäť a preto sa pre tabuľky vyberá jeden kľúč ako primárny identifikátor.) Na zaistenie konzistencie

tohto vzťahu možno na úrovni databázového systému určiť integritné obmedzenie, podľa ktorého zvolené stĺpce relácie B musia pre každý záznam v B obsahovať hodnoty príslušných stĺpcov kľúča tabuľky A, čím sa identifikuje práve jeden záznam v A. Toto obmedzenie sa nazýva obmedzením referenčnej integrity a stĺpce B odkazujúce sa na kľúč v A tvoria cudzí kľúč B. Špeciálne je možné povoliť nevyplnenie stĺpcov cudzieho kľúča na strane B, čím sa modeluje vzťah 0..N:0..1, kde entita z B nemusí byť nutne priradená žiadnej entite A.

Problém nastáva pri vzťahu typu M:N. Každá entita z A môže byť vo vzťahu k viacerým entitám z B a preto nutne potrebujeme zoznam kľúčov entít vo vzťahu k nej. Obsiahnutie zoznamu kľúčov v stĺpci tabuľky by však bol v rozpore s prvou normálnou formou (ide o štandardnú požiadavku na štruktúru databázových tabuliek, podľa ktorej musia byť dáta uložené v každom stĺpci neštruktúrované). Tento problém sa na relačnej úrovni bežne rieši pridaním pomocnej tabuľky T, ktorá obsahuje dvojice kľúčov entít z A a B. Tým vzniká nová skupina entít, ktorá nefigurovala v konceptuálnom návrhu. Táto skupina nereprezentuje žiadnu nezávislú entitu z modelovanej reality (kľúč T musí nutne obsahovať časti kľúčov A a B). Ide o takzvaný slabý entitný typ.

3 Architektúra a použité technológie

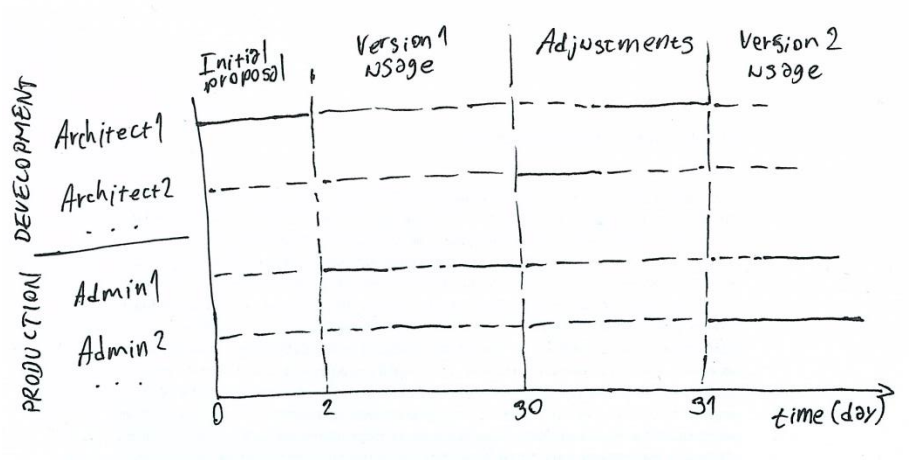
Návrh architektúry systému vychádza z cieľov vytýčených v prvej kapitole.

3.1 Použitie aplikácie

Ciele aplikácie by sa dali rozdeliť do dvoch skupín. Najprv sú to vlastnosti, ktoré systém preberie od riešení spomínaných v časti, 1.1, ktoré má zastúpiť, najmä teda CMS a generátory kódu. Druhú skupinu cieľov predstavujú tie, ktorými sa má toto riešenie od predošlých odlíšiť, predstavujú vlastnosti, vďaka ktorým bude mať nahradenie uvádzaných predchodcov touto aplikáciou skutočne zmysel, ako je napríklad kombinácia prostriedkov RDE a manuálnej úpravy výsledku v grafickom prostredí. Zlúčením týchto požiadaviek získame predstavu o predpokladanom spôsobe používania aplikácie, ktorému musí byť jej návrh prispôsobený.

3.1.1 Užívateľské skupiny

Treba rozlišovať dve základné skupiny užívateľov – návrhárov prostredia, ktorí upravujú výsledok automatizovaného generátora používajúceho RDE a koncových užívateľov, ktorí spravujú produkčné dáta, pričom sa štruktúra ich pracovného prostredia už pochopiteľne nemení.



Obr. 1 Užívateľské skupiny a príklad ich práce v rôznych fázach vývoja administračného prostredia – skupina „DEVELOPMENT“ predstavuje dodávateľa software, zatiaľ čo „PRODUCTION“ zastupuje zákazníka. Prerušovaná čiare predstavuje „existenciu“ zákazníka, spojitá potom jeho aktivitu v rámci daného projektu. Ako možno vidieť, obdobia aktivity jednotlivých skupín sú striktne oddelené.

Tieto dve skupiny užívateľov budú k aplikácii zrejme pristupovať v iných obdobiach životného cyklu aplikácie. Je zrejme nežiaduce, a to z pohľadu oboch zúčastnených strán, aby boli produkčné dáta upravované, zatiaľ čo ešte prebiehajú časté funkčné zmeny prostredia, v ktorom sa majú tieto zmeny odohrávať. Preto v prvej fáze bude s návrhom pracovať výlučne dodávateľ systému. Následne bude

systém odovzdaný do užívania. Aj v tejto, neskoršej, fáze sa ešte môže vyskytnúť potreba zásahu zo strany dodávateľa a aplikácia takýto zásah musí umožniť, nič menej, tieto prípady budú predstavovať výnimočné udalosti. Aplikácia s nimi síce musí počítať, nie sú však primárnym cieľom optimalizácie, či už po stránke výkonu alebo užívateľského pohodlia. Preto bude väčšina návrhu vnímať tieto dve skupiny užívateľov a operácie nimi vykonávané oddelene.

3.1.2 Časť návrhára

Časť aplikácie používaná návrhárom sa ďalej delí na automatizovaný návrh generovaný systémom a následné úpravy vykonávané podľa zváženia návrhára. Pred tieto fázy, z ktorých druhá opäť závisí na výsledkoch prvej, sa na základe 1.2 predradí ešte jedna, v ktorej systém vyžiada od užívateľa tie informácie potrebné k vygenerovaniu návrhu, ktoré nie je možné s dostatočnou istotou odvodiť z fyzickej štruktúry databázy.

Keďže s návrhom prostredia v tejto fáze môže pracovať viacej užívateľov, prípadne jeden užívateľ opakovane v priebehu niekoľkých dní a výsledok tejto práce má byť následne opakovane používaný pri správe dát koncovým užívateľom, musí byť tento model priebežne uchovávaný v permanentnom úložisku. Model musí reprezentovať všetky potrebné aspekty administračného prostredia, navigáciu v rámci neho, štruktúru použitých formulárov a ich prepojenie na objekty spravovanej databázy. Všetky tieto aspekty musí mať návrhár možnosť podľa potreby meniť, ale vždy v súlade s integritnými obmedzeniami danej databázy.

3.1.3 Prostredie koncového užívateľa

Pre koncového užívateľa je aplikácia podstatne jednoduchšia. Tento užívateľ sa pohybuje po administračnom prostredí naplnenom dátami produkčnej databázy, môže prehliadať a upravovať dáta jednotlivých entít. Operácie tohto užívateľa s dlhodobým efektom sa dajú zhrnúť tromi základnými databázovými operáciami – `INSERT`, `UPDATE` a `DELETE`, ktoré však vykonáva sprostredkovane v medziach stanovených stanovených jeho užívateľským prostredím. Do štruktúry prostredia ako takej nezasahuje.

3.2 Druh aplikácie

Rozhodnutím, ktoré výrazne ovplyvňuje možnosti a obmedzenia následného návrhu riešenia, je výber platformy vyvíjanej aplikácie. Presnejšie je nutné rozhodnúť sa medzi vývojom štandardnej desktopovej a webovej aplikácie. Hlavnými zvažovanými aspektmi sú užívateľská prístupnosť (užívatelia sú historicky zvyknutí na natívne desktopové aplikácie, ale značnú časť úkonov už bežne vykonávajú prostredníctvom webových portálov), technologické limity jednotlivých platforiem (tu sa prejavujú niektoré obmedzenia webových portálov, napríklad bezstavovosť protokolu HTTP), zložitosť implementácie (plynie z predchádzajúceho) a možnosti následného rozširovania a opravovania nasadeného software a podpory užívateľov na diaľku (webové aplikácie umožňujú ľahkú centralizáciu, zatiaľ čo desktopové vyžadujú distribúciu aktualizácií k užívateľom).

3.2.2 Webové riešenie

Po zvážení predností a nedostatkov obidvoch platforiem bolo zvolené riešenie formou webovej aplikácie.

Hlavným dôvodom tejto voľby sú už spomínané dobré možnosti centralizovanej správy a zabezpečenie prístupu k rovnakej verzii aplikácie pre všetkých užívateľov. Program má pisárne dva druhy užívateľov. V prvej fáze to je vývojár, ktorý jeho prostredníctvom vytvorí administračné prostredie podľa potrieb zákazníka. Následne mu toto prostredie sprístupní k užívaniu. Dá sa však predpokladať, že sa bude aplikácia ďalej upravovať a rozširovať, čo sa odzrkadlí aj v jej dátovej vrstve a vo výsledku povedie na potrebu úpravy prostredia, ktoré zákazník už medzičasom začal používať a vytvoril jeho prostredníctvom množstvo dát. Pri väčších aplikáciách sa tiež predpokladá viac užívateľov – moderátorov spravujúcich obsah jedného informačného systému. Po dodatočnej úprave tohto prostredia zo strany vývojového tímu by sa jeho nová verzia musela nanovo rozdistribuovať medzi užívateľov. Užívatelia by si museli novú aplikáciu nainštalovať, čo by pri častých zmenách bolo značne nepohodlné najmä pokiaľ by sa intenzívne rozširoval aj obsah niektorých inštancií IS. Alternatívnym riešením by bolo vytvorenie systému automatických aktualizácií. Aplikácia by pravidelne kontrolovala aktuálnosť prostredia poskytovaného užívateľovi voči centrálnej databáze a následne si sťahovala a inštalovala zmeny. To by ale podstatne zvýšilo komplexnosť riešenia, nehovoriac o tom, že takto navrhnutá aplikácia by sa týmto výrazne posunula k princípom webového riešenia, pretože by šlo o model klient-server.

Naviac, lokálne inštancie aplikácie by museli centralizovane komunikovať so spravovanou produkčnou databázou, takže by sa nevyhli prenosom veľkých objemov dát po sieti. Naopak, pri webovom riešení je možné umiestniť produkčnú databázu aj databázu interne používanú generátorom administračného prostredia na jeden server spolu so samotnou aplikáciu, čím sa minimalizujú náklady na sieťový prenos vo fáze spracovania HTTP požiadavku a užívateľovi sa posiela už len finálny kód webovej stránky.

3.2.2 Možnosti desktopovej alternatívy

Na druhej strane, desktopové aplikácie majú za sebou dlhú tradíciu, vývojové prostriedky sú pri nich vyspelejšie a pomerne ustálené. Vytvorenie bohatého užívateľského prostredia je oproti webovým aplikáciám tiež značne jednoduchšie, nakoľko webové aplikácie si musia v každej situácii vystačiť s HTML, CSS a JavaScriptom.

Hlavná prednosť desktopového riešenia ale spočíva v natívnom prostredí operačného systému, ktorý umožňuje omnoho ľahšie uchovávať stav aplikácie. Tá si môže svoj stav uchovávať teoreticky roky. Naproti tomu protokol HTTP je bezstavový. Každá požiadavka od klienta musí prejsť komplexným procesom spracovania nanovo, i keď existujú efektívne mechanizmy (ako napríklad používanie cache výstupu alebo dynamické načítavanie obsahu po častiach prostredníctvom asynchrónnych požiadaviek), ktoré túto záťaž znižujú. Doba odozvy webovej aplikácie je naviac predĺžená prenosom dát po sieti na veľké vzdialenosti.

Nič menej, tieto výhody desktopového riešenia sa časom stávajú menej a menej výraznými, ako v platformách určených pre web pribúdajú nové formy uchovávania stavu aplikácie (cookies, sedenia (session), uchovávanie stavu v databáze a pod.). Nadstavby týchto technológií umožňujú použiť pri vývoji webovej aplikácie postupy zaužívané pri desktopových systémoch a to bez výrazných obmedzení. Konečne, priemerná rýchlosť internetového pripojenia sa vo vyspelých krajinách pohybuje rádovo v jednotkách Mb, čo je pre aplikáciu, kde väčšina zobrazovaného obsahu bude mať textovú podobu a multimediálny obsah bude skôr výnimkou, vcelku dostatočné.

3.3 Celková štruktúra

Architektúra aplikácie musí reflektovať spôsob jej používania tak, ako bol popísaný v časti 3.1. V prvom rade je potrebný vhodný model reprezentujúci administračné prostredie. Toto prostredie sa bude skladať z bežných formulárových prvkov, ktoré ale zväčša nebudú pevnou súčasťou layoutu, ale ich rozloženie a funkcie budú výsledkom automatizovanej analýzy databázy a následných manuálnych úprav. Preto musí objektový model poryť základné časti formulárov, ako sú editačné polia, tlačidlá a rôzne menu.

3.3.1 Skladba užívateľského prostredia

Vychádzajúc z potrieb aplikácie rozobraných v časti 3.1, premietnime ich teraz do komponent systému. Prejdime tieto komponenty odhora nadol a rozdeľme medzi ne potrebné úlohy. Každý z uvedených objektov sa odzrkadlí tak v dátovej vrstve, ako aj na úrovni aplikačnej logiky.

Obálka návrhu: projekty

Dá sa predpokladať, že užívateľ systému (IT firma) bude chcieť tento systém použiť na správu databáz viacerých aplikácií. Je teda žiaduce takéto použitie umožniť a zaviesť základnú entitu, zaobaľujúcu návrh administračného prostredia. Túto entitu budeme nazývať *projekt*. Projekt je určený pre konkrétnu databázu a všetky operácie s návrhom prostredia sa vykonávajú v rámci konkrétneho projektu.

Okrem samotnej sady formulárov by mal projekt uchovávať špecifické užívateľské preferencie a oprávnenia. Vďaka tomu bude možné poskytnúť klientom maximálnu mieru personalizácie systému im dodávaného. Krajným (a nepravdepodobným) prípadom takejto personalizácie je vytvorenie viacerých projektov pre tú istú databázu. Je nič menej, zaujímavé, že koncept projektu takéto použitie umožňuje.

Prehľad entít

Zadefinujme, kvôli prehľadnosti, tieto základné objekty, s ktorými bude aplikácia pracovať.

- Pole je editačný formulárový prvok, do ktorého má užívateľ možnosť zadať určitú hodnotu.
- Ovládací prvok je formulárový prvok slúžiaci k navigácii, k odoslaniu formulára, k uloženiu zmien alebo k potvrdeniu inej užívateľskej akcie. T ento prvok nie je editovateľný.

- Panel pozostáva z polí a ovládacích prvkov a prípadných ďalších panelov. Zvyčajne ide o jednotlivý formulár. Každé pole a ovládací prvok sú viazané k práve jednému panelu.
- Projekt predstavuje súbor všetkých dát týkajúcich sa konkrétneho návrhu prostredia. Obsahuje panely tohto návrhu a ďalšie, pomocné atribúty.

Toto rozdelenie (s vynechaním projektov) zodpovedá štandardnej štruktúre formulárov tak webových, ako aj klasických desktopových aplikácií. Z funkčného hľadiska by sa dali úlohy týchto prvkov rozdeliť nasledovne.

- Pole zodpovedá za validáciu svojho obsahu, vlastné vykreslenie, naplnenie tohto vykreslenia dátami poskytnutými inou komponentou a vrátenie týchto dát po ich úprave.
- Ovládací prvok vyvoláva udalosti (ako prechod na iný panel alebo odoslanie obsahu aktívneho panelu), ktoré spracováva centrálna riadiaca jednotka (viď nižšie).
- Panel zastrešuje polia a ovládacie prvky a je prostredníkom medzi poľami a dátovou vrstvou.
- Projekt oddeľuje jednotlivé spravované databázy. Všetky aspekty návrhu a užívateľov s ním pracujúcich sú, pokiaľ to nie je v priamom protiklade s logikou funkcionality, určované lokálne v rámci projektu alebo na jednej z nižších úrovní.

Koncept panelov, polí a ovládacích prvkov je skutočne jednoduchý. Lepšiu predstavu o notoricky známych prvkoch, ktoré sa skrývajú za uvádzanými definíciami, si čitateľ môže spraviť s pomocou obrázku 2.

Obr. 2 Polia a jeden ovládací prvok na príklade jednoduchého formulára (t.j. panelu)

Uvedené prvky spolu s väzbami medzi nimi tvoria návrh prostredia. Do úvahy síce prichádzajú aj mierne odlišné dátové štruktúry, ale vychádzajúc z toho, že sa prostredie bude skladať z formulárov, textových a iných editačných polí, tlačidiel a odkazov, čo je pre užívateľa intuitívne prostredie používané väčšinou bežných aplikácií, boli by tieto štruktúry tomuto konkrétnemu riešeniu podobné, s obdobným rozdelením funkčných úloh.

Panely

Pozastavme sa pri úlohe panelov. Panel, t.j. formulár s poľami a ovládacími prvkami, bude zväčša priradený jednej konkrétnej tabuľke v spravovanej databáze a jeho polia budú zodpovedať jej editovateľným stĺpcom. Je totiž prirodzené, že užívateľ edituje jednotlivé entity reálneho sveta, ako

objednávky alebo články, ktoré sú uložené v jednotlivých tabuľkách. Je, samozrejme, možné, že k správe dát v jednej tabuľke budú potrebné dáta z iných tabuliek, ktoré sú s danou tabuľkou v istom relačnom vzťahu. Uvážme ešte typický postup užívateľa, ktorý chce zmeniť niektoré hodnoty konkrétnej entity, akou môže byť nezaplatená faktúra alebo záznam v obchodnom registri. Intuitívnym postupom by preňho bolo najprv zvoliť „Faktúry“ v hlavnom menu prostredia, následne v zozname faktúr vybrať tú, ktorú chce editovať a nakoniec zmeny uložiť.

Špecializované panely

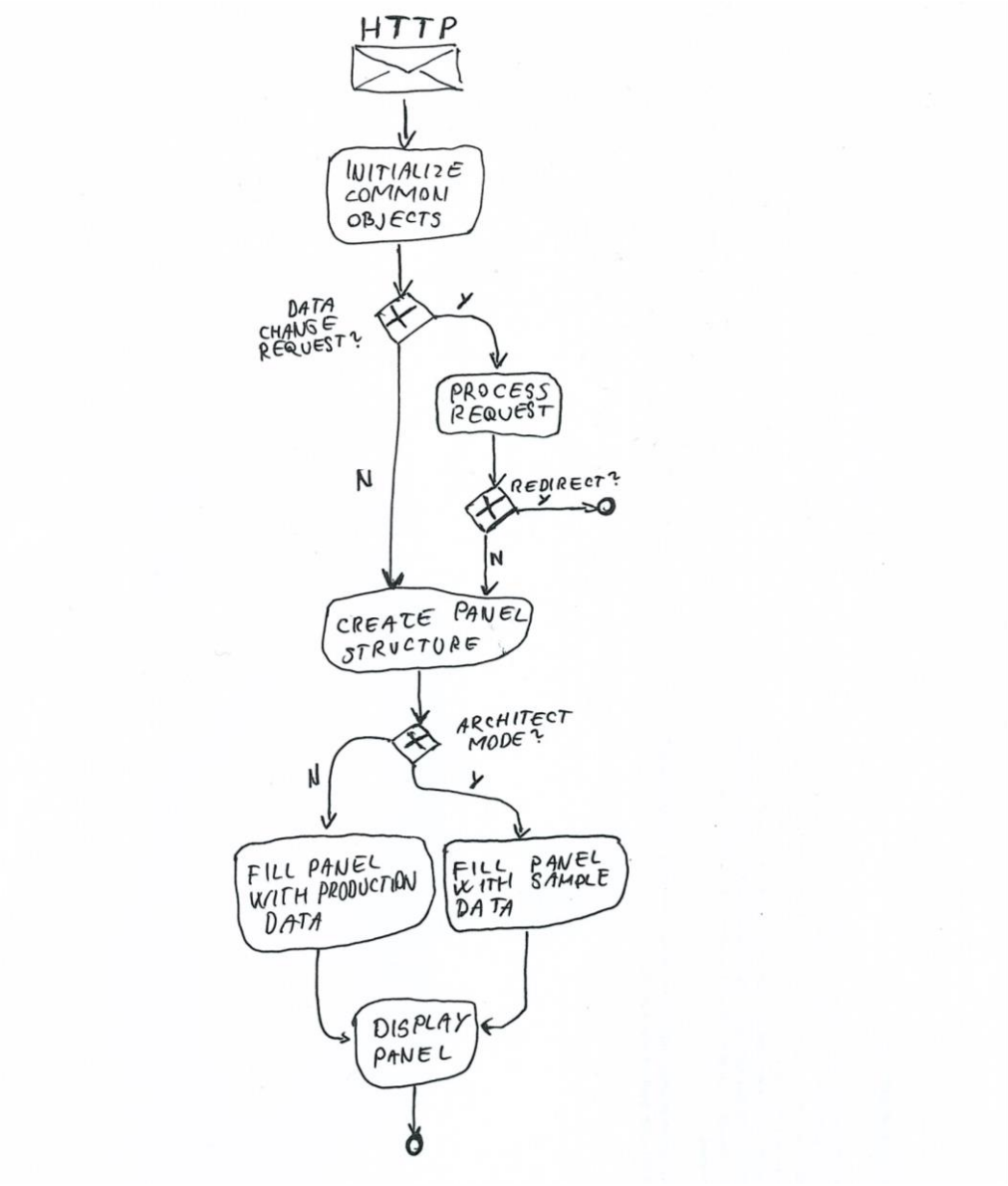
Panely v tomto ponímaní pokrývajú primárne poslednú časť tohto procesu, t.j. samotnú editáciu. Pokiaľ ale pripustíme panely neobsahujúce žiadne polia, môžeme si predstaviť panel obsahujúci len navigačné ovládacie prvky, ktorými sa užívateľ presúva na iné panely. Zatiaľ čo editačné panely pracujú s jediným záznamom, navigačné pristupujú k celej tabuľke, prípadne dokonca k žiadnej z tabuliek, pokiaľ by sme chceli aj ono hlavné menu modelovať ako špeciálny panel. Síce sa tieto panely zdajú byť diametrálne odlišné, táto odlišnosť nie je záležitosťou panelu ako celku, ale len niektorých z jeho ovládacích prvkov. V prípade editačných panelov ovládacie prvky nepotrebujú prístup k databáze, navigačné ovládacie prvky budú musieť obsahovať dáta príslušnej tabuľky, prípadne až niekoľkých tabuliek.

K problému špecializovaných panelov možno pristupovať dvojako. Tieto panely môžu ostro vyčlenené od štandardných a spracovávané úplne odlišne. To by znížilo nároky na všeobecnosť štandardného panelu, ktorých bude v návrhu väčšina, ale sťažila by sa tým údržba kódu, viaceré jeho časti by boli duplicitné a základný všeobecný panel by pokrýval iba minimum funkcionality. Na druhej strane, pokiaľ by sa dostatočne všeobecný koncept panelu, ktorý by už v základnej forme zvládal bežné operácie ako vykreslenie svojich ovládacích prvkov a to bez ohľadu na to, či sú tieto ovládacie prvky jednoduchými tlačidlami bez prepojenia s databázou alebo rozsiahlymi navigačnými zoznamami, umožnilo by to jednotné spracovanie všetkých stránok administračného prostredia a tiež vytváranie zaujímavých kombinovaných panelov, ako napríklad navigačného zoznamu, ktorý by ale obsahoval polia pre rýchle pridanie nového záznamu alebo panelu umožňujúceho in-line editáciu. Preto je zväčša preferovaný druhý prístup.

3.3.2 Proces spracovania stránky

Popíšme teraz základné úkony, ktoré sú súčasťou každého úspešného generovania stránky administračného prostredia. Pokiaľ je súčasťou spracovania stránky aj úprava dát v databáze, ako napríklad vloženie nového záznamu do tabuľky na základe odoslaného webového formulára, vykoná sa táto operácia po inicializácii základných objektov a pred vykreslením panelu. V prípade, že sa užívateľ presúva na iný panel, zastaví sa spracovanie stránky po prípadných úpravách v databáze a dôjde k presmerovaniu na novú stránku, kde proces začína odznova.

Pri spracovaní stránok určených návrhárovi prostredia je postup mierne odlišný. Je to spôsobené najmä tým, že návrhár, či už v automatickom alebo manuálnom režime, potrebuje prístup k metadátam databázy. Naopak, návrhár nepotrebuje pristupovať k produkčnej databáze. Tento rozdiel však nie je markantný – k jeho ilustrácii slúži obrázok 3.



Obr. 3

Priebeh spracovania HTTP požiadavky

Pokiaľ sa návrhár v rámci kontroly výsledkov automatického návrhu pohybuje po modele a nezasahuje doň, dochádza k inicializácií panelov podobne ako v predošlom prípade, avšak s tým rozdielom, že panely nie sú napĺňané produkčnými dátami. Aby sa takto prezentovaný model priblížil tomu, s ktorým bude pracovať koncový užívateľ, generujú polia náhodné dáta, ktoré sa zobrazujú vo formulároch. Týmto spôsobom sa zamedzí duplicite veľkej časti logiky vykresľovania formulárov a návrhár si bude môcť urobiť presnú predstavu o výsledku svojej práce bez toho, aby musel pri finálnej kontrole prechádzať do role administrátora. Možnosť návrhára pristupovať k produkčným dátam totiž môže byť pre zákazníka sama o sebe nežiaduca.

Tieto generované dáta by naviac nemali predstavovať celkom náhodné reťazce, ale hodnoty, ktoré sa budú svojim formátom blížiť dátam, s akými bude neskôr pracovať koncový užívateľ. Základnú predstavu o tomto formáte možno získať z dátových typov príslušných databázových stĺpcov. Preto by napríklad v poli pre stĺpec typu `FLOAT` malo byť reálne číslo zo zvoleného intervalu s istou

presnosťou. Presná voľba týchto parametrov náhodných dát nie je z funkčného hľadiska podstatná, slúžia výlučne k väčšej názornosti v GUI návrhára.

Odlišnosti v spracovaní stránok budú patrne väčšie, pokiaľ bude návrhár priamo editovať štruktúru jednotlivých panelov, vlastnosti ich ovládacích prvkov a polí (viď 3.5). V tomto prípade sa bude spracovanie viac podobať klasickej MVVM (Model, View, View-Model) aplikácii. Síce budú panely modelu a ich časti nebudú pevne určené, ale ovládacie prvky na úpravu jednotlivých polí alebo iných častí možno špecifikovať vopred.

3.3.3 Prehľad dátovej vrstvy

Keďže s návrhom prostredia sa bude pracovať v dvoch primárnych fázach (návrh a správa reálnych dát) a tieto fázy sa môžu naviac striedať vo viacerých iteráciách úprav tohto prostredia po jeho nasadení, je potrebné projekty a všetky ich súčasti dlhodobo uchovávať. Aplikácia teda potrebuje vlastné, interné dátové úložisko a pridruženú dátovú vrstvu schopnú serializovať editačné prvky, polia, panely a celé projekty a následne ich z tejto serializovanej formy rekonštruovať.

Dvomi základnými typmi úložísk dostupných aplikáciám je súborový systém a databáza. Zatiaľ čo sa uloženie štruktúry panelov do súborov v presne stanovenom formáte (napríklad ako XML) môže zdať implementačne priamočiarejšie, pri komplexnejších návrhoch s väčším množstvom panelov by sa však dáta v súboroch stali pravdepodobne v krátkej dobe neprehľadnými a ťažko udržiavateľnými, berúc do úvahy fakt, že o každom jednom formulárovom poli sa musí uchovávať dostatok informácií, aby mohlo toto bolo schopné plniť svoju funkciu správy dát príslušného databázového stĺpca. Možným riešením je rozložiť návrh do viacerých súborov, tieto by však museli byť medzi sebou previazané tak, aby sa z polí a ovládacích prvkov dal spätne zložiť panel. S dekompozíciou objektov sa vie oveľa lepšie vysporiadať relačná databáza, ktorá umožňuje určiť a následne kontrolovať korektnosť väzieb medzi objektmi, t.j. referenčnú integritu. Navyše, prenositeľnosť databázy týmto neutrpí a príslušný návrh prostredia bude možné preniesť na iný server automaticky generovaným SQL skriptom.

Ďalej je nutné určiť „rozumnú“ mieru dekompozície modelu. Výhodou takejto dekompozície je možnosť lokálnych úprav v modeli – zmena konkrétneho panelu alebo poľa bez prepisovania celého návrhu môže priniesť výrazné zrýchlenie vo fáze návrhu. Na druhej strane, úplná dekompozícia na elementárne atribúty, ako je napríklad popisok konkrétneho poľa alebo odkaz na akciu, ktorá sa má vykonať pri kliknutí na konkrétne tlačidlo (ovládací prvok) formulára, by znamenala vytvorenie množstva tabuliek a identifikátorov jednotlivých vlastností týchto objektov. Čím detailnejšie sa bude dátová vrstva pri serializácii a deserializácii objektov „zaoberať“, tým väčšia bude pravdepodobnosť, že táto vrstva sa bude musieť upravovať aj pri minoritných zmenách v internej štruktúre objektov. Táto cesta vedie k pomerne krehkému systému, v ktorom by už zaistenie jednoduchej rovnosti

$$x = deserializuj(serializuj(x))$$

vyžadovalo značné úsilie. Zlepšenie výkonu aplikácie je v tomto prípade už podstatne menšie a aby vôbec nejaké nastalo, bolo by potrebné detailne detekovať, ktoré vlastnosti objektov návrhu sa zmenili a ktoré nie.

Preto je vhodné vykonať dekompozíciu na základnej úrovni (panely, polia, ovládacie prvky) a následne tieto časti serializovať ako celky do textového alebo binárneho formátu. Serializácia týchto celkov by mala byť zaistená jednotne pre všetky prvky danej kategórie (všetky polia, všetky ovládacie prvky,...) tak, aby si neskoršie menšie úpravy týchto objektov nevyžadovali zložité úpravy v dátovej vrstve, ale len lokálnu „registráciu“ týchto zmien na úrovni príslušných objektov. Dôsledkom je prenechanie serializácie a deserializácie týchto častí na ne samotné. Dátovej vrstve potom pri serializácii stačí vyžiadať si serializované verzie jednotlivých objektov tak, aby ich bolo možné pri deserializácii spätne identifikovať a previazať (napr. naplniť panely poľami).

Systémová databáza bude plne v správe aplikácie a dá sa predpokladať, že požiadavky na prácu s touto databázou budú predstavovať najkomplexnejšiu časť databázovej vrstvy. Nie je to však jediná databáza, s ktorou bude aplikácia pracovať. Z predchádzajúceho a zo základných cieľov systému (viď 1.4) plynie použitie celkom troch databáz, kde každá bude použitá rôznym spôsobom rôznymi typmi užívateľov. Zhrňme úlohy týchto databáz, resp. komponent dátovej vrstvy s nimi pracujúcich.

- **Produkčná databáza.** Je úložiskom produkčných dát, ktoré sa majú spravovať. Bude k nej pristupovať iba koncový užívateľ, budú sa z nej načítať dáta do príslušných panelov a každá úprava týchto dát bude vykonaná na priamu žiadosť tohto užívateľa.
- **`INFORMATION_SCHEMA`.** Obsahuje metadáta produkčnej databázy a je používaná pri automatickom návrhu a kontrole návrhu po manuálnych úpravách. Koncový užívateľ ju nepoužíva.
- **Systémová databáza.** Uchováva projekty a príslušné návrhy prostredia v serializovanej podobe. Je využívaná všetkými užívateľmi, z toho koncovým užívateľom len ako „read-only“.

3.3.4 Centrálna riadiaca jednotka

Dátová vrstva predstavuje podklad pre ďalšie operácie aplikačnej logiky a ako taká je prakticky nezávislá do toho, či je aplikácia navrhnutá ako desktopový „tučný klient“ alebo webový portál (viď 3.2). Vyššie vrstvy však už musia odlišnosť týchto prostredí zohľadňovať. Keďže bolo zvolené webové riešenie, ďalšie aktivity po získaní potrebných dát musia smerovať k vykonaniu užívateľom požadovanej operácie, ktorá vychádza z obsahu HTTP požiadavku a v konečnom dôsledku k zobrazeniu webstránky – vygenerovaniu výsledného HTML formulára, ktorý je, ako bolo popísané v 3.3.1, grafickou realizáciou panelu.

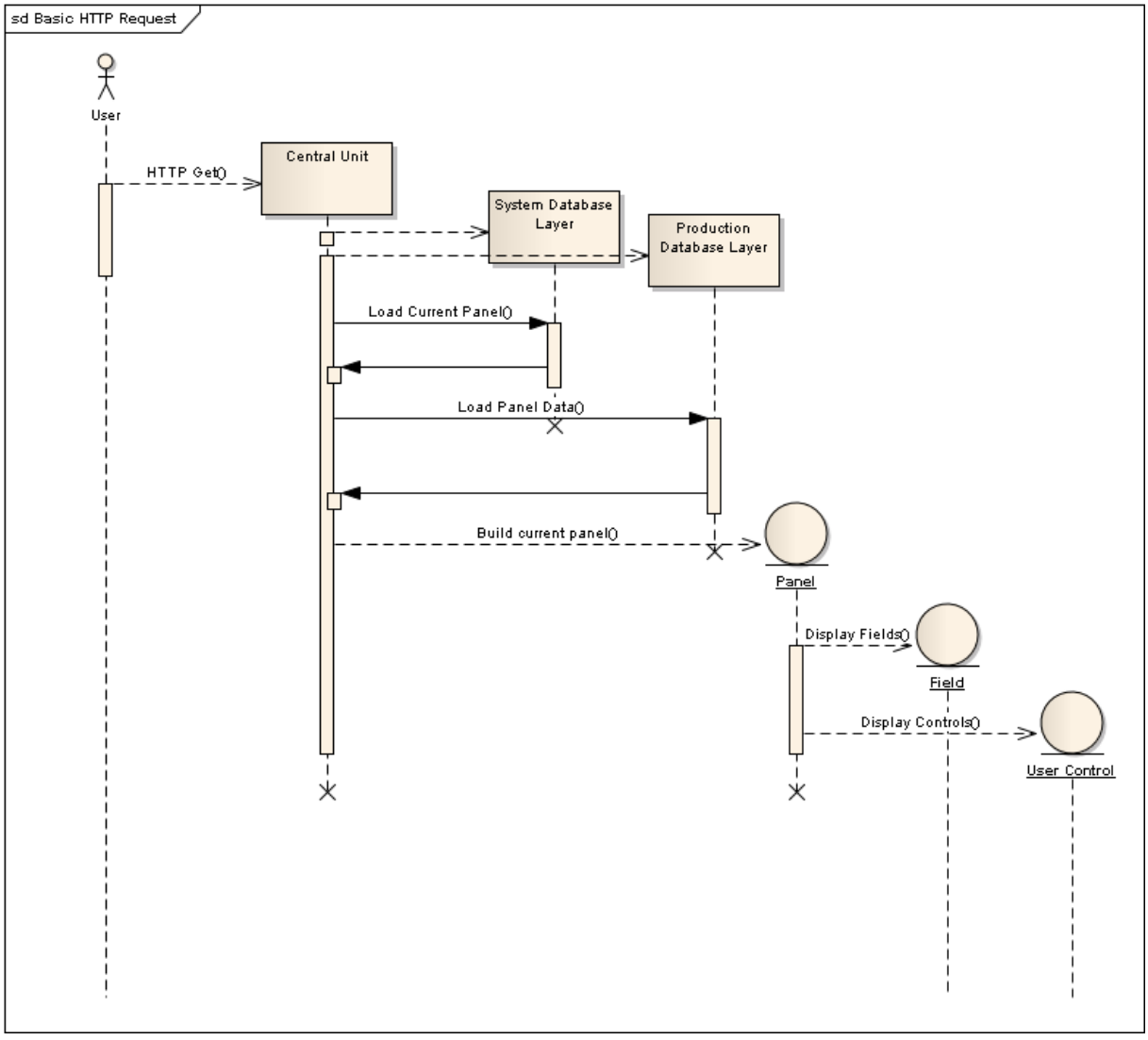
Úlohu mediátora medzi databázovou vrstvou a objektmi návrhu prostredia zohráva centrálna riadiaca jednotka (na obrázku 4 označená ako Central Unit). Táto na začiatku spracovania každého HTTP požiadavku inicializuje databázovú vrstvu (tak pre produkčnú ako aj systémovú databázu), načíta panel alebo panely, ktoré majú byť vo výslednom zobrazení obsiahnuté a vyžiada od produkčnej databázy dáta, ktorými sa tieto panely majú naplniť. Existencia tohto centrálneho bodu aplikácie so sebou prináša niekoľko výhod.

- Zjednoduší sa kontrola užívateľských oprávnení a pokus o neoprávnený prístup bude možné zastaviť ešte pred prístupom do produkčnej databázy. Centrálna jednotka overuje identitu užívateľa a jeho oprávnenia voči projektu a panelu, na ktorý jeho HTTP požiadavka smeruje.

K tomu stačí prístup do systémovej databázy, bez prístupu k citlivým produkčným dátam alebo metadátam databázy.

- Prezentačná vrstva je takto oddelená od vrstvy databázovej a do databázovej vrstvy sú posielané len korektné a kompletné dáta odovzdané panelmi.
- Hlavné objekty aplikácie, ako rozhrania jednotlivých databáz a aktívny panel, sú vytvorené raz a následne poskytované centrálnou jednotkou ďalším komponentám.

Priložme ešte kompletný pohľad na spracovanie HTTP požiadavky, zahŕňajúci centrálnu riadiacu jednotku a naznačujúci fázy aktivity jednotlivých zložiek. Obrázok 4 nepokrýva všetky alternatívy priebehu spracovania, len základné časti, ktoré pri používaní prostredia administrátorom (nie pri jeho úprave návrhárom) prebehnú, až na chybové stavy, vždy. Ide o vytvorenie, naplnenie a zobrazenie jedného panelu, čo zodpovedá prechodu ľavou stranou diagramu na obrázku 4.



Obr. 4 Úloha centrálnej riadiacej jednotky pri načítaní stránky – CRJ inicializuje databázovú vrstvu, získava z nej model prostredia, vyberá aktuálny panel, napĺňa ho dátami a dáva pokyn na jeho vykreslenie. Panel zas tento povel deleguje na svoje polia a ovládacie prvky.

3.4 Reverzné dátové inžinierstvo

Hlavnou funkciou programu je návrh administračného prostredia. Tento návrh musí vychádzať iba z dát v databáze. Preto treba identifikovať vhodný zdroj metadát a metodiku, ktorou z týchto dát vyťažiť logické vzťahy medzi objektmi v databáze uloženými.

Táto, analytická časť aplikácie, bola ovplyvnená prácou Mgr. Hany Kozelkovej [Kz], ktorá rozoberá niektoré zo známych algoritmov reverzného dátového inžinierstva. Niektoré idey boli použité v zjednodušenej forme a je prípustné ich ďalšie zlepšovanie. Naopak, niektoré princípy predstavené v tejto práci sa v kontexte danej aplikácie ukazujú ako nepraktické a bolo použité vlastné náhradné riešenie, ako je ukázané ďalej v tejto časti.

3.4.1 Zdroje metadát

Väčšina RDBMS poskytuje špeciálne systémové funkcie a tabuľky, ktorými možno získať informácie o tabuľkách, stĺpcoch, indexoch a podobne. Tieto príkazy sa však medzi jednotlivými databázovými systémami výrazne líšia nielen syntakticky, ale hlavne rozsahom takto dostupných dát. Ich použitie by výrazne sťažilo postupné pridávanie podpory pre ďalšie RDBMS a v prípade neexistencie potrebného ekvivalentu by podporu pre niektoré systémy nebolo možné pridať vôbec.

Ako východisko bola zvolená `INFORMATION_SCHEMA`, štandardizovaná databáza metadát, ktorej základ je súčasťou všetkých hlavných RDBMS. Naviac, k tabuľkám (presnejšie pohľadom) `INFORMATION_SCHEMA` možno pristupovať štandardnou syntaxou SQL. Táto databáza obsahuje potrebné údaje o tabuľkách, stĺpcoch a ich dátových typoch, ktoré sa dajú využiť pri automatizovanom návrhu užívateľských ovládacích prvkov, ako aj súpisy primárnych a cudzích kľúčov a ďalších integritných obmedzení, na základe ktorých možno vygenerovať validačné pravidlá formulárov a tiež identifikovať kľúčové stĺpce tabuľky, ktoré reprezentujú príslušný záznam a sú vhodným obsahom prehľadových výpisov obsahu tabuliek.

Hlavné algoritmy obsiahnuté v [Kz] vychádzajú z analýzy dát. Je pravdepodobné, že podrobným skúmaním dát databázy by sa mohli dosiahnuť presnejšie výsledky a dali by sa tak pokryť vlastnosti relačného modelu, ktoré nie sú vyjadrené vlastnosťami databázy zachytenými v `INFORMATION_SCHEMA`. Napríklad sa tak môžu detekovať cudzie kľúče, ktoré nie sú vynútené obmedzeniami referenčnej integrity na úrovni logického návrhu databázy. Tento konkrétny prípad je skôr výnimkou, pretože vývoj databázových aplikácií bez náležitého zavedenia RI je pomerne „nebezpečný“ a súčasné RDBMS RI väčšinou podporujú. Výnimkou je však napríklad engine MyISAM MySQL. Na druhej strane, InnoDB RI podporuje a preto je tneto modernejší engine pre MySQL až na niektoré špeciálne prípady preferovaný.

S priamym použitím produkčných dát ako zdroja informácií pre RDE je ale spojených niekoľko zásadných problémov.

- **Tieto dáta nemusia existovať.** Tieto dáta budú k dispozícii iba v prípade, že sa databáza už istú dobu používa a ide o refactoring alebo rozšírenie existujúceho riešenia. Je však na mieste

otázka, za akých okolností by systém, ktorý je predmetom tejto práce, bol nasadený v rámci rozšírenia zavedeného funkčného riešenia. Pokiaľ je pre zákazníka vyvíjaný nový systém začínajúci od prázdnej databázy, analýza z dát jednoducho nebude možná. Dá sa dokonca predpokladať, že toto bude väčšinový prípad.

- **Pre zákazníka môže byť toto použitie produkčných dát nežiaduce.** Je možné, že zákazník poskytne len vzorku alebo len všeobecný popis dát.
- **Produkčná databáza môže obsahovať miliardy záznamov.** Ak by sa napríklad určovanie kandidátnych kľúčov vykonávalo prostredníctvom viacerých SQL dotazov zisťujúcich unikátnosť určitej kombinácie stĺpcov skrz záznamy tabuľky, automatizovaný návrh by sa mohol výrazne spomaliť.
- **Dôveryhodnosť výsledkov analýzy dát.** Ak istá k-tica stĺpcov predstavuje unikátny identifikátor záznamu v rámci dostupných dát, táto unikátnosť nemusí byť nutne zachovaná po pridaní ďalšieho záznamu. Na druhej strane, metadáta v INFORMATION_SCHEMA sa dajú považovať za platné (aj keď nie úplné) a k zmene tohto stavu je potrebná úprava schémy databázy zo strany vývojára.

Z týchto dôvodov nebola priama analýza dát použitá. Priamym dôsledkom je tiež nepoužitie algoritmov uvedených v [Kz].

3.4.2 Upresnenie metadát užívateľom

Využijúc informácie o väzbách medzi tabuľkami možno tiež s dobrou mierou istoty rozhodnúť, ktoré tabuľky predstavujú samostatné entity a ktoré, naopak, reprezentujú pomocné slabé entitné typy a na základe toho danú tabuľku zahrnúť alebo vylúčiť z administračného prostredia. (Napríklad tabuľka sprostredkúvajúca vzťah M:N sa zrejme nebude editovať samostatne, ale v priamej väzbe na dáta tabuliek v tomto vzťahu.)

„Dobrá miera istoty“ však nie je dostatočná a preto bude mať návrhár prostredia možnosť zmeniť toto rozhodnutie už pred vygenerovaním prvotného návrhu. Podobná situácia nastáva pri určovaní názvu, pod ktorým bude tabuľka pri editácii vystupovať. Názov tabuľky v databáze môže a nemusí byť intuitívny a pravdepodobne nebude plne v zhode s názvom, ktorý sa má zobrazovať užívateľovi. Tieto a ďalšie údaje preto treba vyžiadať od užívateľa pred generovaním návrhu a použiť ich spolu s dátami INFORMATION_SCHEMA.

3.4.3 Dodatočné zdroje

Ďalšou, pokročilejšou možnosťou, je umožniť návrhárovi definovať istú „firemnú kultúru“, ktorá by určovala, aké konvencie pomenovania sa používajú pre rôzne typy tabuliek a stĺpcov, ktoré by systému slúžili ako dodatočné vodítko pri generovaní návrhu. Súčasťou definície takejto kultúry by bol slovník mapujúci často používané názvy tabuliek na názvy, ktorými sa majú označovať v návrhu. Iným využitím konvencií by bolo určenie regulárnych výrazov, ktoré charakterizujú špeciálne tabuľky a stĺpce, ako napríklad

- Systémové tabuľky, ktoré majú byť z návrhu vynechané (napr. „`sys_.*`“).
- Tabuľky mapujúce entity vo vzťahu M:N, kt. sú často označované ako `tab_a2tab_b`, kde `tab_a` a `tab_b` sú tabuľky vo vzťahu M:N.
- Cudzie kľúče sa často označujú ako `tab_a_ID`, kde `tab_a` je referenčná tabuľka. Týmto spôsobom by mohli byť cudzie kľúče detekované aj v prípade, že nie sú definované na úrovni fyzického návrhu databázy.

Je tiež častou praxou, že sa časť databázy opakuje ako základný framework vyvinutý pre interné potreby firmy a používaný vo viacerých podobných projektoch. Bolo by možné vytvoriť administračné rozhranie pre tento základ separátne a v konkrétnych projektoch už dotvárať len správu ostatných tabuliek databázy. To by sa dosiahlo postupným aplikovaním viacerých predpripravených návrhov na jednu databázu a v závere vygenerovaním návrhu len pre zvyšné, nepokryté tabuľky. Bolo by potrebné, aby aplikované návrhy neboli v konflikte so štruktúrou databázy a tiež riešiť prípadné konflikty medzi jednotlivými návrhmi.

Obdobou idey „firemnej kultúry“ v [Kz] je koncept šablón názvov stĺpcov pri identifikácii cudzích kľúčov, ako však bolo ukázané vyššie, možnosti využitia tohto druhu informácií sú podstatne širšie.

3.4.4 Vzťahy medzi entitami

Aplikácia bude pri automatickom návrhu administračného prostredia vychádzať z fyzického modelu tak, ako je implementovaný v príslušnej databáze. Informácie obsiahnuté v tomto modele tvoria nadstavbu relačného modelu, ktorý tým pádom nebude ťažké získať.

Ako už bolo spomínané v stati 1.2, problematickejšie bude odvodenie koncepčného modelu. Aby sa mohla vygenerovať formulárová štruktúra, je nutné najprv identifikovať silné entitné typy a vzťahy medzi nimi. Základom k určeniu vzťahov medzi entitami je analýza cudzích kľúčov medzi reláciami. Všeobecne neplatí, že by databázové systémy obsahovali podporu pre špecifikáciu cudzieho kľúča na fyzickej úrovni, ale systémy bez tejto podpory sú málo používané, prípadne ide o zastarané technológie, ku ktorým sa radí formát tabuliek MyISAM v MySQL.

Spoľahlivá identifikácia referenčných obmedzení v databázovom systéme bez podpory referenčnej integrity by vyžadovala priame skúmanie dát uložených v databáze, ktorá však bola odmietnutá z dôvodov uvedených v časti 3.4.1. Pre systémy bez podpory RI to znamená, že väzby jednoducho nebudú nájdené. Možným doplnkom riešenia je poskytnutie možnosti špecifikácie referenčných vzťahov zo strany návrhára pred vytvorením prvotného návrhu.

Aby sa mohli určiť slabé entitné typy, nutno určiť vlastné a cudzie kľúče tabuliek. Pri cudzích kľúčoch treba naviac rozlíšiť medzi vzťahom 0..N:1 a 0..N:0..1. Tieto vzťahy možno rozlíšiť podľa toho, či stĺpce cudzieho kľúča povoľujú vloženie prázdnych hodnôt (`NULL`).

Okrem podpory RI sa ďalej predpokladá definícia primárneho kľúča pre každú tabuľku spravovanú systémom. Definícia PK síce nebýva u RDBMS povinná, ničmenej vedie na značne nižší výkon tabuliek [TK461] a preto sa jeho definícia vo všeobecnosti doporučuje. Systém bude tento nedostatok

detekovať, pričom užívateľ by mal dostať možnosť primárny kľúč dodefinovať a znova spustiť analýzu databázovej štruktúry. Existencia primárneho kľúča je dôležitá pre vyhľadávanie a navigáciu medzi entitami. Všeobecne sa nedá predpokladať ani unikátnosť záznamov v tabuľke a teda nemožno použiť celý záznam ako zástupný primárny kľúč, ale bolo by potrebné siahnuť po automaticky generovanom `ROW_ID`, prípadne použiť pre rozlíšenie záznamov iný údaj, špecifický pre daný RDBMS. Vzhľadom na komplikácie, ktoré tento aspekt prináša (a na to, že ide o síce komplikovaný, ale okrajový problém), budú tabuľky postrádajúce primárny kľúč z návrhu vynechané.

3.4.5 Komplexnejšie riešenia

Okrem podmienok referenčnej integrity a primárnych kľúčov tabuliek je potrebné zistiť dátové typy stĺpcov vrátane príznakov ako `NOT NULL` alebo `AUTO_INCREMENT`, ktoré sa využijú k výberu vhodného editačného poľa pre daný stĺpec.

Naviac by sa mohli získavať indexy a uložené procedúry. Je napríklad bežnou praxou zaobaľovať dotazy modifikujúce dáta kvôli výkonu do uložených procedúr (databázový stroj nemusí parsovať textový reťazec a kontrolovať, či môže použiť niektorý z exekučných plánov v cache). Umožnenie volania uložených procedúr programom by ale prinieslo dodatočné komplikácie. K detailnejšej analýze by tiež mohlo prispieť vyhodnocovanie všeobecných integritných obmedzení, teda obmedzení mimo referenčnú integritu (obmedzenia typu `CHECK` v SQL).

Na druhej strane, prehnane komplikovaný algoritmus generujúci návrh prostredia by tento návrh urobil ťažšie uchopiteľným nielen po stránke implementačnej, ale aj užívateľskej, keď by užívateľ nedokázal napriek znalosti databázy, pre ktorú návrh generuje, odhadnúť jeho výslednú podobu a musel by ho vždy dôsledne kontrolovať. Preto bol pre prvú verziu programu zvolený jednoduchý, ale rozšíriteľný algoritmus, v dôsledku čoho sa využívajú len informácie uvedené vyššie.

Toto zjednodušenie však nie je bez následkov. Nemožno predpokladať, že návrh prostredia pokrýva všetky integritné obmedzenia databázy (čo by, patrne, nebolo možné ani pri vynaložení oveľa väčšieho úsilia na vyťaženie metadát a to už len z toho dôvodu, že niektoré z týchto údajov nemusia byť obsiahnuté v `INFORMATION_SCHEMA` alebo ich nemusí byť možné modelovať na úrovni validácie formulárových polí a panelov). Krajným prípadom je použitie databázových triggerov, ktoré môžu vykonávať integritné kontroly, ktoré nebolo možné vyjadriť prostriedkami príslušného RDBMS. Pri práci s produkčnou databázou je teda nutné počítať s tým, že vloženie dát do databázy môže zlyhať aj po úspešnej validácii priamo pri vykonávaní databázového dotazu. Na túto situáciu musí byť aplikácia schopná reagovať a podať užívateľovi, pokiaľ možno, čo najkonkrétnejšiu informáciu o príčine chyby.

3.4.6 Kompromisné riešenie

Výsledkom tohto rozboru možností RDI je algoritmus založený na metadátach `INFORMATION_SCHEMA`, ktorý si nerobí nároky na pokrytie všetkých prípustných väzieb medzi reláciami, ale usiluje sa o poskytnutie dobrého približného výsledku v únosnom čase pri obmedzení potreby zapojenia užívateľa do iniciačnej fázy návrhu. Zo všeobecného pohľadu bude zrejme väčšina reálne nasaditeľných postupov riešenia tejto úlohy kompromisom medzi týmito aspektmi

- **Predikovateľnosť, „čitateľnosť“ výstupu vs. zachytenie špeciálnych prípadov.** Čím komplikovanejší algoritmus je použitý, tým viac sa z neho stáva „black-box“ a užívateľ sa nemôže spoľahnúť, že vykoná to, čo od neho zo skúsenosti očakáva. Naviac, snaha o pokrytie zriedkavých úkazov databázového designu môže narušiť kvalitu identifikáciu jednoduchších konceptov. Preto by sa mal algoritmus zamerať najmä na tieto bežné prípady. Dôležité však je, že „zvláštnosti“ modelu môže návrhár ošetriť dodatočne.
- **Rýchle dokončenie prvotnej analýzy vs. zapojenie užívateľa a zvýšenie presnosti.** Viaceré vlastnosti relačného modelu sa nedajú z fyzického modelu nedajú s istotou určiť. Riešením je zapojiť do návrhu užívateľa, ktorému však táto činnosť stále musí zabrať podstatne menej času než vytvorenie návrhu bez použitia generického návrhu. Možným riešením je zapojiť užívateľa implicitne, skrz „firemnú kultúru“, ako bolo popísané v 3.4.4.

3.5 Základné vrstvy systému

Všeobecne prospešným princípom návrhu software je rozdelenie funkcionality medzi menšie komponenty, ktoré spolu komunikujú cez spoločné rozhranie. Toto umožňuje oddelený vývoj jednotlivých častí, pričom vývojár nemusí poznať detaily iných komponent. Zároveň to umožňuje zameniteľnosť týchto komponent v neskoršej fáze životného cyklu aplikácie a selektívne dopracovávanie kľúčových častí. Nájdenie vhodných miest takéhoto delenia v danom systéme je predmetom tejto časti.

3.5.1 Oddelenie časti závislej na type RDBMS

Piaty z cieľov projektu, vytýčených v kapitole 1, vyžaduje rozšíriteľnosť riešenia o podporu ďalších RDBMS. Pritom je žiaduce minimalizovať objem kódu, ktorý nutno reimplementovať pre každý nový RDBMS. Dvoma základnými časťami aplikácie sú preto vrstva databázovo špecifická, ktorá musí preklenúť syntaktické a funkčné rozdiely medzi databázovými systémami a poskytovať jednotnú reprezentáciu dát nadradenej, databázovo nezávislej vrstve. Treba naviac počítať s možnosťou, že systém bude pracovať nad dvoma rôznymi databázovými systémami súčasne – to nastane v prípade, keď by produkčná databáza spravovaných dát a interná databáza systému, v ktorej si uchováva podobu návrhu prostredia pre správu produkčnej databázy, boli prevádzkované na dvoch rôznych databázových serveroch. V rámci servera s produkčnou databázou bude aplikácia ešte pristupovať k databáze `INFORMATION_SCHEMA`, aby získala metadáta, ako bolo zdôvodnené v 3.4. Ku každej z týchto troch databáz sa pristupuje za iným účelom, preto je logické rozdeliť dátovú vrstvu na tri časti podľa databázy, s ktorou pracujú. Ich hlavnou úlohou je konvertovať dáta z databázy na univerzálne objekty a späť. Celkovo bude systém pracovať s troma databázami – internou, tzv. systémovou databázou, databázou metadát, t.j. INFORMATION_SHCEMA a produkčnou databázou obsahujúcou samotné spravované dáta. Keďže každá z týchto databáz bude využívaná inak a inými užívateľmi, bude pre každú vytvorená samostatná databázová komponenta.

3.5.2 Oddelenie častí návrhára a administrátora

Druhým dôležitým rozdelením aplikácie je oddelenie časti určenej architektovi, ktorý navrhuje užívateľské prostredie (formuláre, typy polí a podobne) pre danú databázu, a časti administrátora, ktorý toto prostredie používa pre správu produkčných dát. Prvá menovaná bude intenzívne využívať metadáta a ukladať a meniť vyprodukovaný model administračného prostredia. Pritom ale nepotrebuje prístup ku konkrétnym dátam databázy, pre ktorú sa toto prostredie navrhuje. (Keby, naopak, vyprodukovaný návrh závisel od týchto dát, šlo by o zásadný nedostatok a riešenie by sa nedalo považovať za všeobecné.) Administračná časť, naopak bude aktívne pracovať s produkčnou databázou, a zobrazovať jej dáta v štruktúre načítanej z internej databázy. Túto však nepotrebuje meniť.

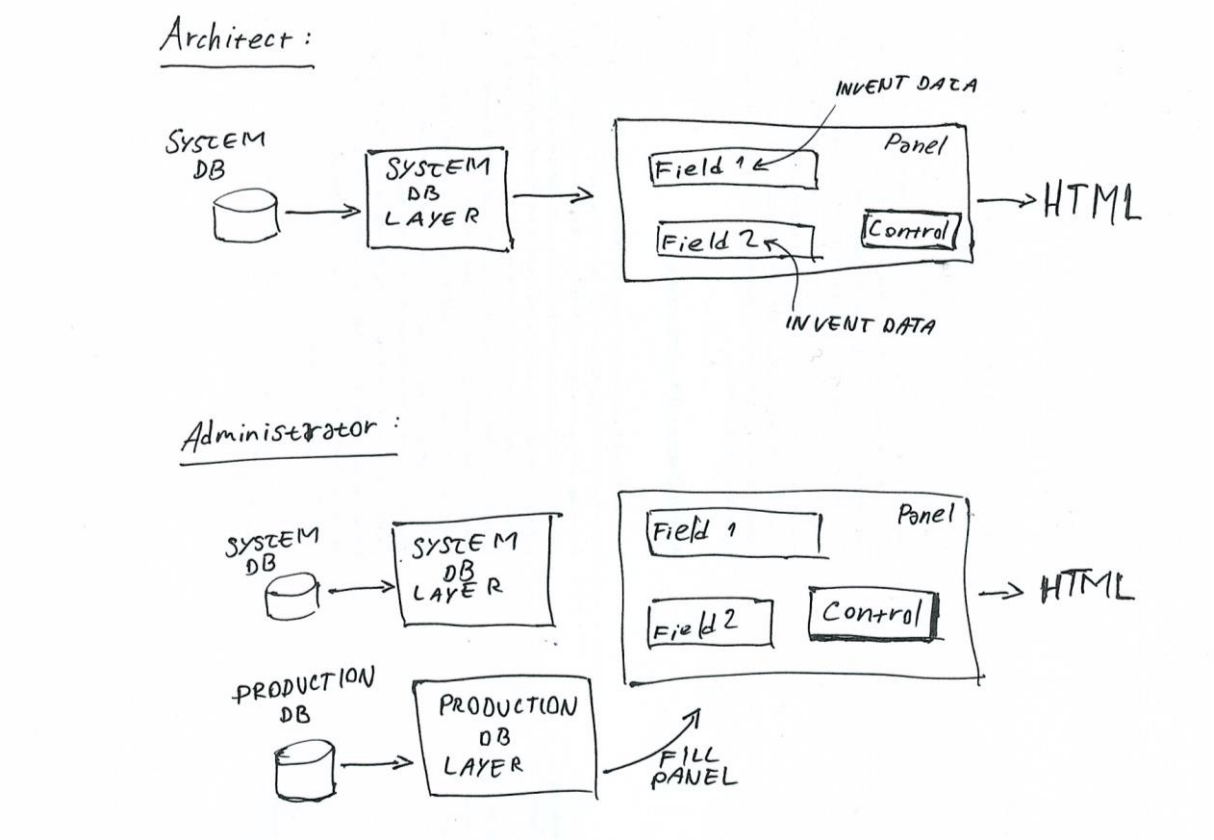
Ďalším dôvodom pre oddelenie prostredia návrhára a administrátora sú rozdielne požiadavky na nástroje a rozhranie pre tieto typy užívateľov. Automatizovaná časť návrhu vychádza z INORMATION_SCHEMA. Pri následnom ladení návrhu ľudským návrhárom sa táto databáza využíva na kontrolu konzistencie úprav s databázovým modelom. Návrh sa priebežne aktualizuje v systémovej databáze. Produkčnú databázu však návrhár vôbec nepotrebuje a, z bezpečnostných dôvodov, by k nej dokonca nemal mať prístup. Na druhej strane, administrátor pracuje primárne s produkčnými dátami, ktoré systém obalí do formulárov uložených v systémovej databáze. Na tomto základe vznikajú dve oddelené skupiny objektov používaných v aplikačnej logike – objekty týkajúce sa dát z `INFORMATION_SCHEMA` využívané exkluzívne návrhárom a objekty s obálky produkčných dát určené pre koncového užívateľa. Ako je zvýraznené na obrázku 1, tieto objekty už nie sú „databázovo špecifické“, t.j. majú rovnakú štruktúru a použitie bez ohľadu na typ podkladových RDBMS.

V rámci užívateľského rozhrania sa pre sekciu návrhára predpokladá použitie formulárov jednotného vzhľadu, v ktorých bude prebiehať konfigurácia typov polí, validačných pravidiel a ďalších aspektov prostredia pre koncového užívateľa. Toto konfigurovateľné prostredie preto nemôže byť pokryté konečnou sadou prístupných formulárov, ako býva zvykom u štandardných webových aplikácií, ale jeho podoba musí byť vytváraná prevažne dynamicky z menších komponent, ktoré reprezentujú časti výsledného formulára. Tieto komponenty môžu byť v zásade troch typov: prvky obsahujúce dáta, ovládacie prvky a kontajnery zaobaľujúce ďalšie prvky. Užívateľské prostredie je zložením týchto komponent a preto musí každá z nich mať vlastnú prezentačnú vrstvu, podporovať príslušné rozhranie pre komunikáciu s nižšou vrstvou a implementovať vnútornú logiku podľa potreby. Aby mohol návrhár dobre odhadnúť výsledok svojej práce, musí vidieť vytvorené prostredie v podobe blízkej tej, v akej sa zobrazí koncovému užívateľovi. Preto budú mať oba typy užívateľov prístup k jednotnej zobrazovacej komponente tohto prostredia, prirodzene s tým rozdielom, že architekt nebude pracovať s produkčnými dátami. Pre modifikačné nástroje používané výlučne architektom sa použijú samostatné zobrazovacie komponenty realizované formou štandardných webových ovládacích prvkov.

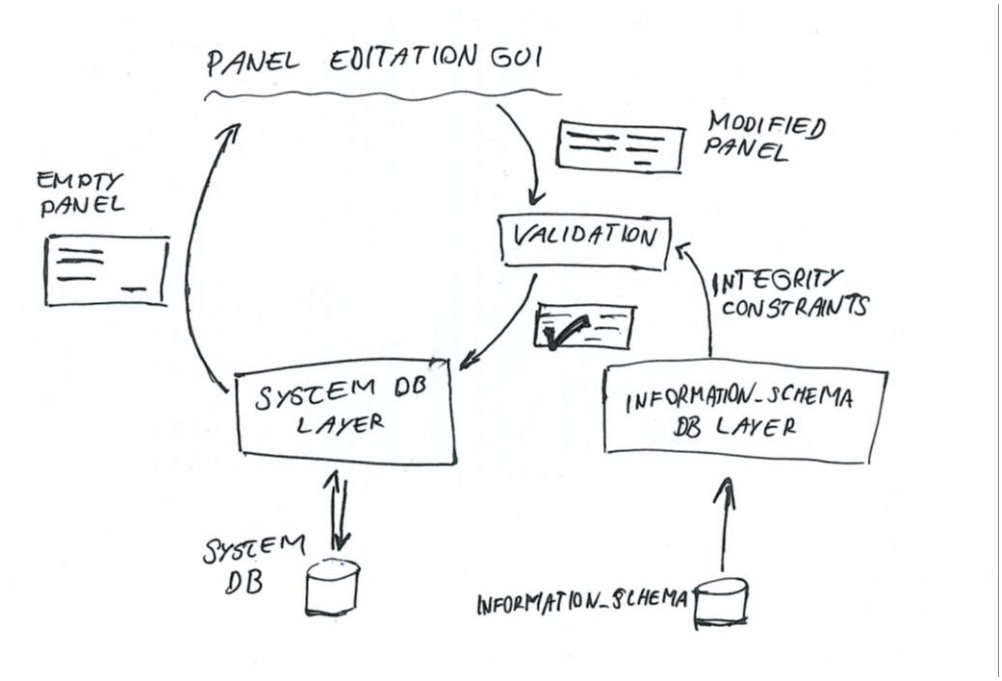
Rozdelenie objektovej a prezentačnej vrstvy pre administračnú časť spôsobilo, že sa tento návrh vymyká bežnej architektúre MVC, kde spravidla požiadavku užívateľa spracuje jeden controller (čo je v tomto prípade pravda len po určitú vrstvu) a vyvolá vykreslenie jedného view. Napriek tomu systém nesie viaceré znaky tohto vzoru.

Obrázok 5 demonštruje rozdiel medzi užívateľom-adminstrátorom a návrhárom prostredia pri prehliadaní dát. Síce obaja rovnako používajú rozhranie systémovej databázy, aby získali štruktúru panelu, ktorý sa má zobraziť, zatiaľ čo administrátor získava dáta na vyplnenie tohto panelu prostredníctvom rozhrania produkčnej databázy, v móde architekta sa využije rozhranie polí, ktoré implementuje funkcie generujúce vzorový obsah týchto polí, ktorý sa následne použije pri zobrazení panelu.

Obrázok 6 ukazuje využitie INFORMATION_SCHEMA pri úprave návrhu prostredia. Najprv dôjde k načítaniu príslušného panelu ako v predchádzajúcom prípade. Následne je zobrazené prostredie grafického návrhára panelov. V tomto rozhraní užívateľ vykoná úpravy panelu a výsledný panel odošle späť systému. Panel môže byť do systémovej databázy uložený iba za predpokladu, že nie je v rozpore s integritnými obmedzeniami produkčnej databázy. To sa overí pri validácii návrhu, za využitia metadát, ktoré z `INFORMATION_SCHEMA` získa príslušná databázová komponenta.



Obr. 5 Porovnanie načítanie panelu pre architekta a administrátora.



Obr. 6 Kontrola upraveného návrhu panelu voči integritným obmedzeniam načítaným z INFORMATION_SCHEMA.

3.6 Výber programovacieho jazyka a frameworku

Programovacie jazyky a knižnice v nich implementované sa líšia najmä výkonom vytvorených programov, pracnosťou vývoja (ktorá je odvodená od komplexnosti dostupných knižníc; pohodlie vývojára a výkon výslednej aplikácie sú zväčša antagonistické) a množinou podporovaných platforiem (vzhľadom na vytýčené ciele do úvahy prichádzajú len jazyky schopné prevádzky na Windowsom aj Linuxom, tieto sa ale líšia v náročnosti zaistenia multiplatformovosti riešenia – jednoduchou cestou sú PHP alebo Java, zložitejšie to je pri C#). Naviac treba zohľadniť prostriedky na komunikáciu s databázou, keďže aplikácia bude musieť v rôznych databázach alebo súboroch uchovávať de facto celý svoj stav vrátane užívateľského rozhrania pre administrátorov.

Ovládnutie každého jazyka na adekvátnej úrovni si vyžaduje nezanedbateľné množstvo znalostí a prinajmenšom niekoľkomesačnú praktickú skúsenosť s používaním jazyka pri riešení netriviálnych úloh. Takúto skúsenosť mal pred započatím projektu autor s dvoma jazykmi, konkrétne C# (v kombinácii s ASP.NET) a PHP, po porovnaní ktorých zvolil implementáciu v C#. Vybrané detaily porovnania možno nájsť nižšie.

3.6.1 Prístup k dátam

Aplikácia je dátovo intenzívna. Musí pracovať s produkčnou databázou, so zdrojom metadát o tejto databáze a naviac si získané informácie a vlastný návrh prostredia správcu ukladať vo vlastnej, internej databáze. Celkovo teda bude pristupovať k trom zdrojom dát, ktoré určujú nielen dáta, ktoré sa užívateľovi zobrazia, ale aj ich grafické zaobalenie a validačnú a navigačnú logiku. Tak PHP ako C#

umožňujú pristupovať k rôznym databázovým systémom a iným dátovým zdrojom, akými môžu byť XML alebo CSV súbory, prípadne webové služby tretích strán. Rozdiel je v dostupnej nadstavbe nad základným databázovým pripojením. Technológia ADO.NET a jej nadstavby v rámci .NET frameworku predstavujú komplexné riešenie práce s dátami, umožňujúce vytvoriť aplikáciu s dátovou vrstvou na rôznych stupňoch abstrakcie – od konkrétnych SQL príkazov po automaticky generované POCO objekty zastupujúce databázové entity v podaní Entity Frameworku.

3.6.2 Multiplatformovosť

Jednou z hlavných predností PHP je jednoduché nasadenie na IIS v prípade systémov Windows aj Appache pri linuxových systémoch. Má tiež podporu všetkých hlavných databázových systémov. ASP.NET má natívnu podporu v prostredí Windows a SQL Serveru. K použitiu iných databázových systémov však stačí implementovať základné rozhranie poskytovateľa databázového spojenia. Takáto implementíácia je dostupn pre Oracle, MySQL a ďalšie RDBMS. Naviac existuje všeobecná implementácia pripojenia ODBC, ktorá je slabšia po stránke výkonu, ale umožňuje sa pripojiť k ľubovoľnej relačnej databáze. Okrem základnej podpory prístupu k databáze sú pre ASP.NET dostupné aj implementácie systému správy užívateľov (ASP.NET Membership, Roles a Profile) pre MS SQL Server, Oracle aj MySQL, čo urýchli vývoj základných prvkov webu, ktoré nie sú hlavným predemtom práce.

ASP.NET síce nie je primárne určené pre linuxové operačné systémy, projekt Mono však predstavuje použiteľnú implementáciu .NET pre Linux. Nemá podporu niektorých najnovších vlastností platformy .NET a nemá ani jej mieru stability, ale pokiaľ sa nepoužijú niektoré špecifické technológie vyvíjané Microsoftom mimo základ frameworku (ako je napríklad ASP.NET AJAX), je Mono použiteľným základom pre linuxovú verziu aplikácie. V prípade potreby je tu stále možnosť rozdeliť projekt na linuxovú a windowsovú vetvu, čo je ale nežiaduce z dôvodu údržby.

3.6.3 Nepoužitie ASP.NET MVC

Po výbere programovacieho jazyka C# a základného frameworku ASP.NET bolo ešte potrebné zvoliť medzi novým trendom v ASP.NET – frameworkom MVC s vykresľovacím enginom Razor a klasickými stránkami ASP.NET so zabudovanými prvkami MVVM a vykresľovaním cez WPF. Vzhľadom na celkový návrh aplikácie popísaný v predchádzajúcej stati bola zvolená druhá z možností. Dosiahnutie potrebnej miery flexibility v ASP.NET MVC by totiž pravdepodobne znamenalo použitie niekoľkých hlavných controllerov, ktoré by spracovanie komponent formulára delegovali premennému počtu subcontrollerov, a vytvorenie HTML view pre každý prvok zvlášť. I za predpokladu, že takáto funkčnosť je dosiahnuteľná, dá sa u nej predpokladať väčšia pracnosť než pri využití WPF.

Keďže je Razor založený na HTML, znamenalo by jeho použitie tiež viac práce na grafickej obálke aplikácie, čo nie je predmetom tejto práce. Objekty WPF sa v ASP.NET vytvárajú jednoducho a ich východzia realizácia do kódu HTML je pre formulárovú aplikáciu bez zvláštnych vyšších grafických nárokov postačujúca. WPF predstavuje pomerne flexibilné riešenie, neobmedzujúce zanorenie kontajnerových prvkov a druh ich obsahu.

Použitie ASP.NET MVC by na druhej strane mohlo uľahčiť vývoj časti aplikácie používanej návrhárom prostredia, bola však zvolená cesta menšieho rizika kolízie požiadaviek tohto frameworku a zamýšľaných funkcionalít programu.

3.7 Databázová vrstva

Táto časť ďalej rozvíja stať 3.3.3, zaoberajúcu sa dátovou vrstvou a problémami s ňou spojenými na všeobecnejšej úrovni. Ako bolo v tejto časti vyložené, aplikácia bude pracovať celkom s troma databázami. Z ich účelu možno odvodiť niektoré ich vlastnosti a nároky na prácu s nimi.

Administračné prostredie je vytvárané pre konkrétnu produkčnú databázu. Práve o tejto databáze môžeme učiniť len minimálne predpoklady. Aj keď jej schéma nie je vopred známa, po jej preskúmaní musí byť systém vykonávať nad touto databázou základné až stredne zložité dotazy, aby sa získali dáta pre naplnenie jednotlivých panelov. Pokiaľ sú tabuľky tejto databázy previazané cudzími kľúčmi, čo je pri OLTP systémoch skôr pravidlom než výnimkou, bude sa pri získavaní prehľadových tabuliek tiež často vykonávať operácia spojenia tabuliek (klauzula `JOIN`). Avšak vzhľadom na to, že štruktúra databázy nie je vopred daná, je potrebné obmedziť sa na čo najjednoduchšie dotazy – pokiaľ možno, nepoužívať vnorené dotazy alebo dotazy, ktoré by mohli byť potenciálne časovo náročné. Nie je totiž známe ani množstvo dát v databáze.

`INFORMATION_SCHEMA`, z ktorej sa budú vyťažovať metadáta o štruktúre produkčnej databázy, je síce štandardizovaná, ale dodávatelia DBMS ju zvyknú o niečo rozšíriť. Tu je vhodné obmedziť sa na tabuľky určené štandardom. Keby sa systém spoliehal na niektoré dáta mimo tento štandard, môže to spôsobiť problémy pri implementácii databázového rozhrania pre ďalšie systémy.

Aj keď každá z týchto databáz bude použitá iným spôsobom inými užívateľmi a obsluha každej z týchto databáz bude predstavovať vlastnú systémovú komponentu, z dôvodu minimalizácie duplicitného kódu a sprehľadnenia celého riešenia je vhodné vytvoriť základnú databázovú vrstvu, ktorá bude slúžiť ako podporný nástroj pre vyššie vrstvy a zároveň zvýši znovupoužiteľnosť ich kódu tým, že zakryje drobné syntaktické rozdiely medzi rôznymi RDBMS.

Obľúbeným a moderným prostriedkom pre prácu s databázou v prostredí .NET je LINQ to SQL. Ide o sadu rozširujúcich funkcií, ktoré umožňujú mapovanie objektov C# na databázové tabuľky a vykonávanie dotazov LINQ nad týmito objektmi, ktoré sa následne preložia do SQL a vykonajú v databáze. Síce použitie tohto rozšírenia má mnoho výhod a dokonca existujú jeho implementácie pre MySQL, PostgreSQL, Oracle a ďalšie DBMS, pre vyvíjanú aplikáciu nebolo vhodné. Predovšetkým, štruktúra produkčnej databázy nie je vopred známa a preto nie je možné vytvoriť jej objektový model. V dôsledku toho tiež nie je potrebné ORM voči produkčnej databáze, stačí jednoduchšie riešenie, ktoré dáta vracia ako dátové riadky alebo tabuľky. Ďalším aspektom je podoba výsledného SQL dotazu. Pri zložitejších dotazoch, ako sú napríklad tie vykonávané nad pohľadmi v `INFORMATION_SCHEMA`, je žiaduca plná kontrola nad podobou vykonávaného dotazu, aby programátor mohol tento dotaz optimalizovať.

Na druhej strane, skladanie textového reťazca SQL dotazu všade, kde aplikácia potrebuje pristupovať k databáze, je neprehľadným a ťažko udržiavateľným riešením, ktoré by radikálne sťažilo dopĺňanie podpory pre ďalšie DBMS. Databázová vrstva teda musí spĺňať tieto požiadavky:

- umožní skrátený zápis dotazov nezávislý na type DBMS
- preklad tohto zápisu do SQL zaistí oddelená komponenta, ktorej implementácia sama o sebe zaistí podporu pre ďalšie databázové systémy
- tento zápis jasne určuje stavbu výsledného dotazu až na syntaktické detaily zápisu v danom DBMS, t.j. je ním určené, kde sa použijú klauzuly ako `JOIN`, `WHERE` a podobne

Z týchto požiadaviek vyšla implementácia základnej databázovej vrstvy (viď 3.7.2).

3.7.1 Systémová databáza

Podkladom aplikácie je databáza uchovávajúca primárne dva druhy informácií. Sú to dáta súvisiace so správou užívateľov a existujúcich projektov, teda ich prihlasovacie údaje, prístupové oprávnenia, zámky zabraňujúce súčasným, potenciálne konfliktným úpravám vykonávaným viacerými užívateľmi a podobne. Otázkami bezpečnosti sa bližšie zaoberá časť 3.12.

Podstatnou otázkou je reprezentácia objektov prostredia – panelov, polí a ovládacích prvkov (viď 3.1). Každý z týchto objektov zrejme bude mať množstvo vlastností – svoje umiestnenie, zobrazovaný popisok, identifikáciu tabuľky a stĺpca / stĺpcov, ktoré reprezentuje, a ďalšie špecifické nastavenia.

Jedným z možných riešení je nasledovanie príkladu databázy používanej systémom WordPress a umiestnenie týchto dát do tabuľky typu kľúč – hodnota obsahujúcej konfiguráciu všetkých objektov. Pri inicializácii objektu by sa načítali všetky relevantné položky a podľa nich by sa objekt nastavil. Ukladanie do databázy by predstavovalo opačný proces, pri ktorom by každý objekt vygeneroval slovník svojich vlastností a poslal ho databázovej vrstve.

Tento model má ale mnoho obmedzení. Vychádza z predpokladu, že vlastnosti objektov nemajú bohatú štruktúru. Keby napríklad niektorá vlastnosť predstavovala objekt, ktorý má ďalšie vlastnosti, mapovanie by sa skomplikovalo. K ešte väčším komplikáciám by došlo, keby vlastnosti obsahovali pole hodnôt, prípadne iných objektov. Tu by sa musel používať štrukúrovaný kľúč, čo by viedlo k neprehľadnému a ťažko udržiavateľnému modelu, kde by každá zmena serializovaných objektov vyžadovala niekoľko zásahov do mapovania v rámci serializácie, nehovoriac o tom, že výsledná databáza by nespĺňala prvú normálnu formu. Už len splnenie základnej požiadavky na serializačný engine, ktorú môžeme symbolicky vyjadriť rovnicou

x = deserializuj(serializuj(x))

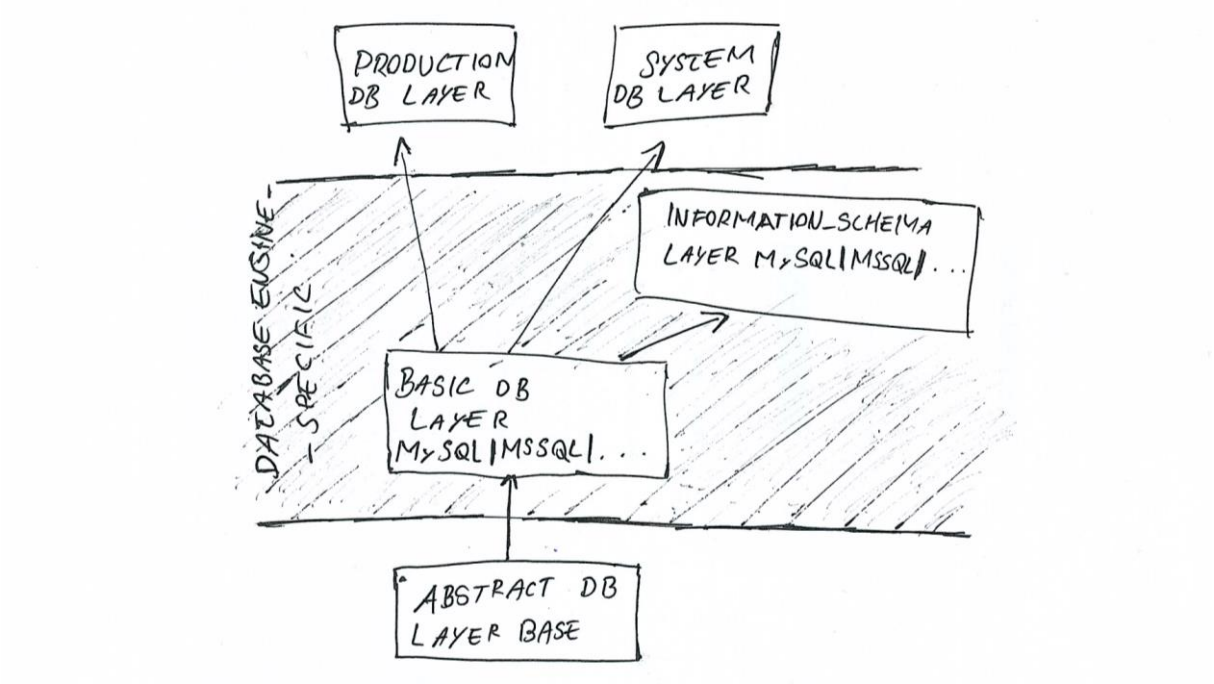
by vyžadovalo značné úsilie.

3.7.1 Základná databázová vrstva

Podľa pôvodného návrhu mala byť celá databázová vrstva špecifická podľa typu RDBMS. Väčšina komponent odpovedajúcich si komponent by sa na seba ale veľmi podobala. Systémová databáza sa skladá z rovnakých tabuliek bez ohľadu na databázový systém a podobne objekt pracujúci s produkčnou databázou používa vždy rovnakú logiku spracovania, prekrýva len syntaktické rozdiely. Objekt získavajúci štatistiky z `INFORMATION_SCHEMA` špecifický byť musí, nakoľko formát tejto databázy nie je u rôznych systémov totožný a je možné, že pri niektorých RDBMS sa `INFORMATION_SCHEMA` pre časť potrebných metadát nebude dať použiť a objekt bude musieť siahnuť po špecializovaných dotazoch a systémových tabuľkách daného databázového systému.

Pre prvé dva objekty sa ale oplatí vytvoriť ešte jednu tenkú databázovú vrstvu, ktorá ošetrí malé syntaktické rozdiely a, po vzore LINQ to SQL, spomínaného v úvode časti 3.6, uľahčí písanie databázových dotazov. Pre tvorbu dotazu bol zvolený návrhový vzor Factory. Výrobná trieda produkuje jednouché objekty zastupujúce vstupné hodnoty parametrov, stĺpce, tabuľky a niektoré tabuľkové operácie ako EQUIJOIN. V kombinácii s kľúčovými slovami SQL ako `SELECT` a `FROM` vzniká prehľadný a kompaktný zápis dotazu, ktorý základná databázová vrstva transformuje do podoby čistého textu SQL, pričom ošetrí tak syntaktické rozdiely ako aj parametrizáciu dotazu, aby sa zamedzilo SQL injection. Komponenta tiež poskytuje niekoľko metód vykonávajúcich dotaz, ktoré sa líšia formou vráteného výsledku – tabuľka, riadok alebo jediná hodnota. Viac informácii k časti tejto vrstvy určenej pre MySQL možno nájsť na adrese https://github.com/rjankovic/MySQLNEaT .

S pomocou tejto základnej vrstvy možno obmedziť databázovo-špecifickú časť aplikácie na komponentu zbierajúcu metadáta a túto vrstvu samotnú. Prejdime si výslednú štruktúru databázovej vrstvy odspodu podľa obrázka 7. Podklad tvorí abstraktná databázová vrstva, ktorá zaisťuje pripojenie k databáze, vykonanie príkazu a vrátenie jeho výsledkov v želanej forme. Nestará sa ale o preklad sekvencie objektov dotazu na príkaz SQL. Ten implementujú jednotlivé základné (ale už nie abstraktné) databázové rozhrania pre zvolené databázové systémy. Každá z týchto komponent implementuje spoločné rozhranie, ktoré následne využívajú odvodené triedy, ktoré už môžu plnohodnotne využívať abstraktný prístup k stavbe databázového dotazu.



Obr. 7 Oddelenie komponent závislých na type použitého RDBMS pri návrhu databázovej vrstvy spôsobilo jej ďalší rozpad.

3.7.2 Vrstva nad INFORMATION_SCHEMA

Automatizovaný návrh vychádza z vlastností databázy získaných z `INFORMATION_SCHEMA`. Informácie o integritných obmedzeniach a dátových typoch sa tiež využijú pri validácii návrhu po jeho manuálnej úprave. Potrebné sú nasledovné údaje.

- Zoznam tabuliek v databáze – v základnej verzii sa nepočíta s editáciou pohľadov alebo iných databázových objektov než sú tabuľky.
- Primárny kľúč každej tabuľky, prípadne informácia o tom, že primárny kľúč chýba. Tento kľúč sa využije na vyhľadanie záznamu pri inicializácii editačného formulára a bude dôležitý pri navigácii medzi entitami, ako už bolo spomínané v časti 3.3. Pokiaľ je tento kľúč automaticky generovaný (použitím príznaku `AUTO_INCREMENT` v MySQL, resp. `IDENTITY` v SQL Server alebo sekvenčných objektov), nemala by byť umožnená priama editácia tohto stĺpca či stĺpcov.
- Referenčné väzby medzi tabuľkami (viď 3.4).
- Dátové typy vrátane príznaku `NOT NULL` sa využijú pri výbere typu editačného poľa a príslušných validačných pravidiel.

Komponenta pristupujúca k `INFORMATION_SCHEMA` vykonáva spomedzi častí databázovej vrstvy najkomplexnejšie dotazy, ktoré sa odzrkadľujú na výkone. Pri návrhu editačného panelu pre ľubovoľnú tabuľku sú síce potrebné všetky z uvedených metadát, avšak tieto dáta sa len načítavajú a dá sa predpokladať, že sa produkčná databáza nebude meniť počas práce návrhára prostredia, respektíve zmeny budú tak zriedkavé, že užívateľa nebude obťažovať po zmene štruktúry databázy prostredie pregenerovať. Za týchto podmienok možno ukladať získané štatistické údaje v relácii

užívateľa webovej aplikácie, získať ich pre všetky tabuľky naraz pri prvom požiadavku a následne vracať dáta z relácie bez ďalších dotazov na `INFORMATION_SCHEMA`. Toto správanie je zapúzdrené v objekte databázovej vrstvy a nadradené vrstvy o ňom nepotrebujú vedieť. Tým sa vytvára priestor pre ďalšiu optimalizáciu lokálnymi zásahmi do tohto objektu.

3.7.3 Vrstva nad systémovou databázou

Táto kompoenta zodpovedá za správu dát v systémovej databáze popísanej v stati 3.3.3.

Voľba serializačnej technológie

Podľa predbežnej analýze v časti 3.3.3 je potrebné zvoliť metódu, ktorou sa budú serializovať jednotlivé komponenty panelov po základnej dekompozícii. Pritom by sa použitá technológia mala adaptovať na menšie zmeny v štruktúre serializovaných objektov bez výraznejšieho zásahu do kódu a všetky objekty jedného druhu by mali byť serializované uniformne. To implikuje použitie niektorej z generických serializačných technológií dostupných pre platformu .NET. Framework už vo svojej základnej forme poskytuje prostriedky tak pre binárnu, ako aj XML serializáciu pomocou technológie DataContracts, ktorá sa stala finálnym riešením, ako je zdôvodnené nižšie.

K serializácii .NET objektov možno použiť aj niektoré z riešení tretích strán. Ako príklad spomeňme systém vyvinutý firmou Google, Protocol Buffers. Tento binárny serializačný formát je zaujímavý najmä svojím výkonom [Pb], ktorý je podľa niektorých meraní (podľa tvorcov technológie) lepší než u serializačných techník dodávaných Microsoftom.

Dôvodom, prečo bola napriek tomu zvolená serializácia pomocou WCF Data Contracts, je podpora dedičnosti a celkovo komplexnejších scenárov použitia. Serializovať sa totiž musí prakticky celý objektový model, v ktorom možno predpokladať použitie abstraktných bázových tried a viac úrovní dedičnosti medzi postupne sa rozširujúcimi objektmi. Príkladom je editačné pole zobrazené ako WYSIWYG textový editor pre rozsiahle textové dáta produkčnej databázy, ktorý je potomkom základného textového poľa, ktorý dedí od všeobecného editačného prvku (ktorý môže obsahovať popisok a niektoré validačné pravidlá).

Druhotný význam pri výbere metódy serializácie má formát serializovaných dát. Binárna serializácia je síce kompaktnejšia, ale serializované dáta sú človeku neprístupné, čo by mohlo skomplikovať ladenie serializačného procesu. Preto som preferoval serializáciu do XML, ktorú Protobuff tiež nepodporuje.

Výkon systémovej databázy

Pri serializácii je potrebné rozlišovať dva prípady. Prvým je pridávanie nového formulára do databázy, kedy sa musia vytvoriť väzby medzi novo pridávanými objektmi na základe identifikátorov vygenerovaných databázou. Je tiež nutné zaistiť zápis do databázy v správnom poradí, aby sa neporušila referenčná integrita. Druhým prípadom serializácie je aktualizácia štruktúry formulára alebo jeho časti, ktorá sa vykoná jednoduchým prepísaním jeho definície pri zachovaní identifikátora. Jednoduchšou alternatívou, ktorá by ale mala výrazný dopad na výkon aplikácie pri práci návrhára, je

prepísanie celého modelu novou verziou. V tomto prípade by sa funkcionalita „aktualizácie“ vôbec nemusela implementovať.

Efektivita práce so systémovou databázou je kľúčom k výkonu aplikácie pri bežnom používaní správcom produkčnej databázy. Pri každom presunutí sa na iný formulár je potrebné získať jeho objektovú reprezentáciu, naplniť ju dátami z produkčnej databázy a vykresliť formulárové prvky. Pritom objem dát popisujúcich stavbu formulára bude často väčší než množstvo produkčných dát v ňom zobrazené. Tieto dáta naviac nutno deserializovať z XML. Je nežiaduce, aby deserializácia prebiehala po každom odoslaní formulára na server. Namiesto toho sa zvolil podobný postup ako pri `INFORMATION_SCHEMA`. Pri prvom načítaní aplikácie sa vytvorí kompletný objektový model prostredia, ktorý sa uloží v relácii užívateľa. Komplikáciou oproti databáze metadát je možná zmena podoby prostredia, ktorú môže návrhár vykonať bez vedomia koncového užívateľa. Problém sa rieši uzamknutím administračného prostredia, pokiaľ je k projektu prihlásený návrhár. Po dokončení úprav sa musia zmeny u užívateľa prejaviť, ale to len v prípade, že skutočne nastali a návrhár si prostredie napríklad len neprezeral. Preto komponenta systémovej databázy po každej zmene návrhu inkrementuje v databáze jeho verziu. Pred použitím objektového modelu uloženého v relácii užívateľa sa skontroluje verzia tohto modelu oproti aktuálnej verzii v databáze a prípadne sa vykoná opätovné načítanie.

Aj keď sa deserializácia objektov vykoná len pri prvom načítaní stránky, nemala by trvať príliš dlho. Otázne je, či sa viac oplatí načítať najprv z databázy kompletný obsah tabuliek a potom záznam po zázname deserializovať formuláre alebo spustiť pre každý formulár alebo skupinu formulárov samostatné vlákno a vykonávať paralelnú deserializáciu menších častí. Podľa niekoľkých meraní sa prvá z možností javí cca dvakrát efektívnejšia. V tomto konkrétnom prípade bola totiž deserializácia výrazne rýchlejšia než získanie dát z databázy (a `DataContracts` sú pritom jedným z menej výkonných serializačných riešení, viď 3.6.1), hoci to rozhodne nemusí byť pravidlom.

3.7.4 Vrstva nad produkčnou databázou

Zdanlivo najjednoduchšiu úlohu má časť databázovej vrstvy pracujúca priamo s produkčnými dátami. Musí len získavať časti záznamov alebo tabuliek a napĺňať nimi objekty, ktorých deserializáciu zaistil vyššie popísaná systémová vrstva, a ukladať ich po tom, ako mu nadradená vrstva vráti tieto objekty s novým obsahom. Táto komponenta ale pracuje priamo s dátami klienta a preto naň treba klásť vyššie bezpečnostné nároky. Akúkoľvek akciu by mal vykonať len v prípade, že nedošlo k zmene verzie prostredia, ktorú generuje vrstva systémovej databázy. Pokiaľ užívateľ upravuje záznam, zmeny by nemali byť uložené, pokiaľ sa zistí, že záznam medzičasom už niekto upravil. Nemožno sa ani spoliehať na to, že dotazy spúšťané nad produkčnou databázou skončia bez chyby. Mohlo dôjsť k zmene databázovej štruktúry alebo v databáze môžu byť definované špeciálne integritné obmedzenia, ktoré nebolo možné reprezentovať dostupnými validačnými pravidlami a typmi editačných polí systému. V prípade chyby pri vykonávaní dotazu objekt chybu propaguje do vyššej aplikačnej vrstvy, ktorá ju zobrazí užívateľovi.

Väčšina úprav, ktoré táto komponenta vykonáva, sa síce týka len jedného záznamu, v niektorých špeciálnych prípadoch, akým je úprava vzťahu M:N reprezentovaného pomocnou tabuľkou, dochádza

k zmene viacerých záznamov a to dokonca vo viacerých tabuľkách. Takéto operácie musia prebiehať v tranzakčne, aby sa nenarušila integrita dát.

Ďalším problémom, s ktorým sa komponenta potýka, je získavanie dát pre prehľadové navigačné tabuľky, kde si užívateľ má možnosť vybrať, ktorý záznam chce editovať, prípadne ho zmazať. Pokiaľ takáto tabuľka obsahuje stĺpec cudzieho kľúča, je vhodné, ako bolo už spomenuté, aby sa na mieste stĺpca zobrazila hodnota reprezentujúca odkazovaný záznam, ktorá je obsahom niektorého stĺpca príslušnej tabuľky. Komponenta produkčnej databázy preto vykonáva databázový `JOIN`, prípadne niekoľko `JOIN`-ov. Musí sa zabezpečiť unikátnosť názvov stĺpcov vo výslednom dotaze a ich spätné mapovanie na stĺpce cudzích kľúčov.

Dodatočnou funckionalitou, ktorá uľahčuje prácu návrhára pri tvorbe prostredia, je generovanie náhodných dát, ktoré sa návrhárovi zobrazia tak, ako by sa koncovému užívateľovi pri danom modele prostredia zobrazovali produkčné dáta. Tieto náhodné dáta by mali byť formátom a rozsahom podobné prodkučným, k čomu sa využívajú vlastnosti editačných polí formulára.

Naviac, ako bolo spomínané v časti 3.7.3, musí táto vrstva pri vykonávaní príkazov počítať s ich možným zlyhaním následkom pokusu o narušenie integrity databázy. V tejto situácii komponenta chybu zachytí, v prípade, že ide o známu chybu, môže k nej pridať vlastný komentár, a posunie ju nadradenej vrstve (centrálnej riadiacej jednotke), ktorá zaistí jej zobrazenie vo formulári.

3.8 Objektová vrstva

V predchádzajúcej časti boli okrajovo popísané logické vzťahy objektov, za ktorých serializáciu a deserializáciu zodpovedá systémová databáza. Nižšie je zhrnutá povaha týchto objektov, ktoré sú centrálnym bodom aplikácie a priamym výsledkom návrhu editačného prostredia.

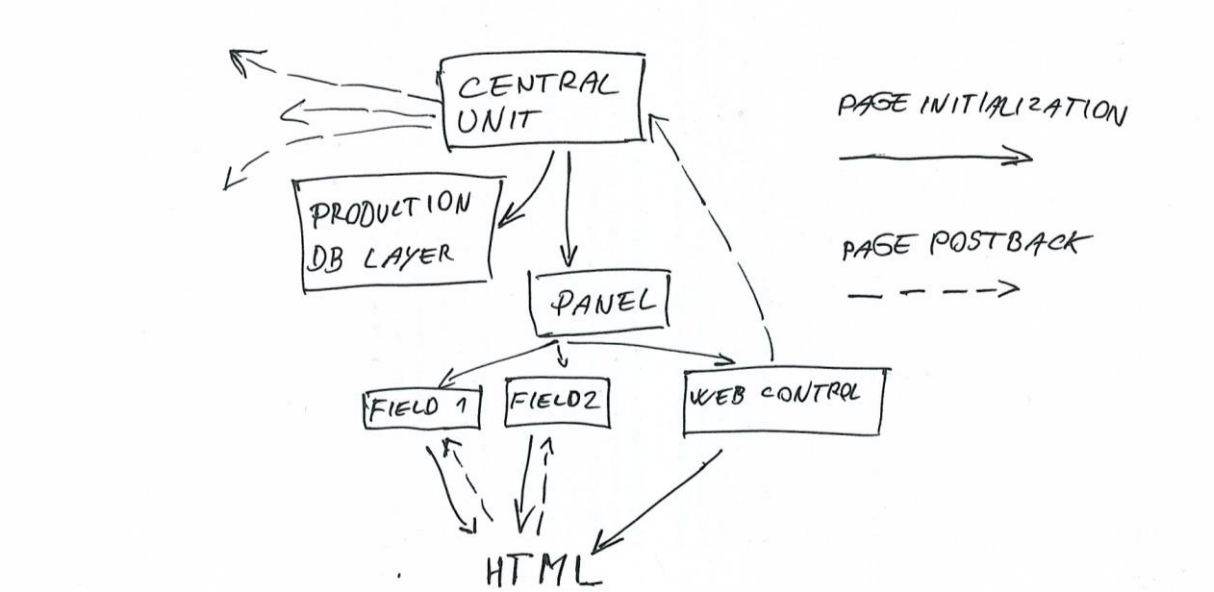
Každý projekt (spravovaná databáza) pozostáva z panelov, teda editačných obrazoviek, na ktorých užívateľ – administrátor prehliada alebo modifikuje dáta. Panely, ktorých úloha bola bližšie popísaná v stati 3.3, sú hierarchicky usporiadané, čo umožňuje vytvoriť obalový panel so základnými ovládacími prvkami a ako jeho potomkov zaradiť konkrétnejšie panely, ktoré predstavujú formuláre, skupiny navigačných ovládacích prvkov pre jednotlivé tabuľky, prípadne iné väčšie celky. Vo väčšine prípadov sa predpokladá existencia jedného primárneho panelu, ktorý bude obsahovať navigáciu viditeľnú počas celej doby práce užívateľa a bude obsahovať priestor pre vykreslenie podriadených panelov, ktoré sú už orientované na konkrétnu tabuľku produkčnej databázy. Pre jednu tabuľku budú potrebné minimálne dva panely, jeden pre navigáciu na záznam, ktorý chce užívateľ editovať, a druhý editačný, ktorý môže slúžiť zároveň pri editácii aj vkladaní nových záznamov. Prípadne by pre danú tabuľku mohli existovať ďalšie panely s jednoduchším rozhraním pre špecializované operácie nad záznamami (napríklad tabuľka s článkami blogu by mala plnohodnotný editačný panel na správu všetkých prístupných stĺpcov a zjednodušený na editáciu nadpisu a textu, čo bude väčšinový prípad). Štruktúra panelov je teda všeobecná, v prvej verzii programu však bola pre zjednodušenie plne implementovaná podpora len pre jeden hlavný panel a dva panely pre jednu tabuľku (navigačný a editačný).

Panel môže okrem iných panelov obsahovať editačné polia a ovládacie prvky. Ovládacím prvkom sa pritom rozumie akákoľvek súčasť, ktorá umožňuje užívateľskú interakciu a vyvoláva reakciu zo strany serveru. Typicky sú to tlačidlá a tabuľky s prehľadmi záznamov. Editačné polia sú naopak interaktívne súčasti, ktoré okamžité spracovanie nevyvolávajú. Oba druhy prvkov formulára sa môžu ďalej špecializovať. Editačné polia sa prirodzene špecializujú podľa dátového typu, ktorého editácii majú slúžiť. Tomu prispôsobujú nielen svoju grafickú stránku, ale najmä validačné pravidlá, ktoré zaručujú typovú korektnosť pred pokusom vložiť dáta do databázy.

Vysoká špecializácia navigačných prvkov je menej častým javom, tlačidlá pridania / úpravy / odobrania záznamu a tabuľka, obsahujúca niekoľko reprezentatívnych stĺpcov podkladovej databázovej tabuľky, umožňujúca zoradiť, prípadne filtrovať záznamy a presunúť sa do editačného panelu pre konkrétny záznam sú zväčša dostatočnými prostriedkami efektívnej manipulácie s dátami. Zvláštny prístup je však vhodný pre hierarchické dáta, kde efektívnejšie než vyhľadať želaný záznam v zozname zoradenom podľa obsahu vybraných stĺpcov je z pohľadu užívateľa vyhľadanie príslušného záznamu priechodom hierarchie do hĺbky.

Aby mohli riadiace objekty aplikácie s panelmi a ich časťami efektívne pracovať, musia tieto objekty vo svojej základnej a ktorejkoľvek špecializovanej podobe podporovať základné rozhrania podľa typu objektu. Panely zaručujú správne použitie záznamu vráteného databázovou vrstvou, ktorým naplnia svoje polia a tiež získanie užívateľských dát po odoslaní formulára a vytvorenie záznamu v rovnakom formáte ako predtým dostali na vstupe. Panely tiež určujú rámcové umiestnenie editačných polí a navigačných prvkov, t.j. najmä poradie týchto prvkov. Nemajú dosah na grafickú stránku. Editačné polia zabezpečujú obdobnú funkcionalitu na nižšej úrovni, vrátane grafického rozhrania. Získajú vstup od užívateľa, vykreslia sa na do HTML na miesto pridelené panelom, po odoslaní formulára sa aktivujú pred nadradeným panelom a získajú upravené dáta, ktoré zvalidujú, ohlásia prípadné chyby riadiacej jednotke (popísaná v 3.9) a poskytnú panelu nové hodnoty.

Popísaná propagácia dát z GUI do objektovej vrstvy sa vyvolá použitím ľubovoľného navigačného prvku. Tomu zostáva len úloha informovania riadiacej jednotky o vyvolanej akcii. Tá následne deleguje spracovanie databázovej vrstve, pokiaľ je požiadavkou úprava dát, respektíve navigačnej jednotke, pokiaľ sa užívateľ presúva na iný panel alebo iný záznam, alebo spracovanie zastaví, alebo pokiaľ došlo k chybe pri validácii alebo iným okolnostiam, ktoré zabránili vykonaniu operácie. Komunikáciu na úrovni objektovej vrstvy ilustruje obrázok 8.



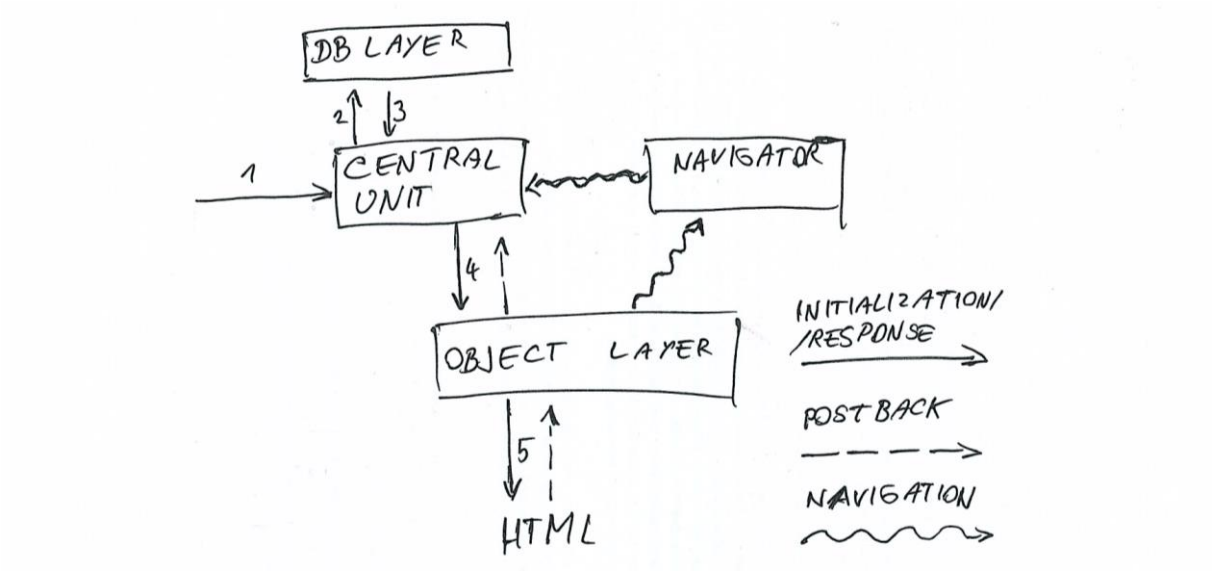
Obr. 8 DataFlow Pri spracovaní vykreslení stránky a následnom postback-u.

3.9 Centrálna riadiaca jednotka a navigácia

Zatiaľ čo panely sú jadrom návrhu prostredia pre konkrétnu databázu, aplikácia ako celok je riadená separátnou riadiacou jednotkou. Tá je prvou časťou programu spúšťanou pri každom požiadavku na server. V prvej fáze zistí, či je užívateľ prihlásený, overí jeho prístupové oprávnenia a prípadne mu odoprie prístup (viď 3.11). Následne vytvorí základné objekty aplikácie, ako sú triedy databázovej vrstvy a objekt automatizovaného návrhára prostredia (popísaný nižšie). Tiež zistí, či je možné použiť objekty panelov uložené v užívateľskej relácii (viď vrstvu systémovej databázy, 3.7.4) a prípadne vyžiada ich opätovné načítanie.

Druhá fáza činnosti riadiacej jednotky je špecifická podľa momentálne zobrazovaného panelu. Systém vyvolá generovanie GUI prvkov panelu delegovaním tejto úlohy na polia a navigačné súčasti panelu. Tie prípadne vykonajú ďalšie operácie popísané v predchádzajúcej časti. V poslednej fáze inicializácie sa udalosti navigačných prvkov napoja na navigačnú jednotku. Následne prebehne spustenie týchto udalostí. Pokiaľ užívateľ klikol na odkaz alebo tlačidlo smerujúce na iný panel, vyvolá navigačná jednotka presmerovanie, ďalšie spracovanie sa zastaví a celý proces sa opakuje na novej stránke. Pokiaľ užívateľ zvolil uloženie, prípadne zmazanie dát, odovzdá navigátor riadenie späť centrálnej jednotke, aby tá mohla vyvolať validáciu polí na serveri a v prípade úspešného zvalidovania celého panelu vyvolať príslušnú akciu v databázovej vrstve. Inak propaguje nájdené chyby do užívateľského rozhrania.

Spracovanie stránky končí po spracovaní všetkých udalostí generovaním HTML z objektov ASP.NET, na ktoré už ale aplikačná logika nemá vplyv. Celkový proces spracovania HTTP požiadavky je zachytený na obrázku 9.



Obr. 9 Zapojenie objektu navigátora do spracovania HTTP požiadavky.

3.10 Návrh prostredia

Návrh administračného prostredia prebieha v dvoch fázach. V prvej užívateľ využije údaje z `INFORMATION_SCHEMA` a niekoľko dodatočných informácií interaktívne zadaných užívateľom, aby vygeneroval iniciačný návrh. V druhej fáze je tento návrh prezentovaný užívateľovi podobne, ako bude neskôr zobrazovaný správcovi dát po nasadení informačného systému. Užívateľ tu môže formuláre ďalej upravovať a dopĺňať. Dôvod tohto rozdelenia je bližšie popísaný v časti 1.2.

3.10.1 Automatizovaný návrh

Umožnenie automatického generovania administračného prostredia pre ľubovoľnú relačnú databázu je jedným z hlavných cieľov projektu. Tento návrh vychádza z metadát, ktoré boli získané databázovou vrstvou (viď časť 3.7.3) zo zdrojov popísaných v 3.4. Od užívateľa sa naviac získa:

- Výber tabuliek, pre ktoré sa má návrh generovať, pričom je tento výber vopred obmedzený na tabuľky spĺňajúce základné predpoklady uvedené v časti 3.3.
- Reprezentatívny stĺpec každej zo zvolených tabuliek, t.j. stĺpec, ktorý sa bude objavovať v navigačných tabuľkách na prvom mieste. Komponenta pracujúca s metadátami sa predtým pokúsi zoradiť stĺpce podľa ich vhodnosti k reprezentácii tabuľky na základe dátových typov, toto poradie však môže byť nepresné.
- Špecifikácia začlenenia editácie mapovacích tabuliek do editačného panelu niektorej z mapovaných tabuliek (prípadne oboch alebo žiadnej). Potenciálne mapovacie tabuľky systém vopred detekuje.
- Podobne sa detekujú potenciálne hierarchické tabuľky a užívateľ môže zvoliť typ vygenerovanej navigácie pre ich navigačné panely (tabuľka / strom).

Návrh je pomerne jednoduchý, nakoľko získavanie rozsiahlych informácií o integritných obmedzeniach databázy by zaťažilo komponentu pracujúcu s `INFORMATION_SCHEMA` a ich následná analýza by z návrhu mohla spraviť ťažko odhadnuteľný black box, ako už bolo vyložené v stati 3.7.3. Najprv sa vygeneruje hlavný panel, ktorý obsahuje hierarchické menu s odkazmi na ostatné panely. Pre každú zo zvolených tabuliek sa vytvoria dva panely, jeden editačný a jeden navigačný.

Editačný panel obsahuje polia pre všetky stĺpce tabuľky, s výnimkou stĺpca s príznakom AUTO_INCREMENT, ktorého ručná editácia zväčša postráda zmysel a viaceré databázové systémy ju pri východzom nastavení neumožňujú.

Automatizovaný návrhár má k dispozícii sadu dostupných editačných polí, pričom každé pole predstavuje samostatný modul aplikácie (viď 3.12). Tento modul obsahuje „továreň na polia“ (návrhový vzor Factory), ktorá na základe informácií o type stĺpca rozhodne, či je ňou generovaný typ poľa vhodný pre editáciu tohto typu dát. Zároveň deklaruje mieru špecifickosti tohto poľa, aby sa mohol automatický návrhár rozhodnúť medzi viacerými použiteľnými typmi editačných polí. Napríklad pre editáciu dátumu sa dá použiť jednoduché textové pole, ale vhodnejšie je pole vo forme kalendára, ktoré má vyššiu špecifickosť a teda nie je použiteľné pre všeobecné textové dáta. Niektoré špecializované polia potrebujú viac informácií než typ jedného stĺpca. Príkladom je pole cudzieho kľúča, ktoré validuje svoje hodnoty oproti hodnotám referenčnej tabuľky. Modulárna funkčnosť by sa v tomto prípade dosahovala obtiažnejšie a preto tieto polia vytvára priamo návrhový automat. Doplnením panelu o ovládacie tlačidlá vloženia, zmazania a úpravy záznamu je dotvorený editačný panel tabuľky.

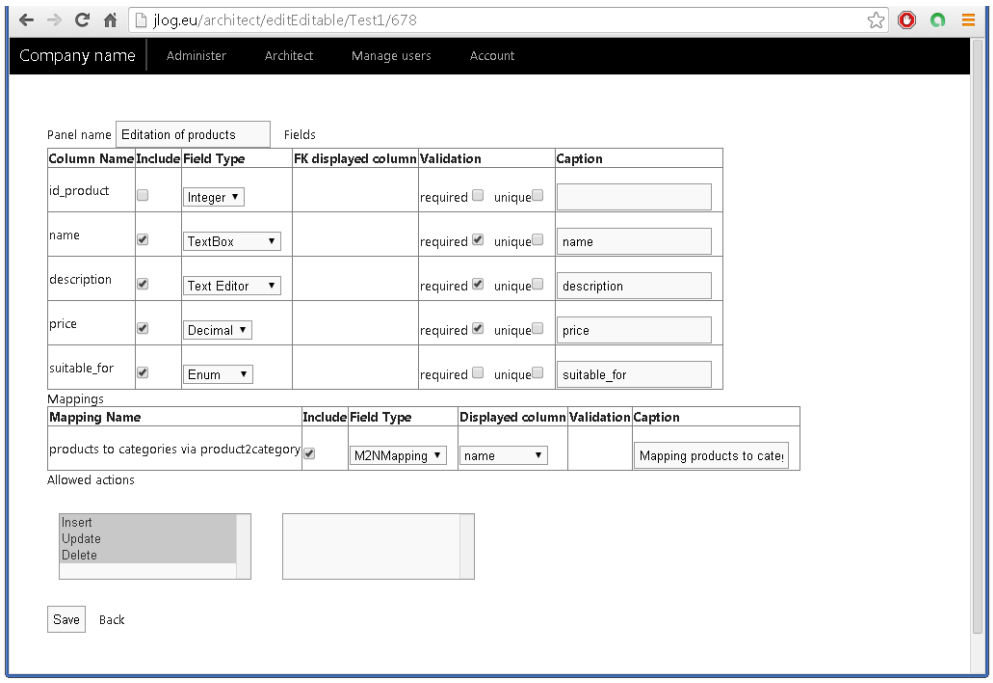
Navigačný panel pozostáva z jedného primárneho ovládacieho prvku, ktorým je buď navigačná tabuľka alebo navigačný strom. Popri štandardných editačných možnostiach (pridať / zmazať / upraviť záznam) a možnosti triedenia záznamov je zaujímavá a z pohľadu užívateľskej prístupnosti dôležitá voľba zobrazovaných stĺpcov, na základe ktorých musí byť koncový užívateľ schopný rýchlo identifikovať hľadaný záznam. Na prvom mieste sa zobrazuje stĺpec zvolený užívateľom v prípravnej fáze. V prípade navigačného stromu je to z praktických dôvodov jediný zobrazovaný stĺpec, v tabuľke je možné zobraziť niekoľko ďalších stĺpcov záznamu (vodné sa ukazujú byť 3-4 stĺpce). Tieto stĺpce vyberá kompnenta štatistickej databázy, pričom uprednostňuje kratšie textové typy, ktoré môžu obsahovať názov entity. Môže to však byť aj adresa alebo telefónne číslo, čo pri hľadanie záznamu zrejme veľmi neuľahčí. Druhoradou možnosťou sú dlhšie texty (z ktorých sa použije len časť) a dátumy. V krajnom prípade možno použiť číselné údaje. Výber reprezentatívnych stĺpcov má užívateľ možnosť po vytvorení návrhu upraviť a predpokladá sa, že túto možnosť využije.

Návrhy panelov pre jednotlivé tabuľky sú nezávislé. To je výhodné, pokiaľ sa štruktúra databázy bude neskôr meniť. V tomto prípade môže užívateľ vyžiadať pregenerovanie časti návrhu pre konkrétnu tabuľku alebo pridanie novej tabuľky bez zmeny ostatných častí. Nepríjemné je, že k vytvoreniu tohto dodatočného návrhu sú potrebné informácie o celkovej štruktúre databázy (pravidlách referenčnej integrity a mapovacích tabuľkách odkazujúcich sa na novú tabuľku) a preto musí databázová vrstva nanovo získavať veľkú časť metadát.

3.10.2 Editácia zo strany návrhára

Po automatickom vytvorení návrhu sa návrhár môže po prostredí pohybovať podobne, ako neskôr koncový užívateľ, pričom sa formulárové prvky vypĺňajú zástupnými dátami generovanými jednotlivými poľami (viď 3.3). Nevyhnutná je možnosť pridávať a odoberať panely, pričom sa nový panel vždy najprv automaticky vygeneruje, ako bolo spomenuté vyššie. Panely sa generujú v pároch (editačný a navigačný) a pri každom z týchto typov sú možnosti a postup úprav odlišné.

Pri editačných paneloch možno pridávať a odstraňovať polia, meniť ich typ a niektoré vlastnosti (ako popisok vo formulári). Tieto úpravy môžu viesť na návrh panelu nekonzistentný so schémou databázy. Vhodným prístupom je preventína restrikcia možností návrhára, t.j. sprístupniť pre každý stĺpec tabuľky len použiteľné typy polí a nepovoliť odobranie editačného poľa, pokiaľ stĺpec ním editovaný nemá príznak NULL ani prednastavenú východziu hodnotu. Presne detekovať množinu prípustných úprav a obmedziť možnosti GUI na základe tohto skúmania ale môže byť príliš komplikované. Úpravy podmienené komplexnejšou logikou je preto vhodné návrhárovi umožniť a na chyby ho upozorniť až dodatočne. To je vhodné realizovať rozšírením systému automatického návrhára o funkciu validácie návrhu panelu, pretože práve tu sú koncentrované všetky potrebné informácie o metadátach a užívateľských preferenciách. Pozmenený panel sa odošle návrhovému automatu a ten ho buď schváli alebo informuje o zistených problémoch a vráti ho užívateľovi na prepracovanie.



Obr. 10 Príklad možností návrhára-korektora pri úprave vygenerovaného návrhu.

Úprava navigačných panelov spočíva najmä v určení reprezentatívnych stĺpcov, prípadne v obmedzení editačných možností (zakázanie mazania záznamov a pod.).

3.11 Bezpečnosť

Aplikáciu je potrebné aspoň na základnej úrovni zabezpečiť proti neoprávnenému prístupu a prípadným chybným úpravám vedúcim k strate dát. Pri webovej aplikácii je táto úloha na jednej strane zjednodušená tým, že aplikáciu je možné spravovať centrálne, oproti modelu samostatných klientských staníc s takzvanými tlstými klientmi, na druhej strane riziko zvyšuje otvorená povaha webového prostredia a tak, síce to nie je kľúčovým miestom aplikácie, je vhodné použiť prinajmenšom štandardné bezpečnostné prvky, väčšina z ktorých je dostupná v podobe systémových knižníc.

3.11.1 Prístupové práva

Užívatelia aplikácie sa delia do troch hlavných skupín – administrátori, architekti a správcovia systému. Administrátori majú najnižšie oprávnenie, ktoré im dovoľuje editovať dáta prostredníctvom hotového návrhu rozhrania. Predpokladá sa, že do tejto kategórie bude spadať väčšina užívateľov. V súčasnej verzii systému sa rola administrátora ďalej nešpecifikuje a povoľuje spravovať všetky panely projektu, pri väčšom množstve administrátorov produkčných dát však má význam odstupňovanie ich oprávnení. Dosiahlo by sa to pridaním príznaku niektorých panelov alebo ovládacích prvkov, ktorý by určoval minimálne požadované oprávnenie na prístup k tomuto objektu.

Vyššiu mieru oprávnenia predstavuje rola architekta, ktorý môže meniť jednotlivé návrhy. Hoci sú týmto intuitívne nadradení administrátorom, ich právomoci nemusia byť nadmnožinou práv administrátorov. Naopak, má zmysel nepovoliť návrhárom prostredia prístup k produkčným dátam. V tomto prípade vzrastá význam automaticky generovaných dát vypĺňajúcich polia pri pohybe návrhára po vytvorenom prostredí, nakoľko si reálne dáta prehliadnuť nemôže.

Najvyššie oprávnenia má správca systému, ktorý môže prideľovať oprávnenia iným užívateľom, pridávať a rušiť projekty spravované danou inštanciou systému. Pre zjednodušenie systému je takýto užívateľ s absolútnymi oprávneniami v každej inštancii jeden a vytvára sa pri inštalácii. Je prípadne možné oddeliť oprávnenia správy projektov a správy užívateľských práv, kde správa projektov zostáva najvyšším oprávnením, správca projektov prideľuje oprávnenia správcom užívateľov a tí spravujú architektov a administrátorov.

Väčšina oprávnení (okrem správcu projektov) môže byť udelená buď v rámci jedného projektu alebo globálne pre všetky projekty evidované v inštancii systému.

3.11.2 Paralelný prístup

Zamedzenie konfliktnému paralelnému prístupu užívateľov je tradične problematickou časťou návrhu aplikácií, pričom možno najviac pozornosti sa jeho riešeniu venuje práve v databázových systémoch. V danej aplikácii táto potreba vznikla až druhotne a preto bolo zvolené pomerne jednoduché riešenie, ktoré je zrejme restriktívnejšie, než je nevyhnutné.

Pri vstupe do návrhárskeho módu návrhár získa exkluzívny zámok pre celý projekt. Tento zámok zamedzí prístupu tak iných návrhárov, ako aj administrátorov s adekvátnymi oprávneniami. Pokiaľ administrátor v momente vstupu architekta do projektu pracuje s niektorým panelom, neumožní sa mu uloženie nových dát. Toto priamočiare riešenie vychádza z predpokladu, že k zmene návrhu, ktorý už bolo odovzdaný k užívaniu administrátorom, nebude dochádzať často a na takýto zásah budú zo strany dodávateľa software zrejme dopredu upozornení.

Riešenie paralelného prístupu viacerých administrátorov k jednému záznamu bolo popísané v časti 3.7.5.

3.12 Modulárnosť

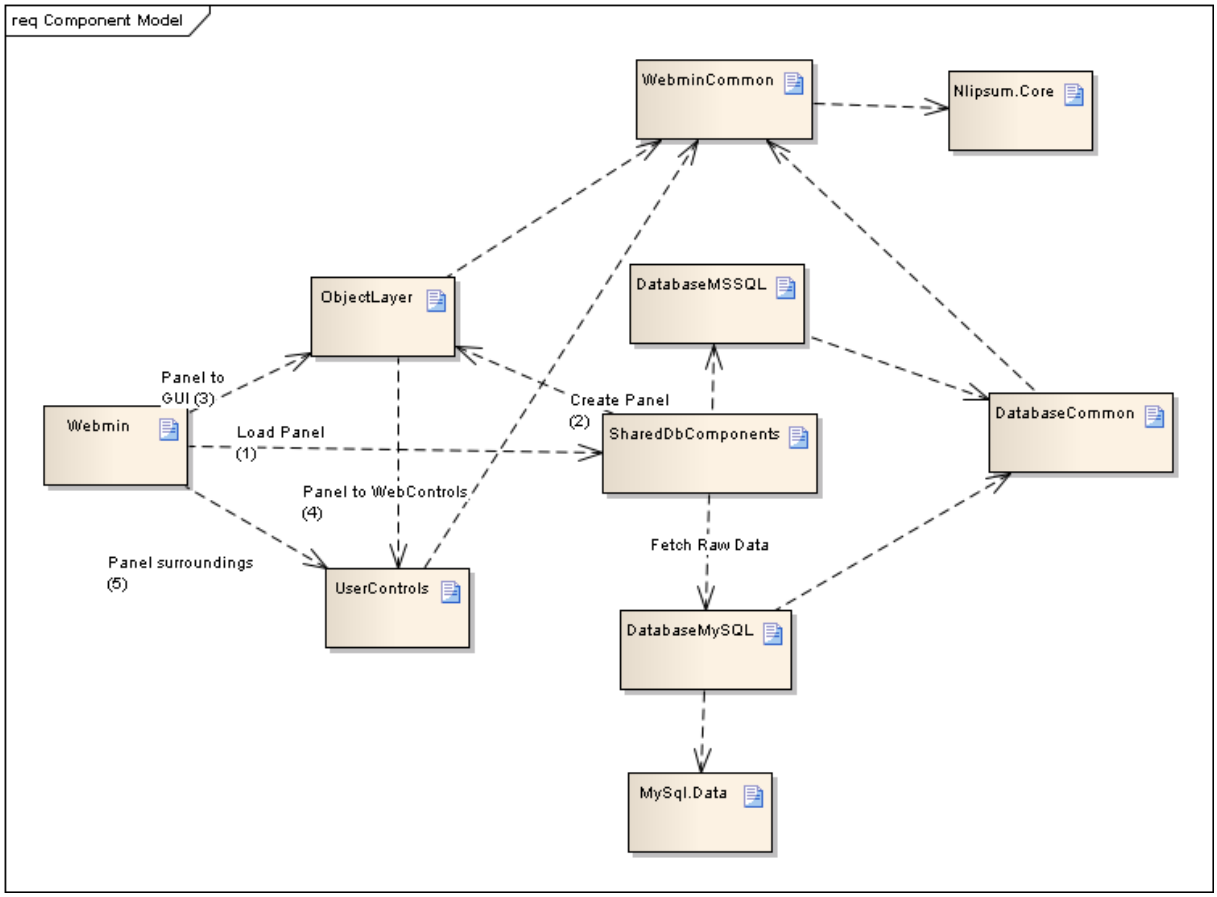
Jedným z cieľov aplikácie je poskytnúť podporu pre rôzne databázové systémy. Tento cieľ je rozobraný v časti 3.6. Ďalším smerom modulárneho prístupu je vytvorenie jednotného rozhrania editačných polí, ktoré, podľa 3.10.1, vystupujú ako samostatné jednotky schopné komunikovať s automatizovaným návrhárom, popísať svoju vhodnosť pre editáciu toho-ktorého stĺpca a následne plniť svoju funkciu v rámci panelu.

Ako moduly by potenciálne mohli vystupovať aj navigačné ovládacie prvky. Tieto moduly by mohli reagovať na súhrnné informácie o stĺpcoch tabuľky, detekovať špeciálne typy tabuliek a vytvoriť špecializované navigačné prostredie. Napríklad pre logovaciu tabuľku by mohol vzniknúť prehľadový výpis, ktorý by vykonával pomocou asynchrónnych volaní priebežné aktualizácie a podával tak online informácie o dianí v databáze. Iným príkladom je galéria obrázkov uložených v databáze alebo v súborovom systéme. Ničmenej, podobné moduly by museli v dohodnutom formáte informovať o rozsahu dát z príslušnej tabuľky, ktoré potrebujú k svojmu zobrazeniu. Situácia by sa ešte viac komplikovala, keby tento modul potreboval pristupovať k iným tabuľkám, napríklad aby získal údaje zo záznamov, na ktoré sa základná tabuľka odkazuje cudzím kľúčom, o iných zdrojoch, ako súborový systém v prípade spomínanej galérie obrázkov, nehovoriac. Je naviac otázne, koľko rôznych zmysluplných navigačných panelov sa môže vyskytnúť. Štandardná navigačná tabuľka napokon môže ľahko zobraziť väčšinu bežných dátových typov. Z týchto dôvodov sa od implementácie tejto časti upustilo a v prípade potreby špecializovaných navigačných častí je schodnejšou cestou ich zakomponovanie v jadre aplikácie. Už špecifikácia požiadaviek na implementáciu tohto typu modulu by totiž pravdepodobne odradila väčšinu ich možných autorov.

4 Vývojová dokumentácia

Implementácia webovej aplikácie vychádzajúcej z analýzy poskytnutej v tretej kapitole implementovaná pre platformu .NET v jazyku C# sa skladá z projektov (assembly), ktoré postupne prechádzajú od implementácie základných metód prístupu k databáze cez vrstvu objektov internej business logiky až k užívateľskému rozhraniu. Prejdime si tieto projekty v poradí odhora nadol. Každý z nich nájde čitateľ na priloženom CD v zložke `Source\{názovProjektu}\{názovProjektu}.csproj`. Cieľovým frameworkom riešenia je .NET 4.0 a súbor `Webmin.sln` je kompatibilný s VisualStudio 2010 aj 2012.

- **Webmin -** webové užívateľské rozhranie. Sú tu obsiahnuté všetky stránky ASP.NET (súbory .aspx) a k nim prislúchajúci kód na pozadí. Projekt využíva všetky nižšie vrstvy k získaniu podkladov pre návrh užívateľského prostredia, jeho uloženiu do databázy a následnému vykresleniu.
- **ObjectLayer -** obsahuje objekty predstavujúce komponenty užívateľského rozhrania. Tieto komponenty vychádzajú z návrhu v 3.7. Sú vytvárané webovou vrstvou alebo deserializované za použitia databázovej vrstvy. Sú tiež schopné poskytnúť vlastnú reprezentáciu ako objekty typu `System.Web.UI.Control` a spravovať obsah týchto objektov. Tým vytvárajú bloky, z ktorých sa skladá výsledné rozhranie.
- **UserControls -** ide o kolekciu špecializovaných užívateľských webových ovládacích prvkov, vytváraných v `ObjectLayer`. Poskytujú niektoré neštandardné funkcie ako napríklad roletové menu používajúce výhradne CSS alebo editor hierarchických vzťahov v tomto menu.
- **SharedDbComponents -** dve databázové komponenty, konkrétne pre systémovú a produkčnú databázu (viď 3.6.4 a 3.6.5), ktoré sú nezávislé na type RDBMS.
- **DatabaseMySQL** a **DatabaseMSSQL -** tieto assembly poskytujú implementáciu databázového rozhrania založenú na zdieľanej abstraktnej vrstve – implementujú rozhranie `IDbDeployableFactory`, ktoré zahladzuje jemné rozdiely medzi databázovými systémami. Naviac každý z týchto projektov obsahuje vlastnú komponentu pre prístup k `INFORMATION_SCHEMA` (viď 3.6.6).
- **DatabaseCommon -** obsahuje rozhrania implementované vyššími vrstvami pre MySQL a SQL Server a tiež základný abstraktnú databázovú vrstvu (viď 3.6.2).
- **Common -** obsahuje zdieľané konštanty, enumerácie, objekty `Project` a `User`, ktoré sú obálkami základných údajov o užívateľovi a projekte a ostatné vrstvy si ich medzi sebou posielajú. Je tu tiež niekoľko pomocných funkcií (ako napríklad konverzia textového reťazca na objekt typu `System.IO.Stream` a podobne).



Obr. 11 Graf závislostí projektov riešenia. Niektoré priame závislosti plynúce z tranzitivity sú kvôli prehľadnosti vynechané. Pri vybraných závislostiach je naznačený ich význam, čísla 1-5 vyjadrujú chronologickú náväznosť operácií pri spracovaní bežnej stránky.

Okrem uvedených projektov využíva riešenie ešte niekoľko komponent tretích strán.

- `MySql.Data` je implementácia ADO.NET pre MySQL. Táto assembly je súčasťou sady MySQL Connector .NET, ktorá je voľne dostupná na http://dev.mysql.com/downloads/connector/net/.
- `NLipsum` je generátor náhodného textu podobného Lorem Ipsum, voľne dostupný z https://code.google.com/p/nlipsum/.

Všetky tieto komponenty sú uložené na priloženom CD v zložke `External`.

4.1 Prehľad procesu

Analýzou dátového toku a spracovania HTTP požiadavky sa zaoberá stať 3.3, najmä v časti 3.3.2. Popíšme zbežne znova tento proces, odkazujúc sa už na konkrétne komponenty implementácie. Pomôckou pritom môže byť obrázok 11.

Uvažujme požiadavku na zobrazenie „bežnej“ stránky administračného rozhrania s použitím produkčných dát.

Spracovanie začína a končí v `MasterPage` ASP.NET, `MinMaster` ktorá je spoločná pre všetky stránky systému a obsahuje len kontajnery na dynamicky vytvárané rozhranie. Kód na pozadí tejto stránky overí, že užívateľ je prihlásený a má právo pristúpiť k požadovaným dátam. Tiež sa určí, o aký typ stránky ide (editačný panel, navigačná tabuľka, úprava panelu návrhárom a podobne) a na základe toho sa zvolí dcérska stránka, ktorá má spracovanie dokončiť. Pre potreby tejto stránky sa inicializujú základné objekty prostredia ako rozhrania jednotlivých databáz.

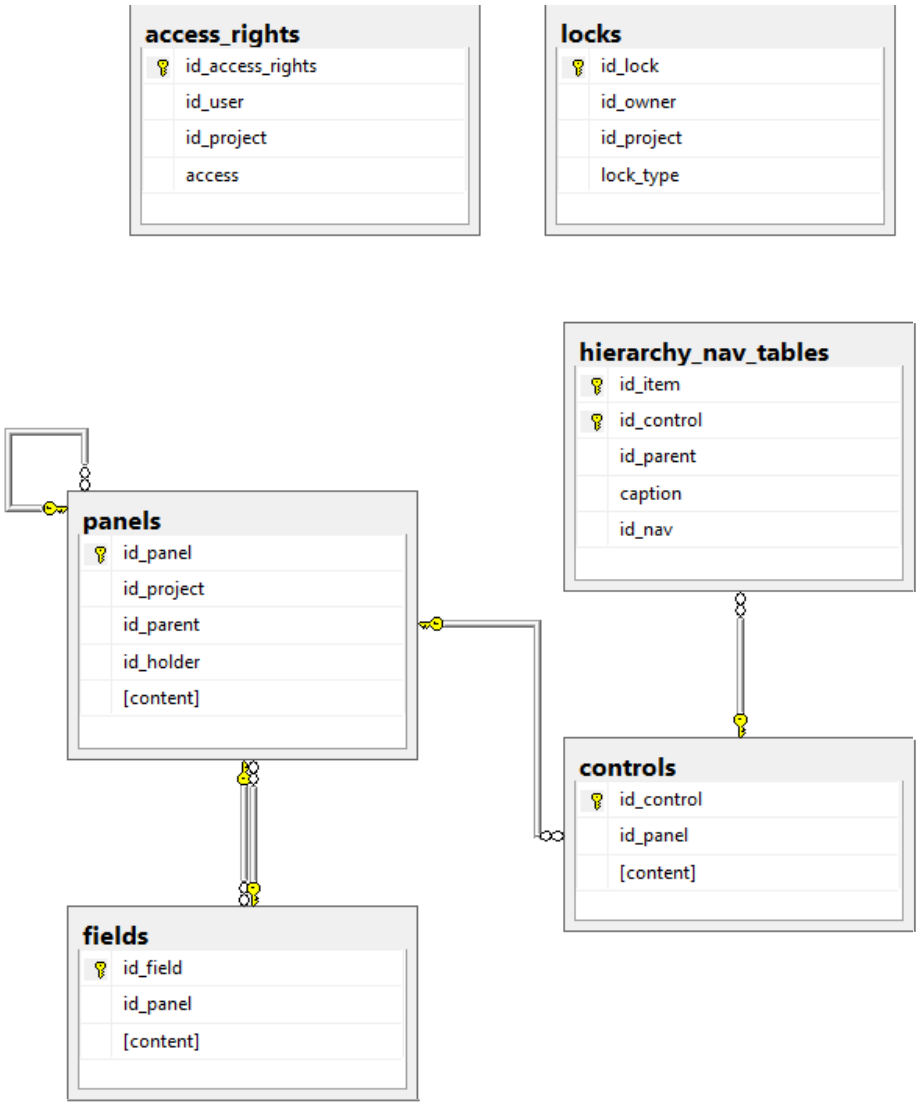
Po dokončení základnej inicializácie v uvažovanom prípade prevezme kontrolu stránka `Show`, ktorá zastrešuje pohyb po prostredí tak pre administrátora ako aj návrhára (nie však jeho úpravu). Táto trieda zistí, ktorý panel je aktívny a určí, či užívateľ zostal na tom istom paneli, z ktorého bola HTTP požiadavka vyvolaná, alebo sa presunul na iný. Tiež sa skontroluje verzia modelu prostredia v systémovej databáze. Ak je táto vyššia ako pri aktuálne načítanom modele, musí dôjsť k opätovnému načítaniu zo systémovej databázy, k čomu využije predtým inicializované databázové rozhranie, ktoré je definované v projekte `SharedDbComponents` (viď bod (1) na obrázku). Tým získa aktuálny panel ako entity objektovej vrstvy (2). Inak sa použije model uložený v užívateľskej relácii, čím sa zabráni opakovanej deserializácii tých istých dát.

Po získaní panelu ho musí stránka naplniť dátami, k čomu využije produkčnú databázu, resp. jej rozhranie implementované v `SharedDbComponents.dll`, ktorá zas použije základné databázové rozhranie pre príslušný RDBMS z `DatabaseXXX.dll`. Po vyplnení panelu sa musia panely a im podriadené objekty konvertovať do formy webových ovládacích prvkov ASP.NET (3). To vykoná každé pole a každý ovládací prvok (tlačidlá a tabuľky implementujúce `IControl`) samostatne (4). Kód na pozadí stránky `Show` tieto ovládacie prvky potom už len zostaví do jednoduchej tabuľky (5) dokončí spracovanie.

4.2 Štruktúra databázy

Štruktúra časti databázy použitej ako podklad výslednej implementácie je znázornená na obrázku 12. Tento diagram však nezahŕňa všetky tabuľky, ktoré sú potrebné k fungovaniu aplikácie. Sú vynechané tabuľky, procedúry a funkcie používané komponentou MembershipProvider a to najmä preto, že táto časť databázy bola automaticky vygenerovaná príslušným nástrojom či už pre MySQL alebo SQL Server. Naviac, práve v tejto časti sa odlišujú databázy použité pre tieto dva RDBMS. Tabuľky v uvedenom diagrame sú súčasťou oboch verzií a názvy a relačné vzťahy sa zhodujú.

Jedinou väzbou medzi tabuľkami z diagramu a ostatnou časťou databáz sú cudzie kľúče `id_user` a `id_owner` v tabuľkách `access_rights` a `locks`, ktoré odkazujú na primárny kľúč tabuľky `aspnet_users` (resp. `my_aspnet_users` vo verzii pre MySQL) užívateľov spravovanej príslušným poskytovateľom členstva ASP.NET.



Obr. 12 Základ systémovej databázy.

Kostru databázy tvoria tabuľky projektov (tabuľka `projects`), panelov (tabuľka `panels`), polí (tabuľka `fields`) a ovládacích prvkov (tabuľka `controls`). Každá z týchto tabuliek obsahuje okrem identifikátora a prípadného odkazu na nadradený prvok (viď 3.2) stĺpec content, v ktorom je uložený statický obsah tohto prvku (vytvorený vo fáze návrhu prostredia) serializovaný do XML. Zatiaľ čo tento model je pomerne priamočiary, niektoré väzby môžu pôsobiť prekvapivo. Jednou z nich je cudzí kľúč panelu odkazujúci sa na tú istú tabuľku – nadradený panel. Dôvodom je zámer umožniť vnorovanie panelov a vytváranie „kondenzovaného“ prostredia, kde má užívateľ možnosť pracovať s viacerými tabuľkami naraz (viď 6.3). S vnorovaním panelov súvisí aj cudzí kľúč „holder“ odkazujúci na špecializované pole, ktoré sa má stať hostiteľom panelu pri výslednom zobrazení. Tieto možnosti sa však v predkladanej implementácii využívajú len minimálne – každý projekt má jeden hlavný panel a ostatné panely sú jeho priamymi potomkami. Automatický návrhár ani manuálna úprava návrhu momentálne nedovoľuje zanoriť panely do nižších úrovní.

K tabuľke `controls` sa naviac viaže `hierarchy_nav_tables` – v tejto tabuľke sa uchovávajú uzly hierarchických menu používaných k navigácii medzi panelmi aktuálneho projektu. Údaje v týchto

tabuľkách sú po vytvorení návrhu statické, na rozdiel od iných tabuľkových alebo stromových navigačných ovládacích prvkov, ktoré svoje zdrojové dáta získavajú z produkčnej databázy. Alternatívnou k tejto tabuľke by bolo priame začlenenie navigačného stromu do XML dokumentu pri serializácii príslušného ovládacieho prvku. Predstava serializovanej dátovej tabuľky uloženej v bunke inej tabuľky sa však zdala byť pomerne scestná, už pri zvážení podmienok kladených prvou normálnou formou relačných databáz.

Tabuľka `access_rights` určuje prístupové oprávnenia užívateľov k projektom. Tieto práva však, ako je vyložené v 3.10, môžu byť priradené tak na úrovni konkrétneho projektu, ako aj na úrovni inštancie systému. Aby sa predišlo potrebe vytvárať dve obdobné tabuľky, boli využitá špeciálna hodnota `NULL`, ktorá je priradená stĺpcu `id_project`, ak sa príslušné oprávnenie vzťahuje na všetky projekty súčasne.

Tabuľka `locks` obsahuje zámky na úrovni projektu, ktoré získavajú návrhári prostredia pri editácii návrhu. Zamykanie zabraňuje konfliktným úpravám pri paralelnom prístupe viacerých užívateľov; bližšie informácie ohľadom zámkov sú uvedené v stati 3.10.

4.3 Triedy databázovej vrstvy

Databázová vrstva sa sama o sebe skladá z troch vrstiev – všeobecného základu v triede `BaseDbLayer`, nutnej špecializácie v projektoch **`DatabaseMSSQL`** a **`Databa`**

`seMySQL` a zdieľaných databázových komponent, opäť použiteľný pre všeobecný RDBMS, pre ktorý je implementované rozhranie `IDbDeployableFactory`.

`IDbDeployableFactory` vychádza z myšlienok v stati 3.6.2 a určuje metódy ako `Join(FK)`, alebo `InsVals(Dictionary<string, object>)`, ktoré vytvárajú objektovú reprezentáciu databázového EQUIJOIN-u na základe zadaného cudzieho kľúča alebo zoznam stĺpcov a ich hodnôt, ktoré sa majú použiť v dotaze `INSERT`. Základné DB komponenty pre konkrétne systémy implementujúce `IDbDeployableFactory` musia týmito metódami vytvárať vlastné objekty implementujúce prípadné všeobecné rozhranie (ktoré napríklad v prípade `IDbVals` umožňuje ďalej manipulovať s kolekciou hodnôt, ktorá sa neskôr použije pre `INSERT` alebo `UPDATE`), ktoré naviac príslušná databázová vrstva dokáže preložiť do kódu SQL. V prípade implementovaných databázových rozhraní tento preklad zaisťuje každá realizácia týchto rozhraní samostatne metódou `Deploy(SqlCommand)` alebo `Deploy(MySqlCommand)`, ktorá pridá príslušnú klauzulu do databázového dotazu.

`BaseDbLayerMySQL` a `BaseDbLayerMSSQL` vytvárajú databázové spojenie a spájajú výstupy prekladu jednotlivých častí do výsledného dotazu, ktorý potom posunú všeobecnej základnej databázovej vrstve (`BaseDbLayer` v `DatabaseCommon`), ktorý zaistí, že sa príkaz vykoná tranzakčne a prípadné výsledky vráti ako dátovú tabuľku, riadok alebo jedinú hodnotu.

Vďaka tomu dve z databázových komponent, ktoré používajú len objekty produkované IDbDeployableFactory spojené kľúčovými slovami ako WHERE a SELECT, môžu pracovať s rôznymi databázami rovnakým spôsobom.

To nie je prípad komponenty INFORMATION_SCHEMA, ktorý bol zvlášť implementovaný v DatabaseMSSQL a DatabaseMySQL oddelene. Síce tiež používa základnú DB komponentu na vykonanie dotazu a vrátenie výsledkov, ale nevyužíva továreň objektov. Dotazy diktuje priamo ako textové reťazce. Síce by aj v tomto prípade bola továreň použiteľná, zdieľanie tejto komponenty rôznymi databázovými vrstvami by však stále nebolo možné z dôvodov uvedených v 3.6.2. Z tohto dôvodu pre je túto časť ako pre jedinú vzniklo rozhranie IStatsDbLayer, ktoré určuje informácie, s ktorých dostupnosťou môže počítať návrhár pri tvorbe prostredia. Tieto informácie nezahŕňajú len jednotlivé tabuľky, ale aj ich previazanosť cudzími kľúčmi. Pri implementácii automatizovaného a manuálneho návrhára sa naviac ukázalo, že sú často potrebné údaje pre celú databázu, nielen pre jednotlivé tabuľky. Preto táto komponenta pri prvej požiadavke na vlastnosti stĺpcov tabuľky získa údaje pre celú databázu a pri následných dotazoch už len vracia časti tejto štruktúry uchovávanej v session. Výsledkom je pomalšia prvotná inicializácia, ale vyšší výkon pri dlhšej práci s modelom jedného projektu.

Trieda WebDbLayer implementuje vrstvu produkčnej databázy a vychádza z 3.6.5. Dodatočná komplexita spočíva v ošetrení niektorých špeciálnych stavov. Napríklad je nutné vytvoriť korektný dotaz pomocou objektov poskytovaných IDbDeployableFactory aj v prípade, že sa má v prehľadovej tabuľke v paneli použiť viacero cudzích kľúčov, pri čom treba predísť konfliktom v názvoch stĺpcov na jednej strane a vrátiť tabuľku vo forme požadovanej objektovou vrstvou na strane druhej.

Trieda SystemDbLayer odpovedá systémovej komponente z 3.6.4. Okrem správy všetkých trvácnych súčastí návrhu v databáze, ich serializácie, deserializácie a zachovania prepojenia medzi týmito objektmi pomocou cudzích kľúčov zodpovedá aj za správu užívateľov a ich oprávnení. V dôsledku toho je rozhranie ISystemDbLayer vôbec najobsiahlejším v celej implementácii.

4.4 Triedy objektovej vrstvy

Všetky triedy projektu ObjectLayer sú serializovateľné pomocou DataContracts. Serializujú sa vlastnosti nastavené návrhárom prostredia, na základe ktorých sa tieto objekty pri deserializácii opätovne vytvoria.

Základným prvkom tejto vrstvy je trieda Panel, ktorá obsahuje kolekcie editačných polí IField a ovládacích prvkov Control. Môže tiež obsahovať ďalšie vnorené panely, v súčasnej implementácii sa táto možnosť však využíva len pri základnom paneli projektu, ktorý obsahuje menu a priestor pre vykreslenie aktívneho dcérskeho panelu. Panel predstavuje editačný alebo navigačný formulár pre jednu tabuľku. Slúži ako prechodová vrstva medzi jednotlivými poliami, ovládacími prvkami a centrálnou riadiacou jednotkou (viď 3.9). Vyvoláva načítanie hodnôt polí z formulára alebo naopak ich naplnenie hodnotami z databázy. Panely určené pre editáciu konkrétneho záznamu naviac uchovávajú hodnotu jeho primárneho kľúča.

Objekty typu Control poskytujú metódu `ToUControl()`, ktorá vracia `System.Web.UI.Control.IField` má zas podobnú vlastnosť `MyControl` tohto typu. Týmto tieto objekty poskytujú realizáciu seba samých do užívateľského rozhrania.

Dôvodom rozdielnej implementácie je to, že polia musia „udržať kontakt“ s vytvorenými ovládacími prvkami, aby ich mohli lokalizovať v odoslanom formulári a načítať odoslanú hodnotu, ktorú bude vyžadovať nadradený panel. Z toho istého dôvodu sa poliam neprideľuje v HTML ID automaticky generované ASP.NET, ale statické ID obsahujúce identifikátor tohto poľa z databázy.

Zatiaľ čo každé pole svoju hodnotu z formulára načítava samo, ovládacie prvky nechávajú spracovanie udalostí na nich užívateľom vyvolaných na centrálnu jednotku (aby tá mohla vyvolať validáciu formulára, skontrolovať oprávnenia užívateľa a podobne). Je to zaistené tým, že metóda `UControl` preberá okrem iného handlery adekvátne udalostiam, ktoré môže vygenerovaný ovládací prvok vyvolať.

4.5 Prezentačná vrstva a návrhár prostredia

Prezentačná vrstva je z časti klasickou webovou stránkou. Ide o časť správy užívateľských účtov a základných vlastností projektov (ako sú pripojovacie reťazce k databázam) v menných priestoroch `_min.Web.Account` a `_min.Web.Sys`.

Zaujímavejšou časťou tejto, finálnej, vrstvy je centrálna riadiaca jednotka aplikácie implementovaná čiastočne v triedach `Web.MinMaster` a `Web.Show`. Prvá menovaná je vstupným bodom aplikácie. Preto zisťuje typ stránky, ku ktorej sa užívateľ snaží pristúpiť, na základe toho kontroluje užívateľské oprávnenia a vytvára všetky tri komponenty pre databázy aktuálneho projektu. S výnimkou prípadu, keď užívateľ pracuje s na manuálnom návrhu administračného prostredia, na úkony tejto triedy nadväzuje trieda `Web.Show`, ktorá má na starosti zobrazenie aktuálneho panelu v projekte a spracovanie prípadných udalostí vyvolaných jeho ovládacími prvkami. Keďže je model prostredia objemný, ukladá sa do session a `Web.Show` (ktorá je v skutočnosti kódom na pozadí `Show.aspx`) kontrouje, či je tento model ešte stále aktuálny alebo je potrebné znovu ho načítať s využitím systémovej databázy.

Ak užívateľ odoslal editačný formulár, prebehne validácia, získanie hodnôt z panelu a ich predloženie vrstve produkčnej databázy. Po spracovaní udalostí sa vyžiada vykreslenie polí a ovládacích prvkov do hlavného kontajnera aktívneho panelu, nastavia sa validaátory a stránka sa zobrazí.

Pri odoslaní navigačného formulára (t.j. po kliknutí na položku v ovládacom prvku prehľadového panelu) sa kontrola predáva objektu triedy `Navigator`, ktorá vykonáva väčšinu presmerovaní. (S výnimkou chybových stavov). Pri načítaní novej stránky sa toto detekuje a `Web.Show` získa z produkčnej databázy dáta, ktorými sa naplnia polia panelu.

Stránky v mennom priestore `_min.Web.Architect` sa používajú pri manuálnej úprave modelu. Často využívajú služby komponenty `INFORMATION_SCHEMA`, aby sa overila korektnosť modelu voči

integritným obmedzeniam databázy a komponety systémovej databázy na uloženie návrhu po úprave.

Súčasťou projektu prostredia je aj automatizovaný návrhár (`_min.Architect.Architect`), ktorého hlavnou metódou je `Propose()`, ktorá vytvára prvotný návrh modelu pre celý projekt. Tá volá `ProposeForTable()`, ktorá vytvorí pre danú tabuľku dva panely = editačný s poľami pre všetky stĺpce (s výnimkou stĺpca s AutoIncrement) a navigačný, ktorý bude buď navigačnou tabuľkou (`NavTableControl`) alebo stromom v prípade hierarchických dát (`NavTreeControl`). Validácia modelu po úprave je tiež zaisťovaná triedou `Architect`, konkrétne metódou `CheckPanelProposal`. Podoba východzieho návrhu aj validácia sú vhodnými adeptmi na prepracovanie a rozšírenie v ďalších verziách systému.

4.6 Užívateľsky definované polia

Táto stať, podobne ako 4.7, slúži dvom účelom. Prvým je predstavenie implementácie konceptu, ktorý robí systém rozšíriteľným o nové prvky GUI, resp. o podporu ďalších databázových systémov. Druhým je podanie stručného návodu k vytvoreniu takého rozšírenia a upozornenie z možných úskalí pri jeho implementácii.

Do systému je možné pridávať moduly implementujúce nové druhy editačných polí, ktoré sú funkčne rovnocenné so základnými zabudovanými poľami, s výnimkou `FKField` a `M2NMappingField`, ktoré plnia špeciálne úlohy. Užívateľom vytvorené pole môže mať vlastný vzhľad definovaný na úrovni objektov z namespace `System.Web.Controls`, môže implementovať ľubovoľnú validáciu na strane serveru a vykonávať potrebné transformácie dát pred ich zobrazením, resp. odoslaním do databázy.

Pole je definované dvojicou tried C# implementujúcich stanovené rozhrania. Konkrétne je to samotný objekt poľa implementujúci `IColumnField`, ktorý zodpovedá za vykreslenie, naplnenie, validáciu a transformáciu dát z webového ovládacieho prvku a k nemu priradenej „továrenskej“ triedy implementujúcej `IColumnFieldFactory`, ktorá zodpovedá za vyhodnotenie, či tento typ poľa môže editovať daný typ stĺpca v tabuľke, vytvorenie objektu tohto poľa a prípadnú ďalšiu konfiguráciu, ako je popísané nižšie.

Ako ukážka tejto funkcionality bolo vytvorený projekt `ExternalDecimalField` uložený na priloženom CD, v ktorom je implementácia poľa vhodného na spravovanie stĺpcov obsahujúcich čísla s pevnou i pohyblivou desatinnou čiarkou. Implementáciu všetkých ostatných používaných polí možno nájsť v hlavnom projekte v `ObjectLayer/Field.cs`. V tomto súbore sa nachádza aj trieda `ColumnField`. Všetci potomtkovia tejto triedy sú rovnocenní s pluginami a mohli by na ne byť potenciálne prerobení. Keďže je táto trieda súčasťou assembly `ObjectLayer.dll`, musí ju projekt pluginu referencovať. Referencia tejto assembly je dostatočná pre vytvorenie základného poľa. Pre niektoré pluginy by mohli využiť existujúcu implementáciu špecializovaných ovládacích prvkov v `UserControls.dll`. Nová assembly, ktorá vznikla po preložení projektu pluginu, bola skopírovaná do zložky `bin/ColumnFields` hlavného projektu aplikácie. Táto zložka sa prehľadáva pri každom spustení aplikácie a nájdené triedy implementujúce príslušné rozhranie sú pridané do internej kolekcie dostupných typov polí.

4.6.1 IColumnField

Prejdime potrebné časti implementácie na príklade základného poľa určeného pre celočíselné hodnoty, `IntegerField`. Trieda musí dediť od `ColumField`, čo znamená, že bude spravovať jeden konkrétny stĺpec a nebude pritom využívať informácie o iných stĺpcoch alebo tabuľkách. Preto napríklad nie je možné týmto spôsobom implementovať vlastnú verziu `FKField` alebo `M2NMappingField`, ktoré závisia na dátach mimo jeden konkrétny stĺpec. Ďalej musí byť trieda poľa serializovateľná, aby mohla byť jednoducho uložená do internej databázy. Serializovať by mala všetky trvalé nastavenia, pochopiteľne však nie užívateľom vložené hodnoty. Serializáciu zaistí priradenie atribútov WCF k tomu určených, ako možno vidieť v kóde.

Keďže je trieda odvodená od `ColumnField`, ktorá je sama o sebe serializovateľná (a odvodená od FieldBase), musí `ColumnField` vedieť o odovodenom type a k tomu využíva funkciu `GetKnownTypes()`. Tá je implementovaná cez `Common.Environment.GetKnownTypes()`, ktorá jednoducho vráti všetky typy polí nájdené pri štarte aplikácie. Môže byť prekvapivé, že táto metóda je implementovaná nanovo v každom potomkovi `ColumnField`, keď sa jej funkcia v žiadnom smere neodlišuje od zdedenej verzie. Skutočne, pokiaľ sa bude nové pole používať iba v prostredí IIS, nie je táto implementácia potrebná. Vynechanie tohto kódu však môže spôsobiť problémy užívateľom Mono. Implementácia mod_mono dostupná v čase vývoja totiž dedičnosť metódy používanej v atribúte KnownTypes nedokázala korektne spracovať.

Východzí konštruktor ColumnField nastavuje názov stĺpca tabuľky, ktorý bude editovaný skrz toto pole, a popisok zobrazovaný vľavo od webového ovládacieho prvku vo formulári. Môžu sa v ňom vykonať úvodné nastavenia interných štruktúr prvku, ak sú potrebné.

Nižšie možno vidieť implementáciu konštruktoru a metódy GetKnownTypes pre IntegerField. Za povšimnutie stojí atribút `[DataContract]`. Tento musí trieda mať, aj keď k svojmu predkovi nepridáva žiadne nové serializované polia, aby bola rozpoznaná ako serializovateľná.

```
[DataContract]
    public class IntegerField : ColumnField {
        private int? value;
        private WC.TextBox myControl;


        public static IEnumerable<Type> GetKnownTypes()
        {
            return FieldHelper.GetKnownTypes();
        }

        public IntegerField(string columnName, string caption)
            : base(columnName, caption)
        { }
```

Vlastnosť `MyControl` obsahuje objekt webového ovládacieho prvku. Tento buď už existuje alebo sa vytvorí. Nastavenie statického ID vykonávané pri inicializácii prvku je potrebné, kvôli zachovaniu stavu zobrazenia dynamicky vytváraného ovládacieho prvku po odoslaní na server.

Metóda `RetrieveData` sa volá po odoslaní formulára užívateľom a pred validáciou. Malo by dôjsť k transformácii dát z webového prvku do interných štruktúr objektu.

Nižšie je uvedená implementácia vlastnosti `MyControl` a metódy `RetrieveData()` v príklade `IntegerField`. Vlastnosť `MyControl` ukazuje vhodný spôsob vygenerovania unikátneho a jednoduchého ID poľa. Tento formát názvu sa v rámci aplikácie nepoužíva pre žiadny iný druh ovládacích prvkov. Prípadné úpravy jadra aplikácie by túto vlastnosť mali zachovávať. RetrieveData() zas berie do úvahy možnosť, že dáta v textovom poli nepredstavujú číslo (môže to byť napríklad prázdny reťazec).

```
public override UControl MyControl
{
    get {
        if(myControl == null){
            myControl = new WC.TextBox();
            myControl.ID = "Field" + FieldId;
            myControl.ClientIDMode = System.Web.UI.ClientIDMode.Static;
        }
        return myControl;
    }
}

public override void RetrieveData()
{
    int parsed;
    if (Int32.TryParse(myControl.Text, out parsed))
        value = parsed;
}
```

Metóda `FillData` nastaví hodnotu ovládacieho prvku podľa aktuálnej internej hodnoty. Táto je vopred nastavená inou cez vlastnosť `Data` z nadradenej časti aplikácie.

Zaujímavá je metóda `InventData`. Túto metódu volá systém, keď generuje hodnotu poľa pre náhľad návrhára. Mala by vytvoriť náhodné ukážkové dáta podľa reprezentatívneho typu a uložiť ich do privátnej premennej `value`.

Vlastnosť `Data` je, ako už bolo spomenuté, používaná pri nastavovaní hodnôt polí podľa záznamu vráteného databázou a tiež pri opačnom procese, teda zbieraní dát užívateľa a tvorení databázového dotazu na ich uloženie. Ak je ovládací prvok nevyplnený alebo v ňom nie je žiadna prípustná hodnota, mala by táto vlastnosť mať hodnotu `null`.

Implementácia FillData() a InventData() pre IntegerField je priamočiara, pričom InventData() generuje náhodné číslo od 0 do 99 999. Vlastnosť `Data` kvôli odolnosti počíta s dvoma možnými reprezentácie null – tak v zmysle platformy .NET ako aj relačnej databázy.

```
public override void FillData()
{
    myControl.Text = value == null ? "" : value.ToString();
}

public override void InventData()
```

```
{
    value = (int)(DateTime.Now.Ticks % 100000);
}

public override object Data
{
    get
    {
        return value;
    }
    set
    {
        if (value == null || value == DBNull.Value) {
            this.value = null;
            return;
        }
        this.value = Int32.Parse(value.ToString());
    }
}
```

Metóda `Validate()` je volaná vždy po odoslaní formulára užívateľom. Úlohou poľa je nastaviť hodnotu `ErrorMessage` na vhodnú chybovú hlášku, ak obsahuje príslušný objekt `WebControl` neprípustnú hodnotu. Znakom úspešnej validácie je hodnota `null` vlastnosti `ErrorMessage`. Okrem vlastnej validácie formátu by sa mala zohľadňovať aj hodnota vlastnosti `Required`, ktorá je určená pri návrhu rozhrania editačného panelu zaškrtnutím položky „Required“ v stĺpci Validation tabuľky návrhára. Hodnota `Unique` by sa naopak zohľadňovať nemala. Kontrola unikátnosti je vykonávaná automaticky na inom mieste a to aj preto, že objekt poľa nemá priamy prístup k databáze, kde by musel unikátnosť hodnoty skontrolovať.

Poslednou metódou je `GetValidators`, ktorá vytvára zoznam validátorov. Každý z týchto validátorov je potom umiestnený na výslednú stránku a jeho chybové hlásenia sú zobrazované štandardným spôsobom. Tvorcovia nových polí by sa však nemali spoliehať na validáciu JavaScriptom a to už aj z toho dôvodu, že sú validátory v implementácii Mono pomerne nespoľahlivé (napríklad `RequiredFieldValidator` nikdy svoju funkciu nevykoná – ide o známy bug).

Nižšie je opäť uvedená implementácia týchto metód pre `IntegerField`, čím je implementácia tejto triedy dokončená. Zaujímavé však je, že zatiaľ čo `Validate()` kontroluje aj neprázdnosť poľa, pokiaľ je toto pole povinné, `GetValidators()` nevytvára obdobný validátor. Ten totiž vytvára už predok `ColumnField`, `FieldBase`. To môže, pretože má k dispozícii ID príslušného objektu `WebControl`, cez ktoré naň naviaže validátor. Ten už potom funguje oddelene od poľa a výsledok jeho spustenia je otázkou príslušnej implementácie ASP.NET (treba stále pamätať na bug spomínaný vyššie). Abstraktný predok poľa však nedokáže korektne vykonať internú validáciu typu „required“, pretože má k dispozícii len všeobecný objekt `WebControl` a hodnotu poľa v objekte `Data`. Ten ale môže mať hodnotu null aj v prípade, že je pole vyplnené, avšak jeho hodnota nie je číselná a neprešla parsovaním pri predchádzajúcom volaní `RetrieveData()`. Preto je interná validácia ponechaná plne na konkrétnej implementácii poľa.

```
using WC = System.Web.UI.WebControls;

…

        public override void Validate()
        {
            ErrorMessage = null;
            if(Required && String.IsNullOrEmpty(myControl.Text))
                ErrorMessage = "The field " + Caption + " is required";
            else if(!String.IsNullOrEmpty(myControl.Text) && (value == null))
                ErrorMessage = "The value of " + Caption + " must be an integer";
        }

        public override List<WC.BaseValidator> GetValidators()
        {
            List<WC.BaseValidator> res = base.GetValidators();
            WC.RegularExpressionValidator numVal =
                new WC.RegularExpressionValidator();
            numVal.ErrorMessage = "The field " + Caption + " must be an integer";
            numVal.ControlToValidate = myControl.ID;
            numVal.ValidationExpression = "[0-9]+";
            numVal.Display = WC.ValidatorDisplay.None;
            res.Add(numVal);
            return res;
        }
```

4.6.2 IColumnFieldFactory

Druhou triedou, ktorú treba pri programovaní nových polí implementovať, je továreň na tieto polia, založená na rozhraní `IColumnFieldFactory`. Prvým, ľahkým krokom je voľba názvu, pod ktorým bude tento typ poľa vystupovať pri editácii rozhrania editačného panelu návrhárom. Tento názov trieda zverejňuje vo vlastnosti `UIName`.

Ďalšou konfiguračnou vlastnosťou je `Specificity`. Táto vlastnosť určuje, nakoľko je daný typ poľa špecializovaný. Napríklad, rozsiahle textové dáta možno editovať v jednoduchom prvku `TextBox`, ale možno použiť aj zabudovaný textový editor. V prvotnom návrhu sa vyberie druhá možnosť, pretože továreň editoru deklaruje vyššiu špecifickosť. Týmto údajom sa riadi automatický návrhár a vyberá najšpecifickejšie z prípustných polí. Špecifickosť môže mať tri stupne – nízku, strednú a vysokú. Vysokú špecifickosť má napríklad `DateField`, pretože toto pole nemôže byť použité k editácii iného dátového typu než je dátum.

Továreň musí informovať systém o type poľa, ktoré produkuje. Toto zariadi jednoduchá vlastnosť `ProductionType`.

Metóda `CanHandle` rozhoduje, či môže byť daný stĺpec tabuľky spravovaný poľom vyrábaným touto továrňou. Väčšinou pôjde o jednoduchý test dátového typu, možno však využiť aj ďalšie vlastnosti štandardného objektu typu `DataColumn` ako dĺžku textu, unikátnosť a podobne. Nie je však zaručené, že všetky vlastnosti stĺpca budú správne nastavené, hoci sa databázová vrstva pri tvorbe tohto stĺpca dodatočne získava informácie zo štandardných pohľadov `INFORMATION_SCHEMA`.

Metóda `Create` vytvára samotný objekt poľa na základe interných nastavení triedy. Nemala by však tomuto objektu nastavovať jeho hodnotu a iné vlastnosti, ktoré závisia od vstupu užívateľa.

Je dôležité poznamenať, že trieda `IColumnFactory` musí tiež poskytovať bezparametrický konštruktor, aby sa mohli vytvárať jej inštancie.

Nižšie je výpis implementácie továrne na polia typu `IntegerField`. Táto implementácia je opäť priamočiara, až na metódu `CanHandle(),` ktorá špecifikuje, že `IntegerField` je použiteľné pre všetky celočíselné typy.

```
public class IntegerFieldfactory : IColumnFieldFactory
    {
        public string UIName
        {
            get
            {
                return "Integer";
            }
        }
        public FieldSpecificityLevel Specificity
        {
            get
            {
                return FieldSpecificityLevel.Low;
            }
        }
        public Type ProductionType
        {
            get
            {
                return typeof(IntegerField);
            }
        }
        public bool CanHandle(DataColumn column)
        {
            return column.DataType == typeof(long) || column.DataType == typeof(int)
                || column.DataType == typeof(short)
                || column.DataType == typeof(ulong) || column.DataType == typeof(uint)
                || column.DataType == typeof(ushort);
        }

        public ColumnField Create(DataColumn column)
        {
            return new IntegerField(column.ColumnName, column.ColumnName);
        }
    }
```

4.6.3 ICustomizableColumnFieldFactory

Nakoniec prejdime implementáciu pokročilej verzie továrne na polia. Pre konfiguráiu väčšiny typov polí je dostatočná základná konfiguračná tabuľka, ktorú návrhár vidí pri editácii panelu. Niektoré polia však môžu vyžadovať ďalšie nastavenia. Ak má pole prijímať len reťazce vyhovujúce istému regulárnemu výrazu, je možné pre každé takéto pole samostatný objekt `Field`. Druhou možnosťou je vytvoriť všeobecný objekt poľa validovaného podľa regulárneho výrazu, ktoré bude tento regulárny výraz prijímať v konštruktore a tento výraz sa bude serializovať do internej databázy s

ostatnými vlastnosťami, ako je názov poľa a vlastnosť `Required`. Nastavenie tohto regulárneho výrazu sa potom musí vykonávať vždy pri vytvorení poľa tohto typu popri ostatných štandardných nastaveniach, ako bolo popísané v časti 3.2.2 tejto dokumentácie. Iným, implementovaným, príkladom je ImageField, ktorého kód sa nachádza v súbore `ObjectLayer\ImageField.cs`. Implementácia poľa musí zabezpečiť funkcie súvisiace s uploadom obrázkov, preto je mierne zložitejšia. Zaujímavejšia však je trieda ImageFieldFactory, implementujúca `ICustomizableColumnFieldFactory`.

Táto trieda zodpovedá za zobrazenie a spracovanie formulára dodatočných nastavení (v tomto prípade výber zložky pre ukladanie obrázkov, zmenšenia nahrávaných obrázkov a toho, či sa majú dodatočne vytvárať miniatúry nahraných obrázkov). Všetky prvky formulára si uchováva v súkromných premenných podobne, ako to robí `ColumnField`.

Metóda `ShowForm` preberá ako argument webový ovládací prvok ASP.NET Panel, do ktorého má byť formulár vykreslený. Môže doň vložiť ľubovoľný webový ovládací prvok, v prípade `ImageField` je to tabuľka s popiskami a poľami formulára dodatočných nastavení.

Metóda `LoadProduct` sa volá len v prípade, že pole tohto typu už v editačnom paneli figuruje a majú sa na ňom vykonať dodatočné úpravy. Táto metóda je vhodným miestom na načítanie parametrov poľa do interných premenných továrne. Po jej zavolaní musia byť tiež vyplnené polia formulára dodatočných nastavení poľa pôvodnými hodnotami.

Poslednou metódou je `ValidateForm`, ktorá má podobnú sémantiku ako `Validate` pri `IColumnField`. Trieda musí byť schopná validovať formulár, ktorý sama vytvorila, a adekvátne upraviť hodnotu vlastnosti `ErrorMessage`, pričom za znak korektného vyplnenia formulára sa považuje hodnota null priradená tejto vlastnosti. Metóda `Create` vytvárajúca objekt poľa sa, pochopiteľne, volá len v prípade, že validácia prebehla úspešne a preto v nej už hodnoty vyplnené vo formulári netreba kontrolovať.

4.7 Rozšírenie o podporu ďalších RDBMS

Aby mohol byť systém rozšírený o podporu konkrétneho RDBMS, musí preň existovať nasledovné:

- Implementácia rozhrania ADO.NET.
- Implementácia `System.Web.Security.MembershipProvider`.

Na rozdiel od pridávania nových editačných polí, ktoré bolo popísané v 7. časti tejto dokumentácie, nie je možné rozširovať sadu popdporovaných RDBMS formou pluginov, ale je potrebné prekompilovať časť databázovej vrstvy. Základom je implementácia sady rozhraní požadovaných objektovou vrstvou. Následne je potrebné nové triedy pridať medzi dostupné databázové rozhrania rozlišované pri inicializácii databázových spojení.

Pri implementácii je vhodné nasledovať príklad jedného z už existujúcich databázových rozhraní, či už pre MySQL alebo SQL Server.

4.7.1 IDbDeployableFactory

IDbDeployableFactory reprezentuje továreň na SQL dotazy, ktorá definuje objekty zastupujúce štandardné časti SQL dotazov, ako klauzule JOIN a WHERE, ktorých zápis nie je závislý od konkrétneho dialektu SQL. V ďalšej časti implementujete prekladač týchto časti dotazov na výsledné textové reťazce SQL.

Pre odlíšenie novej implementácie je potrebné vytvoriť rozhranie, ktoré dedí od IQueryDeployable. Nazvime ho IQueryX. Toto rozhranie musí implementovať metódu Deploy(XCommand cmd, StringBuilder sb, ref int paramCount), kde XCommand je objekt SQL dotazu z implementácie ADO.NET pre cieľový RDBMS a sb je text dotazu, ku ktorému objekt pripojí svoj preklad do textu. Hlavným využitím objektu SQL dotazu pri preklade do textu je parametrizácia dotazu, aby sa predišlo SQL injection. Ak príslušný objekt generuje časť kódu, ktorá by pri nevhodnom nastavení parametrov mohla viesť na SQL injection, je hodné preklad parametrizovať a parametre pridajť do objektu dotazu. Počítadlo paramCount je pred začiatkom prekladu nastavené na 0. A je postupne predkladané každej prekladanej časti dotazu. Každý objekt, ktorý pridáva k dotazu nový parameter, ho môže využiť k odlíšeniu svojho parametru od ostatných, aby vo výsledku vznikol dotaz s parametrami param0, param1,... Alternatívou je využitie vlastnosti objektu dotazu.

Jednoduchý príklad prekladu časti dotazu do SQL je uvedený nižšie. Ide o implementáciu jedného stĺpca danej tabuľky pre SQL Server. Po tom, ako je tento objekt inicializovaný pomocou názvu tabuľky, stĺpca a prípadným aliasom, je pripravený na preklad. V momente volania Deploy() je už vytvorená inštancia SQL dotazu a tento dotaz môže byť čiastočne preložený. Objekt k preloženej časti dotazu pripojí text vo forme [t].[c] AS “a”, kde t, c a a sú názov tabuľky, stĺpca a alias, v tomto poradí. Ak alias alebo tabuľka nie sú definované, z prekladu sa vynechajú, pričom je zodpovednosťou tohto objektu, aby výsledok predstavoval vždy korektnú časť kódu príslušného dialektu SQL, v tomto prípade T-SQL.

Všimnime si, že táto trieda implementuje dve rozhrania – jedným z nich je rozlišovacie IMSSqlQuerzDeployable odvodené od IQueryDeployable, druhým IDbCol, ktoré požiadavky na reprezentáciu databázového stĺpca nezávisle na type RDBMS.

```
class Column : IDbCol, IMSSqlQueryDeployable
{
    public string table { get; set; }
    public string column { get; set; }
    public string alias { get; set; }

    public Column(string column)
    {
        this.column = column;
    }
    public Column(string column, string alias)
    {
        this.alias = alias;
```

```
            this.column = column;
        }
        public Column(string table, string column, string alias)
            : this(column, alias)
        {
            this.table = table;
        }

        public void Deploy(SqlCommand cmd, StringBuilder sb, ref int paramCount)
        {
            sb.Append(" "
                + (table == null ? "" : ("[" + table + "]."))
                + ("[" + column + "]")
                + (alias == null ? "" : (" AS [" + alias + "]")));
        }
    }
```

Implementácia metódy Deploy pre všetky základné časti dotazov, ktorých rozhrania sú definované v `DbLayerInterfaces.cs` v `Source\DatabaseCommon` na priloženom CD. Vo zvyšku implementácie `IDbDeployableFactory` možno vychádzať z `DbDeployableFactoryMSSQL` alebo `DbDeployableFactoryMySQL`.

4.7.2 `IBaseDbLayer`

Po implementácii továrne na časti dotazu a ich oddeleného prekladu môžeme pristúpiť k zloženiu a vykonaniu celých SQL dotazov. To je hlavná úloha tried implementujúcich `IBaseDbLayer`. Opäť sa doporučuje vychádzať z niektorej z existujúcich implementácií. Hlavnou metódou tohto rozhrania je `Translate`, ktorej parametrami je ľubovoľný počet objektov. Medzi týmito objektmi sa budú vyskytovať hlavne tri dátové typy.

- **Objekty implementujúce Vaše rozhranie IQueryX**. Tieto objekty sa dokážu sami korektne preložiť, a preklad by im mal byť ponechaný na ne. Vhodným dodatočným opatrením je obalenie výsledku medzerami, aby sa predišlo zbytočným syntaktickým chybám, ak prekladaný objekt nezaistil oddelenie svojho prekladu od predchádzajúcich častí dotazu.
- **Textové reťazce**. Môže ísť o kľúčové slová ako WHERE a SELECT alebo iné časti dotazu, ktoré si užívateľ praje vykonať v presne zadanej podobe. Preto by sa mali pridať k dotazu bez úprav.
- **Objekty typu ValueType**. Typicky pôjde o čísla, ktoré sa majú vyskytnúť v dotaze a neboli obalené do objektu implementujúceho `IQueryDeployable`. Jednoduchým riešením je pripojenie nového parametru k dotazu a nastavenie hodnoty tohto parametru na príslušný objekt. Tým sa interpretácia týchto hodnôt prevedie na príslušnú implementáciu ADO.NET.

Preklad iných typov objektov nie je k vytvoreniu funkčnej databázovej vrstvy nutný.

`IBaseDbLayer` tiež implementuje niekoľko pomocných funkcií, ako je zisťovanie posledného vloženého ID v stĺpci `AutoIncrement` či logovanie vykonaných dotazov.

4.7.3 Získavanie metadát

Možno najťažšou časťou implementácie novej databázovej vrstvy je získavanie metadát príslušnej databázy z `INFORMATION_SCHEMA`. Síce aplikácia požaduje len údaje, ktorých prítomnosť je daná štandardom ANSI, názvy tabuliek a stĺpcov tejto databázy sa môžu líšiť, dáta môžu byť rozdelené do viacerých tabuliek než určuje norma alebo naopak zlučované do veľkých prehľadov. Opäť sa odporúča vychádzať z niektorej, prípadne z oboch z existujúcich implementácií.

Vrstva nad `INFORMATION_SCHEMA` musí implementovať rozhranie `IStats`, ktoré požaduje tieto metódy, resp. vlastnosti:

- **ColumnTypes**. Vracia slovník, ktorý pre každú tabuľku v databáze obsahuje zoznam jej stĺpcov ako objekty `DataColumn`. V záujme presnosti automatizovaného návrhu je potrebné správne vyplniť vlastnosti `AllowDbNull` a `DefaultValue`.
- **ColumnsToDisplay**. Pre každú tabuľku vráti zoznam jej stĺpcov v poradí od stĺpca najvhodnejšieho k reprezentácii záznamu tabuľky. Vhodným reprezentatívnym stĺpcom, pod ktorým môže záznam vystupovať v prehľadových tabuľkách, je napríklad unikátny názov článku/užívateľa/tovaru. Použitý algoritmus by mal vychádzať z dátových typov a ďalších atribútov príslušných stĺpcov.
- **TwoColumnTables**. Zoznam tabuliek s dvoma stĺpcami. Používa sa ako pomôcka pri identifikácii mapovacích (asociačných) tabuliek.
- **TablesMissingPK**. Tabuľky postrádajúce primárny kľúč. Tieto budú vynechané z automatizovaného návrhu.
- **SelfRefFKs**. Cudzie kľúče odkazujúce na vlastnú tabuľku. Používajú sa pri identifikácii hierarchických tabuliek.
- **Mappings**. Zoznam detekovaných asociačných tabuliek v databáze.
- **PKs**. Primárne kľúče tabuliek – pre každú tabuľky s definovaným primárnym kľúčom bude slovník obsahovať zoznam stĺpcov primárneho kľúča v poradí danom databázou.
- **FKs**. Zoznam cudzích kľúčov pre každú tabuľku.
- **Tables**. Jednoduchý zoznam tabuliek v databáze.
- **UnsiutableTables**. Tabuľky, ktoré nemožno systémom spravovať na základe ich dátových typov – môže ísť napríklad o binárne dáta. Momentálne automatizovaný návrhár považuje textové pole za univerzálne, t.j. predpokladá, že sa hodnota všetkých spravovaných stĺpcov dá editovať ako text a po následnej konverzii uložiť do databázy. Ak tabuľa obsahuje stĺpec, pri ktorom takáto editácia nie je možná, mala by byť zaradená do tohto zoznamu.
- **SetDisplayPreferences**. Pred spustením automatizovaného návrhára má užívateľ možnosť ručne definovať preferované reprezentatívne stĺpce jednotlivých tabuliek. Po nastavení preferencií by tieto mali byť odzrkadlené v hodnote vlastnosti `ColumnsToDisplay` (človekom určená voľba má zrejme väčšiu prioritu než poradie určené algoritmicky na základe dátových typov).

4.7.4 „Registrácia“ novej databázovej vrstvy

Nakoniec treba nové objekty databázovej vrstvy prepojiť s nadradenou vrstvou, ktorá ich bude používať. Tento proces je v súčasnosti mierne nešikovný a zlepšenie modulárnosti databázovej vrstvy jedným z vhodných smerov ďalšieho rozvoja projektu.

V projekte `SharedDbComponents` nájdete triedy `WebDbLayer` a `SystemDbLayer`. Obe pre prístup k databázam nimi spravovaným používajú rozhranie implementované `IBaseDbLayer`. Ich konštruktory preberajú objekt tohto typu a podľa neho inicializujú príslušnú implementáciu `IDbDeployableFactory`. Do týchto konštruktorov je potrebné doplniť rozskok pre typ Vašej základnej vrstvy a vytvorenie inštancie vašej implementácie `IDbDeployableFactory`.

Je tiež potrebné vykonať obdobné rozšírenie metódy `Page_Init` v `Site.Master.cs` v projekte `Webmin` a rozšíriť enumeráciu `DbServer` definovanú v projekte `Common`.

Pokiaľ má byť nový typ databázy využitý ako systémová databáza, v ktorej bude aplikácia uchovávať modely administračných proetredí, treba vytvoriť nový záznam v sekcii `<connectionStrings>` v konfigurácii webu a, samozrejme vytvoriť na príslušnom serveri databázu so štruktúrou identickou so systémovými databázami použitými pre MySQL či SQL Server. Do tejto databázy sa musia doplniť tabuľky príslušného databázového Mebership Providera a tento provider musí byť tiež zaregistrovaný v sekcii <membership> vo `Web.config`.

5 Problémy pri implementácii

Veľká časť implementácie projektu predstavovala aplikovanie štandardných postupov vývoja webových aplikácií. Preto na tomto mieste nie je uvedená úplná implementačná dokumentácia, ale len výber niekoľkých zaujímavých problémov, ktoré vyvstali zo špecifických potrieb projektu, ktoré miestami nie sú v súlade s tradičným poňatím webu a preto im bola venovaná zvláštna pozornosť.

5.1 Dynamické vytváranie GUI

Štruktúra formulárov zobrazovaných administrátorovi ako celok závisí od schémy spravovanej databázy. Preto nebolo možné použiť štandardné stránky ASP.NET s explicitne určenou sadou ovládacích prvkov, a väčšina týchto objektov sa musela vytvárať dynamicky pri spracovaní každej HTTP požiadavky. Výkon aplikácie, ktorá svoje GUI vytvára vždy od základu, bol otázny, a preto sa hľadala cesta ako sa tomu vyhnúť. Ako možné riešenie sa javila komponenta Dynamic Placeholder vyvinutá Denisom Bauerom http://www.denisbauer.com/Home/DynamicControlsPlaceholder . Ako sa ale ukázalo, tento modul vytvorený v roku 2006 nedokázal korektne spracovať asynchrónne volania pomocou ASP.NET AJAX, ktoré aplikácia na niektorých miestach využívala. Preto sa prešlo k vytváraniu prvkov stránky vždy od základu. Efektivita tohto riešenia sa ukázala ako postačujúca, nastali však problémy s uchovávaním stavu zobrazenia ASP.NET, ktorý udržuje stav formulárových prvkov pri opakovanom načítaní tej istej stránky, pretože novovytvorené objekty sa pri opakovanom odoslaní stránky nenamapovali na svoje pôvodné ekvivalenty, aj keď mali rovnakú štruktúru ako predtým. Kľúčom riešenia bolo správne rozdelenie spracovania medzi fázy životného cyklu stránky - vytváranie objektov pred inicializáciou stavu zobrazenia (štandartná metóda stránky ASP.NET `Page_Init`), načítanie stavu zobrazenia tesne po jeho vytvorení (`Page_Load`) a ukladanie vytvoreného panelu do užívateľskej relácie v náväznej fáze `Page_LoadComplete`. Naviac boli potrebné niektoré konfiguračné úpravy, ako zrušenie cachovania stránky, ktoré spôsobovalo stratu dát pri kliknutí na tlačidlo „späť“ internetového prehliadača, a explicitné priraďovanie statických identifikátorov webových ovládacích prvkov namiesto automaticky generovaných identifikátorov ASP.NET, ktoré bránili korektnému mapovaniu, ako je spomenuté vyššie. [MSDN]

5.2 Ovládací prvok správy vzťahu relácií M:N

Prirodzeným spôsobom správy vzťahu M:N medzi dvoma tabuľkami z pohľadu užívateľského prostredia je zvoliť pri editácii záznamu jednej z týchto dvoch tabuliek sadu záznamov druhej tabuľky, s ktorými bude tento záznam vo vzťahu. Výber podmnožiny možno zaistiť skupinou zaškrtávacích polí, ich počet sa ale už pri pomerne malých tabuľkách nepríjemne prejaví na prehľadnosti celého formulára. Preto bola zvolená pomerne bežná alternatíva pozostávajúca z dvoch zoznamov – jeden pre záznamy, ktoré v danom vzťahu sú a druhý pre zvyšné záznamy relácie. Kliknutím možno položku presúvať z jedného zoznamu do druhého a tým efektívne spravovať daný vzťah. V zdrojovom kóde túto funkciu zabezpečuje trieda `M2NMappingField`.

Hlavným problémom pri vytváraní tohto prvku bol výkon aplikácie pri presune položky z jedného zoznamu do druhého. Táto funkcionalita sa v objektovom modele ASP.NET dá dosiahnuť pomerne jednoducho, ale vyžaduje to odoslanie stránky na server, vytvorenie a odoslanie odpovede, čo je pri tak jednoduchom úkone mimoriadne neefektívne a odozva stránky je príliš pomalá. Riešením je vykonať zmenu na strane klienta prostredníctvom JavaScript-u bez odosielania stránky na server. Tento postup je jednoduchý a efektívny, ale framework ASP.NET pri ňom detekuje potenciálny útok HTML Injection, keď v kóde HTML nájde nový element, ktorý vyhodnotí ako pochádzajúci z iného serveru alebo webstránky. Problém bol vyriešený odstráneným tejto kontroly pri spracovaní stránky. Ide o potenciálne bezpečnostné riziko a je žiaduca revízia, ktorá zaručí nevykonanie kontroly len v prípade daného ovládacieho prvku.

5.3 Podpora Linuxu operačných systémov: ModMono

ModMono je implementáciou .NET (zahŕňajúcou ASP). V čase vývoja bola dostupná vezia 2.8 podporujúca .NET 4.0, teda nie vtedy najnovšiu verziu 4.5. Aplikácia bola preto od začiatku vyvíjaná pre .NET 4.0. Dosiahnutie kompatibility s Mono-m si vyžiadalo množstvo úprav. Nedostatočná je zo strany Mono najmä podpora JavaScriptu, konkrétne ide o bug pri validácii formulárov na strane klienta a to už pri použití základného validačného nástroja, akým je `RequiredFieldValidator`. Mono síce vygeneruje validačnú funkciu, ale k jej volaniu nikdy nedôjde. Ide o bug známy už niekoľko rokov. Preto sa pri Mone nemožno na klientskú validáciu spoľahnúť a odoslané dáta validovať aj na serveri. Serverová validácia, samozrejme, prebieha aj pri funkčných javascriptových validátoroch v prostredí Windows, ale pre Linux systémy to znamená nutnosť posielať stránku na server aj prípadoch, keď by sa chyba dala zistiť už pred odoslaním. Pomerne pracným riešením by bola priama implementácia validačných funkcií v JavaScript-e, pravdepodobne za použitia knižnice jQuery. K tejto implementácii však nedošlo.

S obdobnými problémami sa autor potýkal pri ovládacom prvku Menu, ktorý je štandardom ASP.NET vytvárajúcim roletové menu. Mono menu síce vygenerovalo, ale k roletovému efektu nedochádzalo, a to ani po nastavení príslušného objektu na generovanie potrebného efektu len za pomoci CSS a HTML, bez použitia JavaScriptu. Krajným riešením, ku ktorému sa z núdze pristúpilo, bolo vytvorenie potomka triedy `Menu` a vlastná implementácia generovania HTML a CSS na základe techník použitých v niektorých vzorových implementácií z http://www.cssplay.co.uk/menus/ .

Nefunkčnosť dynamických ovládacích prvkov z balíčku `AJAX Control Toolkit` (http://www.asp.net/ajaxlibrary/ajaxcontroltoolkitsamplesite/) po predchádzajúcich zisteniach nebola prekvapením. Dôležité bolo najmä nájsť vhodnú náhradu za komponentu `HTMLEditorExtender`. Implementácií textového editoru založených na JavaScripte je však k dispozícii dostatok.

Nakoniec nutno spomenúť problém pri vytváraní GUI inštalácie, kde Mono nedokázalo opätovne načítať pozmenený konfiguračný súbor, čo pri testovaní na Windows nebol problém. Problémy pri vytváraní inštalátora sú bližšie popísané v nasledujúcej časti.

5.4 Inštalátor

Pôvodným zámerom bolo vytvorenie webového inštalačného sprievodcu, ktorý si od užívateľa vyžiada potrebné informácie (pripojovací reťazec systémovej databázy a meno a heslo základného užívateľa aplikácie, ktorému ako jedinému budú priradené všetky práva (viď 3.10.1)), následne vytvorí databázu a príslušný užívateľský účet, vykoná zmeny v konfiguračnom súbore aplikácie a presmeruje užívateľa na úvodnú stránku systému pripraveného na vloženie prvého projektu. Tento cieľ sa podarilo naplniť len pre platformu Windows, a to z nižšie popísaných dôvodov.

Ak má inštalátor vytvoriť prvého užívateľa a uložiť ho do databázy, potrebuje mať najprv správne nastavený objekt `MembershipProvider`, ktorý okrem iného obsahuje informácie o pripojení k databáze, v ktorej sú užívatelia uložení. Problémom je, že táto databáza v prvej fáze inštalácie ešte neexistuje a po jej vytvorení a upravení konfiguračného súboru preto treba znovu načítať konfiguráciu a tento objekt prenastaviť. Pokiaľ však nemá dôjsť k pádu stránky, musí sa toto udiať bez reštartu aplikačnej domény. Tento objekt je ale automaticky vytváraný infraštruktúrou .NET pri inicializácii stránky a samotná zmena konfiguračného súboru ani následné znovunačítanie stránky nevyvolajú jeho aktualizáciu, konkrétne aktualizáciu pripojovacieho reťazca. V prostredí Windows sa nastavenie inak neprístupného objektu dá vykonať, ale iba pomocou `Reflection`, čo určite nie je vhodná programátorská praktika. V prostredí Mono sa týmto, ani rôznymi inými prístupmi nepodarilo aktualizáciu objektu `MembershipProvider` docieliť, následkom čoho nie je možné vykonať obe časti inštalácie (vytvorenie databázy a užívateľa) za behu aplikačnej domény. Preto boli úkony importu databázy a nastavenia databázového pripojenia (čo predstavuje zmenu dvoch hodnôt v konfiguračnom súbore) ponechané na manuálny zásah užívateľa. Inštalácia pre Mono následne len vytvorí užívateľa so zvoleným menom a heslom.

6 Záver

Práca vychádzala z myšlienky využitia dostupných metadát databázy a vytvorenia CMS na mieru bez nutnosti opätovnej konfigurácie základných administračných panelov, ktorá je jednou z rutinných činností pri mnohých webových projektoch, zväčša nevyžaduje kreatívne riešenie a ako taká by mala byť vykonávaná automaticky.

Pretože databázové systémy štandardne poskytujú detailné informácie tak o štruktúre databáz, ako aj o dátach v nich uložených, vytvorenie systému, ktorý by túto myšlienku premietol do praktického použitia, nie je nereálne. Riešenie tejto úlohy však nie je jednoduchým cvičením. Mnohé problémy vyvstali už pri analýze a ešte podstatne viac sa ich ukázalo pri samotnej implementácii. Následkom toho bola aplikácia v niektorých miestach dodatočne zjednodušená, vždy však so snahou zachovať jej rozšíriteľnosť a umožniť tak ďalší vývoj, ktorý postupne odstráni nedostatky, doplní špecializované súčasti a spraví aplikáciu skutočne užitočnou.

6.1 Zhodnotenie naplnenia cieľov

Zopakujme ešte raz primárne ciele projektu, ktoré boli stanovené v prvej kapitole.

Pri automatizovanom návrhu dochádza k odvodeniu základných rysov konceptuálneho modelu – na základe cudzích kľúčov sa určia vzťahy medzi entitami, vyčlenia sa slabé entitné typy a na základe toho sa rozhodne o začlenení alebo vylúčení tabuľky z návrhu. Pri vytváraní editačného panelu pre konkrétnu tabuľku sa potom využívajú dátové typy stĺpcov a ich ďalšie vlastnosti, ako je napríklad príznak `NOT NULL`, aby sa určilo, či má byť tento stĺpec priamo editovaný alebo je jeho hodnota generovaná databázovým systémom a aby sa v prvom prípade k poľu naviazali vhodné validačné pravidlá. Pre editáciu väzieb 1:N a M:N boli vyvinuté samostatné typy polí.

Na druhej strane, týmto postupom rozhodne nie sú zúžitkované všetky informácie obsiahnuté v `INFORMATION_SCHEMA` a pri použití pokročilejších algoritmov RDI by entitný model mohol byť presnejší. Otázkou zostáva, nakoľko by toto rozšírenie prospelo použiteľnosti výslednej sady editačných panelov, ako bolo popísané v stati venovanej `INFORMATION_SCHEMA` v kapitole 3.

Formuláre, ktoré sa vygenerujú po získaní metadát z `INORMATION_SCHEMA`, majú vo výsledku celkom jednoduchú štruktúru – pre každý stĺpec tabuľky (s výnimkou stĺpcov spravovaných samotným databázovým enginom) sa vytvorí jedno editačné pole, ktorého užívateľské rozhrnie a validačné pravidlá sa určia podľa dátového typu príslušného stĺpca. Momentálne sú implementované polia pre editáciu numerických a textových dát, zahŕňajúce WYSIWYG textový editor s možnosťou formátovania. Tieto formuláre postačujú väčšine bežných operácií. Do aplikácie je tiež relatívne ľahko možné doplniť implementáciu ďalších, špecializovaných editačných polí ako pluginov implementujúcich spoločné rozhranie.

Požiadavka na možnosť upravovať návrh prostredia za behu aplikácie nepredstavuje výraznejšiu komplikáciu pri vývoji aplikácie. Je tomu tak vďaka použitiu webovej platformy, čím sa uľahčila implementácia centrálneho riadenia verzií návrhu a zamedzenia prístupu užívateľov k prostrediu v čase, keď prebiehajú zmeny. Ničmenej, pravidlá určujúce, za akých okolností môže užívateľ ešte použiť predošlú verziu návrhu a kedy už musí dôjsť k opätovnému načítaniu prostredia, by mohli byť menej restriktívne, prirodzene, za cenu komplexnejšej aplikačnej logiky.

Pokiaľ by sme chceli hodnotiť systém ako CMS, výsledok by do značnej miery závisel od potrieb užívateľa. Ak užívateľ potrebuje poskytnúť svojim zákazníkom špecializované užívateľské rozhranie a dostupné CMS ho v tomto limitujú, je tento projekt vhodným riešením. Je taktiež použiteľný, pokiaľ užívateľ nevyžaduje pokročilé redakčné nástroje ako napríklad správu vydaní, verzovanie článkov, detailné nastavenie užívateľských oprávnení, emailové notifikácie a podobne. Ak je užívateľ, naopak, zvyknutý na systémy ako Wordpress či Joomla, mnohé funkcie mu budú chýbať. Aplikácia však bola navrhnutá s ohľadom na rozšíriteľnosť a tieto špecializované funkcie môžu byť doplnené.

Rozšíriteľnosť o nové typy editačných polí už bola viackrát spomínaná. Pridanie podpory ďalších RDBMS vyžaduje implementáciu dvoch rozhraní (základného databázového rozhrania a komponenty pristupujúcej k `INFORMATION_SCHEMA`), pričom v oboch prípadoch sa dá vychádzať z existujúcej implementácie poskytnutej pre SQL Server a MySQL a v prvom prípade postačia minimálne úpravy zohľadňujúce syntax daného RDBMS. Toto rozšírenie však zatiaľ nie je možné vykonať formou pluginu, ale je nutný zásah, do exitujúceho kódu, hoci len na jednom mieste. Týmto rozšírením sa ďalej zaoberá časť 7.3.

Systém podporuje tak Windows ako aj Linux a to bez oddelenia implementácií pre tieto dva systémy. Inštalácia na Linuxe však vyžaduje väčšie úsilie. Najprv je potrebné na cieľový server nainštalovať Mono a niektoré kroky inštalácie samotnej webovej aplikácie, ktoré sa na Windows vykonávajú už vo webovom rozhraní, je potrebné na Linuxe vykonať ručne úpravou konfigurácie alebo spustením priložených SQL skriptov.

6.2 Porovnanie výsledku práce s podobnými produktmi

Podobným produktom sa primárne venuje časť 1.1. Návrh projektu od začiatku nesmeroval k vytvoreniu konkurenčného software jedného z týchto riešení, ale k poskytnutiu funkcionality, ktorú neposkytujú ani klasické CMS ani generátory CRUD panelov. Preto hľadanie podobných systémov nebolo jednoduché.

Možno najpodobnejšie zameranie má technológia ASP.NET Dynamic Data, ktorá sa z časti stala inšpiráciou pre tento projekt. Dynamic Data sú nepochybne vyspelejšie, užívateľsky prívetivejšie a funkčne bohatšie než predkladaný systém. Ničmenej, tento má oproti Dynamic Data najmä tieto dve výhody:

- Úprava prostredia v GUI za behu aplikácie. Takáto úprava v Dynamic Data nie je možná, pretože vo výsledku ide o generátor kódu, ktorý sa musí skompilovať.

- Multiplatformovosť. Síce by sa aj pri Dynamic Data dalo použiť Mono ako podkladová implementácia .NET pre Linux, ale funkčnosť výsledku je otázna, nakoľko sa pri vývoji ukázalo, že Mono chybne vykresľuje niektoré prvky, nespúšťa validačné funkcie na strane klienta a obsahuje niekoľko ďalších nedostatkov, ktoré síce môžu byť a možno už aj sú opravené v nových verziách frameworku, ale rozhodne sa nedá povedať, že by vývoj Dynamic Data prebiehal s ohľadom na kompatibilitu s Monom.

Ďalším frameworkom, ktorý sa v istých aspektoch podobá na tento projekt, je Symphony 2 vyvinutý pre PHP. Opäť síce ide o generátor zdrojového kódu, je ale zaujímavý prenositeľnosťou a spôsobom definície formulárov. Ich podobu určujú konfiguračné súbory, ktoré obsahujú mapovanie polí na stĺpce tabuiek, aplikované validačné pravidlá ich začlenenie do obrazoviek, priradenie použitej HTML šablóny a niekoľko ďalších nastavení. K editácií týchto súborov nie je potrebný žiadny komplexný nástroj (zatiaľ čo správa webstránky založenej na Dynamic Data bez použitia Visual Studia by bola značne sťažená). Po úprave konfigurácie formulára nie je potrebné aplikáciu prekompilovávať, čím sa Symphony 2 blíži možnosti úprav za behu. Dalo by sa povedať, že táto miera priamosti úprav bez použitia externých nástrojov je dostatočná, nakoľko úpravy v nasadenom prostredí nebudú časté, budú sa zrejme diať s vedomím užívateľa a mimo jeho pracovnú dobu. Ďalej, skúsený správca môže potrebné úpravy v konfiguračnom súbore vykonať rýchlejšie než pri použití webového GUI. Pri vývoji aplikácie sa oproti Symphony 2 uprednostnil iný aspekt, a síce zníženie nárokov na návrhára prostredia, ktorý nemusí byť programátorom a nemusí mať priamy prístup k serveru, kde aplikácia beží.

6.3 Ďalší vývoj

Aplikácia v súčasnosti podporuje len niektoré z najpoužívanejších databázových systémov, základné dátové typy a editačné prvky. Viaceré prvky, ktoré mali byť súčasťou systému, neboli implementované alebo bola implementovaný len ich podklad v databázovej vrstve. O možnosti Ďalšieho vývoja software teda nie je núdza. Uveďme aspoň niektoré z nich.

- **Vnorené panely.** Možnosť vnoriť panel do iného panelu je podporovaná na úrovni databázovej vrstvy. V súčasnosti sa však využíva len na vnorenie všetkých panelov do jedného hlavného (obsahujúceho menu projektu). Vnorovanie panelov by umožnilo prácu s viacerými tabuľkami súčasne, čo je často potrebné, pokiaľ medzi tabuľkami existujú logické závislosti.
- **Zohľadnenie „kultúry" pri automatizovanom návrhu.** Mnohé spoločnosti majú zažité konvencie pokiaľ ide o pomenovávanie databázových objektov. Automatický návrhár by mohol tieto (konfigurovateľné) konvencie využívať k presnejšiemu odlíšeniu tabuliek, ktoré má či nemá zahrnúť pri tvorbe prostredia a aké panely a polia pre ne vytvoriť.
- **Publikačný systém.** Systém v súčasnosti neumožňuje posunúť publikovanie obsahu na určenú hodinu, správu verzií alebo vyžiadanie kontroly vloženého obsahu povereným užívateľom pred jeho publikovaním. Toto sú však niektoré z bežných schopností CMS.
- **Pluginy databázovej vrstvy.** Časti databázovej vrstvy závislé na použitom RDBMS sú pomerne dobre vyčlenené, nie sú však oddelené do tej miery, aby sa podpora pre ďalšie databázové systémy dala získať pridaním pluginu tak, ako je to možné pri editačných

poliach. Riešenie by mohlo byť podobné analógiou pluginov polí – triedy implementujúce verejné rozhranie by na základe pripojovacieho reťazca určili, či sa používa databázový systém, s ktorým sú schopné pracovať a v prípade kladného výsledku by vrátili inštanciu príslušného objektu databázovej komponenty. Táto funkcionalita nebola implementovaná pre nedostatok času.

- **Pluginy samotné.** Prvým z týchto databázových pluginov by mohol byť plugin pre najpoužívanejší komerčný databázový systém, Oracle. Ďalšie pluginy editačných polí by mohli rozšíriť množinu podporovaných dátových typov, zlepšiť podporu multimediálneho obsahu alebo dodať jemnejšie validačné pravidlá (napríklad text validovaný regulárnym výrazom).

A Užívateľská dokumentácia

Táto dokumentácia zahŕňa inštaláciu, správu a použitie systému Webmin tak z pohľadu vývojára a administrátora, ako aj koncového užívateľa.

Kapitola A.1 pokrýva inštaláciu systému na nový server vrátane konfigurácie, ktorú je potrebné vykonať pred prvým spustením, prípravy podkladovej databázy (najmä pre MySQL) a riešení niektorých častých problémov.

Kapitoly A.2, A.5 a A.7 sú určené primárne pre administrátorov aplikácie. Pokrývajú pridávanie nových databáz, ktoré majú byť spravované pomocou Webmin-u, nastavenie užívateľských oprávnení a personalizáciu užívateľských účtov.

Kapitola A.3 sa venuje návrhu administračného prostredia, ktoré bude vykonávať vývojár alebo designér dodávateľa. Zahŕňa automatizované vygenerovanie prvotného prostredia pre daný projekt a možnosti jeho následných úprav a rozšírení.

Kapitola A.4 nahliada na už vytvorené administračné prostredie očami koncového používateľa – správcu produkčných dát. Popisuje pohyb po tomto prostredí, vkladanie a modifikácie dát.

Spomeňme tiež, že k systému Webmin možno pridávať užívateľsky vytvorené pluginy, ktoré môžu poskytovať špecializované editačné polia alebo umožňovať pripojenie aplikácie k zatiaľ nepodporovaným databázovým systémom. Návod na tvorbu týchto pluginov je obsiahnutý v kapitolách 4.6 a 4.7. K vytvoreniu pluginov je potrebná mierne pokročilá znalosť C#, prípadne príslušného RDBMS.

A.1 Inštalácia

Tento oddiel popisuje inštaláciu aplikácie na platforme Windows (IIS) pri použití databázy MS SQL Server, na Linuxe s použitím MySQL a prípadnú možnosť kombinovať IIS a MySQL. Popis nezahŕňa konfiguráciu webového serveru a nasadenie webstránky naň. V prípade Linuxu toto zahŕňa inštaláciu Mono, ktorá má rôzne varianty a závisí na distribúcii Linuxu. Je však dôležité, aby bola nainštalovaná novšia verzia s podporou minimálne .NET 4.0.

Zdrojové súbory aplikácie sa nachádzajú na priloženom CD v zložke `Source\Webmin`. Obsah tejto zložky treba skopírovať do koreňového adresára stránky na webovom serveri. Následné kroky sa líšia v závislosti na operačnom systéme serveru. Popis inštalácie pre Microsoft IIS možno nájsť v A.1.1, obdobný popis pre Linux zas v A.1.2.

A.1.1 IIS + MS SQL

Aplikácia bola testovaná na IIS 6/7 a MSSQL Server 2008/2012. Na aplikačný server ani databázový stroj teda nie sú kladené žiadne neštandardné nároky. Prvým krokom inštalácie pri použití IIS a MS SQL je príprava databázy. Na serveri treba vytvoriť novú prázdnu databázu, ktorá bude aplikácii slúžiť ako základná systémová databáza pre uloženie štruktúry administračného rozhrania pre databázy spravované týmto systémom a ďalšie potreby aplikácie. Taktiež treba vytvoriť užívateľa, ktorý bude vlastníkom tejto databázy (bude členom role db_owner) a bude môcť byť použitý systémom Webmin.

Uistite sa tiež, že aplikácia bude mať právo na zápis do súboru web.config v koreňovom adresári. Tento súbor bude upravovaný počas automatického konfiguračného procesu.

Pred inštaláciou otvorte tento súbor a uistite sa, že v časti „connectionStrings“ je zakomentovaný element s atribútom `name=”MySqlServer“`. Naopak, podobný element pre `”MsSqlServer“` zakomentovaný byť nesmie. Podobne upravte konfiguráciu poskytovateľov členstva `(providers)` v časti `”membership“`.

Po týchto krokoch je aplikácia pripravená na prvé spustenie. Pri ňom sa zobrazí formulár, v ktorom sa vyplní pripojovací reťazec k prázdnej databáze v štandardnom formáte a prihlasovacie údaje prvého užívateľa. Po kliknutí na „Configure Webmin“ sa prípadné nedostatky zobrazia pod formulárom, formumulár bude možné upraviť a znova odoslať. Po zadaní správnych údajov sa systém pripojí k databáze, v ktorej vytvorí potrebnú štruktúru a pridá prvého užívateľa podľa zadaných údajov a pridelí mu neobmedzené práva na správu všetkých častí systému. Na záver presmeruje užívateľa na úvodnú stránku s nápisom „Webmin“, kde sa môže prihlásiť a začať aplikáciu používať. Je však možné, že sa zmeny v konfigurácii neprejavia automaticky. V tomto prípade je potrebné po vytvorení prvého užívateľa reštartovať webový server.

A.1.2 Linux + MySQL

Pre Linux bola aplikácia testovaná na Mono 2.5 a je kompatibilná s MySQL 4.1+. Najprv je potrebné importovať štruktúru databázy z priloženého skriptu, ktorý sa nachádza v Dumps/MySQLSchema.sql. Druhým krokom je úprava súboru web.config v koreňovom adresári aplikácie. Vo vrchnej časti tohto súboru v časti appSettings treba zmeniť hodnotu kľúča FirstRunMono z False na True, hodnotu kľúča ServerType na MySql a v sekcii connectionStrings vložiť správny pripojovací reťazec v elemente MySqlServer nahradiť reťazec „MySql connection string“ korektným pripojovacím reťazcom k práve importovanej databáze (viď poznámku pre používateľov MySQL v časti 2 tohto manuálu). Ďalej treba nastaviť poskytovateľa členstva ASP.NET. V časti „membership“ treba zmeniť hodnotu atribútu „defaultMembershipProvider“ na „MySqlMembershipProvider“.

V konfigurácii zakomentujte pripojovací reťazec `MsSqlServer` a odkomentujte pripojovací reťazec `MySqlServer. Podobne upravte konfiguráciu poskytovateľov členstva.`

V tejto chvíli je aplikácia pripravená na prvé spustenie, pri ktorom si vyžiada prihlasovacie údaje prvého užívateľa, ktorý bude následne vytvorený a budú mu pridelené neobmedzené správcovské oprávnenia. Aby sa nové nastavenia prejavili, treba po nastavení užívateľa reštartovať webový server.

Je vhodné upozorniť, že kompilácia možno neprebehne bezchybne na prvýkrát, pravdepodobne sa zobrazí výnimka oznamujúca nenájdenie niektorých programových častí. Tento problém v čase implementácie riešilo opakované spustenie aplikácie vo webovom prehliadači (bez reštartu serveru).

A.2 Správa projektov

Projektom v tomto kontexte rozumieme jednu konkrétnu databázu, užívateľské prostredie k nej vytvorené a v širšom kontexte aj pridruženú časť INFORMATION_SCHEMA, z ktorej sa čerpajú informácie o spravovanej databáze. Projekty v rámci aplikácie spravuje jediný užívateľ a to ten, ktorý bol vytvorený pri prvom spustení aplikácie po inštalácii. Tento užívateľ nájde v hornom menu po prihlásení položku „Manage projects“, ktorá vedie na jednoduchý prehľad dostupných projektov. Tlačidlo „Insert“ pod výpisom umožňuje vložiť nový projekt a odkazy „Manage“ spravovať už existujúce projekty. Pri oboch operáciach sa zobrazí rovnaký formulár, v ktorom nutno zadať unikátny názov projektu. Formulár naviac poskytuje možnosť overiť platnosť a dostupnosť zadaného databázového spojenia. Využitie tejto funkcie sa doporučuje. Platné databázové spojenie však nie je nutnou podmienkou, pripúšťa sa vytvorenie projektu pre budúcu databázu. Vloženie nového projektu, resp. uloženie zmien v už existujúcom projekte, prípadne zrušenie projektu sa iniciuje tlačidlami v spodnej časti formulára.

Poznámka pre používateľov MySQL. MySQL môže mať pri spolupráci s ASP.NET potiaže so znakovou sadou. Preto by pripojovacie reťazce mali explicitne špecifikovať nastavenie „CharSet=utf8“. Ďalším možným problémom je parsovanie dátumov, ktorým MySQL, pokiaľ užívateľské nastavenia neurčujú inak, priraďuje východziu hodnotu „0000-00-00 00:00“. Toto nie je korektná hodnota dátumu v prostredí .NET. Databázové rozhranie MySQL však dokáže vykonať konverziu na prípustnú nulovú hodnotu dátumu. Toto správanie zaisťuje atribút „Allow Zero Datetime=true“ pripojovacieho reťazca. Tieto dva atribúty sa odporúča používať vo všetkých pripojovacích reťazcoch projektov, ako i v pripojovacom reťazci k systémovej databáze.

A.3 Rozhranie návrhára prostredia

Po vytvorení projektu sa jeho názov pridá do horného roletového menu pod položkami Architect aj Administer, avšak použiteľná je v tejto chvíli len položka Architect. Po zvolení príslušného projektu v tejto ponuke za predpokladu, že k projektu ešte neexistuje žiaden návrh administračného prostredia alebo bol tento návrh zo systémovej databázy odstránený, sa zobrazí sprievodca inicializáciou projektu. Cieľom je upozorniť na možné problémy pri návrhu prostredia a tiež zistiť prvotné preferencie užívateľa ohľadom podoby tohto návrhu v sémantickej rovine.

A.3.1 Detekované problémy

Prvý krok je sumarizáciou zistených potenciálnych nedostatkov. Detekované sú dva druhy závad – absencia primárneho kľúča v tabuľke a prítomnosť nespravovateľných dátových typov v jednotlivých tabuľkách.

V prvom prípade chýbajúci primárny kľúč okrem iného znemožňuje systému vytvoriť efektívne navigačné prvky na pohyb medzi záznamami tabuľky a preto táto tabuľka nebude môcť byť touto cestou spravovaná.

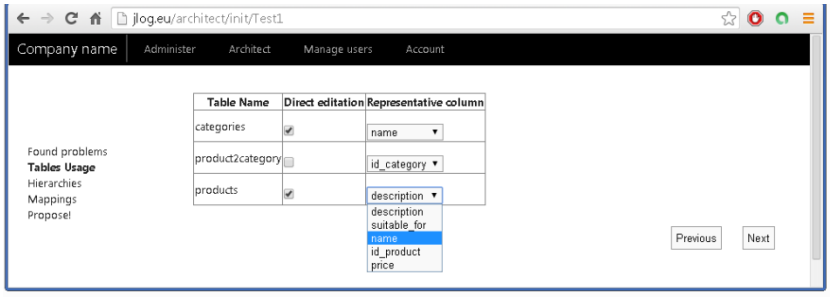
V druhom prípade tabuľka obsahuje binárne dáta, ktoré tiež nemôžu byť spravované, nakoľko nemožno jednoduchým spôsobom určiť, čo je obsahom stĺpca tabuľky a poskytnúť pre tento stĺpec vhodné editačné rozhranie alebo bol použitý niektorý z dátových typov, pre ktoré ešte v systéme nebol implementovaný vhodný formulárový prvok.

Ak nebude zistený žiadny z uvedených nedostatkov, zobrazí sa pozitívne ladená hláška „Seems OK...“. Ak sú tieto nedostatky ľahko odstrániteľné, možno tak učiniť a voľbou „Check again“ požiadať o opätovnú kontrolu stavu databázy.

A.3.1.2 Spravované tabuľky

V druhom kroku sa zobrazí súpis tých tabuliek databázy, ktoré neboli vyradené v prvom kroku. Pre každú tabuľku systém heuristicky stanoví, či je alebo nie je vhodná k priamej správe a prípadne zvolí reprezentatívny stĺpec tabuľky. Ukážka tohto výpisu je zachytená na obrázku 13.

Predpokladá sa, že tabuľky ktorých primárny kľúč je z časti alebo ako celok cudzím kľúčom alebo je pokrytý niekoľkými cudzími kľúčmi, nereprezentujú samostatné entity a ide len o pomocné štruktúry databázového modelu. Užívateľ má ničmenej možnosť tieto tabuľky medzi spravované zahrnúť zaškrtnutím príslušného políčka v stĺpci „Manage directly“ alebo naopak zahrnuté tabuľky zo správy vyňať.



Obr. 13 Oprava výsledkov analýzy pred vygenerovaním prvotného návrhu prostredia.

Z dôvodu ľahšej navigácie medzi záznamami tabuliek je vhodné zvoliť reprezentatívnu množinu stĺpcov, ktorá by bola dostatočne kompaktná a zároveň identifikovala záznamy tabuľky človeku prístupnou formou. Program preto zvolí z každej tabuľky niekoľko takýchto reprezentatívnych stĺpcov. Primárne preferuje kratšie textové polia. Číselné údaje majú naopak nízku prioritu. Táto

voľba ale nemusí byť optimálna a preto je v tomto kroku na užívateľovi, aby skontroloval a prípadne zmenil voľbu prvého z týchto stĺpcov. Voľbu vykoná v roletovom menu v riadku s názvom tabuľky, pričom prednastavená hodnota predstavuje výsledok dedukcie systému.

A.3.1.3 Hierarchie

V treťom kroku systém detekuje hierarchické dáta. Za hierarchickú sa považuje tabuľka, ktorá má primárny kľúč pozostávajúci z práve jedného stĺpca a niektorý z cudzích kľúčov v tejto tabuľke sa odkazuje na tento primárny kľúč. Zobrazený zoznam obsahuje tieto tabuľky a príslušné cudzie kľúče. Má len informatívnu funkciu a užívateľ nemá možnosť pridať inú hierarchiu, ktorá tieto podmienky nespĺňa. Po iniciačnej fáze návrhu však bude mať možnosť zmeniť štruktúru danej tabuľky a opakovať návrh pre túto tabuľku.

A.3.1.4 Správa vzťahu M:N

Častou potrebou pri návrhu databázy je reprezentácia vzťahu M:N, teda relácie medzi entitami typu A a B, kde každá entita A môže byť priradená niekoľkým entitám typu B a naopak. Zvyčajným riešením tohto problému v relačných databázach je vytvorenie pomocnej mapovacej tabuľky, ktorá obsahuje dvojice cudzích kľúčov odkazujúcich sa na záznamy tabuliek A a B. Správu tejto relácie možno vykonávať vo vyčlenenej ovládacej obrazovke, často je však vhodnejšie pričleniť ju k ovládaciemu panelu jednej z mapovaných tabuliek. Príkladom môže byť panel editácie tabuľky s výrobkami zahŕňajúci výber kategórií, do ktorých tento výrobok patrí, pričom vzťah medzi výrobkami a kategóriami je určený pomocnou tabuľkou.

V štvrtom kroku inicializácie návrhu sa aplikácia pokúsi takéto pomocné tabuľky detekovať. Za mapovaciu tabuľku sa považuje každá tabuľka o dvoch stĺpcoch, ktoré sú oba cudzími kľúčmi odkazujúcimi na rôzne tabuľky. Tieto tabuľky sa objavia vo výpise a užívateľ dostane možnosť zvoliť, ku ktorej z dvoch mapovaných tabuliek sa správa tejto relácie priradí. Môže prípadne zvoliť obe alebo žiadnu z nich.

Editačné prostredie pre zvolené tabuľky bude potom obsahovať špeciálny ovládací prvok, ktorý je bližšie popísaný v časti 3.7 tejto dokumentácie.

A.3.1.5 Pôvodný návrh

V tejto chvíli má Webmin všetky potrebné informácie k vygenerovaniu prvotného administračného rozhrania pre danú databázu. Po kliknutí na tlačidlo „Finish“ sa toto rozhranie vygeneruje a jeho presná podoba sa uloží do internej databázy. Užívateľ bude presmerovaný na úvodnú stránku editačného prostredia, kde môže návrh prezerať a ďalej dolaďovať. Bližší popis práce s návrhom je obsiahnutý v nasledujúcej stati.

A.3.2 Pohyb v návrhu

Po vytvorení prvotného návrhu prostredia môže návrhár vidieť rozhranie veľmi podobné tomu, ktoré budú po dokončení návrhu používať administrátori. Horná časť pod hlavným menu obsahuje menu

panelov (ovládacích obrazoviek) aktuálneho projektu. Položkám tohto menu boli priradené názvy spravovaných tabuliek a každá z týchto položiek obsahuje dve položky druhej úrovne s názvami „Browse“ a “Add”, ktoré smerujú na prehľad prvkov tabuľky a panel určený k pridávaniu nových záznamov, v tomto poradí. Kliknutie na samotný názov tabuľky je ekvivalentné s voľbou „Add“ pre túto tabuľku. Toto menu však možno ľubovoľne zmeniť a vytvoriť hlbšie členenie podľa logického zoskupenia entít príslušnej databázy.

A.3.2.1 Prehľadové panely

Prehľadové panely pre tabuľky obsahujú v móde návrhára náhodne vygenerované dáta, aby sa viac priblížili podobe, v akej s nimi budú pracovať administrátori a aby tak bolo ľahšie možné posúdiť vhodnosť zvolenej podoby týchto prehľadov.

Prehľadové panely sa vyskytujú v dvoch variantách – častejší prípadom je tabuľka s vybranými reprezentaívnymi stĺpcami (viď 3.1.2). Vedľa záznamov sú ovládacie tlačidlá „Edit“ a „Remove“. Prvé z nich vedie na panel editácie príslušného záznamu prostredníctvom formulára s poliami pre jednotlivé stĺpce tabuľky. V móde návrhára bude tento formulár prázdny, pri reálnom používaní administrátorom, pochopiteľne, predvyplnený hodnotami editovaného záznamu. Tlačidlo „Remove“ slúži k odstráneniu záznamu z tabuľky a „Insert“ k presunu na editačný formulár (pre návrhára je výsledok rovnaký ako po kliknutí na „Edit“). K žiadnym skutočným operáciám voči databáze, samozrejme, nedochádza.

Alternatívny spôsob navigácie predstavuje navigačný strom, ktorý je určený pre hierarchické tabuľky zvolené pri inicializácii návrhu. Každý uzol tohto stromu obsahuje hodnotu prvého reprezentatívneho stĺpca jedného záznamu. Po kliknutí na uzol sa užívateľ presunie do editačného panelu daného záznamu podobne ako pri tlačidle „Edit“ prehľadovej tabuľky. Naviac sa pod stromom nachádza tlačidlo „Insert“ funkciou totožné s tlačidlom pod prehľadovou tabuľkou.

A.3.2.2 Editačné panely

Editácia a vkladanie nových záznamov do tabuľky sa vykonávajú v jednotnom editačnom formulári. Polia formulára odpovedajú stĺpcom tabuľky, ich typ zodpovedá typu editovaných dát tak, ako to odvodil návrhový systém. Napríklad pre textové polia s dlhším obsahom bol zvolený vstavaný textový editor, pre stĺpce ktoré sú cudzími kľúčmi zas roletové menu, ktorého položkami sú prvky odkazovanej tabuľky zastúpené jej reprezentatívnym stĺpcom (v móde návrhára je roletová ponuka naplnená malou sadou náhodných dát).

V pôvodnom návrhu formulára sú zahrnuté všetky stĺpce s výnimkou stĺpca s atribútom Auto Increment, ktorého ručné editovanie sa nepredpokladá a nebude umožnené ani v rámci úprav formulára.

Korektnosť vložených dát je zaisťovaná validačnými pravidlami automaticky generovanými na základe štruktúry databázy. Je takto zaistené vyplnenie polí, ktorých stĺpce majú atribút NOT NULL (a nemajú na úrovni databázového systému určenú východziu hodnotu) alebo číselné hodnoty pre numerické stĺpce. Prípadné chybové hlášky uvidí návrhár nad formulárom pri kliknutí na „Insert“, prípadne

„Update“ pod formulárom. Editačný panel môže naviac obsahovať tlačidlo „Delete“ s intuitívnou funkciou. Ak odoslané hodnoty spĺňajú validačné kritériá, vráti sa užívateľ na prehľadový panel danej tabuľky.

Formuláre a ich funkcie sú v móde návrhára veľmi podobné formulárom, s ktorými bude pracovať administrátor.

A.3.3 Úprava menu

Užívateľ v roli architekta prostredia správy databázy môže po prihlásení a otvorení konkrétneho projektu vidieť pod navigačným menu tohto projektu dodatočný riadok obsahujúci ponuku nástrojov špecificky určených k úprave tohto prostredia. Prvý z týchto nástrojov je označený „Edit menu structure“ a slúži k úprave spomínaného menu projektu.

Ukážku editácie menu možno vidieť na obrázku 7. Pravú časť obrazovky správy menu zaberá zoznam panelov projektu, t.j. editačných formulárov a prehľadových výpisov spravovaných tabuliek. Východzie pomenovanie editačného a prehľadového panelu tabuľky X je „Editation of X“ a „Summary of X“. Toto pomenovanie však možno zmeniť, ako je popísané v nasledovných odstavcoch. Špeciálnou položkou v tomto zozname je „Main page“, ktorá zastupuje úvodnú obrazovku projektu.

V ľavej časti obrazovky je umiestnený strom menu (automaticky generovaná verzia obsahuje len dve úrovne, ako bolo popísané vyššie). Každému uzlu tohto stromu môže byť priradený jeden z panelov uvedených vpravo. Príslušný prvok menu bude potom viesť na zvolený panel. Toto priradenie je ale nepovinné. Prvky menu bez väzby na panel nevyvolajú pri kliknutí žiadnu akciu. Panely, na ktoré sa užívateľ-administrátor nemôže dostať skrz menu, sa nazývajú nedostupné. (Nedostupnosť panelu je jednou z podmienok odstránenia panelu z návrhu prostredia, ako je popísané v časti 3.4.)

Ak je prvok menu viazaný na niektorý panel, tento sa zvýrazní v zozname panelov po kliknutí na daný uzol stromu. Väzbu medzi menu a panelom možno vytvoriť označením uzlu stromu a zvoleného panelu v zozname a následným kliknutím na „Bind to Panel“ v spodnej časti stránky. Podobne „Unbind Panel“ ruší väzbu označeného uzlu stromu na panel. Ak nie je označený žiadny uzol alebo pred naviazaním panelu nie je označený žiadny z panelov, kliknutie na tieto tlačidlá nevyvolá žiadnu akciu. Ak bol uzol už viazaný na iný panel a užívateľ vytvorí väzbu na iný panel, pôvodná väzba sa automaticky zruší.

Premenovanie uzlu menu sa uskutoční označením tohto uzlu a zmenou názvu v textovom poli „Panel Name“.

Pridanie nového uzlu do stromu možno dosiahnuť vpísaním jeho názvu do toho istého editačného poľa a stlačením „Add node“. Nový uzol bude pridaný ak posledný priamy potomok práve zvoleného uzlu pôvodného stromu. Ak nie je v tomto strome označený žiadny uzol, bude nový uzol pridaný do prvej úrovne stromu.

Po označení uzlu a stlačení „Remove node“ bude tento odstránený z menu a prípadná väzba na panel bude automaticky zrušená. Taktiež budú rekurzívne zrušené všetky dcérske prvky stromu.

A.3.4 Pridanie a odstránenie tabuliek

Na druhom mieste v riadku nástrojov architekta je odkaz „Include & exclude tables“. Tento nástroj slúži k správe panelov na globálnej úrovni, teda pridávanie a odoberanie panelov z prostredia.

Centrálnym prvkom je tabuľka informujúca o stave spravovateľných tabuliek. Spravovateľné nie sú tabuľky postrádajúce primárny kľúč alebo tabuľky obsahujúce dátové typy needitovateľné súčasnými prostriedkami systému Webmin tak, ako bolo popísané v časti 3.1.1.

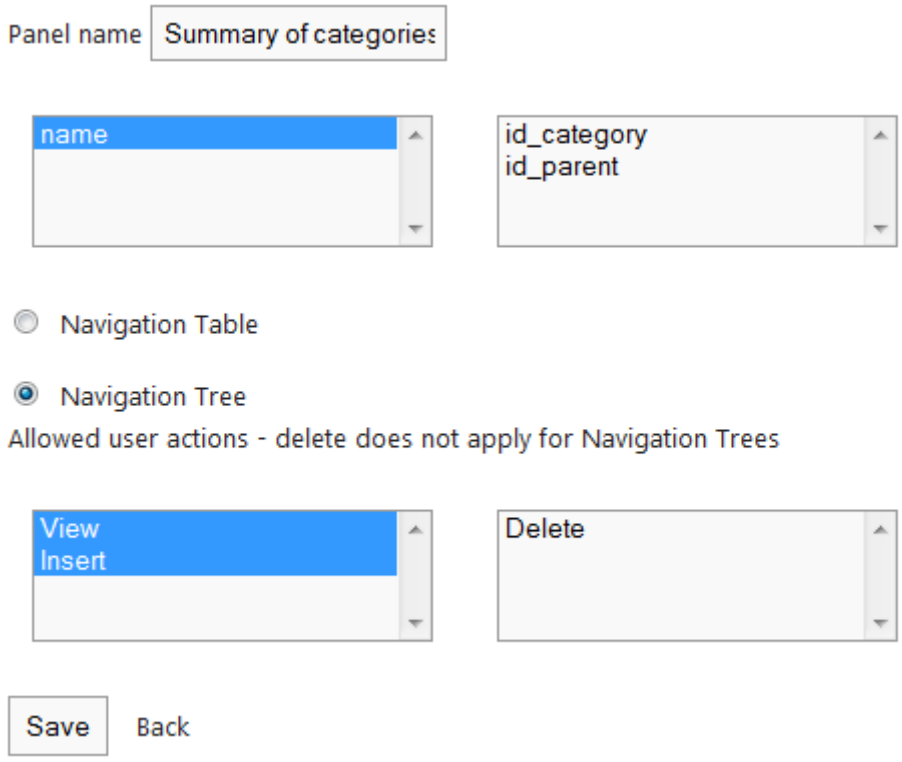
Každú zo spravovateľných tabuliek možno do správy pridať, pokiaľ ešte nie je v prostredí zahrnutá. To indikuje stĺpec „Has Panels“ v tabuľke. Ak je políčko tohto stĺpca v riadku prislúchajúcom danej tabuľke nezaškrtnuté, po kliknutí na voľbu „Add“ pri názve tabuľky sa tento riadok podfarbí zelenou a bude sprístupnené tlačidlo „Confirm“. V opačnom prípade bude odkaz „Add“ neprístupný. Po kliknutí na „Add“ sa taktiež detekujú, podobne ako v štvrtej fáze prípravy iniciačného návrhu, väzby typu M:N, realizované pomocnými tabuľkami a zahrňujúce zvolenú tabuľku a užívateľ bude mať možnosť správu týchto relácii zahrnúť do novo generovaných panelov zaškrtnutím príslušných políčok pod tabuľkou. Po stlačení „Confirm“ systém vykoná podobný sled operácií ako pri vytváraní prvotného návrhu, avšak s obmedzením na zvolenú tabuľku. Budú vytvorené dva nové panely, editačný a prehľadový, ktoré sa pod generickými menami uložia do databázy.

Stĺpec „Independent“ tabuľky spravovateľných tabuliek má informatívny charakter – za nezávislé sa, tak ako v časti 3.1.2, považujú tabuľky s primárnym kľúčom ktorého žiadna časť nie je súčasťou cudzieho kľúča.

Odstránenie panelov tabuľky z prostredia je okrem toho, že pre túto tabuľku existujú panely (je zaškrtnuté príslušné políčko stĺpca „Has Panels“), podmienené neexistenciou odkazu v menu projektu, ktorý by viedol na niektorý z panelov tejto tabuľky. Vylúčenie tejto tabuľky by totiž viedlo k nekorektnému stavu menu. Indikátorom existencie takýchto odkazov v menu je stĺpec „Reachable“ v tabuľke. Pri splnení týchto podmienok sa po zvolení „Remove“ pri názve tabuľky (riadok sa podfarbí červeno) a následnom potvrdení tlačidlom „Confirm“ vymažú z internej databázy oba panely prislúchajúce tejto tabuľke.

A.3.5 Úprava prehľadových obrazoviek

Po tom, ako užívateľ-návrhár prejde v projekte z úvodnej stránky na konkrétny panel, pridá sa do spomínanej sekcie návrhárskych nástrojov odkaz „Edit panel structure“. Cieľ tohto odkazu sa líši podľa toho, či užívateľ v tejto chvíli prehliada editačný alebo sumarizačný panel. V druhom prípade sa vykreslí ponuka stĺpcov tabuľky priradenej tomuto panelu, kde vľavo sú stĺpce momentálne zahrnuté v prehľadovej tabuľke a vpravo zvyšné stĺpce tabuľky tak, ako ukazuje obrázok 14. Kliknutím na názov stĺpca v jednom zo zoznamov ho možno presunúť do druhého zoznamu.



Obr. 14 Úpravy možností prehľadovej tabuľky

Je dôležité poznamenať, že v prípade zaradenia stĺpca, ktorý je cudzím kľúčom, medzi reprezentatívne stĺpce prehľadovej tabuľky sa v príslušnom stĺpci prehľadovej tabuľky nebude zobrazovať priamo hodnota stĺpca, ale hodnota prvého reprezentatívneho stĺpca záznamu, na ktorý tento cudzí kľúč svojou hodnotou odkazuje. Toto správanie dodáva prehľadu výpovednú hodnotu, ktorá by užívateľovi pri jednoduchom zobrazení hodnoty cudzieho kľúča bola zrejme ťažšie prístupná. Toto správanie nemožno zmeniť.

Vizuálne podobnú, avšak logicky odlišnú funkciu má druhá ponuka rovnakého charakteru, obsahujúca prvky „Insert“, „Delete“ a „Update“, slúžiacu k špecifikovaniu operácií, ktoré sú v paneli povolené, t.j. či bude výpis na paneli obsahovať tlačidlo Insert v dolnej časti stránky, respektíve odkazy na editáciu a odstránenie jednotlivých záznamov v prvých dvoch stĺpcoch tabuľky.

Ak sú dáta tabuľky hierarchického charakteru, je v paneli okrem dvoch uvádzaných ponúk naviac možnosť zvoliť medzi reprezentáciou v podobe štandardnej tabuľky a stromu. Nutno upozorniť, že pri voľbe stromového zobrazenia sa z dôvodu prehľadnosti vždy bude zobrazovať len prvý zo stĺpcov zvolených užívateľom vo vrchnej ponuke. Systém ničmenej nebráni užívateľovi vybrať viac stĺpcov a v prípade neskoršej zmeny typu zobrazenia prehľadu na bežnú tabuľku bude tento výber použitý.

A.3.6 Úprava editačných obrazoviek

Ak bude na odkaz „Edit panel structure“ v ponuke nástrojov návrhára kliknuté v kontexte editačného panelu, presunie sa na obrazovku správy polí tohto panelu. Príklad tejto obrazovke je zachytený na

obrázku 9. Centrálnym prvkom tejto obrazovky je tabuľka, ktorej riadky predstavujú stĺpce tabuľky spravovanej týmto panelom. Prvý stĺpec obsahuje názov stĺpca spravovanej tabuľky, druhý zaškrtávacie políčko, ktoré určuje, či tento stĺpec má alebo nemá byť editovateľný z editačného panelu. Stĺpce, ktoré sa nebudú editovať prostredníctvom formulára, sa jednoducho vynechajú aj z databázových dotazov vykonávaných v móde administrátora voči reálnej databáze, následkom čoho sa použijú východzie hodnoty pre tieto stĺpce dané nastavením databázy. Preto nemožno uložiť návrh panelu, ktorý by týmto spôsobom vynechával niektorý zo stĺpcov, ktoré nemajú povolenú hodnotu NULL a nemajú ani nastavenú východziu hodnotu.

V druhom stĺpci možno špecifikovať typ editačného poľa, ktoré sa má pre tento stĺpec spravovanej tabuľky použiť. Táto ponuka je automaticky obmedzená na tie typy polí, ktoré sú vhodné pre dátový typ príšlušného stĺpca. Nižšie je uvedený prehľad základných poskytovaných typov polí, ich správania ako ovládacích prvkov a podporovaných dátových typov.

Názov použitý v obrazovke návrhára	Popis	Dátové typy
TextBox	Jednoduché textové políčko	Textové typy
TextEditor	Integrovaný textový editor umožňujúci formátovanie a WYSIWYG editáciu HTML	Textové typy
Date Field	Textové políčko; po kliknutí zobrazí kalendár, z ktorého možno vybrať dátum	DATETIME
CheckBox	Zaškrtávacie políčko; v databáze nastavuje hodnotu true / false, resp. 0 a 1 pre číselné typy	BOOLEAN, SHORT INTEGER
Enum	Roletová ponuka s prvkami enumerácie	ENUM (iba MySQL)
Integer Field	Textové pole, celočíselnosť vstupu je zaistená validáciou	Celočíselné typy
Decimal Field	Textové pole, číselnosť vstupu je zaistená validáciou	Typy s pevnou a pohyblivou desatinnou čiarkou
File upload	Nahrávanie súboru, bude bližšie popísané nižšie	Textové polia s maximálnou dĺžkou obsahu najmenej 50

FK Field	Roletová ponuka so zoznamom hodnôt reprezentatívneho stĺpca z tabuľky, na ktorú sa odkazuje cudzí kľúč v tomto stĺpci	Stĺpce, ktoré sú cudzími kľúčmi
M2N Mapping Field	Editácia relácie typu M:N v pomocnej tabuľke	

Prvá poznámka k uvedenej tabuľke sa týka cudzích kľúčov. Cudzie kľúče možno editovať iba v poli typu FK Field bez ohľadu na dátový typ tohto stĺpca. Je to kvôli zjednodušeniu validácie, ktorá by v konečnom dôsledku musela zabezpečiť, aby boli do poľa vyplnené len hodnoty z odkazovaného stĺpca inej tabuľky. Editácia v inom type poľa by bola naviac užívateľsky málo prívetivá, nakoľko tieto stĺpce budú pravdepodobne obsahovať číselný identifikátor. Stĺpec referenčnej tabuľky, ktorého hodnoty sa budú zobrazovať v roletovej ponuke tohto poľa, možno zvoliť pre každé pole zvlášť a to pomocou ponuky v štvrtom stĺpci tabuľky.

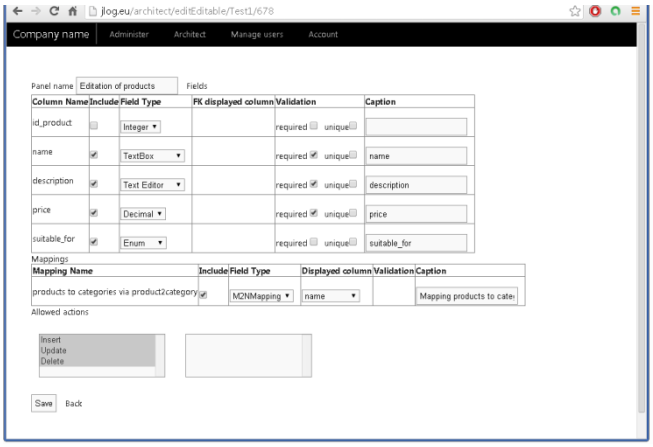
Podobne editácia relácie M:N môže prebiehať len v jednom, k tomu určenom type formulárového prvku. Tento má formu ponuky pozostávajúcej z dvoch zoznamov, ktorej použitie už bolo popísané v sekcii o správe prehľadových panelov. Ľavý zoznam pri editácii konkrétneho riadku tabuľky alebo vkladaní nového riadku obsahuje prvky odkazovanej tabuľky, ktoré majú byť v relácii s touto entitou a pravý zoznam ostatné prvky.

V piatom stĺpci tabuľky možno zvoliť dodatočné validačné podmienky pre dané pole editácie. Políčko označené „Required“ určuje, či má byť vyplnenie tohto poľa povinné. Táto voľba však bude ignorovaná, ak bude pre stĺpec zároveň zvolený typ poľa CheckBox. Ak by totiž bolo zaškrtnutie zaškrtávacieho políčka pri editácii povinné, bolo by políčko ako také vo formulári zbytočné a teda takáto validácia postráda zmysel.

Druhým zaškrtávacím políčkom sa určuje, či majú byť hodnoty v tomto stĺpci unikátne. Pole, ktoré predtým túto validáciu nepožadovalo, možno takto upraviť iba v prípade, že sú hodnoty v danom stĺpci tabuľky v spravovanej databáze unikátne v momente tejto akcie. V prvotnom návrhu je ako „Required“ označené prípustné minimum polí, t.j. všetky polia, ktorých stĺpce majú v databáze príznak NOT NULL a nemajú nastavenú východziu hodnotu. Ako „Unique“ sú označené tie stĺpce, na ktorých bol v databáze vytvorený unikátny index, toto však nie je nutnou podmienkou pre vytvorenie unikátneho poľa vo formulári, systém teda môže simulovať unikátne indexy, i keď to nie je doporučovaný postup. Ak by, naopak, v databáze vznikol unikátny index po vytvorení návrhu a editačné pole ostane bez tohto príznaku, nebude síce prebiehať validácia pred odoslaním dát databázovému serveru, v prípade vloženia duplicitnej hodnoty však systém zachytí špecifickú výnimku databázového rozhrania a podá užívateľovi o udalosti správu zobrazením chybovej hlášky rovnakého formátu ako majú tie pochádzajúce z validácie pred odoslaním.

Druhá tabuľka tejto obrazovky je súpisom relácií typu „M:N“, ktoré sa viažu k tejto tabuľke. Kvôli prehľadnosti má táto tabuľka rovnaké stĺpce ako prvá popisovaná, návrhár tu však má možnosť nastaviť len dve hodnoty – či bude alebo nebude táto relácia spravovaná v rámci tohto editačného panelu (stĺpec „Include“) a stĺpec tabuľky, s ktorou je editovaná tabuľka v relácii M:N, ktorého hodnoty sa budú zobrazovať v zoznamoch špecializovaného formulárového prvku, ktorého príklad uvidí návrhár o niečo nižšie.

Poslednou, už spomenutou časťou obrazovky je výber operácií podporovaných editačným panelom. Jeho funkcia je obdobná nastaveniu povolených akcií prehľadových panelov popísanému v predchádzajúcom oddiele.



Obr. 15 Úprava polí a ovládacích prvkov editačného panelu.

A.4 Rozhranie administrátora

Rozhranie administrátora je výsledkom práce návrhára prostredia a preto sa o jeho podobe nedá hovoriť v detailoch. Preto táto kapitola opisuje len všeobecné vlastnosti administračného rozhrania a niektoré vlastnosti, ktoré návrhár prostredia nemá možnosť ovplyvniť a nie sú celkom intuitívne

Prvým krokom administrátora je výber projektu v menu „Administer“ v hornej časti stránky. Po výbere projektu je administrátor smerovaný na úvodnú obrazovku administrácie projektu, ktorej dominuje menu panelov projektu. Menu má stromovú štruktúru a môže mať ľubovoľnú hĺbku. Niektoré jeho prvky nemusia vyvolať žiadnu akciu, iné môžu viesť na obrazovky s prehľadom záznamov niektorej tabuľky databázy, čiže na prehľad entít istého druhu vystupujúcich v aplikácii. Konečne, niektoré prvky menu vedú na formuláre určené k vytváraniu nových entít istého typu a ich uloženie do databázy.

Prehľadové panely môžu mať dve podoby. Častejšou je jednoduchá prehľadová tabuľka, ktorá v ľavých stĺpcoch môže obsahovať akčné odkazy „Edit“ a „Delete“, čím užívateľ požiada o úpravu príslušnej položky tabuľky a presunie sa do editačného formulára s predvyplnenými pôvodnými hodnotami vzťahujúcimi sa k editovanej entite, respektíve o vymazanie tohto prvku z databázy (v tomto prípade si aplikácia pred zmazaním prvku vyžiada dodatočné potvrdenie). Dolná časť tabuľky slúži k stránkovaniu. Tabuľka naviac poskytuje možnosť zoradiť prvky podľa zvoleného stĺpca.

Indikátor radenia sa nachádza v záhlaví tabuľky vedľa názvu stĺpca v hranatých zátvorkách. Východzia hodnota indikátoru naznačuje, že sa podľa tohto stĺpca prvky neradia. Po kliknutí na názov stĺpca v hlavičke sa obsah tabuľky zoradí podľa tohto stĺpca vzostupne a toto radenie sa znázorní trojuholníkom smerujúcim nahor na mieste indikátora. Pri opätovnom kliknutí na názov tohto stĺpca sa radenie obráti tak isto ako príslušný indikátor. Po kliknutí na iný stĺpec sa tabuľka zoradí nanovo. Tabuľka môže byť vždy zoradená len podľa jedného stĺpca.

Editačné obrazovky sú jednoduché formuláre, v ktorých adminstrátor pridáva nové alebo upravuje už existujúce entity uložené v aplikácii. Po vyplnení a odoslaní formulára sa vykoná validácia vložených hodnôt a súpis prípadných nedostatkov sa zobrazí nad formulárom.

V prípade, že bude užívateľ s formulárom pracovať dlhšie alebo prácu na čas preruší, môže dôjsť k ukončeniu sedenia. Ide o bezpečnostné opatrenie, ktoré automaticky odhlasuje užívateľa po dvoch hodinách nečinnosti v zmysle odoslania formulára alebo kliknutia na jeden z odkazov na stránke. Tento mechanizmus však užívateľa nemá obmedzovať. Preto sa päť minút pred automatickým ukončením sedenia zobrazí informačné okno, v ktorom môže užívateľ sedenie predĺžiť bez straty dát vyplnených vo formulári. Po automatickou odhlásení sa opäť zobrazí dialóg informujúci o tejto udalosti. Ani v tejto chvíli ešte nedochádza k strate hodnôt vo formulári, ale dôjde k nemu, keď sa užívateľ pokúsi formulár odoslať, pretože už nie je prihlásený. Po zhliadnutí správy „The session has expired...“ by si mal užívateľ akékoľvek údaje zo stránky, ktoré nechce stratiť, uložiť externe, napríklad do textového súboru.

A.5 Prideľovanie užívateľských práv

Správa užívateľských oprávnení sa uskutočňuje v jednoduchom prostredí, do ktorého sa užívateľ, ktorý má oprávnenie spravovať užívateľské oprávnenia aspoň k jednému projektu, dostane kliknutím na „Manage users“ v hornom menu. Obrazovka sa skladá z dvoch roletových ponúk a troch zaškrtávacích políčok. V prvej roletovej ponuke sú mená registrovaných užívateľov, v druhej zoznam projektov, prístupové práva ku ktorým má užívateľ právo editovať. Zaškrtávacie políčka predstavujú tri druhy prístupového oprávnenia ku konkrétnemu projektu – administračné oprávnenia umožňujú upravovať dáta v databáze a držiteľ práv návrhára môže upravovať formu administračného prostredia (nemusí ale mať práva administrátora). Posledným typom oprávnenia je právo priraďovať oprávnenia k tomuto projektu. Tieto políčka sa automaticky označia / odznačia pri zmene výberu užívateľa a projektu tak, aby zodpovedali aktuálnym oprávneniam daného užívateľa voči danému projektu. Ak nie je zvolený žiaden projekt, spravujú sa globálne oprávnenia tohto užívateľa k všetkým projektom. (Pri prístupe k niektorému z projektov sa uplatňuje maximum z oprávnení prihláseného užívateľa ku konkrétnemu projektu a jeho globálnych práv.) Po zmene nastavení pre konkrétneho užívateľa a kliknutí na „Save“ sa tieto zmeny uložia a administrátor môže pokračovať v správe užívateľských práv pre iných užívateľov alebo projekty.

Prvý užívateľ, ktorý bol vytvorený pri inštalácii aplikácie, má automaticky priradené všetky práva nad projektmi.

A.6 Nastavenie užívateľského účtu

Pre úplnosť doložme popis registrácie nového užívateľa, jeho prihlasovania a odhlasovania a zmeny hesla. Tieto operácie užívateľ vykonáva v záložke Account horného menu, ktorá je jedinou prístupnou pri príchode neprihláseného užívateľa na stránku. Všetky akcie totiž vyžadujú prihlásenie. Pokiaľ užívateľ ešte nemá vytvorený účet, môže tak urobiť kliknutím na „Register“ po rozbalení roletového menu spomínanej záložky. V zobrazenom formulári vyplní svoje prihlasovacie údaje a ak tieto spĺňajú bezpečnostné požiadavky, je užívateľ registrovaný a automaticky prihlásený do systému. Registrovaní užívatelia sa prihlasujú kliknutím na „Login“ a vyplnením potrebných údajov. Funkciu „Logout“ netreba bližšie vysvetľovať.

Napokon autentizačný systém umožňuje prihláseným užívateľom zmenu hesla („Change password“). V zobrazenom formulári nutno vyplniť pôvodné a nové heslo.

B Obsah priloženého CD

CD obsahuje

- **Adresár Source –** v tejto zložke sú uložené riešenie pre Visual Studio 2010 / 2012. Podadresár Source\Webmin obsahuje súbory potrebné pre inštaláciu systému na server (viď A.1).
- **Adresár HtmlDoc –** tu možno nájsť programátorskú dokumentáciu jednotlivých tried a metód používaných v riešení. Táto dokumentácia ma formu prepojených HTML stránok a vychádza z komentárov v zdrojovom kóde.
- **Adresár FieldPlugin –** toto je ukážka implementácie externého pluginu editačného poľa. Návod pre vytváranie týchto pluginov je uvedený v stati 4.6.
- **Súbor TextBP.pdf –** text tejto práce.

Zdroje

[Kz] Kozelková Hana: Editor datových modelů s podporou reverzního datového inženýrství. Diplomová práca. Univerzita Karlova, MFF, 2009

[DM] Ian H. Witten, Eibe Frank, Mark A. Hall: Data Mining: Data Mining, Third Edition. Elsevier, 2011

[DS] Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom: Database Systems: The Complete Book, Second Edition. Prentice Hall, 2008

[TK461] Dejan Sarka, Itzik Ben-Gan , Ron Talmage: Training Kit (Exam 70-461): Querying Microsoft SQL Server 2012, Microsoft Press, 2012

[Cr] http://www.cmsreview.com/Features/Lists.html

[Pb] https://code.google.com/p/protobuf-net/wiki/Performance

[MSDN] http://msdn.microsoft.com/en-us/library