Technická univerzita v Liberci

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: N 2612 – Elektrotechnika a informatika

Studijní obor: 1802T007 – Informační technologie

Vytvoření webové aplikace pro administrativní účely

Creation of an application for administrative acts

Diplomová práce

Autor: **Bc. Jan Obrátil**

Vedoucí diplomové práce: Ing. Josef Chaloupka, Ph.D.

Konzultant: Doc. Ing. Zdeněk Plíva, Ph.D.

V Liberci dne 16. 5. 2008

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Ústav informačních technologií a elektroniky Akademický rok: 2007/08

ZADÁNÍ DIPLOMOVÉ PRÁCE

Jméno a příjmení: Bc. Jan Obrátil

studijní program: N 2612 - Elektrotechnika a informatika

obor: 1802T007 - Informační technologie

Vedoucí ústavu Vám ve smyslu zákona o vysokých školách č.111/1998 Sb. určuje tuto diplomovou práci:

Název tématu: **Vytvoření webové aplikace pro administrativní účely**

Zásady pro vypracování:

1. Seznamte se s programováním v programovacím jazyku PHP a vytvářením dokumentů v PDF formátu.
2. Vytvořte webovou aplikaci „elektronickou sekretářku" pro administrativní potřeby Ústavu informačních technologií a elektroniky.
3. V této aplikaci by především mělo být umožněno jednoduché vyplňování předdefinovaných šablon formulářů (faktury, cestovní příkazy...). Výstupem by měl být formulář ve formátu PDF.
4. K aplikaci vytvořte podrobnou dokumentaci, aby bylo možné v budoucnu celý systém dále rozšiřovat a upravovat.

Rozsah grafických prací: dle potřeby dokumentace

Rozsah průvodní zprávy: cca 40 až 50 stran

Seznam odborné literatury:

[1] kol. autorů: PHP programje profesionálně, Computer Press, 2001

[2] Ponkrac, M. : PHP a MySQL, Computer Press, 2007

[3] Gruber, M.: Mistrovství v SQL, SoftPress, 2004

[4] PDF Reference: http://www.adobe.com/devnet/pdf/pdf_reference.html

Vedoucí diplomové práce: Ing. Josef Chaloupka, Ph.D.

Konzultant: Doc. Ing. Zdeněk Plíva, Ph.D.

Zadání diplomové práce: **31.10.2007**

Termín odevzdání diplomové práce: **16. 5. 2008**

L.S.

.....

Vedoucí ústavu

.....

Děkan

V Liberci dne 31.10.2007

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že **souhlasím** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mně požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do její skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

zde patří zadání bakalářské práce

Abstrakt

Tato diplomová práce se zabývá vytvořením moderní webové aplikace pro správu formulářů definovaných pomocí externě vložených šablon s možností tisku pomocí formátu PDF a manipulací s nimi – vytváření, editace, správa verzí a přístupů. Jsou zde k nalezení diskutované postupy tvorby jednotlivých částí aplikace, zvolené technologie, návaznosti na standardy a nástroje pracující s daným standardem. Je zde možno nalézt pokyny pro tvorbu formulářů, jejich tiskových výstupů, administrace a údržba aplikace. Poslední část dokumentu obsahuje popis budoucích možných rozšíření aplikace s přihlédnutím na uživatelskou přívětivost.

Klíčová slova: formulář, webová aplikace, správa uživatelů a přístupů, správa verzí, AJAX, XML, XSL-FO

Abstract

Aim of this thesis is creation of a modern web application of form administration defined using external templates with ability to print filled form using PDF export. Application also should create and edit forms with version control with ability of revert changes. There are described successions of creation essential parts of the application, used technologies, standards and tools for working with them. There are directions for creation of new forms, theirs print output, administration and application maintaining. The very last part deals with further development of the application aimed to user experience.

Keywords: form, web application, authentication and access controll, version controll, AJAX, XML, XSL-FO

OBSAH

Úvod

Cílem této diplomové práce je vytvoření webové aplikace umožňující komplexní správu formulářů definovaných pomocí externích definic struktury a instrukcí pro tisk. Výsledná aplikace by měla sloužit Ústavu informačních technologií a elektroniky pro potřeby administrace různých druhů formulářů, jako je úprava nebo vytvoření formuláře předdefinovaného druhu, zachování historie změn nebo tisk.

Vzhledem k popularitě webových aplikací a jejich atraktivitě díky technologiím, jako je AJAX, je možno v dnešní době vyrobit aplikace svou použitelností velmi podobné běžně používaným aplikacím vyžadující instalaci. Výsledným produktem této práce je právě aplikace vyrobená těmito moderními trendy snažící se o maximální interaktivitu s uživatelem během jeho práce.

1 Požadavky na aplikaci

Jak již bylo naznačeno v úvodu, aplikace by měla splňovat požadavky Ústavu informačních technologií a elektroniky, který ji bude primárně používat. Měl být zvážen provoz této aplikace a její následná údržba. Je totiž nutné promyslet všechny možné dopady případných budoucích změn v aplikaci, které jsou zřejmé už nyní, a připravit tuto aplikaci, aby změna byla vůbec možná.

Následující seznam shrnuje ve stručnosti požadavky, které by měla aplikace řešit.

- Přístup více uživatelů do aplikace pomocí přihlašovacího jména a hesla.
- Možnost přidávání, editace a mazání uživatelů.
- Vytváření a editaci formulářů *objednávka*, *příjemka*, *průvodka k faktuře* a případně další. Při vytváření nového formuláře zajistit, aby nevznikly dva různé formuláře se stejným *číslem formuláře*[1].
- Tisk těchto formulářů pomocí exportu do PDF.

V průběhu konzultací se tyto požadavky ukázaly jako nedostačující a v mnohých bodech bylo zapotřebí zajít do daleko větší míry abstrakce, než bylo původně plánováno.

Systém uživatelských práv by měl být daleko více konfigurovatelný, aby bylo možné jednotlivým uživatelům podrobněji nastavovat práva, např. by mělo být možné nastavit uživateli přístup třeba jen do části s objednávkami, ale už ne do části s příjemkami, nebo aby mohl příjemky pouze tisknout.

Z konzultací také vyplynula možnost, že se někdy budou formuláře měnit, možná za rok, možná za dva měsíce, ale musí s tím aplikace počítat. Dále pak bylo naznačeno, že by mohly v budoucnu přibýt další typy formulářů, např. faktury nebo cestovní příkazy. Tento bod tedy bylo zapotřebí řešit daleko obecněji. Těžko si lze představit, že faktura vytvořená v roce 2008 bude po změně šablony formuláře v roce 2009 vypadat jinak a výsledná částka daně DPH bude třeba jiná, protože se legislativa změní. Všechny tyto možnosti musely být promyšleny, aby nakonec nebyla vytvořena aplikace, která nebude vyhovovat potřebám uživatelů.

Z těchto požadavků také vyplynulo, že by měla být aplikace co nejvíce modulární, aby se například definice formuláře dala jednoduše vyměnit za jinou nebo přidat nová

[1] Číslo formuláře je unikátní řetězec, který má každý vytvořený formulář a pod kterým je evidován v papírové verzi.

definice, aniž by to znamenalo úpravu stávajícího kódu aplikace nebo struktury databáze. Bylo tedy zapotřebí navrhnout systém, kterým se aplikace dozví o svých šablonách formulářů nahraným později někam do adresářové struktury.

Řešení problematiky tisku bylo od začátku zvažováno přes export do PDF. Byla zvažována knihovna FPDF s níž má autor již zkušenosti a v praxi se mu osvědčila.[2]

[2]FPDF je knihovna pro generování PDF dokumentu pomocí PHP skriptu. Více informací lze nalézt v [Plathey(2008)].

2 Realizace aplikace

Vzhledem k charakteru požadavků na aplikaci, bylo rozhodnuto pro webovou aplikaci, která má oproti klasickým desktopovým aplikacím spoustu následných výhod:

Snadná aktualizace spočívající v aktualizaci serveru a nikoli všech klientských stanic[3].

Multiplatformnost – aplikace poběží na jakémkoli desktopovém počítači s moderním webovým prohlížečem.

Snadná správa – administrátor se stará o jeden stroj.

Snadná dostupnost – uživatel se může připojit k aplikaci odkudkoli na internetu[4].

Nelze toto řešení pouze vychvalovat, je zapotřebí zmínit i negativa tohoto řešení, které bude zapotřebí minimalizovat použitím nejrůznějších technik, aby negativa nakonec nezpůsobila nepoužitelnost celé aplikace.

Nepohodlnost – webové aplikace bývají dost často pro uživatele nepohodlné a práce s aplikací pomalá; z tohoto důvodu je v maximální možné míře využívána technologie AJAX.

Kompatibilita aplikace mezi různými webovými prohlížeči bývá dost problematická a zajištění kompatibility dost často velmi časově náročné.

2.1 Základní technologie použité při vývoji aplikace

Při výběru použitých technologií byly zvažovány různé technologie. Prakticky přicházely v úvahu tři hlavní technologické proudy.

1. Skriptovací jazyk PHP s volně dostupnými knihovnami a frameworky s napojením na relační databázi MySQL. S touto možností bylo počítáno už od samého začátku, protože autor s těmito technologiemi má mnohaleté zkušenosti při tvorbě komerčních projektů. Je také k dispozici vlastní framework[5] vyvinutý na konci

[3] Je pravda, že toto není nepřekonatelná překážka. Java tento problém řeší pomocí Java Web Start technologie [Sun(2008)].

[4] Aplikování určitých bezpečnostních pravidel je na místě.

[5] Framework je anglický výraz pro označení sady softwarových komponent, podpůrných programů či návrhových vzorů umožňující snazší vývoj softwaru.

roku 2006 a úspěšně nasazený na několika projektech, takže použití v tomto případě bude těžit z nesporných výhod – vyzkoušené nástroje a zkušenosti s jejich nasazením. Další výhodou této možnosti je snadná dostupnost služby. Je dostupná prakticky na jakémkoli hostingu a linuxové distribuce je mají často předkonfigurované, takže instalace a konfigurace je otázkou cca deseti minut. Tomuto řešení se také dost často říká LAMP[6]

2. Programovací jazyk Java s použitím některých dalších technologií J2EE nebo frameworku Spring, který je poslední dobou velmi oblíben. Toto jsou velmi zajímavé technologie, nicméně zkušenosti s těmito technologiemi autor nemá a i když by je rád použil, dostal by se pravděpodobně do velmi nepříjemných časových potíží, protože většinu času by se učil technologie a práce by stála na místě. Mezi nesspornou výhodu tohoto řešení lze přiřadit rychlost této platformy oproti skriptovacím jazykům a snadnou rozšiřitelnost, protože J2EE nástroje nutí vývojáře myslet a programovat v jasně definovaných schématech a nedovoluje mu příliš sklouzávat k nehezkým praktikám vývoje, které jsou jeho kolegové s volností jazyka PHP schopni použít. Využití této technologie tedy umožňuje snazší převzetí projektu jiným vývojářem.

3. Řešení tak trochu „z druhé strany politického spektra“ od firmy Microsoft, která nabízí řešení v podobě frameworku ASP.NET na platformě .NET. Tato varianta má výhody podobné jako druhá zmiňovaná varianta – rychlost, ještě větší míra přehlednosti kódu pro ostatní vývojáře, protože pro tuto platformu existuje pouze jeden webový framework s jedním vývojovým prostředím. Problémem je, že je zde pouze jeden dodavatel technologie, operačního systému a vývojových nástrojů, a to je dost nepříjemné, protože se tato varianta automaticky stává i nejdražší variantou a přitom technologicky ekvivalentní k druhé variantě s použitím jazyka Java a přidružených technologií. Dalším problémem použití této technologie je autorova neznalost této technologie a neochota si na svůj počítač instalovat jiný operační systém než je zvyklý používat.

Z výše uvedených možných technologií byla nakonec vybrána první varianta, a to hlavně z důvodu znalosti těchto technologií. Autor je tedy schopen vytvořit kvalitnější a komplexnější produkt než v ostatních zmiňovaných.

[6]Spojení čtyř technologií: Linux, Apache, MySQL a PHP.

2.2 Použitý framework

Jak už bylo zmíněno v předchozích odstavcích, byla zvolena technologie skriptovacího jazyka PHP, webového serveru Apache a relační databáze MySQL na Linuxovém operačním systému. Jedná se tedy o standardní řešení LAMP nabízené mnoha firmami nabízející webový hosting.

Ve skriptovacím jazyce PHP lze vytvořit aplikace různým způsobem. Těch způsobů je až příliš mnoho a dost často se vývojář, který s touto technologií začne a nemá základy s některými jinými vývojovými nástroji nutící vývojáře k myšlení v návrhových vzorech, uchyluje k nehezkým praktikám. Taková aplikace je dříve nebo později odsouzena k záhubě, protože údržba a rozvoj takovéto aplikace je velmi časově nákladná a náchylná k chybám. Nezřídka je vývoj takové aplikace ukončen a je navrženo její kompletní přepsání.

Když bude ve stručnosti připomenut vznik a průběh vývoje skriptovacího jazyka PHP, snáze lze pochopit jeho klady a zápory a lze tak lépe navrhnout aplikaci odproštěnou od těchto nepříjemných jevů.

2.2.1 Stručná historie jazyka PHP a jeho použití

Kořeny jazyka PHP sahají do roku 1994, kdy se dánský programátor Rasmus Lerdorf rozhodl obohatit své webové stránky o sadu CGI skriptů, které by mu pomohly vytvořit některé dynamické prvky jeho stránek. V létě roku 1995 pak veřejně vypustil jeho verzi nazvanou PHP[7] do světa, aby mu ostatní lidi pomohly v hledání chyb v kódu.

Od verze 2, jejíž charakteristické prvky jsou vidět i v dnešních verzích PHP, si jazyk nese možnost „být vložen“ do HTML kódu a určité úseky jsou pak vykonávány PHP preprocesorem.

Od verze 3 vydané roku 1998 byl kompletně přepsán parser jazyka a přidán Zend Engine, virtuální stroj, který výrazně urychlil běh skriptů.

Verze 4 vydaná v roce 2000 si s sebou nese lepší podporu pro objektově orientované programování (dále jen OOP) a výrazné zrychlení vykonávání kódu. Podpora OOP byla v tu dobu velmi kritizována, protože běžné konstrukce známé z jiných jazyků nešly až tak snadno použít.

Od verze 5 má v sobě PHP Zend Engine II, který opět výrazně urychlil vykonávání kódu. Dále byla kompletně přepracována podpora OOP, takže aplikace využívající

[7] Pretty Home Pages

OOP prvky ve verzi 4 dost často ve verzi 5 nefungovaly. [PHP(2008b)] V současné verzi 5.2.6[8] jsou k dispozici nejdůležitější OOP prvky, které jsou zapotřebí pro bezproblémové psaní, např. abstraktní třídy, abstraktní metody, statické metody, statické atributy, rozhraní, výjimky a modifikátory private, public, protected. Tato verze má OOP na hodně solidní úrovni, ale některé věci stále nejsou takové, jaké by měly být, např. výjimky jsou sice řešeny, ale existuje pouze `try-catch`. Na většinu problémů to stačí, ale někdy se opravdu hodí `try-catch-finally`, které z nějakého důvodu chybí.

Když shrneme historii tohoto jazyka a uvědomíme si, na jaké typy problémů byl nasazován v prvních verzích a jaké problémy s ním řešíme nyní, je jasné, že nebude vše v pořádku.

Původní účel jazyka byl obohatit statickou webovou stránku o dynamický obsah. Nejčastěji se využívalo právě vkládání skriptu do už vytvořené statické stránky a když si otevřeme několik učebnic PHP, tak i tam tento postup bude uveden (a dost často jako jediný). Spousta vývojářů vytvářejí weby právě tímto způsobem. Na každou funkční stránku mají jeden PHP skript, kde mají na samém začátku pomocí funkce `include()` vložené knihovny funkcí a některé inicializační akce, jako napojení aplikace na databázi, počítadlo přístupů apod. Zbytek skriptu je pak pouze HTML stránka s vloženými dynamickými prvky. Tomuto stylu se dost často říká lidově *špagetový kód*. Je to pravděpodobně nejhorší způsob vývoje v PHP, protože prakticky neexistuje znovupoužitelnost kódu, změna grafického prvku stránky je velmi časově nákladná činnost[9] a změna struktury aplikace, byť jen minimální, znamená nepředstavitelné náklady. Tento problém u malých aplikací (maximálně 10 až 15 stránek) není znát, ale u větších aplikací je to velmi nepříjemný jev, který výrazným způsobem prodražuje vývoj a snižuje kvalitu aplikace.

V dnešní době je zapotřebí použít jiné přístupy. Řešení se ukazuje v použití návrhového vzoru *model-view-controller* [Pecinovský(2007)] (dále jen MVC). Tento návrhový vzor ctí myšlenku, že by měly být tři hlavní části aplikace od sebe oddělené a tyto části by se navzájem volaly pomocí dohodnutého rozhraní.

Model je část zabývající se daty, se kterými aplikace bude pracovat. Nejčastěji se bude jednat o data z databáze, s nimiž se v aplikaci pracuje jako s objekty. Této technice se říká *object relation mapping* (dále jen ORM).

[8] Květen 2008

[9] Nemůže ji provézt grafik, protože by se prodírál dost nepřehlednou směsicí PHP a HTML kódu, a úprava dost často není řešitelná z jednoho místa – stejné grafické prvky jsou často nakopírovány ve více souborech bez předem jasné logiky.

View je ve zkratce prezentační vrstva aplikace. V použitém frameworku se o tuto část stará třída využívající šablonový systém Smarty [New(2008)].

Controller obsahuje vlastní aplikační logiku. Příslušný controller je spuštěn frameworkem při dotazu prohlížeče na server a vykoná příslušné operace. Dost často takový typický controller obsahuje definici prostředků, které bude potřebovat, natažení modelů z databáze, vykonání operace s těmito daty a příslušná vizualizace výsledků pomocí view vrstvy nebo přesměrování na jiný controller uvnitř aplikace.

Tímto je možné dosáhnout obrovskou modulárnost, protože je možné vyměnit jakoukoli část třeba za novou verzi toho samého a vše bude transparentně fungovat, např. je-li nutné změnit design webu, stačí změnit šablony ve view vrstvě. Jsou tím eliminovány funkční vazby, které spolu nijak nesouvisí. (Na co míchat prezentační vrstvu s dotazy do databáze, když spolu být nemusí?)

2.2.2 Důvody vzniku a použití daného frameworku

V době[10], kdy autor naléhavě potřeboval framework pro své projekty, neexistoval žádný uspokojivě fungující a využívající v tu dobu relativně nových vlastností jazyka PHP 5.1. Jiných existovala celá řada, ale žádný nepoužíval nativně pro obsluhování chybových stavů výjimky, dědičnost a rozhraní. V tu dobu se už začínal rýsovat velikán dnešní doby – Zend Framework, ale v tu dobu byl zatím jen ve verzi alfa s varováním, že API se může kdykoli změnit. Na takovémto frameworku se nedá stavět stabilní aplikace. Bylo v tu dobu provedeno i několik experimentů s frameworkem CakePHP [Cak(2008)] i přes to, že tento framework je spíše záležitost PHP verze 4. Bohužel tento framework inspirující se úspěchy Ruby on Rails byl velmi omezující a práci to neusnadňovalo.

V prosinci 2006 tedy byl napsán framework od nuly, který se s několika změnami zachoval až do teď. Využívá všechny možnosti jazyka naplno. Jedná se o kompletní OOP projekt bez jakýchkoli potenciálně problémových konstrukcí, jako jsou globální proměnné, zbytečně veřejné atributy tříd a pod. Do dnešní doby s tímto frameworkem bylo napsáno několik úspěšně fungujících aplikací.

Za dobu od jeho vzniku do něj bylo přidáno několik velmi užitečných modulů, které řeší nejrůznější problémy, např. podpora AJAXu apod.

[10] Prosinec 2006

2.2.3 Model frameworku

Část frameworku, model, je možná tou nejužitečnější částí vůbec. Tato část poskytuje základní metody pro práci s databázovými záznamy v tabulce jako s objekty. V projektu lze vytvořit nové mapování na tabulku jako třídu odvozenou od abstraktní třídy modelu `OEModel` poskytovaného frameworkem. V nové třídě se přetíží metoda `_init()`, do níž se vloží definice jednotlivých atributů databázové tabulky, popisy atributů, regulární výrazy pro validaci atributů a další parametry, mezi nimiž je i možnost definice relace na jinou třídu modelu a definovat, přes které tabulky a sloupce bude vedena. V této třídě samozřejmě může být daleko více uživatelsky definovaných metod, které pracují s daty, ale vždy by se mělo zachovat pravidlo, že se v této třídě objevují pouze metody pracující s daty, nikoli metody definující vzhled nebo řízení aplikace.

V aplikaci je tímto způsobem vytvořena např. třída `FormModel` reprezentující tabulku `forms` s informacemi o formulářích a třída `FtypeModel` reprezentující tabulku `ftypes` s informacemi o šablonách vytvořující dané typy formulářů. V kódu pak lze získat z databáze model formuláře s primárním klíčem 123 následujícím způsobem.

```
$formular = OEModel::getByPrimary(new FormModel(), 123);
```

Pokud z tohoto formuláře potřebujeme získat název typu formuláře `ftype_name` nalézající se v tabulce `ftypes`, na kterou je vedena relace, může takový kód vypadat následovně.

```
$ftype = $formular->getRelFtype()->getModel();
$nazev = $ftype->getColumn('ftype_name')->getValue();
```

Tento příklad získání názvu `ftype_name` z relace mohl být i na jedné řádce, ale z typografických důvodů byl rozdělen do dvou. Ale i tak je vidět, jak obrovská úspora kódu toto je. Nikde není zapotřebí psát SQL dotazy a nikde není zapotřebí zajišťovat připojení do databáze – spojení s databází se naváže automaticky až v době, kdy bude opravdu potřeba. V době vývoje aplikace bývá ve frameworku zapnuta kontrola správnosti modelů, která spočívá v kontrole modelu před vlastním použitím oproti databázové tabulce. Pokud v databázové tabulce bude změněna struktura a konfigurace modelu přestane souhlasit se skutečnou strukturou v tabulce, dojde k vyhození výjimky. Jedná se o velmi účinný nástroj pro hledání nesrovnalostí v konfiguraci modelů.

2.2.4 View frameworku

Tato část MVC má za úkol starat se o prezentační vrstvu aplikace. Myšlenka celé view vrstvy aplikace je postavena na šablonovém systému Smarty, který je v dnešní době prakticky zakonzervovaný projekt. Dochází už pouze občas k opravě nějaké bezpečnostní nebo funkční chyby.

Myšlenka šablonového systému spočívá v definování prázdné HTML šablony bez obsahu. V této šabloně jsou obsaženy pouze grafické prvky se značkami pro vložení obsahu. Obsah je pak dodán skrz rozhraní šablonového systému. Šablony Smarty dokáží však více, než jen vložit obsah do šablony. Umí základní prvky skriptování, jako jsou podmínky, cykly přes prvky pole apod. Dost často je toto skriptování využito při zobrazování prvků seznamu. Seznam je připraven šabloně ve formě pole asociativních polí. V šabloně se pak projde pole, kde je v každém cyklu do proměnné vráceno asociativní pole, ze kterého jsou použity hodnoty asociativních prvků.

Systém Smarty není v aplikaci použit v surové podobě. Jeho rozhraní je dost neohrabané a některé konstrukce nejsou příjemné, např. je zapotřebí před každým použitím šablony nakonfigurovat cesty do dočasných adresářů, nakonfigurovat cestu do adresáře s šablonou, nastavit, zda je možné vytvořenou šablonu uchovávat ve vyrovnávací paměti a podobně. Dále zde jsou nepříjemnosti v podobě nutnosti zadat název souboru se šablonou až v metodě, která získává obsah výsledné šablony. Navíc tyto metody pro získání obsahu mají parametry, které jsou dost často irelevantní vzhledem k dřívější konfiguraci objektu šablony. V neposlední řadě je velmi problematické použití této třídy z důvodu využívání ještě starého stylu psaní, kde neexistovaly modifikátory *private*, *public* a *protected*. Spousta atributů je vidět, i když by je vývojář neměl vůbec použít, protože pracuje na úplně jiné vrstvě, dále spousta nastavení není řešená přes settery a gettery, takže nechtěné použití atributu „z venčí“ může vézt ke špatně odhalitelným chybám.

Třída `Smarty` byla tedy obalena[11] třídou `OEView`, aby byl k funkcím příjemnější přístup přes metody, jejichž chování bylo upraveno pro snazší a bezpečnější použití v tomto frameworku.

2.2.5 Controller frameworku

Jak již bylo nastíněno výše, *controller* je část aplikace obsahující vlastní aplikační logiku. Framework na základě požadavku klienta zavolá příslušný controller odkud

[11] encapsulation

podle definovaných pravidel může být požadavek přesměrován na jiný controller nebo provedena příslušná akce, kterou klient po serveru požaduje. Controller tedy vykonává operace nad modely (data aplikace) a zobrazuje výsledky daných operací klientovi pomocí view vrstvy.

V této vrstvě by také měla být řešena veškerá bezpečnost aplikace – kontrola oprávněnosti vykonávat danou akci a případné přesměrování na přihlašovací stránku, zobrazení chyby nebo přesměrování na neprivilegovanou verzi daného tématu stránky.

V tomto frameworku controller zastává zčásti i funkci zvanou *router*. Je to část, která „směruje" tok programu a vybírá správný controller na základě dotazu klienta. Při směrování požadavku frameworkem je standardně počítáno s tím, že je dostupný modul *rewrite*, což je modul webového serveru Apache umožňující zapisovat webové adresy ve tvaru `http://example.com/Forms/PrintForm/29`. Nikde nemusí být vidět název skriptu a přípona `.php` apod. Pomocí modulu rewrite je tato adresa vnitřně „přepsána" na `http://example.com/index.php?url=Forms/PrintForm/29`. Modul rewrite tedy skryje před uživatelem druhý, poněkud neohrabaný způsob zápisu URL. Pěkné adresy (bez zbytečných parametrů) jsou jednak dobře čitelné pro oko uživatele a jednak jsou v některých případech lépe hodnocené internetovými vyhledávači a takováto stránka má větší šanci se stát úspěšněji umístěnou než ostatní (to ale není pro aplikaci v případě této diplomové práci vůbec podstatné).

Směrování se uvnitř controlleru provádí pomocí přetížení metody `customRoute()`. Framework standardně předá řízení chodu aplikace třídě `RootController`, které také v konstruktoru předá pole s parametry vloženými na adresní řádek. Parametry v tomto případě jsou `array("Forms", "PrintForm", 29)`. Třída `RootController` má ve své definici v `customRoute()` pravidlo, které vezme nultý prvek v poli, tedy `"Forms"`, a pokusí se najít controller `FormsController` a předat mu do konstruktoru zbytek pole s parametry bez už použitého nultého prvku, tedy `array("PrintForm", 29)`. Ve třídě `FormsController` sice v aplikaci pravidla jsou, ale v tomto konkrétním případě se na požadavek nepoužijí, protože nevyhoví definovaným podmínkám, což je stejné, jako by v metodě `customRoute()` žádné pravidlo nebylo. Aplikuje se tedy standardní pravidlo, které říká, že se vezme nultý prvek z parametrů a pokusí se nalézt metoda tohoto controlleru. Zavolá se tedy uvnitř objektu `FormsController` metoda `PrintForm` opět s parametrem bez nultého prvku, tedy `array(29)`. V Metodě `PrintForm` je tedy už provedena vlastní akce, kterou uživatel po aplikaci požaduje, s příslušným parametrem potřebným pro splnění úkolu. V této metodě by tedy mělo být ověření správnosti parametrů potřebných pro svou práci, ověření identity uživatele na základě cookies

případně i IP adresy a až následně provedení stanovené akce.

V případě, že nebyla nalezena patřičná třída s controllerem nebo metoda, je provedeno přesměrování na implicitní controller a nebo implicitní metodou. Na implicitní umístění je rovněž směrováno, pokud parametry chybí.

2.2.6 Helper frameworku

Helper je další ještě nezmíněnou částí frameworku, která velmi ulehčuje některé konstrukce. Helper je považován za část patřící do *view* části s tím rozdílem, že si s sebou nese i jistou část funkčnosti. Ve frameworku je zastupován abstraktní třídou `OEHelper` a má velmi podobné vlastnosti jako `OEView` s tím rozdílem, že `OEView` se používá samo o sobě a po programátorovi chce jen jeho naplnění daty a definici souboru s šablonou. `OEHelper` je využit vytvořením vlastní třídy odvozené právě od `OEHelper`. Tento přístup dovoluje, do tímto způsobem vytvořené třídy, vložit další funkcionalitu a řešit tím některé grafické prvky velmi elegantním způsobem. Je možné si jako příklad představit grafickou komponentu *hlavní menu*. Komponenta starající se o vykreslení celé webové stránky jako celku si pak tuto komponentu zkonstruuje a pomocí její metody `addItem()` si přidá všechny potřebné položky v menu v závislosti na kontextu, v jakém se návštěvník stránek nachází, a nakonec obsah menu získá metodou `fetch()`, jež je standardní název metody pro všechny vizuální komponenty.

Pomocí těchto komponent se také usnadňuje vývoj dynamických komponent založených na technologii AJAX. Pomocí helperu je co nejkomplexněji zaobalena funkčnost dané komponenty a pak je tato komponenta bez problémů použita na několika místech. V případě AJAXových komponent to je jednou v controlleru, na kterém je komponenta umístěna, a podruhé na controlleru, na který se komponenta odkazuje a po kterém chce provézt dané interaktivní AJAXové akce (podrobněji je tato problematika rozebírána na straně 21).

2.2.7 Další vlastnosti frameworku

Byly popsány ty největší části frameworku, ale je zapotřebí se zmínit i o zdánlivě nepodstatných vlastnostech, protože v zásadě ovlivňují výkonnost a přehlednost celého systému.

Adresářová struktura

V prvé řadě je zapotřebí zmínit adresářovou strukturu aplikace. Celý framework je

obsažen v adresáři `oe`. Ten obsahuje i další části, jejichž popis by vydal na další diplomovou práci a pro další pochopení aplikace a následný vývoj formulářových šablon není vyčerpávající znalost všech částí frameworku nutná. V případě nutnosti je možné vygenerovat dokumentaci API, protože veškeré třídy a metody jsou striktně komentovány. Jediné, co je zapotřebí vědět, je nastavení práv některých adresářů uvnitř `oe`, ale to je podrobně vysvětleno v kapitole zabývající se instalací. Aplikace je obsažena v adresáři `app`, ve kterém je následující členění.

`controllers` obsahuje soubory s definovanými třídami odvozenými od abstraktní třídy `OEController`, např. pojmenování souboru s třídou `FormsController` je `forms_controller.php`.

`helpers` obsahuje adresáře s helpery, např. definice helperu `MainMenuHelper` je v souboru `main_menu/main_menu_helper.php`. V adresáři `main_menu` je ještě adresář `templates`, ve kterém jsou šablony využívané vizuální částí helperu.

`models` obsahuje soubory s definovanými třídami modelů. Struktura je stejná jako v adresáři `controllers` – pouze soubory s definicemi tříd.

`others` obsahuje soubory s definicemi dalších tříd nesouvisejících s žádnou s výše vyjmenovanou částí frameworku. V případě formulářové aplikace se jedná hlavně o definici tříd struktury formuláře.

`views` je adresář s dalším adresářem `templates` obsahující adresáře pojmenované stejně jako controllery, v nichž jsou šablony stránek využívané controllery daného názvu. Ze zkušeností z několika projektů bude zváženo splynutí adresářové struktury s `controllers`, protože tyto šablony jsou logicky využívány (podobně jako jsou šablony u helperů) pouze svými controllery a je zbytečné tyto šablony mít tímto způsobem oddělené.

V adresáři `app` je ještě adresář `form_templates`, který nebyl diskutován. Je to z toho důvodu, že nemá s frameworkem jako takovým nic do činění, protože se jedná o adresář s definicemi šablon formulářů (podrobnější informace o definicích formulářů je k nalezení na straně 25).

Soubor `web_application.php`, který se jako jediný soubor také nalézá v adresáři `app`, zajišťuje konfiguraci aplikace. Je zde metoda `init()`, v níž se definují přístupové informace do databáze, míra logování informací do logu, je zde možnost zapnout dostupnost zobrazení logu ve spodní části obrazovky, nastavení časové zóny pro zobrazení

času, nastavení locale, nastavení URL, na které se aplikace nachází a kontakt na osobu zodpovědnou za chod systému a schopnou řešit problém, když se něco stane. Tento kontakt je zobrazen při nezachycení výjimky a její probublání kódem až k základní třídě spouštějící aplikaci.

Použití komponent frameworku

Před použitím komponent (model, view, controller nebo jiné třídy definované v adresáři `app/others`) je zapotřebí říct, že si PHP dané třídy musí natáhnout pomocí příkazu `require_once()`, aby je mohlo začít používat. Samozřejmě je zde na místě otázka, proč tomu tak je? Proč si framework nedokáže všechny tyto třídy natáhnout automaticky, aniž by byl tímto programátor obtěžován? Odpověď vyplyne sama, když se upřesní způsob fungování PHP preprocesoru.

PHP preprocesor při každém kliknutí uživatele na stránce načte příslušný soubor, nejčastěji `index.php`, a musí rozparsovat veškeré soubory, které programátor uvedl v `include()` nebo `require()` příkazech. Odpověď na výše položenou otázku je tedy jasná – bylo by to příliš procesorově nákladné a čím by byla aplikace větší, tím by byla pomalejší, i když by k tomu nebyl důvod. Snahou tedy je říkat si v kódu jen o ty komponenty, které jsou skutečně zapotřebí.

Pro účely frameworku, byly vytvořeny příkazy, se kterými se pracuje lépe, než se surovým `require_once()`. Stačí zadat název patřičné třídy, v metodě naznačit, o jaký typ třídy se jedná (model, view, controler nebo soubor v `app/others`) a framework se o zbytek postará. Jsou zde uvedeny příklady použití těchto příkazů.

`OEApplication::useModel("Form")` – připraví k použití třídu `FormModel`.

`OEApplication::useController("Forms")` – připraví k použití třídu `FormsController`.

`OEApplication::useHelper("MainMenu")` – připraví k použití třídu `MainMenuHelper`.

`OEApplication::useOther("Form")` – připraví k použití třídy v souboru `app/others/form.php`.

V případě, že nebude požadovaná komponenta nalezena, bude vyhozena výjimka.

2.3 Použité hotové komponenty v aplikaci

Jelikož tento framework existuje už nějakou dobu, jsou v něm za tu dobu některé univerzální komponenty, které lze bez problémů použít i v této formulářové aplikaci.

2.3.1 Třída `LoggedUser`

`LoggedUser` se používá pro zjišťování stavu přihlášení, při přihlašování do systému ale také při získávání objektu přihlášeného uživatele, tedy objekt `PersonModel`.

Třída byla inspirována návrhovým vzorem *singleton*. O tento vzor se v jeho čisté formě nejedná, ale je z něj vypůjčena jeho hlavní myšlenka – zajistí existenci objektu, který je prakticky globálně dostupný napříč celou aplikací.

Použití je velmi snadné a intuitivní, což je i vidět z následujícího výčtu.

`LoggedUser::getInstance()` – vrátí instanci (třídy `PersonModel`) přihlášeného uživatele. Pokud daný uživatel není přihlášen (jeho session cookie se nenalézá v záznamech v tabulce `persons` nebo čas posledního přístupu je starší než konstanta `LoggedUser::LOGIN_EXPIRE_TIME`, která je nastavena na jednu hodinu), bude vráceno `NULL`.

`LoggedUser::loginUserByPassword("jnovak", "tajneheslo");` – přihlásí uživatele pod zadanými přihlašovacími údaji. Přihlášený uživatel bude k nalezení po zavolání metody `getInstance()`. Pokud přihlášení selže, `getInstance()` vrátí `NULL`.

Výčet není kompletní, ale pro ilustraci dostačující.

2.3.2 Třída `AjaxHelper`

Možná nejzajímavější komponentou je komponenta zabývající se AJAXovými prvky stránek. Termín AJAX vznikl jako akronym ze slov *Asynchronous Javascript And XML*. Pracuje na principu volání příkazu `XMLHttpRequest` pomocí javascriptových událostí z HTML stránky a získání dat ze serveru následně zpracovávané javascriptem. Může se jednat např. o změnu obsahu elementu, změnu stylu elementu, přidání elementu do DOM stromu apod. Původně byly informace posílány ze serveru klientovi ve formě XML, proto to „X“ ve zkratce AJAX. XML však jako transportní formát bylo v této technologii vytěsněno formátem JSON [jso(2008)] nevyžadující takovou režii při parsování a přenosu jako XML.

AJAXové události jsou v tomto frameworku implementovány pomocí javascriptového frameworku Prototype [Pro(2008)] zjednodušující použití javascriptu a vytvářející kompatibilní sadu funkcí odladěných pro chod ve všech hlavních webových prohlížečích. Programátor webové aplikace se pak nemusí starat o kompatibilitu vzniklého javascriptového kódu, protože ji za něj zařídí právě tato knihovna.

Pomocí Prototype byla vytvořena rozšíření DOM elementu o následující metody.

`Element.ajaxRequest($("ElementId", "http://example.com/AjaxRef"))`
: provede dotaz na adresu `http://example.com/AjaxRefresh` a provede příkazy zakódovány serverem do JSONu.

`Element.ajaxFormRequest($("FormId", "http://example.com/AjaxRef"))`
: provede totéž, co předchozí případ, jen s tím rozdílem, že pošle s dotazem i serializovaný formulář.

Příkazy posílané serverem se konstruují právě pomocí třídy `AjaxHelper`. Jednoduchý příklad „Hello World“ zobrazující v klientovi dialogové okno s textem může vypadat následovně.

```
$ajax = new AjaxHelper();
$ajax->addCommandAlert("Hello World");
$ajax->display();
```

Tento kód pošle do prohlížeče javascriptové příkazy v podobě datové struktury kódované formátem JSON, které se postupně v prohlížeči pomocí funkce `eval()` provedou. Na straně prohlížeče tedy stačí knihovna Prototype a malý obslužný skript zpracovávající tuto strukturu. Veškerá další funkcionalita je dynamicky načítána ze serveru. Je to velmi efektivní a robustní přístup.

2.3.3 Třída `SmartGridHelper`

V programu je použita na několika místech komponenta `SmartGridHelper` zajišťující zobrazení seznamu položek. Ať už se jedná o seznam uživatelů v systému nebo seznam dokončených formulářů určitého typu, mají vždy stejné vlastnosti, které zahrnuje právě tato komponenta. Vždy je zapotřebí mít možnost řadit podle různých sloupců vzestupně a sestupně, mít možnost odfiltrovat některé položky, vyhledávat v položkách a podobně. Společné vlastnosti jsou definovány touto třídou, ostatní vlastnosti převezme z vlastností třídy odvozené z `OEModel` a zbytek informací potřebných pro vytvoření tabulky jsou dodány při definici.

Tato komponenta využívá pro svoji funkčnost AJAXová volání. Při změně parametru seznamu se pošle, kromě příslušného změněného parametru, i tzv. *viewstate*, který obsahuje serializované informace nastavení seznamu nakonfigurované při prvním použití. AJAXový controller provádějící změny parametrů a zobrazení tabulky s příslušnými hodnotami, pak může být jen jeden pro libovolný počet tabulek s rozdílnými typy zobrazovaných informací.

2.4 Správa uživatelských účtů

Tak jako pro každou aplikaci, ke které přistupuje více uživatelů, tak i zde je zapotřebí navrhnout systém uživatelských přístupů. Byl použit model, který se v praxi už několikrát osvědčil. Možná je pro tento účel zbytečně obecný a robustní, ale při jakýchkoli změnách či rozšiřování systému, se dá velice snadno použít. Jakékoli změny či přizpůsobení lze provézt pouhou úpravou řádků v tabulce databáze.

2.4.1 Struktura uživatelských účtů

Pro potřeby aplikace byl použit systém oprávnění umožnující mít

- libovolný počet *pravidel* definující prostředky, k nimž se má přistupovat,
- libovolný počet *typů uživatelů* definující oprávnění k prostředkům systému,
- libovolný počet *uživatelů* dostupných přes přihlašovací jméno a heslo.

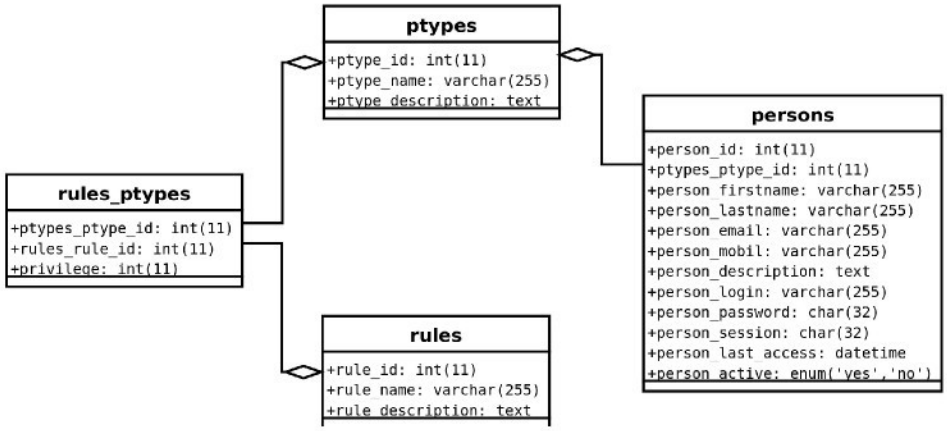
Díky tomuto rozvržení lze popsat jakoukoli kombinaci práv, která je (nebo bude) potřeba.

Pravidla označují místa, ke kterým lze uživateli povolit přístup. Standardně jsou všechny přístupy zakázány, protože ještě neexistují relace na typ uživatele nesoucí danou skutečnost.

Typ uživatele definuje, kam má daný typ uživatele přístup. Mezi typem uživatele a pravidlem je definována N:M relace, která ještě může obsahovat číselnou hodnotu (bitové pole) obsahující přesnější vymezení práv. V aplikaci jsou použity čtyři bity tohoto čísla. Nultý bit čísla definuje právo na vytváření (*Create*), první bit právo na čtení (*Read*), druhý na aktualizaci (*Update*) a třetí na mazání (*Delete*), např. č. 2 znamená právo pouze pro čtení, č. 7 znamená plný přístup k prostředku kromě práva mazání apod.

Uživatel je pak uživatelský účet přístupný pomocí jména a hesla a je v něm definováno o jaký *typ uživatele* se jedná.

Jak je vidět, díky tomuto datovému modelu je zřetelně poznat, jaké právo má uživatel k prostředku (pravidlu). Pomocí položek *typů uživatele* lze vytvořit typické nastavení práv. Třeba osoby mající přístup pouze pro tisk objednávek budou mít u svého typu uživatele nastavena pravidla pro formuláře typu objednávka pro čtení.



Obr. 1. Datový model oprávnění

Z obrázku 1 datového modelu jsou vidět některé atributy u `persons`, které na první pohled nejsou jasné a proto jsou v následujícím výčtu upřesněny.

`active` je atribut značící stav osoby, zda má či nemá oprávnění přihlásit se do systému. Nelze osobu po nějaký čas, když v systému není činná, jednoduše vymazat, protože za tu dobu své existence vytvořila v databázi záznamy se závislostí na dané osobě. Nešlo by tedy smazat osobu, protože by musely být vymazány i formuláře vytvořené touto osobou. Deaktivování osoby je tedy alternativou ke smazání.

`session` udává hodnotu session cookie přihlášené osoby, aby mohla být později identifikována. HTTP je bezstavový protokol a pro uchování si stavu je nutné tento nedostatek obejít a informace session cookie se na to hodí.

`last_access` udává čas posledního přístupu přihlášené osoby. Tímto parametrem je řešena funkce automatického odhlášení po definovaném čase. Hodnota je definována ve třídě `LoggedUser` a lze ji v případě potřeby změnit.

2.5 Navržení formulářů

Nejzajímavější částí celého systému je rozvržení samotných formulářů. Bylo zapotřebí zajistit nezávislost druhů formulářů na vlastní aplikaci. Muselo být navrženo, kde definovat

datovou strukturu formuláře (nebo, v textu také používaný termín, *druh formuláře*) definující vstupní pole pro vložení informací do formuláře, validační informace, implicitní hodnoty, popisky a nápovědy pro vstupní pole atd.,

vyplněná data uživatelem,

historii editace pro případ návratu na předchozí verzi,

tiskové sestavy formuláře pro výstup do PDF.

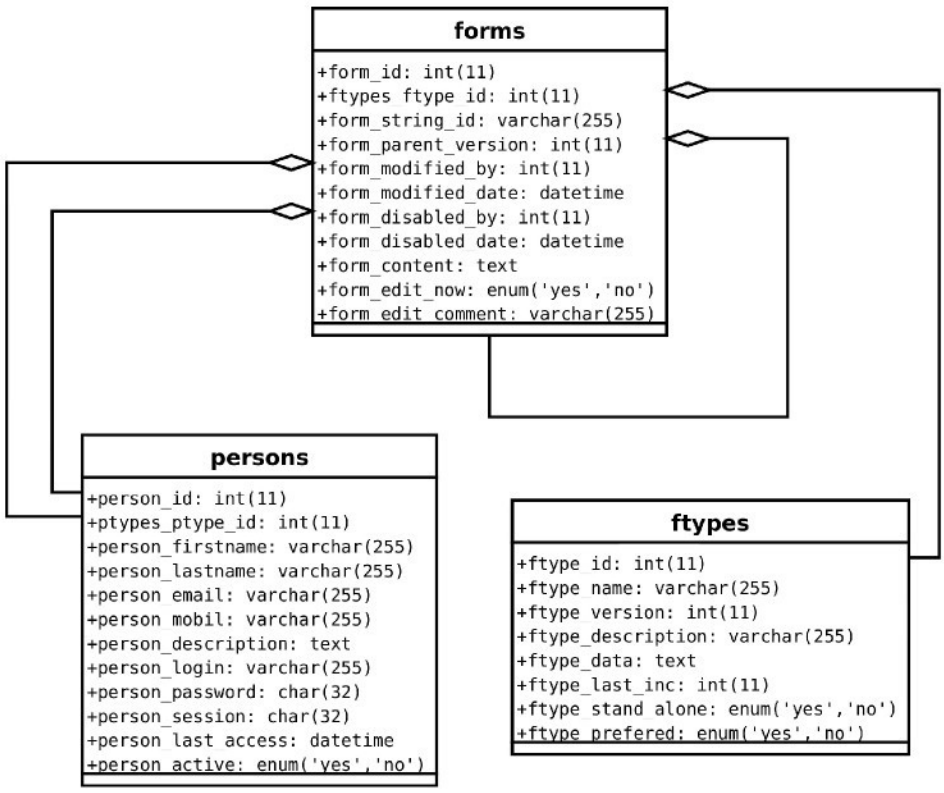
Než bylo nalezeno vyhovující řešení, byly promýšleny varianty, které vypadaly z počátku návrhu slibně, ale postupem času byly zavrženy. Jedno ze zavržených řešení se snažilo veškerou strukturu popsat v několika databázových tabulkách provázaných relacemi. Na tomto řešení není nic špatného až do doby, kdy by mělo být realizováno. Vzniklo by v databázi kolem dvaceti tabulek a středně komplexní formulář by vyžadoval stovky dotazů databáze. Bylo by to jak náročné na implementaci, tak na výkon databázového stroje a při tom toto řešení nic užitečného nepřinese.

Vhodným řešením se nakonec ukázala být kombinace databázového přístupu s využitím XML formátu pro uložení *datové struktury* a *vyplněných dat*. Tento XML dokument s definicí formuláře a vyplněnými daty je pak uložen do databáze včetně dalších metadat formuláře, protože databázový stroj nedokáže efektivně prohledávat XML. Historie je pak řešena ukládáním nových verzí do databáze a ponechání starých pro případnou potřebu. Obrovskou výhodou tohoto řešení, oproti předchozímu, je možnost využití pro tiskový výstup již existující technologii XSL-FO implementovanou několika produkty jak komerčních tak volně dostupných.

Z obrázku 2 jsou vidět atributy databázových tabulek týkající se formulářů. Na první pohled je vidět, že struktura je velmi jednoduchá. Tabulka `ftypes` definuje

`ftype_name` – jméno typu (druhu) formuláře určující typ šablony, který bude použit pro vytvoření nového formuláře.

`ftype_version` – verze typu formuláře. Počítá se s možností aktualizace formulářových šablon, kdy je zapotřebí pozměnit šablonu, ale obecný charakter formuláře



Obr. 2. Databázová struktura formulářů

zůstává stejný. Tímto způsobem je možno i po aktualizaci formuláře vytvořit formulář starší verze.

`ftype_description` je popisek daného typu a verze formuláře, který se objeví při výběru příslušné verze na dialogu pro vytvoření nového formuláře. (Na obrázku 3 na str. 46 je ilustrační příklad zobrazení vytvoření nového formuláře s popiskem „Formulář adresy zákazníka“.)

`ftype_data` obsahuje XML dokument definující šablonu struktury formuláře. Struktura tohoto dokumentu je vysvětlena v kapitole 2.6.

`ftype_last_inc` je pomocný čítač pro účel návrhu čísla formuláře při jeho vytváření. Každý druh formuláře má svůj předpis pro generování čísla formuláře (tedy nejedná se přímo o „číslo“, protože se jedná o řetězec, ale je zvyk tomu říkat číslo). Jedna z položek čísla formuláře je i inkrementální složka. Návrh takového čísla by byl dost časově náročný bez hodnoty naposledy použitého inkrementálního čísla, protože by se musela daná posledně použitá inkrementální složka složitě hledat v databázi a to by bylo velice neefektivní. Při zakládání nového čísla formuláře se tedy použije tento údaj, inkrementuje se a uloží nová inkrementální

složka pro příští použití. Kromě návrhu čísla formuláře, v podstatě odhadnutého programem, je zde i výpis posledních čísel naposledy vytvořených formulářů tohoto druhu. Uživatel tedy má možnost kontroly dané posloupnosti s možností změny navržené hodnoty. Aplikace ještě jednou před vlastním založením formuláře zkontroluje, zda toto číslo formuláře není někým používáno. Tato hodnota je vynulována v případě regenerování typů formulářů.

`ftype_stand_alone` definuje, zda je formulář plnohodnotný formulář (hodnota `yes`), nebo se jedná o formulář, který je pouze využíván ostatními formuláři tzv. *podružný formulář*, jež je pouze vkládán do ostatních formulářů.

`ftype_prefered` označuje preferovanou verzi daného druhu formuláře. Preferovaná verze bude přednostně vybírána při zakládání nového formuláře. Podmínkou však je, aby byla preferovaná verze pouze jedna v daném druhu formuláře. To lze zajistit dvěma způsoby, buď na databázové vrstvě nebo aplikačně. Bylo zvoleno aplikační řešení, kvůli snazší implementaci. Do budoucna by mělo být zváženo i řešení na databázové vrstvě, které je daleko konzistentnější.

Databázová tabulka `forms` pak obsahuje formuláře s udaji

`ftypes_ftype_id` – cizí klíč záznamu vytvořující šablony.

`form_string_id` – číslo formuláře.

`form_parent_version` – primární klíč záznamu formuláře, ze kterého tento záznam vznikl. Nově vytvořený formulář bude tuto hodnotu mít nastavenu na `NULL` a každá jeho další úprava bude zde mít hodnotu primárního klíče svého předchůdce.

`form_modified_by` – cizí klíč na uživatele, který provedl danou změnu ve formuláři.

`form_modified_date` – datum provedené změny

`form_disabled_by` – cizí klíč na uživatele, který daný záznam zneplatnil

`form_disabled_date` – datum zneplatnění

`form_content` – obsah XML dokumentu formuláře s vyplněnými hodnotami

`form_edit_now` je nastaveno na `yes`, když se jedná o rozpracovanou nefinální verzi formuláře. Dokončený formulář má tuto hodnotu `no`. Maximálně jeden formulář s příslušným `form_string_id` by měl mít hodnotu `no`. Toto je opět ošetřeno

aplikačně, aby k tomu nedošlo. Stálo by ale za to tuto vlastnost definovat na databázové úrovni.

`form_edit_comment` uchovává komentář k formuláři. Tento komentář bude vidět v seznamu formulářů daného druhu.

V úvodu bylo zmíněno, že definice šablon (druhů formulářů) budou mimo aplikaci a teď je vidět, že definice jsou vloženy v databázi, do které je dost nepohodlný přístup na to, aby se zde daly testovat nové verze apod. Je tomu skutečně tak. Aplikace sbírá informace o šablonách formulářů z databáze, ale existuje v adresářové struktuře aplikace umístění `app/form_templates`, kde jsou tyto definice v souborech. Při změně těchto definic v souborech stačí v aplikaci provést regenerování šablon. Aplikace při tom projde adresářovou strukturu, zjistí změny, provede některé elementární kontroly, jako jestli jméno adresáře koresponduje s *názvem typu formuláře* apod., a upraví si záznamy o typech formulářů.

2.6 Navržení XML jazyka pro popis formuláře

Pro účely uložení struktury formuláře a vyplněných údajů musel být navržen XML jazyk schopný efektivně pojmout všechny možné potřebné struktury formulářů, které mohou vzniknout. Po několika návrzích se povedlo vytvořit jazyk schopný popsat jakoukoli strukturu formuláře včetně možnosti vkládat do formuláře další formuláře. Hodí se to ve chvíli, kdy jsou někde v podružných formulářích definované např. poštovní adresy nebo údaje o osobách. Při vyplňování formuláře pak stačí vybrat z položek podružného formuláře daného typu a do XML dokumentu se celý tento dokument vloží. To má pak pozitivní vliv na další práci, protože tyto údaje z vloženého formuláře jsou k dispozici jak jakémukoli XML nástroji (např. pro XSL transformace) tak internímu API, který lze ve formulářích také použít ve formě maker.

Pro potřeby vývoje formulářových šablon bylo vytvořeno DTD popisující tento jazyk a vývojové nástroje ve spojení s tímto DTD umí velmi pěkně spolupracovat, takže případná chyba ve formátu dokumentu bude odchycena přímo ve vývojovém prostředí. Velmi dobré zkušenosti jsou s vývojovým prostředím Eclipse, ve kterém byla tato aplikace kompletně napsána.

Protože XML jazyk lze navrhnout různými způsoby, formát XML je velmi pružný a dokáže popsat to samé několika různými způsoby, např. hodnota může být reprezentována jako atribut elementu nebo jako obsah dětského elementu tohoto elementu,

bylo přistoupeno ke konvenci, kdy složitější datové prvky vyžadující víceřádkové členění apod., budou uloženy v obsahu dětského elementu a hodnoty typu jednoslovných identifikátorů, čísel apod. budou uloženy v atributu dětského elementu.

2.6.1 Práce s XML dokumentem šablony

Pro smysluplnou práci s touto XML strukturou v aplikaci, byl použit postup umožňující načtení XML dokumentu do datové struktury a po její použití zpět uložení do aktualizovaného XML dokumentu.

Původní vize této struktury počítaly s několika třídami, které by se vytvářely na základě předaného parametru typu `DOMElement` a opět se ukládaly vrácením objektu `DOMElement`. Tato metodika byla sice správná, ale počet tříd byl hodně jiný, než bylo původně předpokládáno. Až na několik výjimek platí pravidlo: „Co element obsahující další elementy, to třída.“ Celkem to dalo na 24 tříd a jedno rozhraní. Zbytek této kapitoly 2.6 bude o stručném seznámení se s těmito třídami, protože je lze použít při vytváření maker a je dobré mít o nich povědomí při vytváření šablon. Poznámky k těmto třídám jsou totiž i k elementům XML dokumentu šablony formuláře.

2.6.2 Rozhraní `iXMLSerializable` a myšlenka mapování tříd do XML

Vzhledem k tomu, že všechny třídy budou mít vlastnosti týkající se ukládání a načítání do XML stejné, bylo navrženo rozhraní `iXMLSerializable`. Definuje tři metody

`setObjectFromXML(DOMElement $element)` s parametrem typu `DOMElement`, který je referencí na element v rozparsovaném stromu dokumentu. Tato metoda naplní svoje atributy hodnotami z tohoto elementu a rekurzivně vytvoří další podřízené atributy definované třídami mapující XML strom dokumentu.

`getObjectToXML()` je metoda opačného účelu. Má za úkol sestavit a vrátit `DOMElement` obsahující hodnoty atributů objektu.

`getXMLDocument()` vrací hodnotu objektu typu `DOMDocument`, který musí být přístupný jen jeden pro všechny elementy dokumentu. Tato metoda tedy pouze zavolá sama sebe na rodičovském objektu a objekt, který už nemá svého rodiče – pouze jeden typu `FormStruct` – tento atribut vrátí, protože ho jako jediný vlastní.

V interface není definována ještě jedna metoda, které by měla být ve všech těchto třídách implementována stejně, a tou je konstruktor. Není v rozhraní definována, protože

v jednom případě je nutné, aby měla ještě jeden parametr navíc a není jisté, jestli by tato konstrukce byla PHP interpretem pochopena správně. Konstruktor by tedy měl mít dva parametry. První je typu obecně `iXMLSerializable`, ale bývá obvykle definován jako třída obalující tuto třídu (třída, ve které se tato třída, jako atribut, nachází) – pro připomenutí, toto rozhraní je supertypem těchto tříd. Druhý parametr je nepovinný parametr typu `DOMElement`. Pokud tento parametr je nenulový, zavolá se v konstruktoru i metoda `setObjectFromXML()` plnící objekt daty z tohoto elementu.

2.6.3 Třída `FormStruct`, element `Form`

`FormStruct` reprezentuje kořenový element `Form`. Na rozdíl od ostatních tříd implementující `iXMLSerializable` obsahuje konstruktor bez parametrů. Objekt je totiž plněn pomocí metody `loadFromXMLString()` s XML dokumentem v řetězci jako parametrem. Tím lze načíst XML dokument jak ze záznamu v databázi tak ze souboru. Druhou možností, jak naplnit objekt daty je metodou `setObjectFromXML()` definovanou v implementovaném rozhraní. Využívá se v případě použití tohoto objektu ve formě vloženého podružného formuláře. Metodou `getXMLOfForm()` lze získat XML dokument ve formě řetězce.

Důležité atributy třídy

`database_key` (element `DatabaseKey`) obsahuje hodnotu primárního klíče záznamu formuláře v databázi. Element existuje pouze v případě, že se jedná o formulář s hodnotami.

`form_header` (element `FormHeader`) obsahuje objekt typu `FormHeader` definující základní údaje o šabloně a vyplněném formuláři

`form_data_types` je pole objektů typu `FormDataType` (elementy `FormDataType` v elementu `FormDataTypes`) definují datové typy vstupních polí použité ve struktuře formuláře včetně HTML prvku použitého pro vstup dat.

`form_struct_data_definition` (element `FormStructDataDefinition`) je objekt typu `FormStructDataDefinition` a definuje strukturu formuláře včetně vyplněných údajů.

2.6.4 Třída `FormDataType`, element `FormDataType`

`FormDataType` uchovává informace o datovém typu vstupního editačního pole formuláře včetně jeho HTML reprezentace.

Důležité atributy třídy

`data_type_name` (atribut `id`) označuje identifikátor datového typu. Tento identifikátor je využívaný dále v šabloně a uživateli se nikde nezobrazuje.

`data_type_regexp` (element `Regexp`) udává regulární výraz pro validaci vstupních dat.

`data_type_public_name` (element `PublicName`) uchovává veřejně zobrazený název datového typu.

`data_type_public_description` (element `PublicDescription`) obsahuje podrobnější vysvětlení datového typu.

`data_form_input_representation` (element `InputRepresentation`) obsahuje objekt typu `FormDataInputRepresentation` definuje, jakým způsobem bude vstupní datová položka reprezentována v editačním dialogu aplikace.

2.6.5 Třída `FormDataInputRepresentation`, element `InputRepresentation`

`FormDataInputRepresentation` definuje, jakým způsobem bude zařízen vstup informací od uživatele vyplňující pole daného editovaného formuláře. Lze zde definovat několik možných podob vstupního pole formuláře. To je provedeno atributem `form_input`, který je typu `AbstractFormInput`, což je abstraktní třída tvořící základ pro třídy definující konkrétní typ vstupního formulářového prvku.

Jedny z hlavních metod poděděných z abstraktní třídy `AbstractFormInput` jsou

`setTemplateFilename()` se jménem souboru šablony v argumentu. V daném jménu souboru se nachází šablona s formulářovým prvkem příslušného vstupního pole.

`fetchHtmlInput()` s parametrem typu `DataItem` definující příslušný prvek v hierarchii formuláře. Vrátí HTML formulářový prvek pro vstup hodnoty daného prvku.

Potomci třídy `AbstractFormInput` jsou

`FormInputHidden` (element `InputHidden`) je hodnota formuláře, která je skrytá a není dostupná pro editaci uživateli. Touto hodnotou však lze manipulovat pomocí maker a dále může být využitá nástroji manipulující s XML reprezentací formuláře.

`FormInputReadOnlyText` (element `ReadOnlyText`) podobně jako předchozí prvek, slouží jako hodnota pouze pro čtení a není určena pro editaci. Hodnotu však uživatel uvidí.

`FormInputText` (element `InputText`) je jednořádkové vstupní pole omezeno délkou 255 znaky (omezení HTML formulářového prvku `input type="text"`).

`FormInputTextarea` (element `InputTextArea`) je víceřádkové vstupní pole. Tato třída má možnost nastavit si počet sloupců (metoda `setCols()`) a řádků (metoda `setCols()`) vstupního pole.

Další vícenásobné výběry modelu jedna hodnota z N nebo více hodnot z N. Tyto specifické třídy jsou odvozeny od další abstraktní třídy `AbstractFormInputMultisel` definující navíc metody

`addItem()` se dvěma parametry, první je hodnota vybrané položky a druhý je název položky, který bude vidět uživatelem.

`getItems()` vrátí pole položek nabízené ve výběru. Pole je ve formátu asociativního pole (klíč => hodnota), kde klíč je hodnota pole a hodnota je popisek viděný uživatelem.

Potomci jsou pak

`FormInputCheckbox` (element `InputCheckbox`) vytváří vstup pomocí neexkluzivního výběru (žádná až N položek) z hodnot pomocí checkboxů.

`FormInputRadio` (element `InputRadio`) vytvoří exkluzivní výběr (výběr pouze jedné hodnoty) pomocí radiobuttonů.

2.6.6 Třída `FormHeader`, element `FormHeader`

`FormHeader` obsahuje základní informace o šabloně formuláře, jeho verzi, popisky šablony a vyplněného formuláře..., zkrátka základní informace o formuláři.

Důležité atributy třídy

`template_name` (element `TemplateName`) je název šablony (druhu formuláře). Může obsahovat i diakritiku. Tu je však zapotřebí dodržet i v názvu jména adresáře a pod.

`template_version` (element `TemplateVersion`) je verze formuláře ve formě celočíselné hodnoty.

`template_public_description` (element `TemplateDescription`) je popis formuláře. Může se jednat i o víceřádkový popis, který bude zobrazen pod výběrem verze při vytváření nového formuláře. Mělo by zde být uvedeno, pro jaké účely by měl být formulář použit. Proč by měla být použita právě tato verze apod.

`form_description` (element `FormDescription`) obsahuje popisek konkrétního vyplněného formuláře. Tímto popiskem bude formulář prezentován v seznamu formulářů a v případě podružných formulářů tento popisek bude v comboboxu výběru. Popisek by tedy měl být co nejvíce výstižný.

`string_id_pattern` (element `StringIdPattern`) je objekt typu `FormStringIdPattern` definující předpis pro konstrukci čísla formuláře.

`string_id` (element `StringId`) je číslo formuláře.

`stand_alone` (element `StandAlone`) definuje, zda se jedná o podružný formulář (hodnota `no`) nebo ne.

`dependencies` (element `Dependencies`) je objekt typu `FormDependencies` definující závislosti na jiných šablonách. Tyto podmínky dokáží zpřesnit výběr při vkládání podružných formulářů.

2.6.7 Třída `FormStringIdPattern`, element `StringIdPattern`

`FormStringIdPattern` určuje, jakým způsobem bude skládáno číslo formuláře. To je možné složit ze tří stavebních kamenů. Statického řetězce, části data z generované pomocí funkce `strftime()` a inkrementálního čísla.

Důležité atributy třídy

`patterns` je pole jednoprvkových asociativních polí, kde klíč asociativního pole udává typ „stavebního kamene“ čísla formuláře

Klíč `string` (element `String`) definuje řetězec vložený do čísla formuláře.

Klíč `strftime` (element `IncrementalNumber`) definuje formátovací argument funkce `strftime()`[12] pro čas vložený jako druhý argument této funkce. Jako časová hodnota je použita aktuální hodnota času.

Klíč `inc_num` (element `IncrementalNumber`) vloží číslo podle formátovacího řetězce funkce `sprintf()`[13]. Číslo se bere z tabulky `ftypes` sloupce `ftype_last_inc` a při každém získání tohoto čísla se ověřuje, zda toto číslo může být použito, tzn. ověří, že číslo formuláře se s tímto inkrementálním číslem už v databázi nevyskytuje. Pokud se vyskytuje, inkrementuje se o jednu a zkouší to dále dokud nenajde číslo, které by dalo unikátní číslo formuláře. Jakmile ho najde, hodnotu inkrementálního čísla si uloží do databáze pro pozdější použití, protože hledání inkrementálního čísla tímto způsobem je velice časově nákladné.

Tímto způsobem se jednotlivé stavební kameny skládají za sebe a výsledný řetězec (nabídnuté číslo formuláře) je pak posloupností výsledků těchto kamenů.

2.6.8 Třída `FormDependencies`, element `Dependencies`

`FormDependencies` představuje definici závislostí tohoto formuláře na jiných formulářích. Lze zde definovat, že tento formulář pro svou funkčnost vyžaduje jiný formulář určité verze. Verze lze definovat standardními matematickými operátory porovnání. Je tedy možné určit, že pro svou funkčnost formulář potřebuje jiné formuláře verze 2 až 5 a jiný formulář verze větší než 3. Vše je zde možno nadefinovat. Jednotlivá nadefinovaná pravidla jsou omezena pouze na spojku logického součinu (*and*) a nelze vyrobit negaci. Nejedná se tedy o úplný systém logických spojek, pro vyjádření běžných potřeb však zcela postačující.

Důležité atributy a metody třídy

`dependencies` je pole asociativních polí, kde klíče asociativních polí jsou

`template_name` (element `TemplateName`) je jméno šablony

[12] Na stránce `http://cz.php.net/manual/en/function.strftime.php` jsou podrobnosti o formátovacím řetězci této funkce.

[13] Na stránce `http://cz.php.net/manual/en/function.sprintf.php` jsou podrobnosti o formátovacím řetězci této funkce.

`template_version` (element `TemplateVersion`) je verze šablony použitá při porovnávání

`operator` (element `Operator`) je operátor použit při porovnávání. Operátory z důvodu bezproblémového použití v DTD a jinde jsou zapsány zkratkou, ne jejich skutečným symbolem pro daný operátor. Zkratky tedy jsou

`eq` – je rovno (=)

`neq` – není rovno ($\neq$)

`lt` – menší než (<)

`gt` – větší než (>)

`lte` – měnší nebo rovno ($\leq$)

`gte` – větší nebo rovno ($\geq$)

metoda `getDependencyConstrainOf()` se jménem šablony jako parametrem lze získat redukované pole `dependencies` pouze na položky se zadaným jménem šablony.

metoda `getListOfAvailableVersions()` se jménem šablony jako parametrem vrátí primární klíče záznamů z tabulky `ftypes`, které vyhovují omezením zadaných v `dependencies` a mají dané jméno šablony.

2.6.9 Třída `FormStructDataDefinition`, element `FormStructDataDefinition`

`FormStructDataDefinition` definuje základní strukturu uložených dat ve formuláři, popisky k jednotlivým vstupním polím, jejich typy a spouštěná makra. Toto je základní struktura nesoucí jak strukturu tak vlastní vyplněná data. V XML stromu je zde povolen pouze jeden prvek – `DataStruct`.

Důležité atributy třídy

`data_struct` (element `DataStruct`) je objektem typu `FormDataStruct` a definuje prvek struktury formuláře.

2.6.10 Třída `AbstractData`

`AbstractData` je abstraktní třída definující základ pro základní čtyři stavební kameny struktury formuláře.

`DataItem` definující datový prvek formuláře stanoveného datového typu obsahující hodnotu, kterou má možnost uživatel změnit.

`DataStruct` definující strukturovaný prvek, ve kterém mohou být obsaženy všechny možné prvky typu `AbstractData`, tedy `DataStruct`, `DataArray`, `DataItem` a `ForeignForm`.

`DataArray` umožňuje konstruovat vícenásobné seznamy. V této položce může být pouze `DataStruct`, který může být uživatelem jednou až libovolně-krát vložen.

`ForeignForm` umožňuje vložit cizí formulář zadaného druhu.

Důležité atributy a metody třídy

`parent` – rodičovský prvek prvku. Hodnota je typu `AbstractData`.

`name` – název položky datové struktury. Tato hodnota je použita při procházení stromovou strukturou.

`edit_label` – popisek prvku v editovacím dialogu aplikace. Popisek by měl pár slovy vystihovat, co se od uživatele požaduje vyplnit.

`edit_description` – delší popisek prvku. Měl by být více vysvětlující. Může se jednat o víceřádkový text podrobně popisující daný problém.

Metoda `addErrorMessage()` přidá chybovou hlášku vloženou do argumentu k danému prvku.

Metoda `addNoticeMessage()` přidá poznámku vloženou do argumentu k danému prvku. Má využití třeba při ohlášení určité události, kterou by uživatel měl vzít na vědomí, ale nejedná se o chybu blokující dokončení editace formuláře.

Metoda `getDataByPath()` s parametrem pole řetězců. Pole řetězců udává cestu relativně k tomuto prvku, např. `array("..")` vrátí rodičovský prvek, `array("..", "zelenina")` vrátí prvek `zelenina` v rodičovském prvku, `array("/", "nakup", 0, "cena")` vrátí prvek `cena` z nultého prvku pole `nakup`, který je obsažen v kořenovém prvku. Vrácené prvky jsou typu `AbstractData`.

Metoda `removeFilledValue()` odstraní všechny vyplněné hodnoty tohoto a všech podřízených prvků formuláře.

Metoda `getPath()` vrátí pole řetězců, které určují cestu k tomuto prvku. Jedná se o stejné pole řetězců, které je potřebné pro získání tohoto prvku z libovolného prvku formuláře pomocí metody `getDataByPath()`. Cesta je vyjádřena absolutně.

Sada metod `isStruct()`, `isArray()`, `isItem()` a `isForeignForm()` určují o jaké typy prvku se tu jedná. Návratová hodnota je typu `boolean`.

Metoda `addItem()` s parametrem typu `AbstractData` přidá do tohoto prvku prvek vložený v argumentu. To může být použito při pokročilejších technikách manipulace se strukturou formuláře pomocí maker.

Metoda `getItems()` vrátí pole objektů typu `AbstractData`, což jsou podřízené prvky této položce.

Metoda `getItem()` s indexem prvku jako parametrem vrátí prvek příslušného indexu. Prvek je typu `AbstractData`.

Metoda `removeItem()` odstraní prvek s indexem vloženým do argumentu z pořízených prvků.

Metoda `swapItem()` prohodí dva prvky mezi sebou. Indexy obou prvků jsou vloženy jako dva argumenty této metody.

Metoda `getParent()` vrátí vlastníka této metody (typu `AbstractData`). Kořenový prvek formuláře vrací `NULL`.

Metoda `isErrorneous()` vrátí `TRUE`, pokud v tomto nebo podřízených prvcích je obsažena chyba (je nějaký text v chybové hlášce prvku).

Jak je vidět z výčtu metod, předává tato třída svým potomkům velmi mocné nástroje při pohybu a manipulaci strukturou formuláře. Toho lze s výhodou použít při vytváření maker, které se tak stávají velmi mocnými nástroji formuláře.

2.6.11 Třída `DataStruct`, element `DataStruct`

`DataStruct` je třída odvozená z `AbstractData`. Je to strukturový prvek formuláře. V tomto prvku mohou být vloženy další struktury (element `DataStruct`), položky formuláře (element `DataItem`) nebo pole (element `DataArray`) umožňující uživateli zadat libovolný počet prvků tohoto pole pomocí voleb na formuláři.

Důležité metody třídy

`amIArrayItem()` vrátí `TRUE`, pokud tato struktura je prvkem pole. Jen pro připomenutí, v poli (`DataArray`) může být obsažena jen struktura (`DataStruct`).

`getIndexInArray()` vrátí index tohoto prvku v poli. Tuto metodu lze volat jen v případě, že tento prvek je prvkem pole, jinak bude vyhozena výjimka.

Dále jsou dostupné atributy a metody poděděné z `AbstractData`.

2.6.12 Třída `DataItem`, element `DataItem`

`DataItem` je potomek třídy `AbstractData` a slouží pro uchování informací o vstupním datovém prvku formuláře. Je v něm uložena informace o datovém typu položky, z čehož plynou informace o vzhledu vstupního editačního prvku a možnostech jeho editace, implicitní hodnotě a makrech dvou typů – makro spuštěné při změně vyplněné hodnoty uživatelem a druhé makro je spouštěno v iteracích, dokud se hodnota vyplněných polí nepřestane měnit nebo počet spouštění maker nepřekročí určitou mez (řádově desítky spuštění).

Důležité atributy a metody třídy

`data_type` (atribut `dataType`) je objekt typu `FormDataType` definující datový typ a způsob vstupu informací do vstupního pole.

`default_value` (element `DefaultValue`) je implicitní hodnota použitá při založení formuláře nebo po vymazání prvku metodou `removeFilledValue()`.

`filled_value` (element `FilledValue`) je vyplněná hodnota uživatelem.

`on_change_code` (element `OnChangeCode`) je kód makra spouštěný při změně hodnoty `filled_value` uživatelem. Jedná se o PHP kód spuštěn uvnitř tohoto objektu pomocí PHP funkce `eval()`. V kódu lze využít veškerou API dostupného frameworku.

`execute_code` (element `ExecuteCode`) je kód makra spouštěný při každém načtení šablony. Když jsou spuštěna všechna makra v `execute_code` a toto spuštění maker vedlo ke změně hodnoty `filled_value`, bude celý cyklus spuštění maker opakován, dokud se někde ve formuláři tyto hodnoty budou měnit. Je zde zabudován mechanismus, který ochrání aplikaci před chybou návrhu šablony při zacyklení. Po provedení několika desítek cyklů bude spouštění maker ukončeno.

Metoda `setFilledValue()` používaná pro naplnění hodnoty `filled_value`. Pokud tato metoda nebude použita, nemusí správně fungovat některá makra! Inkremen-

tuje totiž vnitřní čítač, podle kterého se zjišťují změny v hodnoty. Podle údaje o změně hodnoty se dále rozhoduje, jestli provézt i ostatní makra.

Dále jsou dostupné atributy a metody poděděné z `AbstractData`.

2.6.13 Třída `DataArray`, element `DataArray`

`DataArray` je potomek třídy `AbstractData` poskytující možnost uživateli vytvářet seznamy opakujících se typů prvků. V dialogu se uživateli zobrazí barevně zvýrazněný rámeček značící, že se jedná o opakující se položky, a zobrazí se možnosti *přidat položku*, *odebrat položku* a možnosti *řazení položek*. V tomto elementu se nachází element `DataStruct`.

Důležité metody třídy

Metoda `addClonedRow()` s indexem prvku jako parametr vytvoří kopii prvku zadaného v indexu a vloží ho na konec seznamu. Vyplněné hodnoty budou zkopírovány.

Metoda `addNewRow()` přidá na konec seznamu prvek pole. Prvek pole bude mít vyplněné hodnoty vynulovány (nastaveny na implicitní).

Metoda `removeRow()` s indexem prvku jako parametr zajistí odstranění prvku se zadaným indexem.

Dále jsou dostupné atributy a metody poděděné z `AbstractData`.

V DTD je možné si všimnout, že u tohoto elementu jsou ještě dva atributy, `min` a `max`. V tuto chvíli zatím nejsou nijak využívány. Při návrhu se počítalo, že by mohlo být užitečné mít nástroj na omezení počtu položek v poli. Na implementaci této funkce už ale nedošlo, protože se postupem času vyvstaly pochybnosti o užitečnosti této funkce. V případě, že se ukáže tato funkce jako důležitá, je možné toto doimplementovat. Stejné funkcionality lze totiž dosáhnout i pomocí maker a to daleko přesněji ušité na míru potřebám konkrétního formuláře.

2.6.14 Třída `ForeignForm`, element `ForeignForm`

`ForeignForm` je potomek třídy `AbstractData` a definuje možnost vložení podružného formuláře. V dialogu pro zadávání dat se na tomto místě objeví combobox s možnými formuláři pro vložení, které jsou reprezentovány popiskem dostupným při editaci

jako první upravitelná hodnota každého formuláře. Podružné formuláře jsou vybírány s ohledem na hodnoty omezení `Dependencies`.

Důležité atributy a metody třídy

`template_name` (element `TemplateName`) obsahuje název typu formuláře (šablony) vybírané v comboboxu.

`form` (element `Form`) obsahuje vložený formulář (objekt typu `FormStruct`), pokud je uživatelem vůbec nějaký formulář vybrán. Jinak obsahuje `NULL`.

Metoda `setFilledValue()` s primárním klíčem záznamu v tabulce `forms` vloží sem tento formulář.

Metoda `getFilledValue()` vrátí primární klíč vloženého formuláře nebo `NULL` v případě neexistence vloženého formuláře.

Metoda `getForm()` vrátí vložený formulář (objekt typu `FormStruct`) nebo `NULL` v případě neexistence vloženého formuláře.

Dále jsou dostupné atributy a metody poděděné z `AbstractData`.

2.7 Tvorba šablon

Pro tvorbu šablon je doporučováno využít DTD dostupné lokálně v adresářové struktuře aplikace nebo vzdálené umístění na hostingu u autora aplikace. Hlavička XML souboru šablony by měla vypadat takto.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Form PUBLIC "-//Obratil//Easy Forms 1.0"
    "http://www.obratil.cz/easy_forms/form.dtd" >
```

Pro vytvoření nové šablony je doporučeno použití vývojového prostředí Eclipse[14], které umí velmi pěkně pracovat s XML. Při vložení této hlavičky si vývojové prostředí stáhne z uvedené adresy DTD a automaticky začne nabízet jednotlivé tagy a jejich atributy, takže psaní nové šablony se tím velice urychlí.

[14]Vývojové prostředí Eclipse s plug-inem PDT pro podporu PHP lze volně stáhnout z `http://www.eclipse.org/pdt/`.

Pro ilustraci tvorby šablon bude v této kapitole vytvořena šablona pro podružný formulář *Adresa klienta*. Tento podružný formulář může být následně použit pro vložení do formuláře pomocí `ForeignForm` tagu. V tomto formuláři bude obsažena adresa klienta složena z několika řádek (řádků může být více či méně, záleží to na uživateli, kolik řádků adresy vloží), PSČ, telefon, email a webové stránky. Dále v příkladu bude požadavek na uložení informace o charakteru zákazníka – jestli je to současný, minulý nebo budoucí zákazník a informace o tom, zda tento zákazník dostává e-mailem upozornění na nové produkty, či nikoli.

První částí šablony je definice datových typů. V tomto případě je zapotřebí definovat datový typ pro *řádky adresy*, *PSČ*, *telefonní číslo*, *e-mail*, *URL*, *status zákazníka* (výčet „současný“, „minulý“, „budoucí“) a informaci o tom, zda dostává *informace o nových službách po emailu nebo poštou*. Definice v XML kódu tedy může vypadat následovně.

```
<Form>
  <FormDataTypes>
    <FormDataType id="line">
      <Regexp value="/^[^\n\r]{0,255}$/"/>
      <PublicName value="Obecný řetězec na jedu řádku"/>
      <PublicDescription>
      Libovolné znaky uložitelné na jednu řádku webového
      formuláře (max 255 znaků).
      </PublicDescription>
      <InputRepresentation>
       <InputText/>
      </InputRepresentation>
    </FormDataType>
    <FormDataType id="psc">
      <Regexp value="/^[0-9]{3}[\ ]?[0-9]{2}$/"/>
      <PublicName value="PSČ"/>
      <PublicDescription>
      Poštovní směrovací číslo.
      </PublicDescription>
      <InputRepresentation>
       <InputText/>
      </InputRepresentation>
```

```
</FormDataType>
<FormDataType id="email">
  <Regexp value="/^([a-z0-9][a-z0-9_\-\.\+]*)
      @([a-z0-9][a-z0-9\.\-]{0,63}\.([a-z]{2,4}))$/i"/>
  <PublicName value="E-mail"/>
  <PublicDescription>E-mailová adresa</PublicDescription>
  <InputRepresentation>
   <InputText/>
  </InputRepresentation>
</FormDataType>
<FormDataType id="www">
  <Regexp value="/^[^\n\r]{0,255}$/"/>
  <PublicName value="WWW"/>
  <PublicDescription>Webová adresa</PublicDescription>
  <InputRepresentation>
   <InputText/>
  </InputRepresentation>
</FormDataType>
<FormDataType id="telefon">
  <Regexp value="/^((\+|00)[0-9]{3})?([\ ]?[0-9]{3}){3}$/"/>
  <PublicName value="Telefonní číslo"/>
  <PublicDescription>
  Telefonní číslo v˜běžně používaném formátu
  </PublicDescription>
  <InputRepresentation>
   <InputText/>
  </InputRepresentation>
</FormDataType>
<FormDataType id="stav_zakaznika">
  <Regexp value="/^(současný|minulý|budoucí)$/"/>
  <PublicName value="Stav zákazníka"/>
  <PublicDescription>Stav zákazníka</PublicDescription>
  <InputRepresentation>
   <InputRadio>
     <Option value="současný" label="Současný zákazník"/>
     <Option value="minulý" label="Minulý zákazník"/>
```

```
        <Option value="budoucí"
            label="Budoucí (potenciální) zákazník"/>
       </InputRadio>
      </InputRepresentation>
    </FormDataType>
    <FormDataType id="upozorneni_na_novinky">
      <Regexp value="/^.*$/"/>
      <PublicName value="Upozornění"/>
      <PublicDescription>
      Způsob upozorňování zákazníka na novinky
      </PublicDescription>
      <InputRepresentation>
       <InputCheckbox>
         <Option value="email" label="Emailem"/>
         <Option value="pošta" label="Papírovou poštou"/>
       </InputCheckbox>
      </InputRepresentation>
    </FormDataType>
  </FormDataTypes>
```

Jak je vidět z výpisu, jsou zde nadefinovány jednotlivé datové typy vstupních polí. (Z typografických důvodů jsou některé řádky zalomeny, konkrétně regulární výraz emailu.) Dále byl trochu zanedbán regulární výraz pro URL. Ten by byl ještě delší než výraz u e-mailu, ale v případě nutnosti lze doplnit.

Šablona formuláře dále obsahuje úsek s definicí hlavních informací o šabloně. Jméno šablony `adresa zákazníka` musí být dodrženo ještě v dalším kontextu.

- Je použito pro reference ve formulářích, kde se formuláře tohoto typu vkládají ve formě podružných formulářů.
- Aby měl uživatel přístup k formuláři, musí být typem uživatele, který na to má právo. Musí tedy existovat relace mezi příslušným záznamem typu uživatele z tabulky `ptypes` přes relační tabulku `rules_ptypes` se správně nastavenou hodnotou atributu `privilege` na záznam v tabulce `rules` s atributem `rule_name` rovným `form_adresa zákazníka`.
- Tento formulář musí být uložen v souboru s přesným názvem a umístěním.

V tomto případě je to
`app/form_templates/adresa zákazníka/1/adresa zákazníka.xml`.

Jak je vidět, diakritika by neměla vadit. V tomto ilustračním příkladu je diakritika použita, kde to jen jde, v praxi by se této možnosti mělo spíše vyhýbat, protože to může způsobit na některých místech problémy. V první řadě je zapotřebí dodržovat všude kódování UTF-8. Dále podpora UTF-8 by měla být všude, kde vyžadujeme tuto funkčnost diakritiky, a to včetně operačního systému, který by toto kódování měl nativně na všech úrovních podporovat. V případě nasazení této aplikace je zvolen Ubuntu Linux, který s UTF-8 nemá žádné problémy.

Další hodnotou v sekci hlavičky šablony je předpis pro generování čísla formuláře, po té údajem, že se jedná o formulář využívaný hlavně jako *podružný formulář*, a nakonec prázdná hodnota v `Dependencies`. Tento formulář tedy ke své funkčnosti nevyžaduje žádný další typ formuláře.

```
<FormHeader>
  <TemplateName value="adresa zákazníka"/>
  <TemplateVersion value="1"/>
  <TemplateDescription>Formulář adresy zákazníka</TemplateDescription>
  <StringIdPattern>
    <String value="ADRZ/"/>
    <IncrementalNumber value="%03d"/>
  </StringIdPattern>
  <StandAlone value="no"/>
  <Dependencies>
  </Dependencies>
</FormHeader>
```

Další částí definice šablony je definice vlastní datové struktury šablony. Pro tento ilustrační příklad byla vybrána následující struktura, kdy na umístění `array("/", "adresa")` je pole jednotlivých řádků adresy následované PSČ, e-mailem, adresou webových stránek a telefonem. Mimo adresu jsou ještě dva údaje `stav_zakaznika` a `upozorneni_na_novinky`.

```
<FormStructDataDefinition>
 <DataStruct name="koren">
  <EditLabel>Zákazník</EditLabel>
```

```
<EditDescription>
Uveďte zde informace o˜zákazníkovi.
</EditDescription>
  <DataStruct name="adresa">
   <EditLabel>Adresa zákazníka</EditLabel>
   <DataArray name="radky_adresy">
      <EditLabel>Řádky adresy</EditLabel>
      <DataStruct name="struktura_radku">
       <EditLabel>Řádek</EditLabel>
       <DataItem dataType="line" name="polozka_radku">
          <EditLabel>Položka řádku</EditLabel>
       </DataItem>
      </DataStruct>
   </DataArray>
   <DataItem dataType="psc" name="psc">
     <EditLabel>PSČ</EditLabel>
     <EditDescription>Poštovní směrovací číslo</EditDescription>
   </DataItem>
   <DataItem dataType="email" name="email">
     <EditLabel>E-mail</EditLabel>
     <EditDescription>E-mailová adresa</EditDescription>
   </DataItem>
   <DataItem dataType="line" name="www">
     <EditLabel>WWW adresa</EditLabel>
     <EditDescription>WWW adresa klienta</EditDescription>
   </DataItem>
   <DataItem dataType="telefon" name="telefon">
     <EditLabel>Telefon</EditLabel>
     <EditDescription>Telefonní číslo klienta</EditDescription>
   </DataItem>
  </DataStruct>
  <DataItem dataType="stav_zakaznika" name="stav_zakaznika">
   <EditLabel>Stav zákazníka</EditLabel>
   <ExecuteCode>
     <![CDATA[
     $upozorneni = $this->getDataByPath(array('..',
```

```
          'upozorneni_na_novinky'))->getFilledValue();
         if($this->getFilledValue() == 'minulý'
             && is_array($upozorneni)
             && in_array('pošta', $upozorneni))
         {
          $this->addNoticeMessage('Opravdu chceme vyhazovat
          peníze za poštu, když s~nimi už nespolupracujeme?');
         }
         ]]>
       </ExecuteCode>
      </DataItem>
      <DataItem dataType="upozorneni_na_novinky"
          name="upozorneni_na_novinky">
        <EditLabel>Upozornění na novinky</EditLabel>
        <EditDescription>
        Vyberte formu, jakou bude
        zákazník informován o~nových službách
        </EditDescription>
      </DataItem>
   </DataStruct>
  </FormStructDataDefinition>
</Form>
```

Jak je na ukázce vidět, bylo použito i jedno makro ve stavu zákazníka. To zkontroluje, jestli stav zákazníka není nastaven na hodnotě `minulý`. Pokud ano, zkontroluje, jestli je upozorňováno na novinky poštou. Pokud i tato hodnota je pravdivá, vypíše do upozorňovacího boxu příslušnou hlášku.

Obr. 3. Založení nového formuláře.

Obr. 4. Ochrana před nechtěným zadáním existujícího čísla formuláře.

Formulář před vyplněním je možno vidět na obrázku 5. Je na první pohled vidět, že na uživatele svítí červené rámečky s chybovými hláškami, protože (ne)vyplněné hodnoty v těchto polích neodpovídají chtěným regulárním výrazům. Vzhledem k tomu, že ve formuláři existují tyto chyby, nelze potvrdit formulář tlačítkem „Dokončit úpravy“ a tím nastavit u této verze formuláře příznak, že se jedná o dokončenou verzí (ne pracovní), a zobrazit ji v hlavním seznamu formulářů daného druhu formuláře, kde se zobrazují jen dokončené verze. Po úpravě filtru lze zobrazit i nedokončené verze a provézt určité úpravy nebo najít přepsané informace.

Během editace jsou k dispozici tlačítka „Vrátit úpravu zpět (undo)“ a „Provézt vrácenou editaci (redo)“. Pomocí nich se lze pohybovat v historii editace zpět a dopředu dle libosti. Tlačítka jsou aktivní pouze v případě, že existuje verze, na kterou by se dalo přejít. V případě, že by se dalo přejít směrem dopředu na více verzí, bude vybrána ta novější varianta. Ke starší variantě se lze vrátit v seznamu rozpracovaných variant formuláře.

Obrázek 6 ilustruje podobu vyplněného formuláře.

2.8 Použití maker

Jak už bylo na více místech této práce uvedeno, formuláře umožňují v sobě spouštět makra. Pro tento účel nebyl navrhován žádný specifický jazyk[15]. Byl použit jazyk PHP vkládaný do tagu `ExecuteCode`. Obsah těchto tagů z celého dokumentu se provede vždy při inicializaci formuláře a v případě, že se změní hodnota `filled_value`, provedou se tato makra znova. Počet iterací, je omezen na několik desítek. V praxi si většina maker vystačí s jedním cyklem, ale více iterací je pro některé typy maker žádoucí. Obsah tagu `OnChangeCode` se spouští pouze v případě změny dané hodnoty uživatelem.

[15]Domain Specific Language nebo někdy jen DSL.

Kód je spuštěn objektem (typu `DataItem`), ve kterém je daný kód nadefinován. Lze se tedy relativně pomocí metody `getDataByPath()` (vysvětlené na straně 36) pohybovat strukturou a využívat prostředků frameworku a aplikace pro dosažení patřičného výsledku.

2.8.1 Typická použití maker

Makry lze řešit nejrůznější úlohy. Pro ilustraci je zde uvedeno několik typických použití maker a nástroje, jimiž se dá dosáhnout další funkcionality.

Validace vstupních dat, na kterou je validace pomocí regulárních výrazů krátká. V kapitole 2.7 na straně 40 je ukázka takové pokročilé validace, kdy se kontrolují hodnoty ve výčtovém typu a obyčejném vstupu, a při určitých hodnotách se vytvoří hláška pro uživatele.
Použité nástroje: `getDataByPath()`, `addErrorMessage()`, podmínky.

Výpočty na šabloně, např. šablony *faktura* by bylo možné tímto způsobem spočítat výslednou částku na faktuře včetně dopočítání DPH atd.
Použité nástroje: podmínky, cykly, `getDataByPath()`.

Výstrahy při změně hodnoty, která by se za běžných podmínek neměla měnit.
Použité nástroje: podmínky, `addNoticeMessage()`.

Vyplnění hodnoty při splnění podmínky. Dá se použít nejrůznějším způsobem, např. po zaškrtnutí checkboxu automaticky v jiném poli formuláře vyplnit předpokládaný text.
Použité nástroje: podmínky, `getDataByPath()`.

Manipulace se strukturou formuláře je trochu náročnější záležitostí, ale je to možné. Je možné v případě nutnosti přidat, smazat nebo různě přestavět strukturu formuláře, změnit za letu styl vstupního datového pole atd. Možná teď není zřejmé, k čemu by to vše mohlo být dobré, ale je to jenom tím, že lidé jsou zvyklí přemýšlet o papírové podobě formuláře, kde toto není možné.
Použité nástroje: nástroje nabízené třídou `AbstractData` a dalšími.

2.9 Tvorba výstupu do PDF

V úvodu bylo zmíněno, že pro export do PDF byla uvažována knihovna FPDF. Ano, počítalo se s ní, že bude použita a že součástí šablony bude i PHP kód pro export

do PDF využívající tuto knihovnu. Postupem času tato myšlenka byla opuštěna a byla implementována funkce exportu do PDF pomocí XSL-FO šablon. Je to daleko standardnější řešení, které je již implementováno a nemá smysl znova vymýšlet kolo.

Pro převod XML dokumentu šablony do PDF je tedy zapotřebí XSL šablona generující FO reprezentující grafické uspořádání dat na papíře. FO je pak už příslušným programem převedeno do PDF souboru. Pro potřeby aplikace byl použit program Apache FOP[16] implementující tyto dvě části (XSL transformace a převedení FO do PDF) do jedné.

Pro řešený ilustrační příklad šablony *Adresa zákazníka* bude vytvořena XSL-FO šablona grafického vzhledu adresy – tedy tiskového výstupu pro tisk obálky.

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <xsl:output method="xml" encoding="utf-8" indent="yes"/>
  <xsl:template match="/Form">
    <fo:root>
      <fo:layout-master-set>
        <fo:simple-page-master master-name="DL"
            page-width="220mm" page-height="110mm"
            margin-top="5mm"   margin-bottom="5mm"
            margin-left="5mm"  margin-right="5mm">
          <fo:region-body   margin="0cm"/>
          <fo:region-before extent="0cm"/>
          <fo:region-after  extent="0cm"/>
          <fo:region-start  extent="0cm"/>
          <fo:region-end    extent="0cm"/>
        </fo:simple-page-master>
      </fo:layout-master-set>

      <fo:page-sequence master-reference="DL">
        <fo:flow flow-name="xsl-region-body" font-family="DejaVu Sans"
            font-size="14pt">
          <fo:block-container position="absolute" top="60mm"
```

[16] Apache FOP je volně dostupný z `http://xmlgraphics.apache.org/fop/`

```
            left="110mm" height="40mm" width="90mm">
          <fo:block text-align="left" >
            <xsl:apply-templates select="FormStructDataDefinition/
              DataStruct[@name='koren']/DataStruct[@name='adresa']"/>
          </fo:block>
        </fo:block-container>
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
</xsl:template>

<!-- vloží adresu -->
<xsl:template match="DataStruct[@name='adresa']">
  <xsl:apply-templates select="DataArray[@name='radky_adresy']/
      DataStruct/DataItem[@name='polozka_radku']"/>
</xsl:template>

<!-- vloží řádky adresy -->
<xsl:template match="DataItem[@name='polozka_radku']">
  <fo:block>
    <xsl:if test="position()=last()">
      <xsl:value-of select="../../../
          DataItem[@name='psc']/FilledValue"/>
      <fo:inline font-weight="bold" padding-left="3mm">
        <xsl:value-of select="FilledValue"/>
      </fo:inline>
    </xsl:if>
    <xsl:if test="position()!=last()">
      <xsl:value-of select="FilledValue"/>
    </xsl:if>
  </fo:block>
</xsl:template>
</xsl:stylesheet>
```

Na příkladu je vidět, jakým způsobem je řešen výstup. V první části je definován vzhled stránky s jejími rozměry včetně tiskového zrcadla. Pak je v těle dokumentu

absolutně pozicován blok, kam je po sléze dopraven obsah, kdy se projdou prvky pole (`DataArray`), které se vypíšou každý ve svém bloku a poslední řádek se zpracuje trochu jinak, protože se předpokládá, že se jedná o město. Vloží se tedy před město PSČ a město se zvýrazní.

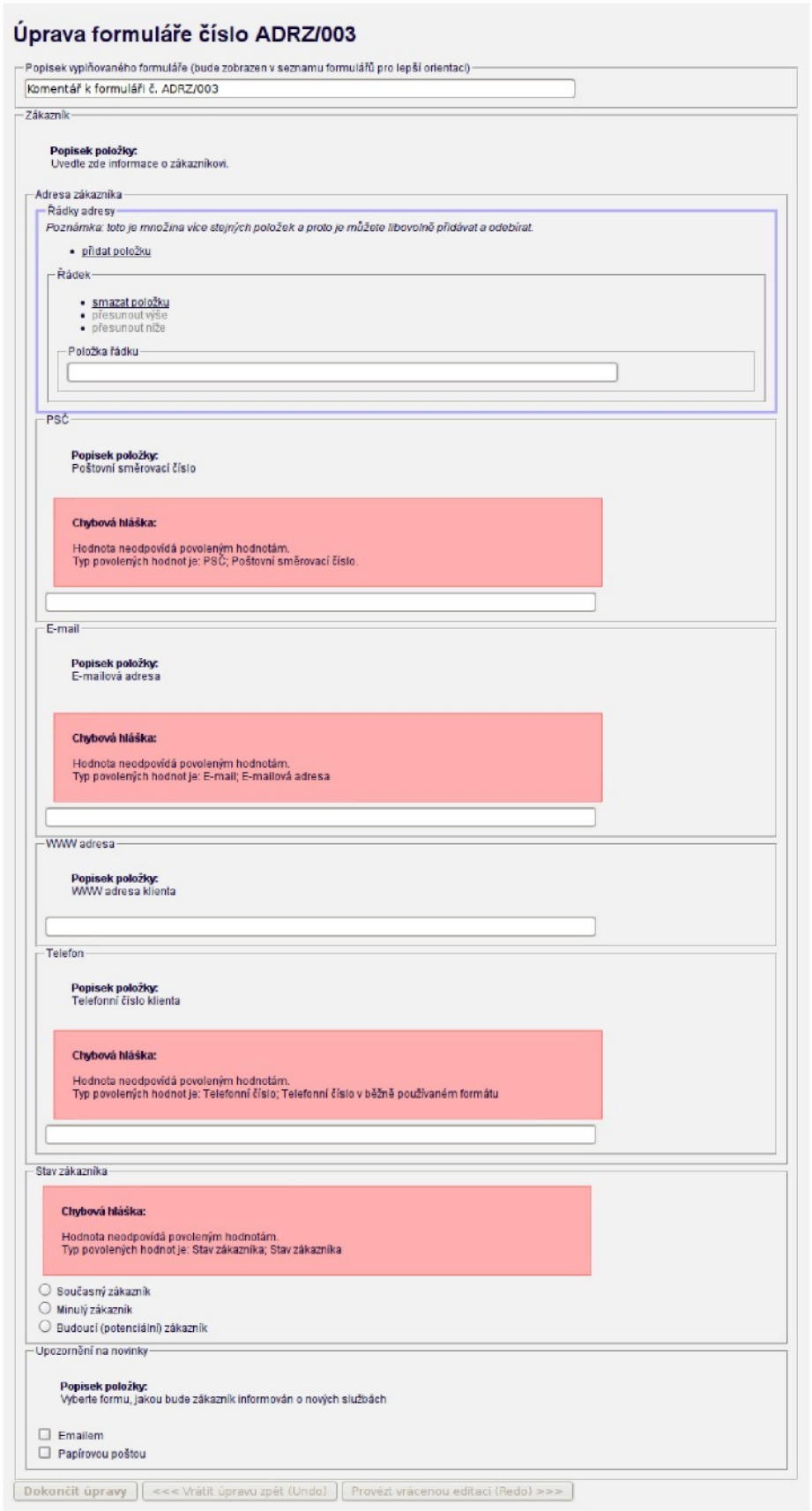
Tento XSL-FO soubor se nahraje do stejného adresáře, kde se nachází šablona formuláře a tím je okamžitě k dispozici možnost tisknout tento formulář. Do adresáře s šablonou je možné nahrát více různých XSL-FO souborů. Aplikace pak dá na výběr, jaký má použít pro tisk. V tomto případě je možné vyrobit tisk pro jiný druh obálky nebo kartičky klienta apod. Fantazii se meze nekladou.

Výsledek exportu do PDF je možno vidět na obrázku 7.

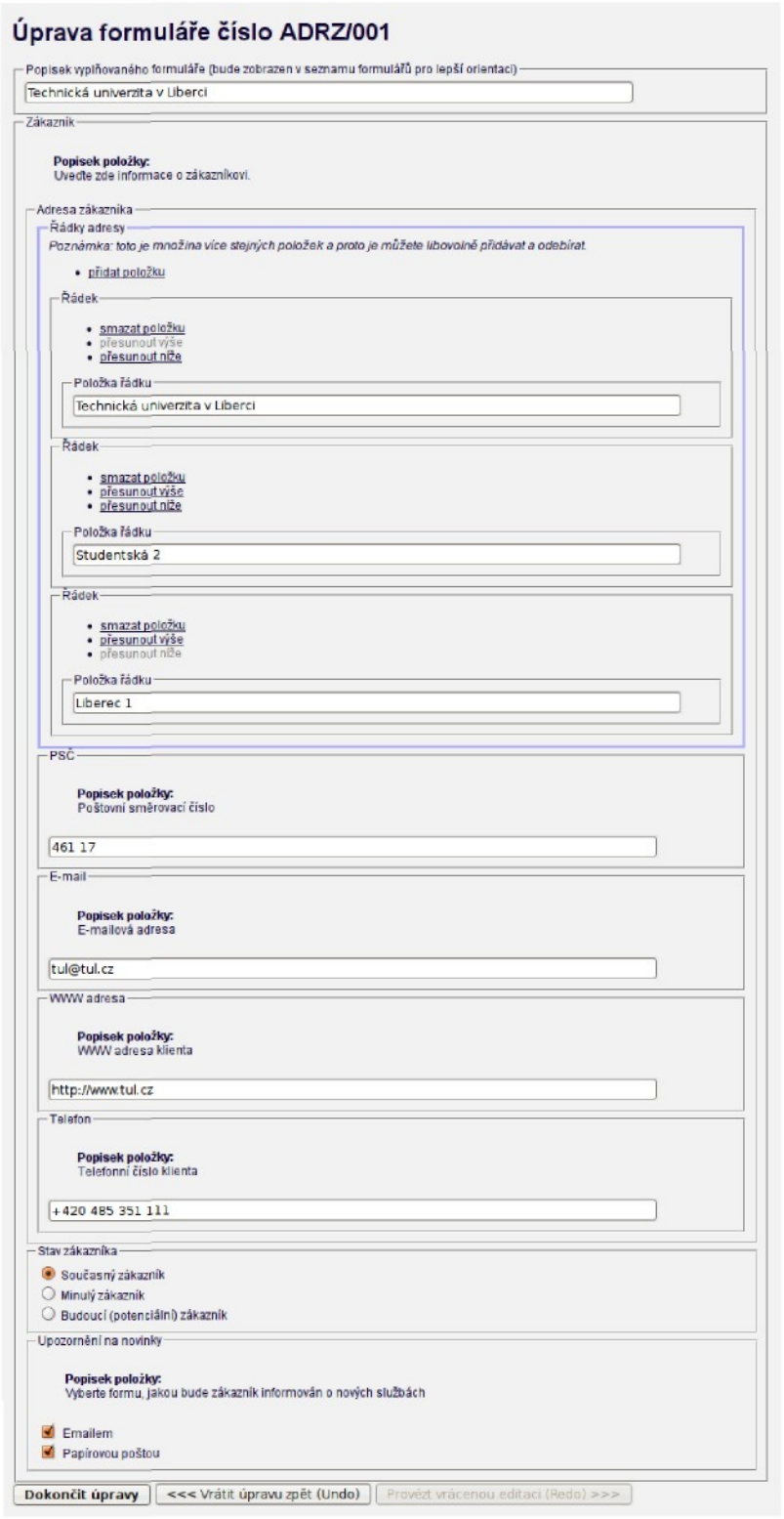
2.10 Doporučení pro vývoj formuláře

Pokud někdo podle těchto bodů bude postupovat, je doporučeno nainstalovat si separátní verzi této aplikace na testovací stroj, protože výroba makra (platí to i pro formuláře bez maker) zcela určitě neproběhne bez spousty zkažených testovacích formulářů a byla by škoda si na ostrém stroji vyrábět poškozené formuláře zabírající zbytečně posloupnost v číslech formuláře.

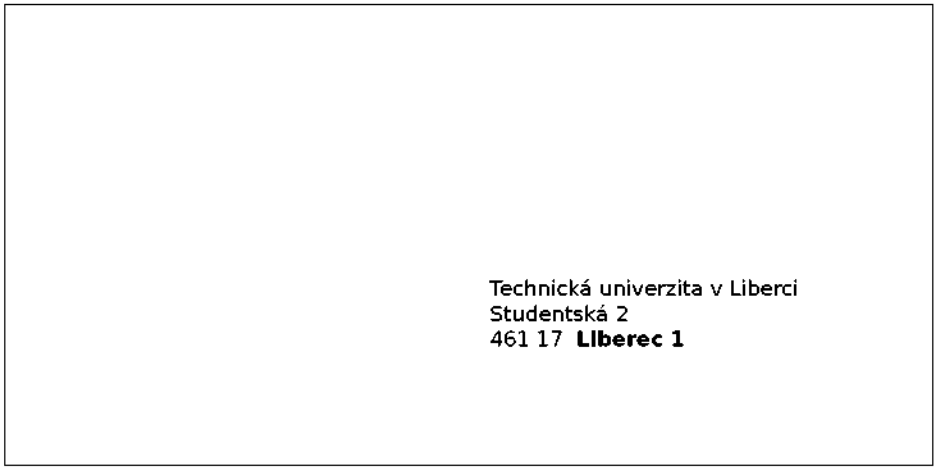
Pro vývoj XML souboru s definicí formuláře je doporučeno použít vývojové prostředí Eclipse s PDT plug-inem, ale je možné použít i jakékoli jiné. Eclipse s PDT plug-inem umí nabízení atributů a metod (tzv. intellisense), které umí nabídnout, pokud jsou třídy, atributy, metody a jejich argumenty správně okomentovány pomocí PHPDoc. Při vývoji makra si lze pomoci založením testovací metody v třídě `DataItem` a v ní vyvíjet kód. Eclipse pomůžou při psaní a až bude kód hotov, nakopíruje se do šablony. Testovací metody se pak mohou odstranit.



Obr. 5. Prázdný formulář typu *Adresa zákazníka*.



Obr. 6. Vyplněný formulář typu *Adresa zákazníka*.



Obr. 7. Výsledný export formuláře typu *Adresa zákazníka* do PDF.

3 Správa aplikace

3.1 Instalace aplikace

Instalace aplikace je popsána v následujících bodech se stručným komentářem. S tímto postupem by měl být schopen i laik aplikaci nainstalovat.

Jednotlivé body instalace počítají s určitým výchozím stavem. Tím je nainstalovaný stroj s běžícím operačním systémem GNU/Linux[17] s vytvořeným uživatelem (v příkladu bude použito jméno uživatele `visper`) s možností vykonávat administrační úkony pomocí příkazu `sudo` a nakonfigurovanou sítí. Jedná se prakticky o výchozí nastavení systému po úspěšném dokončení instalace.

1. Nainstalování OpenSSH serveru umožňujícího vzdálený konzolový přístup, aby se administrátor při každém zásahu nemusel hrbit u terminálu v serverovně.

   ```
   $ sudo apt-get install openssh-server
   ```

   Od této chvíle je možné se na tento stroj připojit z okolní sítě pomocí

   ```
   $ ssh -l visper <adresa>
   ```

2. Instalace webového serveru Apache, interpretu jazyka PHP, MySQL databáze včetně ostatních potřebných knihoven.

   ```
   $ sudo apt-get install apache2 mysql-server php5 php5-mysql
   ```

   Při tomto kroku bude administrátor vyzván k nastavení `root` hesla do MySQL databáze.

3. Instalace běhového prostředí Java a fontů pro správnou funkčnost XSL-FO procesoru Apache FOP.

   ```
   $ sudo apt-get install openjdk-6-jre ttf-dejavu
   ```

4. Instalace XSL-FO procesoru Apache FOP je provedena mimo hlavní repozitář softwarových balíčků, protože aplikace počítá s verzí 0.95beta. Je tedy zapotřebí stáhnout a rozbalit do adresáře `/opt/fop`.

[17] Tento návod počítá s distribucí Ubuntu Linux 8.04 Hardy Heron LTS v serverové edici, na kterou je poskytována podpora a bezpečnostní aktualizace až do roku 2013.

```
$ sudo mkdir /opt/fop
cd /opt/fop
$ sudo wget http://ftp.sh.cvut.cz/MIRRORS/apache/xmlgraphics/fop/\
binaries/fop-0.95beta-bin.tar.gz
$ sudo tar xzf fop-0.95beta-bin.tar.gz
$ sudo mv fop-0.95beta/* .
```

Tímto by mělo být dosaženo rozbaleného Apache FOP v `/opt/fop` a spuštěním příkazu `/opt/fop/fop` by měl program vypsat chybovou hlášku o špatných parametrech na příkazové řádce, což je v této fázi dobrá zpráva. Pokud se tak nestalo, je nejspíš možné, že není správně nainstalováno běhové prostředí Java nebo neproběhlo vše hladce při stahování a rozbalení programu.

5. Konfigurace Apache FOP se provádí v adresáři `/opt/fop/conf`, ale podrobnosti zde nebudou rozebrány, protože by to bylo rozsahově nad rámec této práce [Apa(2008)]. Konfigurační soubor a soubory s definicí fontů jsou k dispozici i s dodávanou aplikací, takže v tomto bodě lze konfiguraci provést prostým nakopírováním příslušných konfiguračních souborů do `/opt/fop/conf`.

6. Ve webovém serveru Apache je zapotřebí aktivovat podporu modulu rewrite.

   ```
   $ sudo ln -s /etc/apache2/mods-available/rewrite.load \
   /etc/apache2/mods_enabled/
   ```

   Po aplikaci změn v nastavení, je zapotřebí restartovat server.

   ```
   $ sudo /etc/init.d/apache2 restart
   ```

7. Volitelně lze nainstalovat program phpMyAdmin, kterým lze pohodlně vzdáleně pomocí webového rozhraní administrovat MySQL databázi. Stejnou službu může poskytnout i jiný software. Volba je čistě na administrátorovi.

   ```
   $ sudo apt-get install phpmyadmin
   ```

   Aplikace je pak dostupná z `http://<adresa_serveru>/phpmyadmin`.

8. Vytvoření uživatele a databáze s právy přístupu na databázovém serveru. V dalším textu bude počítáno se jménem databázového uživatele `elsek` a databází stejného jména.

9. Import základních dat nebo dat ze zálohy do databáze pomocí phpMyAdmin nebo lépe přes příkazovou řádku (příkazová řádka snese větší objemy dat).

```
$ mysql -u elsek -p elsek < data-pro-import.sql
```

10. Instalace vlastní aplikace se provádí rozbalením archívu do adresáře v domovského adresáře a následným ukázáním do tohoto adresáře v nastavení Apache web serveru. Zjednodušenou instalaci ve výchozím nastavení lze dosáhnout pomocí příkazů.

```
$ cd
$ mkdir -r www/elsek
$ cd www/elsek
$ unzip <soubor_s_archivem_aplikace>
$ cd ..
$ sudo mv elsek /var/www
$ ln -s /var/www/elsek .
```

Tímto je dosaženo umístění aplikace na adresu `http://<adresa_serveru>/elsek` a do souborů v adresáři `www/elsek` v domovském adresáři je stále uživatelem přístup bez superuživatelských práv získávaných přes `sudo`.

11. Dalším krokem je nastavení některých adresářů aplikace pro čtení i zápis.

```
$ cd
$ cd www/elsek
$ chmod 777 cache
$ chmod 777 oe/oev/smarty/templates_c
$ chmod 777 oe/oev/smarty/cache
```

12. Nastavení přístupů aplikace do databáze a URL aplikace se provede upravením příslušných řádků souboru `www/elsek/app/web_application.php`.

```
OEDatabase::setDatabaseHost('localhost');
OEDatabase::setDatabaseName('elsek');
OEDatabase::setUsername('elsek');
OEDatabase::setPassword('heslododatabaze');
define('APP_URL', 'http://<adresa_serveru>/elsek/');
```

13. Upravení přepisovacího pravidla pro správnou funkčnost modulu rewrite se provede editací řádky v souboru `www/elsek/.htaccess`

    ```
    RewriteRule ^(.*)$ /elsek/index.php?url=$1 [QSA,L]
    ```

    Na této řádce je důležité, aby adresář za adresou serveru odpovídal. Nic jiného zde měnit není zapotřebí.

14. Protože v Ubuntu je standardně pro aplikace v `/var/www` manipulace s nastavením pomocí souborů `.htaccess` zakázána, je nutné ji povolit. Jedná se o změnu řádky (v této verzi systému se jedná o 12. řádku) v konfiguračním souboru `/etc/apache2/sites-enabled/000-default`.

    ```
    AllowOverride All
    ```

    A restart serveru pro aplikaci změn.

    ```
    $ sudo /etc/init.d/apache2 restart
    ```

15. Poslední úprava spočívá v nastavení práv adresáře `/var/www/.fop` pro zápis uživateli `www-data`, protože Apache FOP je spouštěn z webového serveru, který běží pod uživatelem `www-data` a skupinou `www-data`. Apache FOP je tedy spuštěn se stejnými právy a omezeními vyplývající z daného uživatele. Ten má nastaven domovský adresář na adresář `/var/www`, do kterého mají nejrůznější programy spuštěné pod tímto uživatelem tendenci zapisovat svá dočasná data či nastavení, a Apache FOP do něj vyžaduje zápis.

    ```
    $ sudo mkdir /var/www/.fop
    $ sudo chown www-data:www-data /var/www/.fop
    $ sudo chmod 755 /var/www/.fop
    ```

3.2 Údržba aplikace

Aplikace sama o sobě nevyžaduje žádnou nadstandardní údržbu. Bohatě postačí pravidelné záplaty operačního systému pomocí

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

a neměl by se vyskytnout technický nebo bezpečnostní problém.

3.2.1 Záloha aplikace

Nejdůležitější údržbou aplikace je zálohování dat. Veškerá data vyprodukovaná uživateli se nachází v databázi. Je proto nutné provádět pravidelnou zálohu dat z databáze pro případ selhání disku nebo jiné katastrofy se stejnými následky.

Pro zálohování databáze lze použít phpMyAdmin, ale ten při větších objemech dat začne být nepoužitelný nehledě na zdlouhavost celého procesu. Proto je doporučován postup zálohy pomocí příkazové řádky.

```
$ mysqldump -u elsek -p elsek > elsek_YYYY-MM-DD.sql
$ scp elsek_YYYY-MM-DD.sql elsek@<adresa_zalohovaciho_stroje>:~/backup/
```

První příkaz slouží na kompletní zálohu databáze `elsek` a druhý příkaz zkopíruje zálohovanou databázi na jiný bezpečný stroj ideálně i v jiné budově. Tato procedura by mohla být uložena ve skriptu, který by se prováděl každý den. Přihlašování na zálohovací stroj by pak probíhalo pomocí páru veřejného a soukromého klíče, takže by skript nepotřeboval zadávat heslo a mohlo by jít o plně automatizovaný proces. Případná obnova databáze by pak probíhala podle stejné šablony jako instalace jen s jinými daty v 9. bodě při importu dat do databáze.

Další data, která by bylo vhodné občas zálohovat, jsou soubory v adresáři `www/elsek/app/form_templates`, kde jsou uloženy šablony a tiskové XSL-FO šablony. Definice struktur formulářů lze rekonstruovat ze záznamů v databázi, ale XSL-FO šablony nikoli, přitom to jsou právě tyto tiskové šablony, které pravděpodobně budou dost často měněny a mohla by to být citelná ztráta.

4 Budoucí rozvoj aplikace

Modulární architektura aplikace umožňuje její snadné rozšíření o nové komponenty nebo přepracování těch stávajících. V následujících odstavcích jsou shrnuty části, které si zasluhují dopracovat, a vize dalšího možného rozvoje aplikace větších rozměrů.

4.1 Uživatelské rozhraní pro konfiguraci přístupů

V současné aplikaci je implementována část administrace uživatelských účtů na úrovni založení, editace a (de)aktivace účtu s tím, že v editaci je možno přiřazovat uživatele ke konkrétnímu vytvořenému uživatelskému typu (`PtypeModel`). Tyto uživatelské typy se však musí nakonfigurovat ručně pomocí vytvoření relací na příslušná pravidla (`RuleModel`). Bylo by tedy vhodné vyrobit právě takovéto rozhraní umožňující „klikacím“ způsobem nadefinovat práva příslušným typům uživatelů.

4.2 Přesunutí části funkcionality na databázovou vrstvu

V některých případech aplikace vyžaduje dodržení určitých pravidel uložených dat v databázi. Konkrétně se jedná o zamezení existence více jak jedné dokončené verze formuláře určitého čísla nebo existence pouze jedné doporučované verze typu formuláře a pod. Tato omezení jsou hlídána na aplikační vrstvě. Měla by však existovat kontrola i na databázové vrstvě implementována např. pomocí triggerů. Řešení pomocí triggerů má výhodu v nezávislosti na aplikaci. Je možné takovouto databázi poskytnout i jiné aplikaci a spolehnout se, že daná omezení budou aplikována všude stejně.

4.3 Zrychlení XSL-FO transformací dokumentu

Transformace formuláře do PDF dokumentu trvá v závislosti na výkonnosti serveru 2 až 5 sekund. Většinu z tohoto času spotřebuje virtuální stroj na spuštění sebe a aplikace, a to pokaždé, kdy je to zapotřebí. Bylo by vhodné vymyslet jiný způsob, např. pomocí serverové aplikace využívající knihovnu Apache FOP, která by byla pořád spuštěná a mohla tak promptně vyřizovat požadavky na převody dokumentů. Očekávané zrychlení je v řádech tisíců procent i více.

4.4 Vytvoření grafického návrháře pro tvorbu šablon

Vytvoření grafického návrháře může být poměrně velikým projektem, ale výrazně by to zkrátilo dobu vývoje formulářové struktury. Takový návrhář by mohl být jak ve formě webové tak ve formě desktopové aplikace využívající napojení na webovou aplikaci.

Další metou by mohla být implementace WYSIWYG[18] návrháře pro PDF export formuláře, ale to může být hodně solidním problémem na to, aby jej bylo možné uspokojivě vyřešit.

[18] Je akronym pro *What You See Is What You Get.*

5 Závěr

Výsledkem diplomové práce je nově vyvinutá funkční webová aplikace zajišťující kompletní správu formulářů libovolných druhů nadefinovaných pomocí vytvořeného XML jazyka. Celá aplikace byla navržena modulárně tak, aby části vizuální, databázové a funkční[19] byly programově oddělené a byla tak zajištěna snadná správa a rozšiřitelnost. Odděleny rovněž zůstaly definice struktury formulářů a šablony XSL-FO pro převod vyplněných formulářů do formátu PDF pro kvalitní tiskový výstup. Aplikace má řešen na obecné úrovni víceuživatelský přístup s možností podrobně definovat přístupová práva. Je tedy možné přidat libovolný nový typ formuláře popsaný pomocí vytvořeného jazyka, nadefinovat typy uživatelů, kteří mají mít právo na práci s tímto formulářem, a začít s formulářem pracovat. Není zapotřebí žádný přímý zásah do kódu aplikace nebo databázové struktury.

Aplikace je připravena pro budoucí rozšiřování. Prozatím lze přidávat nové formuláře pouze pomocí ručně vytvořeného XML souboru v jazyce definovaného za pomocí DTD, nicméně v budoucnu by mohl existovat grafický editor pro tvorbu struktury formuláře, který by výrazně ulehčil práci při tvorbě nových formulářů.

[19] Byl aplikován návrhový vzor Model-View-Controller

SEZNAM POUŽITÉ LITERATURY

[jso(2008)] *JSON*, 2008. Dostupné z: `http://json.org/`.

[Apa(2008)] *Apache FOP*. The Apache Software Foundation, 2008. Dostupné z: `http://xmlgraphics.apache.org/fop/`.

[Cak(2008)] *CakePHP: the rapid development php framework*. Cake Development Corporation, 2008. Dostupné z: `http://www.cakephp.org/`.

[Elliote Rusty Harold(2002)] ELLIOTE RUSTY HAROLD, W. S. M. *XML v kostce*. Praha : Computer Press, 2002. ISBN 80-7226-712-4.

[New(2008)] *Smarty: Template Engine*. New Digital Group, Inc., 2008. Dostupné z: `http://www.smarty.net/`.

[Pecinovský(2007)] PECINOVSKÝ, R. *Návrhové vzory*. Brno : Computer Press, 2007. ISBN 978-80-251-1582-4.

[PHP(2008a)] *PHP: Hypertext Preprocessor*. The PHP Group, 2008a. Dostupné z: `http://www.php.net/`.

[PHP(2008b)] *History of PHP and related projects*. The PHP Group, 2008b. Dostupné z: `http://www.php.net/history`.

[Plathey(2008)] PLATHEY, O. *FPDF Library - Home page*, 2008. Dostupné z: `http://www.fpdf.org/`.

[Pro(2008)] *Prototype JavaScript framework: Easy Ajax and DOM manipulation for dynamic web applications*. Prototype Core Team, 2008. Dostupné z: `http://www.prototypejs.org/`.

[Sun(2008)] *Java Web Start Architecture JNLP Specification & API Documentation*. Sun Microsystems, Inc., 2008. Dostupné z: `http://java.sun.com/products/javawebstart/download-spec.html`.