České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídicí techniky

Diplomová práce

Automatizace generování uživatelského rozhraní pro aplikaci eForm

Bc. Zdeněk Dušátko

Vedoucí práce: Ing. Tomáš Černý

Studijní program: Otevřená informatika

Obor: Počítačové inženýrství

2. května 2013

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Tomáši Černému za ochotu, cenné rady poskytnuté při vypracovávání této diplomové práce a hlavně za možnost spolupracovat na velmi moderní technologii. Dále bych chtěl poděkovat své rodině za podporu při studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 5. 5. 2013 .....

Abstract

This thesis deals with automatic generation of fragments of user interface based on aplication data model and its metadata. Diploma thesis consist of analysis, design and implementation of real application eForm. The eForm is based on automatic generation of fragments of user interface that is implemented using Aspect Faces framework. This application will be used to create forms that are requiered in case of setting up a new company and applying to add this company to Commercial Register. In this thesis you can find comparison of developing user interface using automatic generation of fragments of user interface and usual approach. The last part deals with design and implementation of library for automated creation of PDF and XLS forms.

Abstrakt

Tato práce je zaměřena na automatické generování fragmentů uživatelského rozhraní na základě datového modelu aplikace a jeho metadat. Součástí práce je analýza, návrh a implementace reálné aplikace eForm. Aplikace eForm je postavena na technologii automatického generování fragmentů uživatelského prostředí, která je realizována pomocí frameworku Aspect Faces. Aplikace eForm bude sloužit k tvorbě formulářů, které jsou vyžadovány při zakládání nové společnosti pro podání do obchodního rejstříku. Součástí práce je srovnání vývoje uživatelského prostředí pomocí techniky automatického generování fragmentů uživatelského rozhraní s klasickým přístupem. Práce se dále zabývá návrhem a implementací knihovny pro automatizovanou tvorbu PDF a XLS formulářů.

Obsah

1 Úvod **1**

2 Popis problému, vymezení požadavků **3**
2.1 Popis řešeného problému . . . 3
2.2 Specifikace požadavků . . . 4
2.2.1 Funkční požadavky . . . 4
2.2.2 Obecné požadavky . . . 4
2.2.3 Technická specifikace dokumentů k založení do sbírky listin obchodního rejstříku . . . 5

3 Rešerše **7**
3.1 Existující řešení - tvorba formulářů . . . 7
3.2 Automatické generování uživatelského prostředí . . . 8
3.3 Model-Driven Development . . . 8
3.4 Aspect Faces . . . 9
3.4.1 Aspektově orientované programování . . . 9
3.4.2 Inspekce zdrojového kódu . . . 10
3.4.3 Ukázka Aspect Faces aplikace . . . 11
3.4.4 Aspekty generování . . . 13
3.5 Příbuzné projekty . . . 13
3.5.1 Apache Isis . . . 13
3.5.2 MetaWidget . . . 14

4 Analýza **15**
4.1 Případy užití . . . 15
4.2 Analytický doménový model . . . 18
4.3 Popis architektury . . . 19
4.3.1 Prezentační vrstva . . . 19
4.3.1.1 Java Server Faces . . . 20
4.3.2 Business vrstva . . . 21
4.3.3 Datová vrstva . . . 21
4.4 Plán nasazení . . . 21

5 Návrh řešení a realizace **23**
5.1 Podmíněná validace . . . 23
5.1.1 OmniFaces řešení . . . 24
5.1.2 JSF on-deman validation . . . 25
5.1.2.1 Podporované validátory a konfigurace . . . 26
5.1.2.2 Rozšížení pro Aspect Faces . . . 26
5.2 Export formulářů . . . 28
5.2.1 PDF formulář . . . 29
5.2.1.1 Jasper Reports . . . 29
5.2.1.2 PDF Extension . . . 30
5.2.1.3 Flying Saucer . . . 31
5.2.1.4 Optimalizace stránky pro tisk . . . 31
5.2.1.5 Rozšíření pro Aspect Faces . . . 32
5.2.2 XLS formulář . . . 35
5.2.2.1 XLS Extension . . . 35
5.2.2.2 JSoup parser . . . 36
5.2.2.3 JExcelAPI . . . 36
5.2.2.4 Rozšíření pro Aspect Faces . . . 37

6 Testování **39**
6.1 Testování jednotek . . . 39
6.2 Integrační testy . . . 39
6.2.1 Arquillian . . . 40

7 Srovnání s běžným přístupem k tvorbě UI **43**
7.1 Porovnání tvorby uživatelského prostředí . . . 43
7.1.1 UCC . . . 44
7.1.2 Testovací Aplikace . . . 44
7.1.3 Vyhodnocení . . . 45
7.1.4 Údržba aplikace . . . 46
7.1.4.1 Vyhodnocení . . . 47
7.2 Měření výkonu . . . 48
7.2.1 Příprava testu . . . 48
7.2.1.1 JMeter . . . 48
7.2.2 Vyhodnocení . . . 49

8 Závěr **51**

A Seznam použitých zkratek **57**

B Ukázka vyexportovaných formulářů **59**

C Instalační příručka **63**

D Obsah přiloženého CD **65**

Seznam obrázků

3.1 Struktura ZFO formuláře. 7
3.2 Koncept automatizace skrze inspekci. 10
3.3 Vykreslený fragment formuláře. 12

4.1 Diagram případů užití. 16
4.2 Analytický doménový model - akciová společnost. 18
4.3 Architektura aplikace eForm. 19
4.4 Java Server Faces – implementace MVC 20
4.5 Diagram nasazení. 22

5.1 Životní cyklus – Java Server Faces 24
5.2 Konceptuální model automatizovaného exportu formulářů. 29
5.3 Struktura formátu XLSX. 35

6.1 Data Access Object 39

7.1 Testovací aplikace - diagram tříd. 44
7.2 Testovací aplikace - upravený diagram tříd. 46

B.1 HTML náhled na formulář. 60
B.2 Náhled na PDF formulář 61
B.3 Náhled na XLS formulář. 61

D.1 Obsah přiloženého CD 65

Seznam tabulek

2.1 Technická specifikace dokumentů 5

5.1 JSF on-demand validation rozšíření pro Aspect Faces 27
5.2 PDF rozšíření pro Aspect Faces 33
5.3 XLS rozšíření pro Aspect Faces 37

7.1 Srovnání LOC pro editační formulář. 45
7.2 Srovnání LOC pro formulář určený pouze ke čtení. 46
7.3 Srovnání SLOC a MoF po přidání jednoho atributu. 47
7.4 Srovnání SLOC a MoF po změně jednoho atributu. 47
7.5 Srovnání pro oddělené tagy a widgety. 49
7.6 Srovnání pro sjednocené tagy a widgety. 50

Kapitola 1

Úvod

V rámci procesu vývoje informačních systémů tvoří velkou část vývoj grafického uživatelského prostředí (UI). Nemalé náklady také stojí jeho následná údržba. Podle velmi často citovaného článku *Survey on user interface programming*[14] z roku 1992, tvoří až 50% vynaloženého úsilí právě vývoj uživatelského prostředí. Hlavní motivací této práce je nasadit technologii, která toto úsilí významně sníží.

Hlavní problém v klasickém přístupu k vývoji uživatelských prostředí spočívá v duplikaci informací, které jsou již jednou uvedeny v jiných vrstvách aplikace. Např. v datovém modelu aplikace již známe datové typy atributů a omezení, které jsou na ně kladené. U manuálního přístupu k vývoji UI je nutné tyto informace znovu zanést také do prezentační vrstvy aplikace, může tak dojít k chybám způsobených lidským faktorem při duplikaci těchto informací.

Framework Aspect Faces (dříve znám pod názvem JFormBuilder) umožňuje omezit duplikaci stejných informací ve více vrstvách dané aplikace. Framework Aspect Faces (AF) je založen na principech aspektově orientovaného programování. Aspektově orientované programování se soustřeďuje na zapouzdření průřezových koncernů (*crosscutting concerns*), tedy takových koncernů, které existují ve více částech softwaru.

Jedním z cílů této práce je také analýza, návrh a implementace reálné aplikace eForm, která slouží k vytváření formulářů, které jsou vyžadovány při zakládání nové společnosti pro zapsání do obchodního rejstříku. Tuto doménu pro nasazení technologie automatického generování fragmentů UI jsem zvolil z důvodu, že je dostatečně rozsáhlá pro demonstraci výhod této technologie při vývoji i následné údržbě aplikace. Obsahuje rozsáhlé formuláře s mnoha částmi a elementy.

Další motivací zvolit tuto doménu jsou současné možnosti, jak tyto formuláře vytvářet. V obchodním rejstříku je doporučený desktopový program pro úpravu formulářů ve formátu ZFO. Další možností jsou předpřipravené PDF formuláře a poslední možností jsou tzv. inteligentní formuláře. Aplikace Inteligentní formulář je webová aplikace dostupná ze stránek obchodního rejstříku. Odpadá tedy nutnost instalace jakéhokoliv programu. Aplikace eForm nabízí podobnou funkcionalitu jako aplikace inteligentní formulář. Navíc zahrnuje technologii automatického generování fragmentů uživatelského prostředí z datového modelu a jeho metadat. Aplikace se tak stává lépe udržovatelná. Podrobné srovnání současných možností je uvedeno v kapitole 3.1.

Kapitola 2

Popis problému, vymezení požadavků

2.1 Popis řešeného problému

Aplikace eForm obsahuje rozsáhlé formuláře pro sběr informací od uživatele při vytváření formuláře, který je vyžadován při registraci nové společnosti do obchodního rejstříku. Tvorba uživatelského prostředí představuje až 50% úsilí z celkového procesu vývoje informačního systému. V případě aplikací s mnoha formulářovými prvky může toto číslo být i větší. Nemalé náklady také představuje následná údržba aplikace, ve které se každá změna v datovém modelu se musí manuálně zavést do všech vrstev aplikace.

Cílem této práce je dokázat, že pomocí technologie pro automatické generování fragmentů uživatelského prostředí z datového modelu a jeho metadat, lze vytvořit plnohodnotnou aplikaci, která nebude vyžadovat takové úsilí při tvorbě a následné údržbě UI. Aplikace eForm je postavena na frameworku Aspect Faces, která umožňuje automatizovaně generovat fragmenty UI na základě inspekce zdrojového kódu a příslušných konfigurací. Úspora je založena na omezení duplikací informací, které jsou již známé v jiných vrstvách aplikace.

Dalším cílem práce je srovnání techniky automatického generování fragmentů uživatelského prostředí s klasickým přístupem k vývoji UI. Toto srovnání je provedeno z hlediska úspory vynaloženého úsilí na vývoj a následnou údržbu aplikace. Srovnání je provedeno na testovací aplikaci, která je podmnožinou aplikace eForm. Druhým hlediskem srovnání musí nutně být srovnání výkonu aplikací pro oba přístupy. Srovnání technik je věnovaná předposlední kapitola 7 na konci práce.

Aplikace eForm vyžaduje výstup formulářů v PDF a XLS souborech. V této části stojí za úvahu zobecnění technologie pro automatické generování fragmentů UI na libovolný výstup. Kapitola 5.2 se věnuje návrhu a implementaci PDF a XLS knihoven rozšiřujících Aspect Faces framework, které umožňují automatizovanou tvorbu PDF resp. XLS formulářů z datového modelu aplikace.

Posledním řešeným problémem je nutnost vytvoření podmíněných validátorů. Aplikace eForm obsahuje rozsáhlé formuláře, které se skládají z mnoha prvků a částí. Uživatel, s největší pravděpodobností, nevyplní takto dlouhý formulář najednou, ale bude si chtít rozepsaný formulář uložit a později se k němu vrátit. Jelikož současná verze frameworku JSF 2 neumožňuje přeskočení fáze validace formuláře na základě události, která byla vyvolána, tak je nutné

navrhnout a implementovat knihovnu JSF on-demand validation, která tuto funkcionalitu do Aspect Faces frameworku přinese. Více o tomto rozšíření uvádí kapitola 5.1.

2.2 Specifikace požadavků

V této kapitole jsou vymezeny všechny požadavky kladené na systém. Tyto požadavky jsou rozděleny na dvě základní skupiny, požadavky funkční a obecné.

2.2.1 Funkční požadavky

Funkční požadavky definují, jaké funkce bude systém nabízet. Vycházejí z analýzy postihovaného problému. Tato kapitola uvádí jejich výčet.

- Systém umožní uživateli vytváření formulářů pro podání do obchodního rejstříku.
- Formuláře se budou verifikovat podle pokynů pro vyplnění[1] pro každou konkrétní požadovanou právní formu podání do obchodního rejstříku.
- Uživatel bude moci zvolit odpovídající typ formuláře na základě požadované právní formy. Systém nabídne úplnou sadu právních forem. V této práci se zaměřím pouze na některé nejběžnější právní formy, jako jsou společnost s ručením omezeným a akciová společnost.
- Uživatel si bude moci průběžně ukládat rozpracovaný formulář jako koncept a později se k němu vracet.
- Formuláře budou interaktivní, tj. uživatel přidává, resp. odebírá volitelné části formuláře.
- V průběhu vyplňování bude uživateli umožněn náhled na vyplněnou část formuláře a možnost exportu formuláře pro elektronické podání. Výsledný dokument musí splňovat technické parametry, viz technická specifikace dokumentů, která je uvedena v kapitole 2.2.3. Podporované formáty budou PDF a XLS.
- Uživatel si bude moci vyplněný formulář stáhnout, případně ho odeslat na svoji emailovou adresu.

2.2.2 Obecné požadavky

Tato kapitola obsahuje výčet obecných požadavků, které se často také označují jako nefunkční. Obecné požadavky popisují omezení, které musí systém splňovat.

- Aplikace eForm bude nasazena jako webová aplikace.
- V aplikaci bude nasazena technologie automatického generovaní fragmentů uživatelského rozhraní pro snadné přidání další právní formy a pro snadnější údržbu zdrojového kódu aplikace.

[1] Pokyny obchodního rejstříku: https://or.justice.cz/ias/ui/formulare

- Podpora všech významných webových prohlížečů: Mozilla Firefox 4, Opera 11, Google Chrome 10, Internet Explorer 10.
- Přehledné a intuitivní uživatelské rozhraní. Různý pohled na data v závislosti na roli.
- Vysoká modifikovatelnost a rozšiřitelnost. Snadné přidání další právní formy, bez nutnosti změny velké části aplikace.
- Export vyplněných formulářů do dokumentů formátu PDF. Výsledné soubory budou splňovat parametry vyhlášky Ministerstva spravedlnosti 562/2006 Sb.[2], která stanovuje způsob převedení listin do elektronické podoby. Požadavky na dokument jsou shrnuty v tabulce 2.1.
- Systém zamezí neautorizovaným přístupům k datům. Každý formulář bude označen unikátním identifikátorem.

2.2.3 Technická specifikace dokumentů k založení do sbírky listin obchodního rejstříku

Tato kapitola uvádí parametry na výsledné soubory pro podání do obchodního rejstříku[3]. Parametry shrnuté v tabulce 2.1 vycházejí z vyhlášky Ministerstva spravedlnosti 562/2006 Sb., která stanovuje způsob převedení listin do elektronické podoby.

	Popis
1	Každá jednotlivá listina se předává jako právě jeden PDF soubor; není tedy možné rozdělit jednu listinu do více PDF dokumentů ani slučovat více listin do jednoho PDF dokumentu.
2	Velikost předávaných PDF dokumentů je omezena a nesmí přesáhnout 150kB na 1 stránku listiny; např. listina o 20 stránkách může tedy mít maximálně 3 MB.
3	Nedoporučuje se předávat naskenované papírové podklady.
4	PDF listiny je možno ověřit zaručeným elektronickým podpisem i nastavit jejich zabezpečení (encryption); není však možné omezit (ani heslem) jejich plnohodnotné zobrazení nebo tisk.
5	PDF soubory do celkové velikosti 8 MB na jedno podání je možno předávat pomocí elektronické pošty na podatelnu příslušného rejstříkového soudu; pokud bude velikost PDF dokumentu větší, je třeba předat jej na CD-R médiu (možno zaslat i poštou).
6	Soud nepřijímá elektronická podání (e-maily), pokud nejsou opatřeny zaručeným elektronickým podpisem založeným na kvalifikovaném certifikátu vydaném akreditovaným poskytovatelem certifikačních služeb (tzv. elektronickým podpisem).

Tabulka 2.1: Technická specifikace dokumentů

[2] Vyhláška 562/2006 Sb. http://portal.justice.cz/Justice2/ms/ms.aspx?j=33&o=23&k=2299&d=136225
[3] Obchodní rejstřík: https://or.justice.cz/ias/ui/rejstrik

Kapitola 3

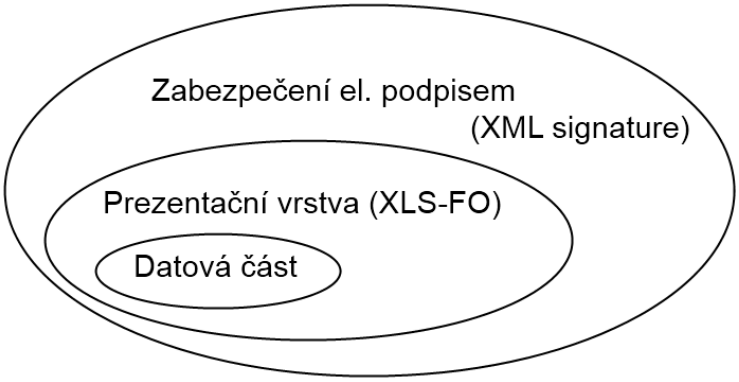
Rešerše

V této kapitole nejdříve uvádím existující řešení pro tvorbu formulářů pro registraci nové společnosti do obchodního rejstříku v České republice. V druhé části se zaměřuji na automatické generování uživatelského prostředí z modelu aplikace a jeho metadat. V kapitole jsou rozebrány příbuzné projekty k frameworku Aspect Faces a jejich vzájemné porovnání.

3.1 Existující řešení - tvorba formulářů

Obchodní rejstřík[1] v současné době nabízí tři typy formátů formulářů pro podání. První možností jsou předpřipravené PDF formuláře, které lze stáhnout ze stránek obchodního rejstříku. Formulář se vyplňuje přímo do PDF souboru.

Druhou možností jsou ZFO formuláře, které se vyplňují pomocí programu Software602 Form Filler. ZFO[36] formát slouží k tvorbě inteligentních formulářů. Je silně inspirován standardem xForms[2], který odděluje data od jejich prezentace. Na následujícím diagramu je zobrazena struktura ZFO formuláře.



Obrázek 3.1: Struktura ZFO formuláře.

[1] https://or.justice.cz
[2] Více o xForms naleznete zde: http://www.w3.org/MarkUp/Forms/#waXForms.

Datová část obsahuje metadata formuláře, informace o formuláři a datové struktuře. Prezentační vrstva je založena na XLS-FO formátovacím jazyku, který je rozšířen o definice formulářových polí a sekcí. Formulář je elektronicky podepsán podle specifikace XML Signature. Formát ZFO má dvě varianty. První možností je komprimovaný balík jednoho, nebo více souborů. Kromě samotného formuláře obsahuje ještě informace v XML nebo obrázky náhledu v JPG. Druhou možností jsou zprávy ISDS.

Software602 Form Filler[3] již nabízí interaktivní způsob přidávání sekcí. Program je zdarma dostupný pro operační systém Windows, Linux i OS X. Aplikace má jednoduché a přehledné uživatelské prostředí. Podporuje přídavné moduly, export PDF formulářů a umožňuje snadné vytváření důvěryhodných dokumentů a ověřování platnosti jejich podpisů a razítek. Jedinou nevýhodou tak zůstává nutnost instalace programu, který má cca 70MB pro Windows a 40MB pro OS X.

Třetím a posledním typem formuláře je tzv. inteligentní formulář. Aplikace Inteligentní formulář je webová aplikace dostupná ze stránek obchodního rejstříku. Odpadá tedy nutnost instalace jakéhokoliv programu. Aplikace nabízí interaktivní možnost tvorby formuláře pro podání do obchodního rejstříku. V současné době je podporována většina právních norem platných v České republice. Aplikace má jednoduché rozhraní a pole s nápovědou. Problém se objevuje ve validaci polí, která hlásí chybu hned po otevření formuláře, dříve než uživatel začne cokoliv vyplňovat. Aplikace dále umožňuje export formulářů do PDF formátu a uložení rozepsaného formuláře jako koncept, ke kterému se uživatel může později vrátit.

Aplikace eForm, která vznikla v rámci této práce, nabízí podobnou funkcionalitu jako aplikace Inteligentní formulář. Navíc zahrnuje technologii automatického generování fragmentů uživatelského prostředí z datového modelu a jeho metadat. Aplikace se tak stává lépe udržovatelná, každá změna formuláře není tak nákladná jako v případě manuální tvorby uživatelského prostředí. Více o výsledcích porovnání s běžným přístupem naleznete v kapitole 7. V následující části jsou rozebrány principy a technologie automatizace tvorby uživatelských prostředí.

3.2 Automatické generování uživatelského prostředí

Více než 50% úsilí při vývoji softwaru patří právě uživatelskému prostředí [14]. Nemalé náklady také stojí jeho následná údržba. V další části jsou naznačeny postupy jak toto úsilí snížit pomocí generování fragmentů uživatelského prostředí z datového modelu.

3.3 Model-Driven Development

Model-Driven Development (MDD)[18], tj. vývoj řízený modelem, ukazuje cestu, jak se vypořádat s častými změnami požadavků při vývoji rozsáhlých aplikací. Hlavním artefaktem není kód programu, ale model. Vývoj a údržba se přesouvá z kódu aplikace do modelu, ze kterého je generovaná celá aplikace nebo její fragmenty.

[3] Software602 Form Filler lze stáhnout ze stránek společnosti 602: http://www.602.cz/602xml_filler/download.

Konsorcium Object Management Group (OMG)[4] přináší širokou škálu standardů pro MDD a nový směr vývoje software - Model Driven Architecture (MDA)[30][2]. MDA odděluje business a aplikační logiku od cílové platformy. Je založena na platformě nezávislých UML (Unified Modeling Language)[5] modelech a dalších podpůrných technologiích. UML je jedním ze základních standardů v rámci MDA. Dalšími stavebními kameny MDD jsou další různé standardy a formalismy, jako jsou např. webové standardy (XML, SOAP, atd).

Klasický přístup k vývoji uživatelského prostředí (UI) příliš nevyužívá praktik MDD[2]. Každá změna v modelu nebo v požadavcích na aplikaci je velice nákladná. Všechny změny se musí manuálně přenést do UI. Dobrým příkladem je formulář pro získávání dat od uživatele. Již v modelu aplikace jsou zaneseny informace o datových typech a o omezeních jednolitých polí. V klasickém přístupu k vývoji UI je nutné manuálně tyto informace duplikovat i v této vrstvě.

Na druhou stranu entitní třídy generované z UML modelu neobsahují všechny nutné informace pro vytvoření UI[3]. Například zde není informace o pořadí jednotlivých prvků, o rozložení, o zobrazení na základě role uživatele, atd. Z výše uvedeného vyplývá potřeba dalších view anotací, které dodefinují finální vzhled aplikace. Sada view metadat by neměla být uzavřená, tak aby se dalo vyhovět i velice specifickým požadavkům uživatelského prostředí dané aplikace. Přesně tuto situaci řeší Aspect Faces framework, kterému se věnuje následující kapitola.

3.4 Aspect Faces

Aspect Faces[6] (dříve znám pod jménem jFormBuilder) je framework pro automatizaci tvorby uživatelského prostředí pomocí generování fragmentů UI z datového modelu a jeho metadat. Pomocí Aspect Faces (AF)[4] nedochází k duplikování informací v UI vrstvě aplikace, které jsou již promítnuty v datovém modelu. Díky této redukci duplikací lze dosáhnout rychlejšího vývoje a podstatně snadnější údržby UI vrstvy. Aspect Faces framework je založen na principech aspektově orientovaného programování. Aspektem v tomto případě je vizuální prezentace dat.

AF je plně kompatibilní s J2EE specifikací. Podporuje všechny knihovny JSF komponent, jako jsou RichFaces, PrimeFaces, Tomahawk, ICEFaces a další. V současné době jsou v přípravě i šablony HTML5 komponent. AF framework je k dispozici zdarma pro open-source projekty nebo také pro akademické využití.

3.4.1 Aspektově orientované programování

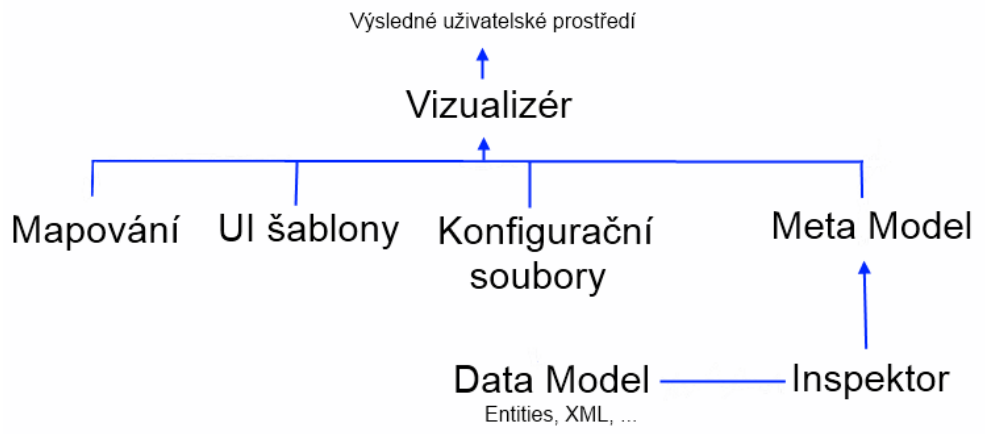
Metodika Objektově Orientované Programování (OOP) se snaží o oddělení a zapouzdření koncernů do samostatných entit. Problémem zůstávají koncerny, které se existují ve více částech softwaru. Tyto koncerny jsou nazývány průřezové (*crosscutting concerns*). Nejběžnějším příkladem průřezového koncernu může být autorizace, transakce, logování, atd. Aspektově orientované programování (AOP)[7] řeší, jak separovat a zapouzdřit část kódu prolínající

[4] http://www.omg.org
[5] http://www.omg.org/gettingstarted/what_is_uml.htm
[6] http://www.aspectfaces.com/

se celým programem do tzv. aspektu a aplikovat ho na různé objekty. Aspekt pozměňuje chování kódu, na který je aplikován, pomocí přidání dodatečného funkcionality.

3.4.2 Inspekce zdrojového kódu

Architektura současných webových aplikací rozděluje aplikaci do tří vrstev: prezentační, business a datovou. Datová vrstva je tvořena pomocí entit, které přestavují persistentní data. Dále ale určuje i to, co se bude od uživatele sbírat, jaká jsou kladena omezení na data a jak se mají zobrazit. Při běžném přístupu k vývoji uživatelských prostředí, je vývojář nucen tyto informace duplikovat. Inspekce zdrojového kódu[5] umožňuje provádět vizualizaci dat bez těchto duplikací.



Obrázek 3.2: Koncept automatizace skrze inspekci.

Inspektor provede inspekci entit v persistentní vrstvě aplikace a vytvoří metamodel. Inspektor vyčte atributy entit a speciální značky. Značky mohou být JPA anotace, Bean Validation, Hibernate Validator nebo i vlastní uživatelské anotace, které jsou zaregistrované s daným deskriptorem. Jak vytvořit vlastní UI anotace je rozebráno v implementační části této práce 5.2.1.5.

V konfiguraci jsou definované všechny anotace, umístění widgetů, layoutů a defaultní nastavení. Mapování definuje, jak jsou provázány atributy třídy s UI widgety. Widgety lze mapovat ve třech různých profilech.

- ReadOnly
- Edit
- Search

Šablony UI Widgetů lze definovat pro libovolnou knihovnu komponent, kterou může být libovolná JSF knihovna komponent nebo i prosté HTML či XML. V aplikaci eForm je použita sada šablon napsaných pro PrimeFaces[7] knihovnu.

[7] http://www.primefaces.org

Dalším důležitým prvkem jsou layouty, které určují rozmístění polí ve výsledném fragmentu UI. AF layout je tvořen HTML tabulkou. Lze tedy vytvářet rozmanité layouty a tím co nejvíce přizpůsobit daný fragment cílové aplikaci.

Na základě konfigurace spojí vizualizér injektovaný atribut třídy s UI widgetem. Vizualizér vytváří výsledné uživatelské prostředí v závislosti na daném profilu, roli uživatele a zvoleným layoutem. Pokud není definovaný žádný layout, tak se použije výchozí. AF framework je navržen tak, aby bylo možné zahájit generování pouze s nutnou minimální konfigurací.

3.4.3 Ukázka Aspect Faces aplikace

V této části je uvedena ukázka použití AF pro vykreslení vstupního formuláře pro zadání adresy uživatele. Entita Address obsahuje doplňkové UI anotace, které určují výsledný vzhled formuláře. Dále jsou zde informace o omezeních kladených na data a samotný datový typ atributu.

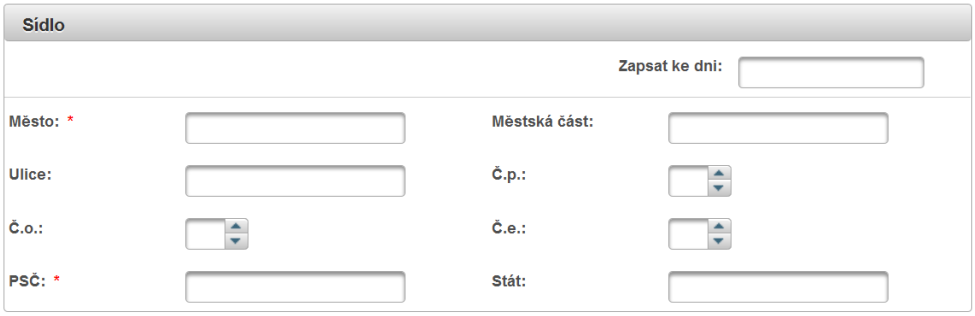
```
@Entity
public class Address extends EntityObject{

    @OdRequired
    @UiOrder(1)
    private String city;

    @UiOrder(2)
    @OdRequired
    @OdRegex(pattern="[1-9]{1}[0-9]{2}\\s?[0-9]{2}")
    private String zipCode;

    ...
}
```

Aby bylo generované uživatelské prostředí co nejvíce uživatelsky přizpůsobitelné dané aplikaci, tak AF negeneruje celé rozhraní, ale pouze jeho zvolené fragmenty. Fragment formuláře pro získání adresy zařadíme do facelet stránky pomocí následující značky.

```
<p:panel header="Sídlo" >
  <af:ui instance="#{fc.address}" edit="true" layout="address-layout.xhtml"/>
</p:panel>
```

Z výše uvedeného výpisu je patrné, že zvolený profil generování je edit a rozmístění bude odpovídat layoutu address, tedy dva sloupce polí. Vykreslený formulář je zobrazen na obrázku 3.3. AF v edit profilu generuje editovatelný formulář. Automaticky přidává všechna omezení na jednotlivá pole. Odpadá duplikování informace o omezeních jednotlivých položek v uživatelském rozhraní. Povinná pole Město a PSČ jsou automaticky označena červenou hvězdičkou. Součástí této práce je také návrh a implementace on-demand validátorů, které validují formulář na základě toho, která akce byla vyvolána. V ukázce je přítomnost

on-demand validátorů vyjádřena pomocí anotací `@OdRequired` a `@OdRegex`. Více o on-demand validátorech naleznete dále v kapitole věnované návrhu a implementaci 5.1.2. V této ukázce a celé práci používám Aspect Faces šablony pro PrimeFaces knihovnu komponent. PrimeFaces knihovna nabízí dostatečný počet komponent pro tvorbu formulářů, podporuje AJAX, validaci, skinování, atd. Více o knihovně PrimeFaces naleznete na stránkách projektu[8].



Obrázek 3.3: Vykreslený fragment formuláře.

Z ukázky Address entity je patrné, že Aspect Faces nabízí podporu pro anotace. AF obsahuje sadu view anotací, pomocí kterých je možné ovlivnit výsledný vzhled atributu. V ukázce je použita anotace `@UiOrder`, která říká v jakém pořadí budou prvky vykresleny. Pokud tato anotace není uvedena, tak se prvky vykreslí v blíže nespecifikovaném pořadí, protože ve vnitřní implementaci AF jsou brány jako set. Úplný seznam podporovaných anotací naleznete v dokumentaci projektu[9]. Seznam anotací lze rozšířit i o vlastní anotace, viz implementační část práce 5.2.1.5.

Důležitým aspektem v generování je také zvolený layout. AF podproruje systém layoutů tak, že ke každému fragmentu lze přiřadit vlastní layout. Pokud nezvolíme žádný, opět se použije výchozí. AF tak snižuje nutnou konfiguraci na minimum. V následujícím výpisu je ukázka address layoutu použitého v ukázce výše.

```
<table class="address-edit-layout" >
  <af:iteration-part maxOccurs="4">
   <tr>
      <td>$af:next$</td>
      <td>$af:next$</td>
   </tr>
  </af:iteration-part>
</table>
```

S použitím Aspect Faces frameworku lze ušetřit až 25% úsilí věnovaného do vývoje uživatelského prostředí a mnohem více při následné údržbě. Vlastní měření úspory kódu v různých případech uvádím na konci práce v kapitole 7.

[8] http://www.primefaces.org/
[9] http://wiki.codingcrayons.com/display/af/Annotation+Descriptor+Reference+Table

3.4.4 Aspekty generování

Generování fragmentů je ovlivňováno mnoha aspekty. Hlavní chod generátoru je dán konfigurací a anotacemi, které se vzájemně doplňují. Dalším aspektem jsou role uživatelů, které ovlivňují co se jak zobrazí danému typu uživatele. Existují zde různé profily generování - Edit, ReadOnly a Search, které určují formu zobrazení (náhled, vstupní pole, atd.). Díky této široké škále aspektů lze generovat unikátní bohatá uživatelská prostředí. Další možností, jak ovlivnit výsledný vzhled, je zanesení dalších podmínek do šablon widgetů ve formě výrazů v Expression Language (EL)[10].

3.5 Příbuzné projekty

V této kapitole jsou rozebrány příbuzné projekty k Aspect Faces a jejich vzájemné porovnání. Nejprve se zaměřím na Apache Isis framework, který vznikl z původního projektu Naked Objects, dále se věnuji projektu MetaWidget, který je také postaven na inspekci kódu. Záměrně se zde věnuji pouze nástrojům a technologiím kompatibilním s J2EE platformou, na které je postavena aplikace eForm a se kterou je kompatibilní AF framework.

3.5.1 Apache Isis

Apache Isis[20] je java framework pro rychlý domain-driven vývoj webových aplikací. Vývoj je soustředěn v datové vrstvě aplikace a Apache Isis vytváří uživatelské rozhraní, ale i persistenci, security a další. Apache Isis framework vznikl rozdělením původního projektu Naked Objects na dvě separátní větve pro Javu a .NET. Větev pro .NET platformu reprezentuje framework pod původním názvem Naked Objects. Apache Isis jako takový vznikl v listopadu roku 2012.

Apache Isis je zaměřen spíše na použití konvencí v modelu, než na dodatečné konfigurační soubory. Vývoj je soustředěn hlavně na datový model, který je složený z POJO objektů a anotací. Isis framework generuje uživatelské prostředí webové aplikace nebo také REST aplikační rozhraní pro webové služby. Apache Isis framework je navržen jako snadno rozšiřitelný a přizpůsobitelný dané aplikaci. Nabízí širokou škálu různých zobrazovacích technologií, jako jsou např. Apache Wicket, Scimpi a HTML5. Dále různé technologie pro security a persistenci. Pro persistenci to jsou např. JDO, noSQL, SQL a XML. Pro security pak např. LDAP, SQL nebo Shiro. Úplný seznam podporovaných technologií je uveden v dokumentaci[11] k projektu.

Hlavním rozdílem mezi Apache Isis a Aspect Faces frameworkem je fakt, že Isis neumožňuje inspekci stávající architektury a vše tak musí být explicitně uvedeno v datovém modelu aplikace. V následující části je uveden MetaWidget framework, který již podporuje inspekci stávající architektury a je tedy Aspect Faces frameworku podobnější.

[10] http://docs.oracle.com/javaee/6/tutorial/doc/gjddd.html
[11] Apachi Isis dokumentace: http://isis.apache.org/documentation.html.

3.5.2 MetaWidget

MetaWidget[31] má velice blízko k Aspect Faces frameworku. Jedná se o open-source framework pro runtime generování fragmentů UI z datového modelu aplikace[12]. Framework provádí inspekci back-end architektury aplikace a generuje sadu nativních widgetů pro front-end framework, který používá daná aplikace. MetaWidget podporuje celou řadu front-end i back-end technologií. Pro front-end to jsou např. JSF, JSP, Google Web Toolkit, Spring web MVC a další. Pro back-end je podpora také velice široká, např. JavaBeans, Java Persistence API, Hibernate, Annotation, Groovy, Seam a mnoho dalších. Celý seznam podporovaných technologií naleznete v dokumentaci k projektu[31].

Hlavním rozdílem mezi Aspect Faces, které používám v rámci této práce, a MetaWidget frameworkem je způsob generování fragmentů. AF sestavuje každý fragment z tagů, které jsou plně uživatelsky modifikovatelné a lze tak dosáhnout rozmanitých uživatelských prostředí. Naopak MetaWidget používá svoji uzavřenou knihovnu a pokud chceme nějaký element přizpůsobit aplikaci, tak je nutný zásah do konfigurace nebo je nutné provést záměnu výchozí značky v místě generování.

V porovnání AF a MetaWidget je zřejmé, že fragmenty generované AF frameworkem lze snadněji přizpůsobit dané aplikaci. Porovnání konfiguračních souborů u obou projektů dojdeme k závěru, že MetaWidget má obsáhlé a složité konfigurační soubory. U AF fragmentů dosahujeme přizpůsobení dané aplikaci pomocí jednoduché modifikace šablon, nikoli změny knihoven v konfiguraci. Naopak MetaWidget má širší podporu front-end frameworků a je distribuován pod licencí LGPL, což vývojáři umožňuje zdarma nasadit framework i v komerčních projektech. Aspect Faces je zdarma pouze pro open-source a akademické projekty.

[12] Tento princip se také někdy označuje jako Object/User Interface mapping (OIM). MetaWidget dokumentace označuje framework jako OIM tool.

Kapitola 4

Analýza

V této analytické části se nejdříve zaměřím na případy užití a jejich popis. Dále se podívám na domény, které se v systému vyskytují a na jejich vzájemné vazby. Nakonec rozeberu architekturu systému a plán nasazení.

4.1 Případy užití

Diagram případů užití je modelem činností, které může systém vykonávat (na základně interakce s uživatelem). Tyto případy užití vycházejí z analýzy funkčních požadavků, viz kapitola 2.2.1. V systému se vyskytují dva aktéři. První aktér (**Uživatel**) je skutečný uživatel, který bude eForm aplikaci používat. Uživatele není nutné rozlišovat do více rolí, protože eForm aplikace je jednoduchá aplikace v režimu SaaS[1]. Uživatel vytvoří požadovaný typ formuláře, uloží a následně vyexportuje. Nelze předpokládat, že by někdo registroval společnost každý den. Není zde tedy třeba implementovat správu uživatelských účtů atp.

Druhým aktérem je **Časovač**. Časovač je automatizovaný skript, který se volá vždy jednou za den a provádí úklid neaktivních formulářů, které jsou v systému déle než určený počet dní. Diagram případů užití je uveden na obrázku 4.1. V následujícím textu popíšu jednotlivé případy užití a jejich scénáře.

UC-1: Vytvoření formuláře

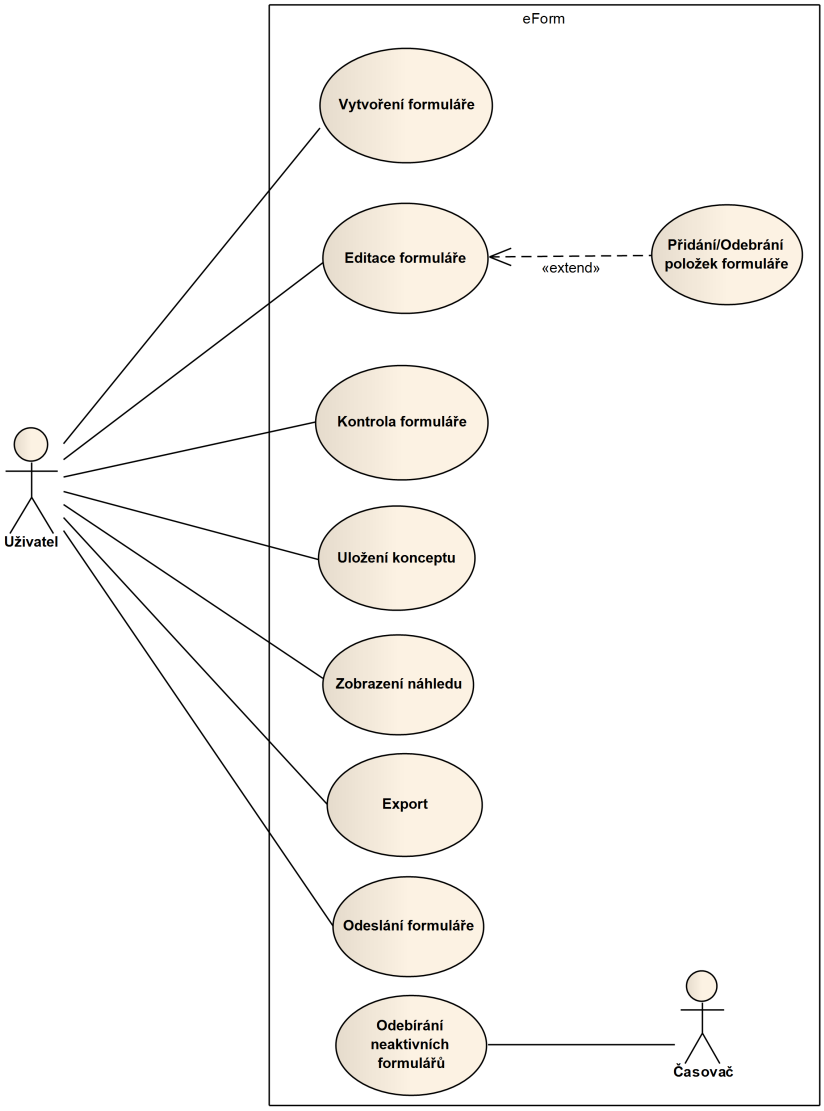
Zahájení práce s formulářem musí být rychlé, proto aplikace nevyžaduje registraci. Uživatel je identifikován pouze na základě emailové adresy a unikátního kódu formuláře.

- Uživatel nejdříve zadá svoji emailovou adresu a vybere požadovanou právní formu registrované společnosti.
- Systém odešle potvrzující email s identifikátorem formuláře. Tímto způsobem dojde k vytvoření nového formuláře a k ověření uživatele.

[1] **Software as a Service** - Model nasazení softwaru, kde je aplikace dostupná přes internet jako služba. Není nutná instalace aplikace na vlastní počítače.

UC-2 a UC3: Editace formuláře (přidávání a odebírání položek)
Pokud již máme vytvořený formulář pro jakoukoli právní formu společnosti, tak známe jeho unikátní identifikátor UID.

- Po zadání UID se uživateli otevře editovatelný formulář.
- Uživatel vyplní jednotlivé položky.
- Uživatel přidá a vyplní volitelné části a položky formuláře.



Obrázek 4.1: Diagram případů užití.

UC-4: Kontrola formuláře

V průběhu editace formuláře si uživatel může ověřit, zda má formulář vyplněna všechna povinná pole a zda uvedené údaje odpovídají požadovanému formátu. Po provedení kontroly je uživatel informován o výsledku a případných chybách. Pokud je formulář vyplněn správně, tak dojde k jeho uložení.

UC-5: Uložení konceptu

Formuláře pro registraci nových firem do obchodního rejstříku obsahují mnoho částí a položek. Takový formulář není snadné vyplnit najednou. Uživatel si v průběhu vyplňování uloží formulář jako koncept a později se k němu může vrátit (na základě identifikátoru UID).

UC-6: Zobrazení náhledu

Před zahájením tisku nebo exportu formuláře si uživatel může zobrazit náhled na vyplněný formulář. Náhled zobrazuje, jak bude výsledný vyplněný formulář vypadat. Tento náhled se může zobrazit přímo z režimu editace formuláře.

UC-7: Export

Z analýzy funkčních požadavků vyplývá požadavek na export do dvou formátů, PDF a XLS, viz kapitola 2.2.1. U tohoto případu užití jsou možné dva přístupy, jak se na export dívat. Povolit export formuláře bez provedení kontroly nebo exportovat striktně pouze ověřené formuláře. Zvolil jsem první možnost, protože by mohlo být užitečné si zobrazit náhled rozepsaného formuláře v cílovém formátu. V případě XLS formátu je také možné dokončit formulář v jiném programu.

- Uživatel vyplní formulář.
- Zvolí výsledný formát.
- Po vygenerování požadovaného souboru si ho stáhne a uloží.

UC-8: Odeslání formuláře

V některých případech může být užitečné neukládat formulář lokálně, ale rovnou ho odeslat jako přílohu emailu.

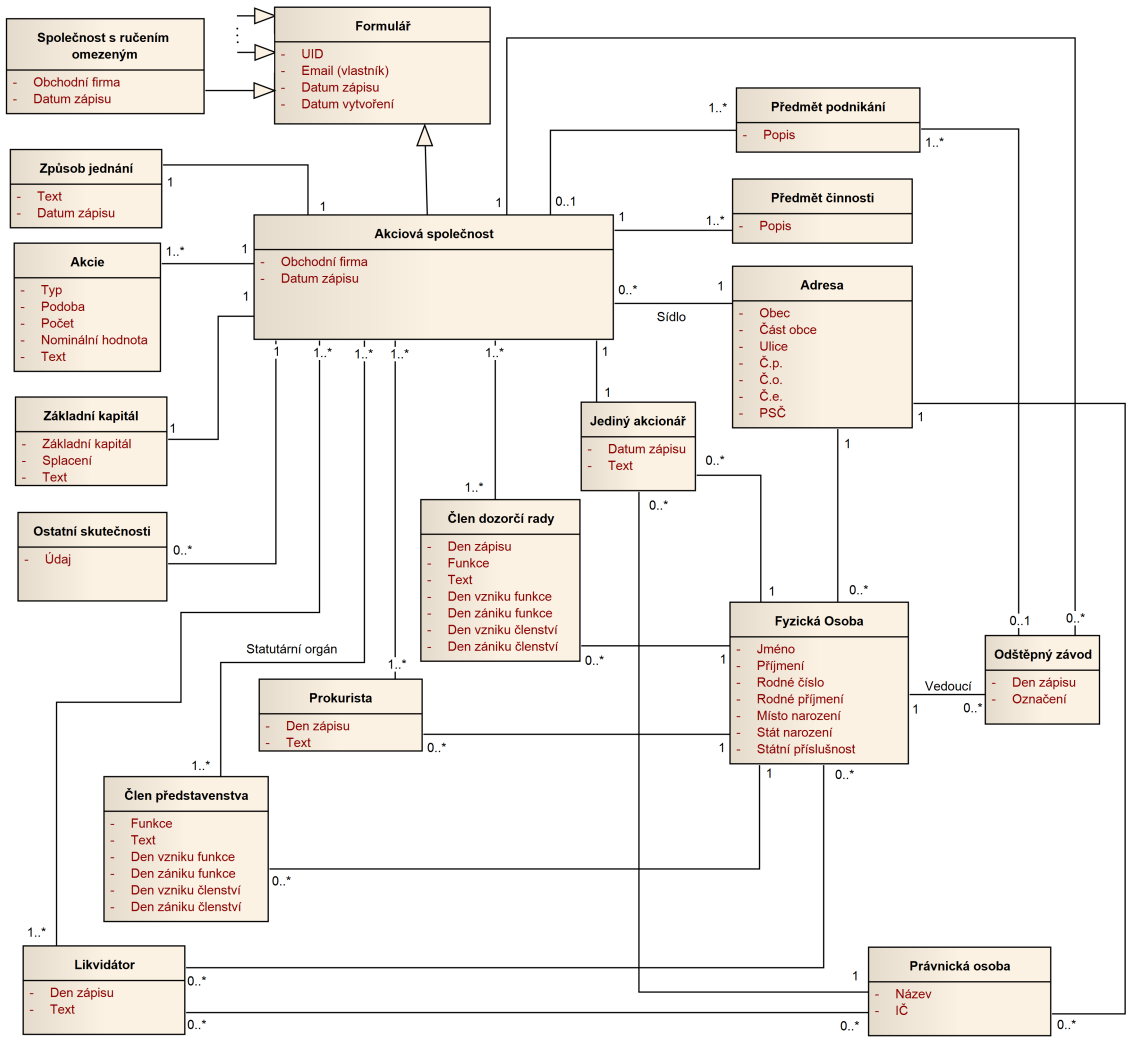
- Uživatel vyplní formulář.
- Vybere požadovaný formát formuláře.
- Zadá cílovou emailovou adresu a odešle formulář.

UC-9: Odebírání neaktivních formulářů

V rámci snahy o zjednodušení uživatelského rozhraní a možnosti práce úplně bez registrace, je nutné zavést nějaký systém, který by automaticky spravoval formuláře – promazával již neaktivní formuláře. V diagramu 4.1 je uveden aktér **Časovač**, který je právě za toto zodpovědný. Jedná se o automatizovaný skript, který se jednou za den spustí a odstraní všechny formuláře, které jsou v systému déle než určený počet dní. Toto časové období je stanoveno na 20 dní.

4.2 Analytický doménový model

Na obrázku 4.2 je znázorněn analytický doménový model aplikace. Pro větší přehlednost se zaměřuje pouze na jeden typ právní formy – akciová společnost. Hlavní doména je **Formulář**. Doména eviduje základní údaje o formuláři a jeho vlastníkovi. Od domény **Formulář** jsou odvozeny všechny typy formuláře pro konkrétní právní formy. Pro urychlení tvorby formuláře v aplikaci eForm bylo upuštěno od nutnosti registrace uživatele. Informace o vlastníkovi jsou zahrnuty v doméně **Formulář**, který má unikátní identifikátor UID a emailovou adresu vlastníka, přes kterou dochází k ověření uživatele. Identifikátor UID musí být dostatečně složitý, protože bude náhradou hesla k uživatelskému účtu. UID je zvoleno jako osm znaků dlouhý řetězec, který se skládá z písmen a číslic. `UID` = [a-zA-Z0-9]{8} .



Obrázek 4.2: Analytický doménový model - akciová společnost.

4.3 Popis architektury

Cílová platforma aplikace je určena přímo ze zadání. Je dána použitím Aspect Faces frameworku, který je implementovaný v jazyce Java. Aplikace eForm je webová aplikace s třívrstvou architekturou. Cílová platforma je tedy Java EE 6 [9], která se využívá u rozsáhlých podnikových řešení, ale i u menších webových projektů. Java EE 6 je komplexní sada pravidel, doporučení a technologií, které slouží k návrhu, vývoji a nasazení vícevrstvých aplikací. Architektura dělí aplikaci do tří základních vrstev – prezentační, business a datové. Obrázek 4.3 zachycuje architekturu i s použitými technologiemi v daných vrstvách.

Obrázek 4.3: Architektura aplikace eForm.

Třívrstvá architektura zajišťuje striktní oddělení dat od řídicí logiky a jejich prezentace. Tento přístup umožňuje vytvoření více pohledů na stejná data. Aplikace je snadno modifikovatelná a udržovatelná. Java EE 6[2] specifikace je přímo navržena pro tvorbu vícevrstvých aplikací a pro každou z nich přináší sadu volitelných technologií a frameworků. V následující části rozeberu jednotlivé vrstvy a technologie použité v každé z nich.

4.3.1 Prezentační vrstva

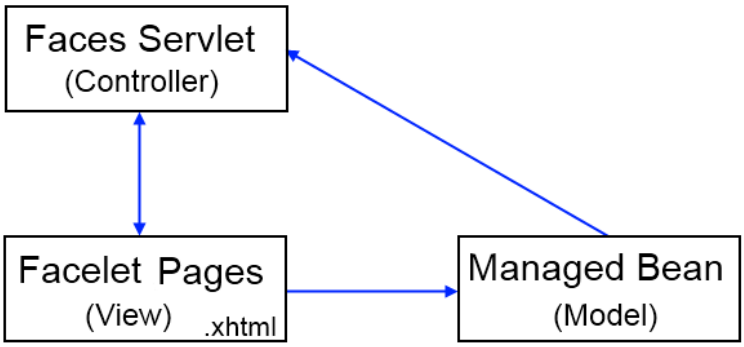
Z pohledu této práce je prezentační vrstva nejdůležitější část. Předmětem práce je nasazení a otestování technologie pro automatické generování fragmentů uživatelského prostředí z datového modelu a jeho metadat. Automatizované generování zajišťuje framework Aspect Faces, viz kapitola 3.4, který v dynamickém režimu[3] pracuje s MVC frameworkem Java Server Faces v aktuální verzi 2.1.

[2] Více o platformě a jednotlivých technologií naleznete zde: http://www.oracle.com/technetwork/java/javaee/tech/index.html.

[3] Aspect Faces framework v dynamickém režimu sestaví facelet šablonu z jednotlivých fragmentů a následně ji interpretuje, tj. vykreslí HTML a JavaScript. Ukázku Aspect Faces v dynamickém režimu nalzeznete zde: http://wiki.codingcrayons.com/display/af/Dynamic+use+of+AspectFaces.

4.3.1.1 Java Server Faces

JavaServer Faces (JSF)[6] tvoří rámec pro tvorbu webových aplikací na platformě Java. Framework je založený na komponentách. Při návrhu specifikace byl kladen důraz na snadnou rozšiřitelnost o nové komponenty. JSF je postaveno nad architektonickým návrhovým vzorem Model–View–Controller (MVC)[12].



Obrázek 4.4: Java Server Faces – implementace MVC

Implementace MVC Java Server Faces frameworkem je znázorněna na obrázku 4.4. Vzor MVC rozděluje prezentační vrstvu na tři separátní komponenty.

- **Model** – V JSF aplikacích je model tvořen pomocí tzv. Managed Bean, které vytváří rozhraní mezi logikou aplikace a DAO vrstvou, viz business a datová vrstva aplikace.
- **View** – Převádí data reprezentovaná modelem do grafické podoby přívětivé uživateli. V JSF je tvořena pomocí tzv. Render Kitu, který pro každou komponentu obsahuje jednu či více vykreslovacích tříd, které danou komponentu vykreslí do formátu interpretovatelného webovým prohlížečem (HTML, CSS, JavaScript).
- **Controller** – Controller reaguje na události, které generuje uživatel a provádí změny datového modelu nebo pohledu. V JSF aplikaci tento řadič nepíšeme, je součástí frameworku (Faces Servlet).

V této práci používám JSF s novější facelet[4] zobrazovací technologií, která umožňuje lepší práci se šablonami, než starší Java Server Pages (JSP)[5] technologie. Aspect Faces v současné době ani nemá žádnou podporu pro JSP, které jsou v praxi stále ještě velmi časté.

[4] Facelet je framework pro tvorbu prezentační vrstvy postavené na JSF. Plně implementuje všechny nové vlastnosti z Java EE 6 specifikace a nahrazuje tak starší JSP. Pro tvorbu stránek využívá XHTML, Expression Language a systém šablon. Více naleznete zde http://docs.oracle.com/javaee/6/tutorial/doc/gijtu.html.

[5] Java Server Pages je původní technologie pro tvoření dynamických stránek na platformě Java. Kombinují statický HTML obsah a JSP skripty. Díky JSP je vývojář odstíněn od Servletu a může se soustředit pouze na vlastní stránky.

4.3.2 Business vrstva

Business vrstvu v aplikaci eForm tvoří Enterprise Java Beans (EJB)[17] ve verzi 3.1. Tato nová specifikace, kterou přináší Java EE 6, usnadňuje práci s EJB. Již se nemusí definovat rozhraní, přináší asynchronní volání a plánování úloh ve stylu cron. Dále přináší EJB Lite pro webový profil. Tato specifikace vznikla v reakci na Spring framework[6]. EJB jsou serverové komponenty, které usnadňují vývoj distribuovaných systémů. EJB specifikace umožňuje vytvářet robustní, škálovatelné a bezpečné aplikace.

4.3.3 Datová vrstva

Datová vrstva tvoří rozhraní mezi databázovým strojem a zbytkem aplikace. Aplikace eForm využívá relační databázi, je tedy nutné nasadit nějaký ORM[7] framework. Součástí specifikace Java EE 6 je Java Persistence API (JPA)[11]. JPA definuje standardní rozhraní pro relačně-objektové mapování. Aplikace eForm využívá Hibernate implementaci JPA v nejnovější verzi 2.0. JPA 2.0 přineslo mnoho nových vylepšení, které umožňují snadnější práci s entitami. Nejpodstatnější změnou představuje Criteria API pro tvorbu dynamických dotazů, rozšířené mapování kolekcí, Cache API a další. Velkou výhodou použití ORM frameworku je snadný potenciální přechod z relační databáze na objektovou.

4.4 Plán nasazení

Aplikace eForm je webová aplikace, která bude nabízena zdarma v distribučním modelu Software as a Service (SaaS). Tento distribuční model přesně vystihuje potřeby aplikace eForm. Kvůli vytvoření několika formulářů, si uživatel nemusí instalovat žádný software, ale využije eForm jako službu na internetu. Aplikace bude dostupná na veřejné adrese. K nasazení EJB aplikace je nutné aplikační server, který implementuje EJB specifikaci. Pro nasazení eForm aplikace jsem zvolil JBoos AS 7.1[8] aplikační server, který má plně certifikovaný webový profil a úplně implementovanou sadu Java EE 6 specifikací.

Na obrázku 4.5 je znázorněn diagram nasazení. Aplikace eForm bude nasazena v cloud prostředí OpenShift[9] od společnosti RedHat[10]. OpenShif jsem zvolil z důvodu, že nabízí JBoss AS 7.1 v distribučním modelu Plarform as a Service (PaaS) a to i zdarma. V PaaS modelu je uživateli nabízena možnost nasazení vlastní aplikace v cloud infrastruktuře majitele. Uživatel nemá možnost ovlivnit tuto infrastrukturu, může spravovat pouze vlastní aplikace. PaaS tak tvoří střední vrstvu cloud technologie. Poskytuje nástroje, aplikační servery, databázové servery, nástroje pro management atp. Dalším příkladem PaaS je např. Google App Engine[11]. Nasazením v cloudovém prostředí získáme vysokou dostupnost, ška-

[6] Spring je open-source framework pro tvorbu J2EE aplikací. Přináší alternativu k EJB a umožňuje tak nasadit aplikace i bez aplikačního serveru pouze na server, který implementuje webový profil.

[7] Objektově relační mapování je obecná technika, která slouží k automatické konverzi relačních dat do objektově orientovaných jazyků.

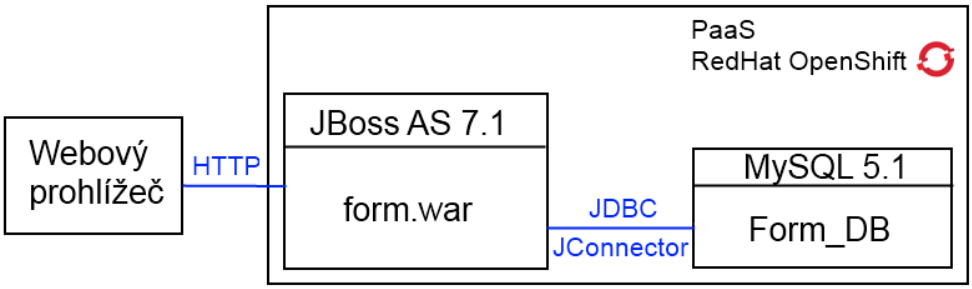
[8] JBoss AS 7.1 lze zdarma stáhnout zde: http://www.jboss.org/jbossas.

[9] https://openshift.redhat.com/

[10] https://www.redhat.com

[11] https://developers.google.com/appengine/

lovatelnost a zároveň nám odpadá starost o samotný hardware. Více o cloud technologiích naleznete zde [1].



Obrázek 4.5: Diagram nasazení.

Z diagramu nasazení dále vyplývá použití MySQL[16] databázového stroje, který je opět hostovaný v OpenShift prostředí. MySQL je velice populární open-source relační databáze. MySQL není tak robustním systémem jako např. Oracle databáze, proto je poměrně rychlá a méně náročná na hardware. Další velkou výhodou je podpora pro celou škálu programovacích jazyků, jako jsou např. Java, PHP, Python, C++ a .NET. Pro platformu Java existuje JDBC ovladač JConnector[12].

[12] http://dev.mysql.com/downloads/connector/j/

Kapitola 5

Návrh řešení a realizace

Tato kapitola se postupně věnuje návrhu a realizaci všech rozšíření pro framework Aspect Faces. Nejprve se zaměřím na provádění validace formuláře na základě podmínek, dále je zde popsán automatizovaný export formulářů do PDF resp. XLS formátu.

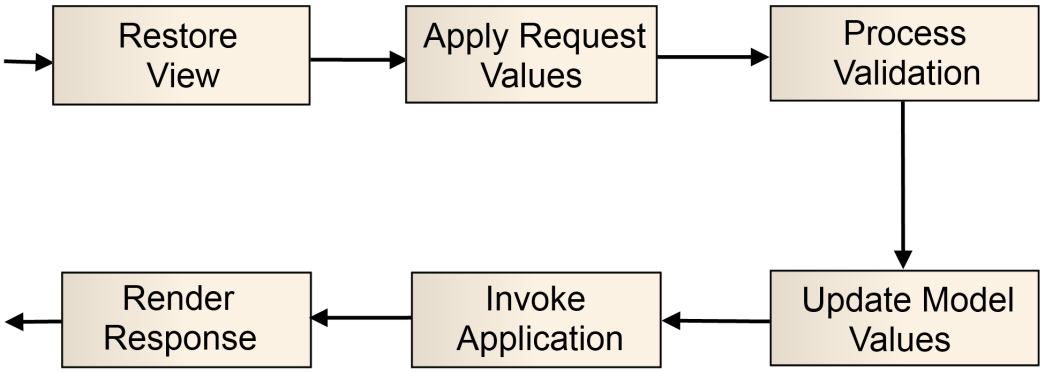
5.1 Podmíněná validace

Formuláře pro registraci nové společnosti do obchodního rejstříku jsou relativně dlouhé. Obsahují mnoho částí a položek. Uživatel, s největší pravděpodobností, nevyplní takto dlouhý formulář najednou, ale bude chtít si rozepsaný formulář uložit a později se k němu vrátit.

Ve výpisu kódu je znázorněn formulář pro odeslání nějaké zprávy. Po stisku tlačítka Send je nutné provést validaci všech polí a v případě, že kontrola polí proběhla v pořádku, se formulář odešle.

```
<h:form>
 <h:inputText  id="date" value="#{msg.date}" >
   <f:validateRequired />
 </h:inputText>
 <h:inputText  id="email" value="#{msg.address}" >
   <f:validateRequired />
 </h:inputText>
 <h:inputText  id="message" value="#{msg.message}" >
    <f:validateRequired />
 </h:inputText>
 ...

 <h:commandButton action="#{msg.save}" value="SaveDraft" id="SaveDraft"/>
 <h:commandButton action="#{msg.send}" value="Send" id="Send"/>
 <h:commandButton immediate="true" value="Cancel" id="cancel"/>
</h:form>
```

Dále chceme rozepsaný formulář uložit jako koncept. Zde vzniká problém, jak přeskočit fázi validace z Java Server Faces životního cyklu [6] a uložit rozepsaný formulář jako koncept. JSF Životní cyklus je znázorněn na následujícím diagramu.



Obrázek 5.1: Životní cyklus – Java Server Faces

Validace je podmíněná ve smyslu, která akce byla vyvolána (které tlačítko bylo stisknuto). Pomocí parametru immediate je možné fázi validace přeskočit, ale zároveň se také neobnoví datový model a formulář tak nebude uložen. V současné verzi frameworku JSF 2 není pro tento scénář žádná podpora, proto je nutné implementovat vlastní rozšíření, viz další podkapitoly.

5.1.1 OmniFaces řešení

OmniFaces[32] je rozšiřující knihovna pro framework Java Server Faces. Soustřeďuje se na zjednodušení běžných úkolů, které se často při psaní JSF aplikací opakují. Knihovna je kompatibilní s velkou množinou JSF knihoven. Aspect Faces framework také podporuje mnoho JSF implementací jako jsou RichFaces, ICEFaces, PrimeFaces, Tomahawk a další, proto je nutné, aby podmíněné validátory byly nezávislé na konkrétní JSF implementaci.

Knihovna OmniFaces přináší tag `<o:ignoreValidationFailed>`, který umožňuje skrýt to, že validace na některých polích selhala. Příklad použití OmniFaces řešení je nastíněn v následující ukázce.

```
<o:form>
  <o:outputLabel for="in" value="Foo" />
  <h:inputText id="in" required="true" />
  ...
  <h:commandButton value="Saveaction="#{crt.save}">
    <o:ignoreValidationFailed />
  </h:commandButton>
</o:form>
```

Problémem tohoto řešení je fakt, že zde nedochází k opravdovému přeskočení fáze validace, ale pouze ke skrytí u polí, kde validace selhala. V datovém modelu jsou tedy obnoveny pouze prvky, u kterých validace neselhala. Z tohoto důvodu a také z důvodu, že Aspect Faces jsou nezávislé na konkrétním dodavateli komponent, jsem se rozhodl napsat vlastní rozšíření pro Aspect Faces - JSF on-demand validation a nepoužívat řešení nějaké konkrétní JSF implementace.

5.1.2 JSF on-deman validation

JSF on-deman validation řeší problém podmíněné validace. S tímto Aspect Faces rozšířením je možné přeskočit fázi validace JSF životního cyklu a zároveň obnovit datový model aplikace. Princip podmíněných validátorů je postaven na překrytí standardních validátorů objekty implementující rozhranní `javax.faces.validator.Validator`.O tomto principu se na javalab blogu zmiňuje André Pankraz [15].

Ve výchozím stavu jsou všechny validátory aktivní. Pokud potřebujeme pro nějakou akci validaci přeskočit, přidáme prefix `_ff` k id prvku, který tuto akci vyvolává. V následujícím kódu je již formulář s podporou on-demand validace.

```
<h:form>
 <h:inputText  id="date" value="#{msg.date}" >
   <f:validateRequired />
 </h:inputText>
 <h:inputText  id="email" value="#{msg.address}" >
   <f:validateRequired />
 </h:inputText>
 <h:inputText  id="message" value="#{msg.message}" >
   <f:validateRequired />
 </h:inputText>
 ...

 <h:commandButton action="#{msg.save}" value="SaveDraft" id="_ffSaveDraft"/>
 <h:commandButton action="#{msg.send}" value="Send" id="Send"/>
 <h:commandButton immediate="true" value="Cancel" id="cancel"/>
</h:form>
```

Pro tlačítko SaveDraft se validace neprovede a formulář bude uložen jako koncept. Toto řešení je bezpečné v porovnání s použitím parametru pro vypínání validace.

```
<f:param name= "validation" value= "true" />
```

Parametr validation je snadné podvrhnout a tak potenciálně narušit bezpečnost aplikace. Pokud někdo přidá prefix `_ff` k POST parametru, tak JSF akci neprovede. Další výhodou tohoto řešení je, že nevyžaduje JavaScript, je tedy bezpečné i v případě, že je JavaScript v prohlížeči zakázaný.

5.1.2.1 Podporované validátory a konfigurace

Knihovna JSF on-demand validation podporuje standardní JSF 2 validátory. Zde je jejich výčet.

- DoubleRangeValidator
- LengthValidator
- LongRangeValidator
- RegexValidator
- RequiredValidator

Pro použití podmíněné validace se musí nejprve přeregistrovat standardní validátory na podmíněné. Není nutné registrovat všechny, stačí jen ty, které chceme, aby byli podmíněné. On-demand validátory se konfigurují jako běžné uživatelské validátory v souboru faces-config.xml.

```
<validator>
  <validator-id>javax.faces.Required</validator-id>
  <validator-class>cz.ctu.odv.OnDemandRequiredValidator</validator-class>
</validator>
<validator>
  <validator-id>javax.faces.Length</validator-id>
  <validator-class>cz.ctu.odv.OnDemandLengthValidator</validator-class>
</validator>
  <validator>
  <validator-id>javax.faces.DoubleRange</validator-id>
<validator-class>cz.ctu.odv.OnDemandDoubleRangeValidator</validator-class>
  </validator>
  <validator>
<validator-id>javax.faces.LongRange</validator-id>
  <validator-class>cz.ctu.odv.OnDemandLongRangeValidator</validator-class>
  </validator>
<validator>
  <validator-id>javax.faces.RegularExp</validator-id>
  <validator-class>cz.ctu.odv.OnDemandRegularExpValidator</validator-class>
</validator>
```

5.1.2.2 Rozšíření pro Aspect Faces

Knihovna JSF on-demand validation rozšiřuje framework Aspect Faces o dalších pět anotací pro podmíněnou validaci. Tyto anotace přináší obdobně jako Bean Validation[1] nebo Hiber-

[1] Nový model validace, který je součástí JEE 6 specifikace. Umožňuje definovat omezení přímo v datovém modelu. Omezení jsou definovány pomocí anotací ve třídě modelu. Refenrenční příručka: http://docs.oracle.com/javaee/6/tutorial/doc/gircz.html.

nate Validation[2] omezení na prvky objektu, která se propagují do prezentační vrstvy. Na základě těchto omezení jsou generovány validační tagy, hvězdička u povinných polí, zvýraznění polí, atd.

Annotation	**Descriptor**	**Variable to use in templates**
@OdRequired	OdRequiredAnnotationDescriptor	required
@OdLength	OdLengthAnnotationDescriptor	minLength, maxLength
@OdLongRange	OdLongRangeAnnotationDescriptor	minimum, maximum
@OdDoubleRange	OdDoubleRangeAnnotationDescriptor	minimum, maximum
@OdRegex	OdRegexAnnotationDescriptor	pattern

Tabulka 5.1: JSF on-demand validation rozšíření pro Aspect Faces

Tyto anotace definují své proměnné, které lze použít v Aspect Faces šablonách pro definici validátorů. JSF značky odpovídají použitému typu validátoru. S knihovnou JSF on-demand validation je také možné použít všechny anotace, které již definují samotné Aspect Faces[3], jako jsou `@UiRequired`,`@UiPattern` a další.

V následujícím výpisu kódu je naznačena entitní třída Address s polem poštovního směrovacího čísla, které je povinné a musí být ve tvaru daném regulárním výrazem, který je naznačený v příslušné anotaci.

```
@Entity
public class Address extends EntityObject{

    @OdRequired
    @UiOrder(1)
    private String city;

    @UiOrder(2)
    @OdRequired
    @OdRegex(pattern="[1-9]{1}[0-9]{2}\\s?[0-9]{2}")
    private String zipCode;

    ...
}
```

Aspect Faces šablona pro vstupní textové pole, na které se bude mapovat pole poštovního směrovacího čísla z entity Address, bude vypadat podobně jako v následující ukázce.

[2] Hibernate implementace Bean Validation. Referenční příručka: http://docs.jboss.org/hibernate/validator/4.0.1/reference/en/html_single.

[3] Úplný seznam podporovaných anotací nalezenete zde: http://wiki.codingcrayons.com/display/af/Annotation+Descriptor+Reference+Table.

V průběhu generování fragmentů uživatelského prostředí se vyhodnotí proměnné `$required$` a `$pattern$`.Na základě obsahu těchto proměnných se k danému poli vloží validátory.

```
<util:inputText
      label="#{text['$entityBean.shortClassName().firstToLower()$.$field$']}"
      value="#{$entityBean.shortClassName()$.$field$}"
      size="$size$"
      printable="$print$"
      required="$required$"
      pattern="$pattern$"
      maxlength="$maxLength$"
      ...
      id="#{prefix}$field$"
      xmlns:f="http://java.sun.com/jsf/core" >

     <f:validateRequired disabled="#{not $required$}" />
     <f:validateRegex pattern="$pattern$"  disabled="#{empty '$pattern$'}"/>
  </util:inputText>
```

Podmíněné validátory jsou omezeny pouze na použití tagů `f:validateXxx`.Validační atributy UI komponenty nelze použít, protože validátor pro tento atribut je svázán přímo s danou komponentou. V různých JSF implementacích se mohou tyto validátory i atributy lišit. Příkladem takového atributu je atribut `required`.

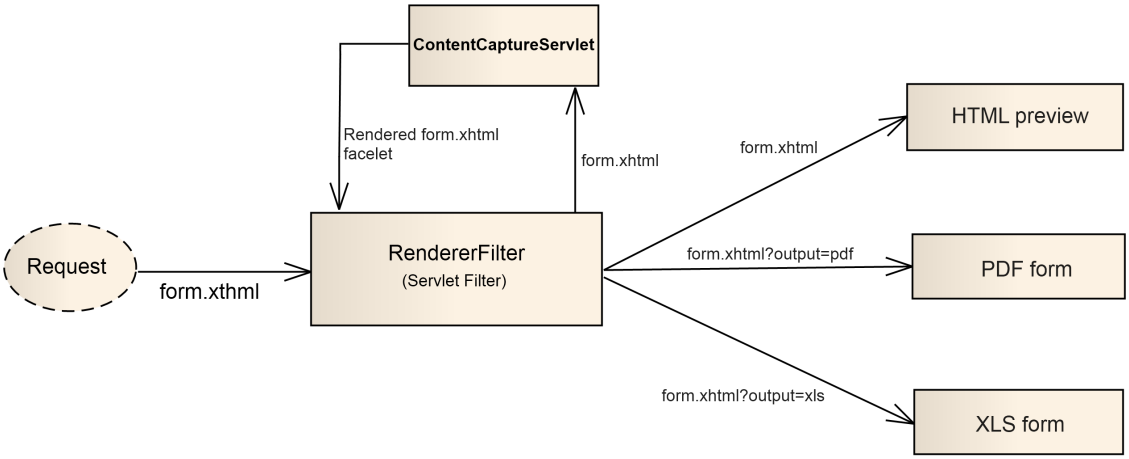
5.2 Export formulářů

Jedním z cílů této práce je návrh a implementace automatizované tvorby PDF a XLS formulářů z datového modelu a jeho metadat. K tomuto účelu lze s výhodou využít generovaných fragmentů formuláře, které tvoří HTML náhled na rozpracovaný formulář. S touto technikou lze definovat pouze jednu facelet šablonu pro HTML, PDF a XLS výstup.

```
<ui:define name="content" >
  <h:panelGroup layout="block" class="form-block">
    <h:outputText value="Adresa" class="block-caption" />
    <util:ui instance="#{fc.address}" edit="false" layout="printable.xhtml"/>
  </h:panelGroup>
  ...
</ui:define>
```

Výše je uveden příklad pro fragment s adresou. Označíme ho jako needitovatelný a přidáme `printable.xhtml` layout, který je optimalizovaný pro tisk. Formát, ve kterém se formulář zobrazí, je dán parametrem `output`. Před zpracováním HTTP požadavku je nutné tento požadavek zachytit a vyčíst požadovaný formát výstupního souboru. Pro tento

účel se na danou URL zaregistruje `RendererFilter`, který na základě `output` parametru dále přepošle požadavek ke zpracování. `RendererFilter` je třída, která implementuje `javax.servlet.Filter`[4] rozhraní.



Obrázek 5.2: Konceptuální model automatizovaného exportu formulářů.

`ContentCaptureServlet` slouží k vykreslení HTML formuláře z datového modelu, Aspect Faces šablon a layoutů. Třída `ContentCaptureServlet` je potomkem třídy `HttpServletResponseWrapper`[5], která implementuje rohraní `HttpServletResponse` a může sloužit jako základ pro úpravu odpovědi od Servletu. Ukázka formuláře v různých formátech je uvedena v příloze B.

5.2.1 PDF formulář

Prvním z požadovaných formátů pro export formulářů je PDF. PDF soubor musí splňovat pravidla daná technickou specifikací dokumentů, která je uvedena v tabulce 2.1 v kapitole 2.2.2, která se zabývá obecnými požadavky na aplikaci.

5.2.1.1 Jasper Reports

Protože kvalitní vestavěná podpora pro tisk na platformě Java chybí, vznikla sada knihoven a nástrojů pod názvem Jasper Reports[10]. Jasper Reports je velice živý open-source projekt, který je k dispozici pod licencí LGPL. V průběhu času se Jasper Reports stal velmi populární reportovací nástroj, který umožňuje vytvářet mnoho typů výsledných souborů, např. XML, HTML, PDF, XLS a další.

[4]Komponenta Servlet Filter je součástí Java Servlet specifikace od verze 2.3. Filtry dynamicky zachycují požadavky resp. odpovědi a mohou je transformovat do jiné formy. Více o této problematice naleznete zde: http://www.oracle.com/technetwork/java/filters-137243.html.

[5]Více viz Java EE 6 dokumentace, kterou naleznete zde: http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponseWrapper.html.

Jasper Reports zde uvádím, protože definice PDF dokumentu je dána XML souborem. První myšlenka byla napsat vlastní XML Aspect Faces šablony ve formátu, který využívá Jasper Reports (JR). Aspect Faces framework ve statickém módu[6] by tento soubor složil z fragmentů a následně by se zkompilovat celý report. Vytvoření reportu z datového modelu a jeho metadat by tedy bylo plně automatizované.

XML soubor, ve kterém je dokument nadefinován, má koncovku .jrxml a po jeho kompilaci se vytvoří jasper binární soubor. Formát souboru jrxml je natolik komplikovaný, že nelze rozložit na jednotlivé fragmenty pro automatické generování. Obsahuje své vnitřní proměnné a absolutní pozice prvků, které nelze spočítat, protože samotné fragmenty ještě nejsou k dispozici.

V JR šablonách lze používat všechny primitivní datové typy a základní objekty, např. String, Date a Object. Vnitřní pomocné třídy dělíme na tři základní skupiny field, variable a parameter.

- **Field** – Základní proměnná reportu. Field tvoří řádek reportu, odpovídá tedy jednomu objektu z datového zdroje aplikace.
- **Parameter** – Parameter je pomocná proměnná pro vkládání částí reportu, které by jinak byly duplikovány na více místech reportu.
- **Variable** – Jsou uživatelsky definované proměnné, které lze použít pro základní výpočty nad daty. Slouží pro tvorbu statistik a grafů v reportu.

O oblíbenosti Jasper Reports reportovacího systému svědčí také fakt, že pro něj byla zavedena přímá podpora ve Spring frameworku nebo plugin pro Java Server Faces. Šablonu reportu lze tvořit ručně, nebo pomocí vizuálního nástroje iReport, který je postavený na platformě NetBeans.

Jelikož nelze vytvořit šablony pro jednotlivé fragmenty jrxml dokumentu, tak jsem se rozhodl vytvořit vlastní Aspect Faces rozšíření pro tvorbu PDF formulářů, PDF Extension.

5.2.1.2 PDF Extension

PDF Extension rozšiřuje možnosti Aspect Faces frameworku o automatické generování PDF formulářů a reportů. Definice PDF stránky se provádí pomocí Aspect Faces layoutů (HTML tabulky) a kaskádových stylů. Není tedy potřeba, aby se vývojář učil novou technologii jako v případě Jasper Reports. Hlavní výhodou tohoto řešení je, že PDF Extension pracuje se stejnou facelet šablonou, která je již vytvořena pro HTML zobrazení. Navíc se přidávají pouze CSS pravidla pro optimalizaci stránky pro tisk, viz kapitola 5.2.1.4. Pomocí tohoto rozšíření vytvoříme z Aspect Faces poměrně silný nástroj pro tvorbu PDF reportů z datového modelu v době běhu aplikace.

[6] Aspect faces umožňuje běh i v tzv. statickém módu. V tomto módu dojde pouze ke složení dokumentu z fragmentů a k uložení tohoto dokumentu. Nedochází zde k interpretaci, např. k vykreslení JSF stránky. Více naleznete v Aspect Faces dokumentaci.
http://wiki.codingcrayons.com/display/af/Static+use+of+AspectFaces

5.2.1.3 Flying Saucer

Ve většině případů bude výstupem Aspect Faces soubor typu XHTML, ale lze definovat XML a jiné šablony. Hlavní myšlenka je využít stejnou facelet šablonu, která slouží pro zobrazení náhledu formuláře ve webovém prohlížeči, i pro tvorbu PDF souboru. Programátor pouze definuje layout a CSS třídy pro tisk a PDF bude automaticky vygenerováno bez dalšího úsilí. Jak optimalizovat stránku pro tisk pomocí CSS je uvedeno v následující kapitole 5.2.1.4.

Nyní je potřeba vyřešit převod XHTML do PDF souboru. Na tento problém je zaměřena knihovna Flying Saucer[28]. Knihovna umožňuje transformaci XML nebo XHTML souborů do PDF. Tuto knihovnu jsem zvolil, protože umí na zdrojový dokument aplikovat kaskádové styly, které nejsou přímo v dokumentu, ale jsou v samostatných souborech.

Flying Saucer má otevřený zdrojový kód[7], není tedy problém zjistit, že pro samotnou transformaci do PDF využívá velice populární knihovnu iText. Podpůrná knihovna iText je volně dostupná pod licencí AGPL[8]. Více o knihovně iText naleznete na stránkách projektu [27] . Převod XHTML stránky by byl možný i s použitím samotné iText knihovny, ale kód PDF generátoru by se stal velice složitý.

5.2.1.4 Optimalizace stránky pro tisk

Stejně jako při tvorbě běžných webových stránek je nutné optimalizovat dokument pro tisk a další různá zobrazení, tak to platí i pro transformaci facelet šablony do PDF. Nástroj Flying Saucer, který tuto transformaci provádí, podporuje standard CSS 2.1. Kaskádové styly ve verzi 2.1[22] přinášejí širokou podporu optimalizace dokumentů pro tisk.

CSS umožňují aplikovat různá pravidla na zobrazení stránky v závislosti na tom, z jakého média se na ni přistupuje. Flying Saucer načte XHTML dokument, aplikuje na něj CSS pravidla a pomocí iText knihovny vytvoří PDF soubor. Flying Saucer se tváří jako tiskové zařízení.

Prvním krokem optimalizace HTML dokumentu je nastavení velikosti tisknuté stránky. Z technických požadavků na dokument vyplývá formát A4, viz tabulka 2.1. Doporučená šířka obsahu stránky je 600px.

```
@page {   size: A4;   }
```

Tisk formuláře vychází ze stejné šablony jako náhled. V tuto chvíli je potřeba skrýt hlavičku, patičku a lištu s nástroji. Skrýt prvek při tisku lze pomocí vlastnosti `display`.

```
.no-print { display: none; }
```

Výsledný formulář se skládá z různě dlouhých částí. Často chceme, aby nějaká část nebyla rozdělena mezi dvě stránky. Pokud aplikujeme následující pravidlo na blok formuláře, tak v případě, že by se již celý nevešel na stránku, tak se vloží až na následující.

[7] Licence GNU Lesser GPL (http://www.gnu.org/licenses/lgpl.html).

[8] Affero General Public License je licence odvozená od GNU GPL. Celé znění licence naleznete zde: http://itextpdf.com/terms-of-use/agpl.php.

```
.form-block { page-break-inside: avoid; }
```

Dalším velkým požadavkem na dokument, který obsahuje tabulky s mnoha prvky, je správné stránkování, tj. pokud tabulka přesahuje na více stránek, aby na každé z nich bylo vykresleno záhlaví a zápatí, pokud jsou definovány. Tagy `THEAD` a `TFOOT` by měli toto podporovat[9], ale v některých prohlížečích a hlavně ve Flying Saucer nástroji to není implementováno[10]. Je tedy nutné na všechny tabulky layoutů aplikovat další CSS pravidlo.

```
table {  -fs-table-paginate: paginate; }
```

Záhlaví a zápatí se často definujeme nejen u tabulek, ale i u celých stránek dokumentu. Například chceme, aby na každé straně nahoře byl umístěn titulek a dole podpis a datum vytvoření dokumentu. V následující ukázce je element `#form-title` umístěn nahoře uprostřed na každé straně dokumentu. Více o této problematice naleznete zde [19].

```
@page{
   @top-center{
        content: element(form-title);
    };
}

#form-title{
    text-align: center;
    position: running(header-center);
}
```

Poslední požadavek kladený na výstupní dokument je vložení pole s číslem aktuální stránky. CSS toto opět umožňuje. V následující ukázce je vloženo číslování na každou stránku dokumentu do pravého dolního rohu.

```
@page{
    @bottom-right{  content: counter(page); };
}
```

Někdy je také dobré zobrazit celkový počet stránek, viz následující ukázka.

```
@page {
    @bottom { content: "Stránka " counter(page) " z " counter(pages);  };
}
```

5.2.1.5 Rozšíření pro Aspect Faces

Velkou výhodou frameworku Aspect Faces je jeho snadná rozšiřitelnost o vlastní anotace a šablony, viz dále. Tuto vlastnost plně využívá PDF Extension a obohacuje Aspect Faces o anotace pro řízení a formátování tisku formulářů do PDF souboru.

[9] HTML 4.0 referenční příručka: http://www.w3schools.com/tags/tag_thead.asp.

[10] Flying Sauces - uživatelská příručka, sekce o rozšíření CSS 2.1 specifikace: http://flyingsaucerproject.github.com/flyingsaucer/r8/guide/users-guide-R8.html#xil_43.

Annotation	**Descriptor**	**Variable to use in templates**
@UiPrint	UiPrintAnnotationDescriptor	print
@UiCaption	UiCaptionAnnotationDescriptor	caption
@UiBold	UiBoldAnnotationDescriptor	bold
@UiItalic	UiItalicAnnotationDescriptor	italic
@UiUnderline	UiUnderlineAnnotationDescriptor	underline

Tabulka 5.2: PDF rozšíření pro Aspect Faces

PDF Extension anotace definují nové proměnné, které lze použít přímo v Aspect Faces šablonách. Na základě hodnoty těchto proměnných, lze prvkům dynamicky přidávat nové CSS třídy, které ovlivňují jejich výsledný vzhled. V následujícím výpisu je uveden příklad použití anotace @UiPrint, která slouží k řízení tisku.

```
@Entity
public class Person  extends EntityObject{

    @UiPrint(disable=true)
    private String nickname;

    @OdRequired
    private String name;
    ...
}
```

PDF Extension je kompatibilní se všemi anotacemi a konfiguracemi Aspect Faces. Je tedy dodrženo stejné pořadí prků, stejné layouty a ve výchozím stavu budou tištěny stejné prvky, které se zobrazují v HTML. Pokud nějaký prvek chceme zobrazit pouze v HTML náhledu, ale ve výsledném PDF dokumentu již být nemá, tak k němu přiřadíme anotaci `@UiPrint(disable=true)`. V našem případě se vytiskne pouze jméno a přezdívka bude pro PDF soubor skryta. Z tabulky 5.2 je patrné, že anotace `@UiPrint` definuje proměnnou `print`. Tuto proměnnou stačí přidat do rodičovské šablony pro zobrazení.

```
<ui:composition  xmlns="http://www.w3.org/1999/xhtml"
                 xmlns:ui="http://java.sun.com/jsf/facelets"
                 xmlns:h="http://java.sun.com/jsf/html" />

     <div class="item #{print?' ': 'no-print'} >
              ...
     </div>
</ui:composition>
```

Pokud platí `print`== false , tak se k danému prvku přiřadí CSS třída no-print, ve které je definováno `display:`none pro media typu print. Na stejném principu jsou postaveny PDF

Extension anotace pro základní formátování výstupu. V současné době knihovna podporuje tři typy formátování.

- `@UiBold` – tučný text
- `@UiItalic` – kurzíva
- `@UiUnderline` – podtržený text

Aspect Faces framework je snadno rozšiřitelný o další anotace, lze tedy obohatit i PDF Extension o vlastní anotace. Prvním krokem je deklarace samotné uživatelské anotace. Celý postup uvedu na příkladu již známé anotace `@UiPrint`.

```
@Retention(RetentionPolicy.RUNTIME)
@Target(value = {ElementType.METHOD, ElementType.FIELD})
public @interface UiPrint {
    public boolean disable() default false;
}
```

Dále se musí vytvořit a zaregistrovat deskriptor[11], který namapuje proměnné z naší anotace na Aspect Faces proměnné, které jsou viditelné v šablonách, ze kterých se tvoří fragmenty uživatelského prostředí.

```
public class UiPrintAnnotationDescriptor implements AnnotationDescriptor{

    @Override
    public String getAnnotationName() {
         return "cz.cvut.pdfextension.annotation.UiPrint";
    }

    @Override
    public List<Variable> getVariables(AnnotationProvider ap) {
        List<Variable> variables = new ArrayList<Variable>();
        variables.add(new Variable("print", ap.getValue("disable")));
        return variables;
    }
}
```

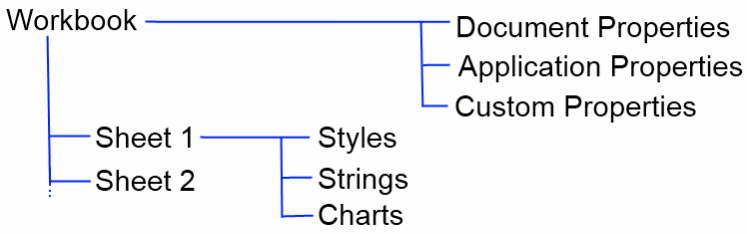
V této fázi se již můžeme odkazovat na proměnnou `print` z deskriptoru anotace `@UiPrint` v šablonách Aspect Faces. Pomocí takto definovaných proměnných lze upravovat šablony a tím přizpůsobit výsledné uživatelské rozhraní specifickým požadavkům.

[11] Annotation Descriptor se registruje v souboru aspecfaces-config.xml. Více naleznete v Aspect Faces dokumentaci: http://wiki.codingcrayons.com/display/af/AspectFaces.

5.2.2 XLS formulář

Druhý z požadovaných formátů formuláře je XLS. XLS je binární formát, který vytvořila společnost Microsoft pro svůj tabulkový editor Excel. XLS formát byl až do roku 2008 uzavřený[12]. Od roku 2007 začal Microsoft používat nový otevřený formát Open XML[35]. Jak již název napovídá, tento formát je založený na XML souborech.

Tabulkové soubory Excel, postavené na Open XML formátu, nesou koncovku XLSX. Open XML je tvořen adresářovou strukturou, která je zkomprimovaná do výsledného ZIP[13] archivu. Na obrázku 5.3 je znázorněna struktura XLSX souboru.



Obrázek 5.3: Struktura formátu XLSX.

Ze struktury je patrné, že obsahuje mnoho adresářů a XML souborů. Všechny části jsou provázané přes relace, které jsou umístěny v `*.rels` souborech. Rozložení tohoto formátu na jednotlivé fragmenty, které by sloužily jako Aspect Faces šablony, je nemožné. Hlavním problémem jsou vnitřní proměnné v `*.rels` souborech, které by se musely dodatečně vygenerovat. Dalším problémem je oddělení dat od samotné specifikace rozložení. V případě jednoduché XML struktury, by se mohl výsledný dokument sestavit z fragmentů generovaných frameworkem Aspect Faces ve statickém režimu[14].

Pro automatizovanou tvorbu XLS formulářů z datového modelu jsem se rozhodl navrhnout a implementovat vlastní rozšíření pro Aspect Faces – XLS Extension.

5.2.2.1 XLS Extension

XLS Extension rozšiřuje možnosti Aspect Faces frameworku o automatické generování základních XLS formulářů. Rozložení stránky generované pomocí Aspect Faces frameworku je tvořeno HTML tabulkou. Této znalosti lze využít pro konverzi formuláře do XLS souboru. Rozložení všech formulářů bude mít stále stejný charakter, mění se pouze parametr počtu sloupců daného layoutu. Dále víme, že jednotlivé položky formuláře jsou tvořeny dvojicí jméno a hodnota.

[12] Všechny specifikace formátů naleznete zde:
http://www.microsoft.com/openspecifications/en/us/programs/osp/office-file-formats/default.aspx.

[13] Souborový formát pro komprimaci a archivaci dat. Na platformě Java je pro práci s ZIP archivem určen balíček java.util.zip.

[14] Aspect Faces ve statickém módu pouze sestaví dokument z jednotlivých fragmentů a uloží ho (nedochází k jeho interpretaci).

Jádro této knihovny tvoří konvertor, který na základě výstupu z uživatelského parseru sestavuje výsledný XLS dokument. Parser nazývám uživatelský, protože není a ani nemůže být součástí XLS Extension knihovny. Tento parser se navrhuje ke každé aplikaci tak, aby byl přizpůsoben zdrojové facelet stránce. Tuto stránku nelze zobecnit, protože by to byl příliš velký zásah do individuality aplikace. Hlavní výhodou tohoto řešení je, že XLS Extension pracuje se stejnou facelet šablonou, která je již vytvořena pro HTML zobrazení. Navíc se přidávají pouze prázdné CSS třídy pro řízení průchodu jednotlivými layouty.

5.2.2.2 JSoup parser

Uživatelský parser prochází Aspect Faces layouty a voláním metod XLS Extension konvertoru vytváří výsledný XLS soubor. Implementace uživatelského parseru XHTML souborů musí být relativně snadná a rychlá. V této práci jsem se rozhodl parser implementovat pomocí knihovny JSoup.

Knihovna JSoup[33] vznikla jako open-source projekt a je distribuována pod MIT[15] licencí. JSoup poskytuje komplexní API pro průchod a tvorbu XHTML souborů. Knihovna umožňuje extrakci dat s využitím DOM a hlavně CSS selektorů.

Struktura formuláře generovaná Aspect Faces je stále stejná. Mění se pouze parametr počtu sloupců v layoutu. Každá část formuláře je identifikovatelná pomocí CSS třídy, každá položka má také svůj identifikátor. Uživatelský parser prochází přes tyto identifikátory a volá metody konvertoru pro přidání titulku, řádku, atd. Pravidla pro výběr prvku jsou triviální, pouze jméno příslušné CSS třídy. Jaké třídy se mají v daném místě použít, definují Aspect Faces šablony resp. layouty a facelet stránka. Aby parser nemusel procházet stránku celou, ale pouze bloky, kde se nacházejí relevantní data, tak je dobré zanést pravidla pro tyto fragmenty do logiky parseru.

5.2.2.3 JExcelAPI

Jádro XLS Extension tvoří konvertor, který vytváří výsledný XLS formulář. Rozhraní pro tvorbu Excel souborů jsem záměrně volil co nejjednodušší tak, aby byl snadno použitelný v uživatelském parseru. Rozhraní tvoří pouze inicializační metody a metody pro přidání řádku a titulku. S touto sadou funkcí lze vytvářet jednoduché formuláře s podporou různých layoutů.

```
public interface XLSConvertor {
     public void init();
     public void create();
     public void addRow(List<Entry> enteries, Coordinate co);
     public void addTitleRow(String title, Coordinate co);
}
```

Konvertor jako vstupní parametry vždy přebírá souřadnice zapisované buňky a záznam resp. popisek. Záznam je tvořen dvojicí jméno a hodnota.

[15] Svobodná licence pro software, která vznikla na Massachusetts Institute of Technology. Software označený touto licencí lze použít i v proprietárním softwaru, ale pouze pokud tam bude tato licence uvedena.

Protože formát XLS nelze snadno modifikovat, tak jsem k implementaci rozhraní konvertoru použil známou otevřenou knihovnu pro práci s XLS soubory – JExceAPI (JXL). Knihovna JXL[34] pracuje s binárním formátem XLS, ale díky oddělenému návrhu lze později přidat i jiné implementace pro podporu novějšího Open XML formátu. Tato knihovna je uvolněna pod licencí GPL resp. LGPL. Knihovna podporuje základní možnosti formátování, jako jsou formáty čísel a data. Dále podporuje práci s fonty, barvou a ohraničením buněk. S touto sadou funkcí je již možné vytvářet základní formuláře, které přesně splní požadavky této aplikace.

5.2.2.4 Rozšíření pro Aspect Faces

Podobně jako PDF Extension, tak i XLS Extension přináší do Aspekt Faces další anotace. Pomocí těchto anotací lze ovlivňovat transformaci Aspect Faces výstupu do XLS souboru. Všechny anotace, které knihovna XLS Extension přinášejí, jsou shrnuty v následující tabulce.

Annotation	**Descriptor**	**Variable to use in templates**
@XLSTransient	XLSTransientAnnotationDescriptor	transient
@UiBold	UiBoldAnnotationDescriptor	bold
@UiItalic	UiItalicAnnotationDescriptor	italic
@UiUnderline	UiUnderlineAnnotationDescriptor	underline

Tabulka 5.3: XLS rozšíření pro Aspect Faces

V současné době je implementována podpora pouze pro základní formátování. Formát buněk výsledného dokumentu je dán vlastnostmi v `xls-creator.properties` souboru.

- `@UiBold` – tučný text
- `@UiItalic` – kurzíva
- `@UiUnderline` – podtržený text

XLS Extension lze podobně jako PDF Extension rozšířit o vlastní uživatelské anotace jak pro formátování, tak i pro řízení vlastní transformace. Pro každou anotaci musíme implementovat a zaregistrovat deskriptor, který namapuje hodnoty z anotace na proměnné viditelné z Aspect Faces šablon. Postup, jak vytvořit vlastní anotaci, je uveden v části věnované knihovně PDF Extension 5.2.1.5.

XLS Extension anotace definují své proměnné, které lze použít přímo v Aspect Faces šablonách. Na základě hodnoty těchto proměnných, lze prvkům dynamicky přidávat nové prázdné CSS třídy, podle kterých lze řídit uživatelský parser a XLS konvertor. Abychom zamezili vedlejším efektům v HTML prezentaci, tak tyto CSS třídy musí mít prázdné definice.

V následující části uvedu příklad použití anotace `@XLSTransient`, která slouží pro skrytí prvku třídy v průběhu XLS transformace. Prvek se tedy zobrazuje pouze v HTML a PDF formátech formuláře. Anotace `@XLSTransient` podle tabulky 5.3 definuje proměnou transient,

tuto proměnnou použijeme v podmíněném výrazu v rodičovské Aspect Faces šabloně pro zobrazení.

```
<ui:composition  xmlns="http://www.w3.org/1999/xhtml"
                 xmlns:ui="http://java.sun.com/jsf/facelets"
                 xmlns:h="http://java.sun.com/jsf/html" />

     <div class="item #{transient?'XLS-param-transient':''}>
     ...
     </div>
</ui:composition>
```

Pokud platí `transient`== true , tak se k danému prvku přiřadí CSS třída `XLS-param-transient`, která nemá definované žádné vlastnosti. Jméno třídy je voleno tak, aby nedošlo k záměně s nějakou jinou třídou definovanou pro prezentaci obsahu. Pro řízení toku parseru používá XLS Extension CSS třídy, protože to je jediný způsob, jak k XHTML elementu přiřadit dodatečnou informaci, a aby zároveň dokument zůstal validní.

Do uživatelského parseru se přidá pouze podmínka, pokud je prvek ze třídy XLS-param-transient, tak ho přeskoč. Z následnující ukázky je patrné, že selektory jsou velmi jednoduché. Složité selektory by měly nepříznivý vliv na výkon.

```
private void processRow(Element tr) {
  if(!tds.select(".XLS-param-transient").isEmpty()){
     return;
  }
  ...
}
```

Kapitola 6

Testování

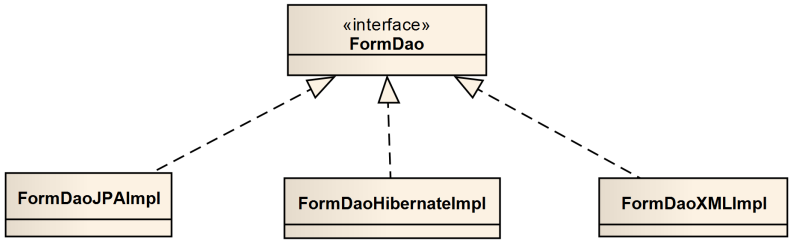
Testování je nedílnou součástí procesu vývoje softwaru. Jednotkové testy by měly být spouštěny při každém sestavení, integrační testy pak v nějakou stanovenou dobu, například při nočním sestavení projektu.

6.1 Testování jednotek

Testování jednotek[13] slouží k testování nějaké malé části kódu aplikace. Testujeme část, která je samostatně testovatelná, pro objektově orientovaný jazyk to je třída. Cílem je izolování části programu a testování dané metody v izolaci. Architektura xUnit pro jednotkové testování je dostupná pro mnoho programovacích jazyků. Na platformě Java to jsou JUnit[1] a TestNG[2]. Pro vytvoření integračních testů jsem zvolil první jmenovaný, tedy JUnit.

6.2 Integrační testy

Integrační testy[13] slouží k ověření provázanosti systémů, popř. komponent v rámci jednoho systému. V aplikaci eForm vytvářím integrační testy na vrstvě Data Access Object (DAO).



Obrázek 6.1: Data Access Object

[1] JUnit je součástí xUnit architektury pro psaní jednotkových testů. Stránka projektu: http://junit.sourceforge.net.

[2] Testovací framework inspirovaný JUnit. Více o projektu naleznete zde http://testng.org/doc/index.html.

Návrhový vzor DAO[23] patří k základním návrhovým vzorům platformy J2EE. DAO vrstva slouží k oddělení perzistentní vrstvy od zbytku aplikace. Hlavní DAO rozhraní jsem implementoval jako generické. Díky použití této techniky zde nehochází k opakování stejného kódu pro každou entitu.

```
public  interface GenericDao <T, PK extends Serializable> {
    void create(T entity);
    T read(PK id);
    T update(T entity);
    void delete(final Object id);
}
```

Implementace DAO je tvořena pomocí EJB a JPA. EJB objekty jsou spravovány kontejnerem. Při testování není možno jednoduše vytvořit jejich instanci, protože ta je dodávána pomocí kontejneru. První možností jak testovat EJB aplikaci je tvorba Mock[13] objektů. Mock objekt je zástupný objekt, který simuluje chování skutečného objektu aplikace. Slouží výhradně k provádění testů. K vytváření mock objektů v EJB aplikaci můžeme využí např. MockEJB[3] framework.

Integrační testy v EJB aplikaci vyžadují běh v EJB kontejneru. V testovací třídě je nejdříve nutné nastavit a spustit Embedded Container. V následující ukázce je použit Glassfish 3.1 Embedded[24].

```
@BeforeClass
public static void initContainer() throws Exception {
  Map<String, Object> props=new HashMap<String, Object>();
  props.put(EJBContainer.MODULES, new File("target/embedded-classes"));
  props.put("org.glassfish.ejb.embedded.glassfish.instance.root",
            "./src/test/testing-domain");
  props.put("org.glassfish.ejb.embedded.glassfish.web.http.port","");
  ec = EJBContainer.createEJBContainer(props);
  ctx = ec.getContext();
}
```

Abychom získali instanci DAO třídy, musíme volat `lookup` v `@Test` metodě. V další části je uveden způsob provádění integračních testů bez nutnosti správy kontejneru z testovacích tříd a zároveň s možností využití všech služeb kontejneru.

6.2.1 Arquillian

Arquillian[21] je testovací platforma pro zjednodušení psaní integračních testů na platformě J2EE. Přináší testy přímo do běhového prostředí aplikace, nemusíme tak spravovat kontejner z testovacích tříd. Jednou z výhod použití Arquillianu je přenositelnost testů mezi různými kontejnery, jako jsou JBoss AS, GlassFish, OpenEJB a další.

[3] MockEJB podporují stateless i statefull session bean, local i remote rozhraní, EJBContext, atd. Více naleznete na stránkách projektu: http://mockejb.sourceforge.net.

Arqullian testy je možné spouštět ve dvou profilech, remote a managed. V remote módu se Arquillian připojí na již běžící server a provede nasazení testovacího archivu, který se definuje v `@Deployment` metodě. Díky nástroji ShrinkWrap[4] provádíme tzv. "micro deployment", nedochází tedy k nasazení celé aplikace, ale pouze části, která je nutná pro provedení daného testu. Touto technikou získáváme kontrolu nad tím, co testujeme a jaké zdroje budou v době provádění testu k dispozici.

```
@Deployment
public static JavaArchive createDeployment() {
  return ShrinkWrap
        .create(JavaArchive.class, "form.jar")
        .addPackages(true, Form.class.getPackage())
        .addPackages(true, FormDao.class.getPackage())
        .addAsManifestResource("META-INF/persistence.xml", "persistence.xml")
        .addAsManifestResource(EmptyAsset.INSTANCE, "beans.xml");
}
```

V managed módu si Arquillian musí nejdříve spustit kontejner a po dokončení testu ho zase ukončit. Výhodou managed profilu je, že se při běhu automatizovaných testů nemusíme spoléhat na to, že je vzdálený server dostupný. Arquillian podporuje oba frameworky pro jednotkové testování, JUnit i TestNG. V této práci jsem použil remote profil aplikačního serveru Glassfish 3.1 a JUnit framework.

Testy spouštěné přímo v kontejneru přinášejí možnost využít Context and Dependency Injection (CDI)[5] a další služby kontejneru přímo v testovacích třídách. Můžeme tedy snadno přistupovat k EJB objektům, načítat perzistentní jednotku a spravovat připojení k databázi. Tento přístup nám umožní definovat alternativní perzistentní jednotku pro testovací databázi, která se vytvoří před každým spuštěním testů a bude tak obsahovat vždy stejná data. Testy nezasahují do produkční databáze, kde se data mohou často měnit a opakované spuštění testů nad produkční databází tak nemusí vrátit vždy stejný výsledek.

```
@Test
public void callServiceToCreateNewForm() {
   Form form = new Form();
   service.create(form);
   Assert.assertNotNull("Id should not be null!", form.getId());
}
```

Po vytvoření `@Deployment` metody již následují `@Test` metody, jako při běžném testování bez kontejneru. Testovací metody pokrývají všecha DAO rozhraní aplikace. Výše je uveden test pro přidání nového formuláře do databáze. Po vložení objektu do databáze se mu automaticky přiřadí unikátní identifikátor ID. Pokud je ID `null`,tak test selže.

[4] JBoss ShrinkWrap nabízí API pro sestavení JAR, WAR a EAR archivů. Více nalezenete zde: http://www.jboss.org/shrinkwrap.

[5] Context and Dependency Injection je součást specififace Java EE 6. Přináší typově bezpečnou injektáž komponent do aplikace. Referenční příručku naleznete zde http://docs.oracle.com/javaee/6/tutorial/doc/gjbnr.html.

Kapitola 7

Srovnání s běžným přístupem k tvorbě UI

V této části práce srovnávám běžný přístup k vývoji uživatelského prostředí s vývojem s použitím technologie pro generování fragmentů UI z datového modelu a jeho metadat. Srovnání je provedeno na dvou úrovních. V první části je uvedeno měření úspory řádků zdrojového kódu oproti manuální tvorbě UI. Ve druhé části pak měření a porovnání výkonu aplikace postavené na AF a běžné JSF aplikace.

7.1 Porovnání tvorby uživatelského prostředí

Cílem této části práce je dokázat, že pomocí technologie generování fragmentů uživatelského rozhraní z datového modelu, lze vytvořit plnohodnotnou JSF aplikaci a zároveň ušetřit náklady na vývoj a následnou údržbu UI vrstvy. V dokumentaci Aspect Faces frameworku se uvádí, že lze docílit úspory až 25% kódu[1] při tvorbě UI a ještě více u následné údržby aplikace. Toto tvrzení ověřuji vlastním porovnáním s klasickou JSF aplikací.

Pro porovnání čisté Java Server Faces a AF aplikace používám následující metriky. U obou beru v potaz pouze view a datovou vrstvu aplikace, protože business vrstva je v obou případech totožná.

- **SLOC** - Počet řádků kódu.
- **MoF** - Počet souborů, které se musejí modifikovat.

SLOC je softwarová metrika pro měření velikosti programu. SLOC udává počet řádků kódu. Rozlišuje fyzické a logické řádky. U datového modelu uvažuji pouze logické řádky kódu, které nejsou závislé na formátování zdrojového kódu. Metrika SLOC, která uvažuje logické řádky, se někdy označuje jako LLOC. LLOC počítá pouze proveditelné řádky, nezapočítá příkaz rozepsaný na dva řádky dvakrát, ale pouze jednou. Při porovnání XHTML souborů je korektnější použít řádky fyzické, protože mohou obsahovat dlouhé mnohořádkové tagy.

[1] http://www.aspectfaces.com/

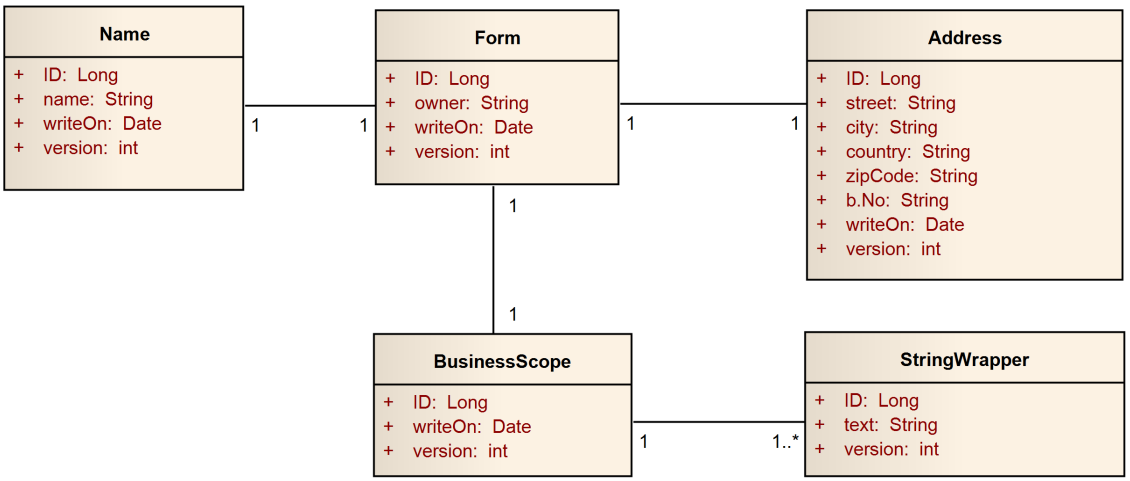
MoF udává počet souborů, do kterých je nutné zasahovat. Protože u AF aplikace může datový model obsahovat dodatečné informace o UI, tak je nutné tuto vrstvu zahrnout do srovnání. Jedná se o tzv. UI anotace, např. `@UiOrder` pro upravení pořadí jednotlivých prvků ve formuláři. Pro měření těchto metrik využívám open-source nástroj UCC, kterému je věnovaná následující podkapitola.

7.1.1 UCC

Unified Code Count (UCC)[8] je komplexní nástroj pro měření různých metrik nad zdrojovým kódem. Umožňuje vytváření statistik o počtu řádků zdrojových kódů, komentářů a prázdných řádků v celém projektu nebo pouze v konkrétní složce či souboru. Déle lze měřit rozdíly mezi dvěma verzemi projektu, což využívám při porovnání následné údržby JSF a AF aplikace. UCC měří logické i fyzické řádky v mnoha jazycích, jako např. jsou Java, C#, C++, ASP.NET, Perl, HTML, XML a mnoho dalších. UCC je open-source projekt, který je distribuován pod GNU GPL licencí a lze zkompilovat libovolným ANSI C++ kompilátorem.

7.1.2 Testovací Aplikace

Testovací aplikace je podmnožinou formuláře pro podání do obchodního rejstříku. V testu budeme uvažovat dva případy – editační formulář a formulář určený pouze pro čtení. Aplikace byla psána dvakrát, jednou manuálně v Java Server Faces a druhá s technologií Aspect Faces. Podobně jako aplikace eForm, i tato testovací aplikace využívá komponent PrimeFaces resp. šablony pro PrimeFaces. Obě aplikace mají shodný datový model, který je znázorněn jako diagram tříd na následujícím obrázku.



Obrázek 7.1: Testovací aplikace - diagram tříd.

Na diagramu 7.1 je zajímavá jen entita `StringWrapper`, která pouze zapouzdřuje datový typ `String`. V běžné aplikaci by byla vyjádřena jako další atribut (typu `List`) třídy `BusinessScope`. `StringWrapper` je zde z důvodu, že AF využívá pro tvorbu uživatelského

rozhraní reflexi[2], která na všech objektech čte getter a setter metody. V současné verzi AF (1.2.1) nelze techniku inspekce zdrojového kódu na kolekci základních datových typů, protože vyčítá všechny jejich atributy, pro které existují getter a setter metody, tedy ne pouze pro `value`. Podporu kolekcí základních datových listů jsem již podal jako návrh na vylepšení AF frameworku. Hledání možných vylepšení frameworku je také jedním z cílů této práce.

Nedílnou součástí každé AF aplikace je sada šablon pro vizuální prvky, ze kterých se skládají fragmenty UI. Pro vytvoření těchto šablon bylo nutné napsat 727 PLOC[3]. V průběhu vývoje eForm aplikace došlo k přidání resp. modifikaci cca 25 PLOC. Většina těchto změn souvisí s přidáním podpory podmíněných validátorů a podpory exportu do PDF a XLS souborů. Dále využívám pět různých layoutů, což je dalších 63 PLOC. Poslední částí je nutná konfigurace frameworku, která dohromady činí 296 PLOC. 296 řádků celkové konfigurace je poměrně vysoké číslo, ze kterého plyne, že použití AF frameworku je vhodné pouze u rozsáhlých projektů s velkým počtem formulářových polí, jako je např. právě aplikace eForm.

Jelikož základní sada šablon je již přímou součástí AF frameworku, tak nejsou zahrnuty v tomto srovnání. Měření bude probíhat pro editační formulář i pro formulář určený pouze pro čtení. V první fázi se zaměřím na tvorbu aplikace, v druhé pak na údržbu. Budu demonstrovat přidání nového atributu entitní třídě a změnu některého stávajícího.

7.1.3 Vyhodnocení

V testovací aplikaci je zahrnuto pět entitních tříd, které dohromady činí 93 LLOC. Dalších 12 LLOC k datovému modelu přidávají informace o rozložení prvků v uživatelském rozhraní. 17 PLOC tvoří layouty pro AF a šablony ve srovnání zahrnuty nejsou, protože změny v nich se týkají pouze eForm aplikace, nikoli této testovací podmnožiny. Souhrn výsledků měření, pro editační formulář i pro formulář, je uveden v následujících tabulkách 7.1 a 7.2.

	manuálně (JSF)	**Aspect Faces**	**Δ LOC / Δ %**
XHTML	85 PLOC	33 PLOC	-52 PLOC / -61%
Java (datový model)	93 LLOC	105 LLOC	+12 LLOC / +12%
Celkem	178 LOC	138 LOC	-40 LOC / **-22%**

Tabulka 7.1: Srovnání LOC pro editační formulář.

Z měření vyplývá, že aplikace splňuje předpoklad úspory kódu při tvorbě uživatelského rozhraní až 25%. V prvním případě bylo provedeno měření pro editační formulář, kde se povedlo dosáhnout úspory 22% LOC oproti běžné JSF aplikace, viz tabulka 7.1. V druhém případě se jednalo o formulář určený pouze ke čtení, kde bylo dosaženo úspory 18% LOC, viz tabulka 7.2. Zajímavý je také fakt, že bylo dosaženo větší úspory pro editační formulář. To je dáno přítomností validátorů a chybových zpráv pro ošetření vstupu v manuálně psaném

[2] Java Reflection API - umožňuje v době běhu získávat informace o třídách. Za běhu lze třídy také vytvářet a upravovat. Více o Java Reflection API naleznete zde:
http://docs.oracle.com/javase/tutorial/reflect/index.html.

[3] Physical Lines of Code

	manuálně (JSF)	**Aspect Faces**	**Δ LOC / Δ %**
XHTML	76 PLOC	33 PLOC	-43 PLOC / -56%
Java (datový model)	93 LLOC	105 LLOC	+12 LLOC / +12%
Celkem	169 LOC	138 LOC	-31 LOC / **-18%**

Tabulka 7.2: Srovnání LOC pro formulář určený pouze ke čtení.

projektu. Aspect Faces framework toto ošetření vstupů plně spravuje sám. Pokud by bylo omezení na více než třech polích, tak by byl rozdíl ještě patrnější.

7.1.4 Údržba aplikace

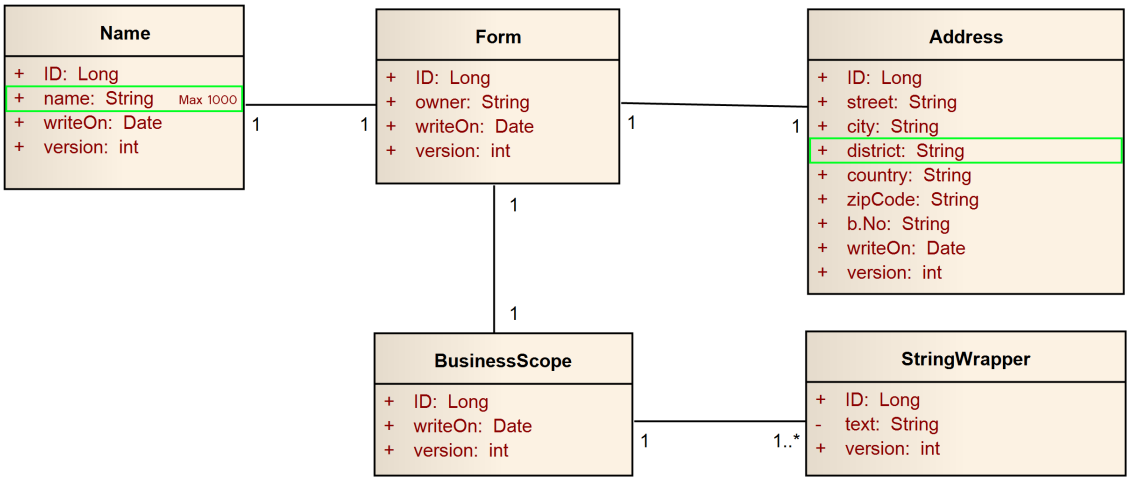
V této části provedeme srovnání v následné údržbě aplikace. Nejprve provedeme přidání atributu k entitní třídě. Ke třídě `Address` přidáme informaci o městské části – atribut `district` typu `String`. Dále ve třídě `Name` zavedeme podmínku, že atribut `name` bude obsahovat dlouhý text – v uživatelském prostředí bude reprezentován pomocí tagu

```
<p:inputTextarea> </p:inputTextarea>.
```

místo stávajícího tagu pro krátký text

```
<p:inputText> </p:inputText>.
```

Změny v modelu jsou zobrazeny v diagramu tříd vyobrazeném na následujícím obrázku.



Obrázek 7.2: Testovací aplikace - upravený diagram tříd.

Předpokládaným výsledkem je, že se uspoří více než 25% kódu, který bude nutno přidat resp. pozměnit při údržbě uživatelského rozhraní. Tento předpoklad vychází z dokumentace[4]

[4] http://www.aspectfaces.com/

k Aspect Faces frameworku. V následující podkapitole je shrnuto kolik řádků a souborů se muselo vytvořit resp. modifikovat při přidání nového atributu. V tomto shrnutí je započítáno zobrazení v editačním formuláři i ve formuláři určeném pouze pro čtení.

7.1.4.1 Vyhodnocení

Tabulka 7.3 přináší souhrnný pohled na výsledky měření pro přidání jednoho atributu do datového modelu.

	manuálně (JSF)	**Aspect Faces**	**Δ LOC**
XHTML	15 PLOC	0 PLOC	-8 PLOC
Java (datový model)	4 LLOC	5 LLOC	+1 LLOC
Celkem	19 LOC	5 LOC	-7 LOC (**-73%**)
MoF	3	1	-2

Tabulka 7.3: Srovnání SLOC a MoF po přidání jednoho atributu.

Pro přidání nového atributu byl modifikován jeden soubor v modelu a z 93 PLOC narostl na 98 PLOC pro AF a o jeden méně pro manuální JSF přístup. Jeden řádek navíc u AF je dán potřebou přidat informaci o pořadí, kde se atribut zobrazí. Explicitně specifikovat pořadí není nutné a vynecháním tohoto kroku by byli obě úpravy datového modelu totožné.

U prezentační vrstvy je situace již odlišná. V případě použití technologie postavené na inspekci zdrojového kódu se nemusí zasahovat do XHTML souboru editačního formuláře ani do předpisu formuláře určeného pouze pro čtení. Nový atribut je inspekcí zjištěn a vykreslen zcela automaticky. V případě klasického (manuálního) přístupu k tvorbě UI je nutné modifikovat celkem 3 soubory s celkovou změnou 19 LOC. U AF přístupu bylo nutné modifikovat 1 soubor s celkovou změnou 5 LOC. Celková úspora úsilí tak činí 73% .

Tabulka 7.4 již ukazuje souhrn výsledku měření v případě modifikace existujícího atributu, což je také velice častý úkon při údržbě aplikace. U atributu `name` zavedeme nový předpoklad, že bude obsahovat dlouhý řetězec o maximální délce 1000 znaků. V uživatelském prostředí se tato změna projeví pouze u editačního formuláře, kde klasické pole nahradí tag `inputTextarea`. Formulář určený pouze pro čtení zůstane nezměněn.

	manuálně (JSF)	**Aspect Faces**	**Δ LOC**
XHTML	3 PLOC	0 PLOC	-8 PLOC
Java (datový model)	1 LLOC	1 LLOC	0 LLOC
Celkem	4 LOC	1 LOC	-3 LOC (**-75%**)
MoF	2	1	-1

Tabulka 7.4: Srovnání SLOC a MoF po změně jednoho atributu.

Z tabulky 7.4 je patrné, že v případě použití AF je opět nutné modifikovat pouze jeden soubor, a to entitní třídu z datového modelu, kde se provede změna omezení kladená na

atribut. V konfiguraci AF je již nastaveno odlišné mapování pro krátké a dlouhé texty, takže není nutná žádná další úprava a AF vykreslí velké textové pole pro pozměněný atribut. V případě klasického přístupu je nutné jít i do XHTML předpisu formuláře a tam ručně přepsat tag a upravit validační zprávu pro případ zadání chybné hodnoty uživatelem.

7.2 Měření výkonu

V této části se zaměřuji na porovnání času odezvy standardní JSF aplikace a aplikace s technologií automatického generování fragmentů uživatelského prostředí z datového modelu. Srovnání provádím pro stejnou aplikaci jako v předchozí části, kde bylo provedeno srovnání vývoje a údržby obou aplikací.

7.2.1 Příprava testu

Měření probíhalo na dvou úrovních. První pro bohaté prostředí, které je tvořeno editačním formulářem a druhé pro jednoduchý výpis formulářových dat uživateli. Cílem tohoto testu je ověření předpokladu, že Aspect Faces aplikace bude mít horší odezvu maximálně o jednotky procent v porovnání s klasickou Java Server Faces aplikací, která byla napsána manuálně. Tento předpoklad vychází z měření autorů Aspect Faces frameworku[4].

Testování probíhalo na aplikační serveru Glassfish 3.1.2[5] na testovacím zařízení Intel Core 2 Duo 1.5 GHz, 4GB RAM a operačním systému Windows 7 Professional (64bit). Měření bylo provedeno nástrojem JMeter, kterému je věnována následující podkapitola.

7.2.1.1 JMeter

JMeter[29] je open-source nástroj pro vytváření zátěžových testů webových aplikací, databází (přes JDBC), webových služeb a emailových serverů přes protokoly IMAP, POP3 a SMTP. JMeter je napsán čistě v jazyce Java a jeho součástí je i grafické uživatelské prostředí.

V této práci využívám JMeter pro měření odezvy testovací aplikace pro porovnání Aspect Faces a čisté Java Server Faces aplikace. V JMeter aplikaci lze nastavit testovací plán, kde volím počet vláken (uživatelů), kteří se budou připojovat k dané aplikaci. Dále volím, s jakou periodou se budou připojovat a kolikrát se celkem každé vlákno připojí. Pro mé měření jsem zvolil jedno vlákno, které provede 250 GET požadavků s periodou 1s.

JMeter umožňuje nastavit libovolný HTTP požadavek a případně jeho parametry. Díky tomu lze testovat odesílání vyplněného formuláře pomocí metody POST. Výsledky provedených testů odchytávají Listener komponenty a zobrazují výsledky naměřených dat v různých volitelných podobách. Jsou zde Listenery pro jednoduché textové a tabulkové výstupy, ale i sofistikované grafické náhledy.

[5] Referenční open-source aplikační server, který plně implementuje Java EE 6 specifikaci. http://glassfish.java.net/

7.2.2 Vyhodnocení

Ve srovnání figuruje manuální JSF aplikace, které byla, pro účely tohoto testu, vytvořena pomocí statického generování Aspect Faces frameworku. V testu se tak srovnávají opravdu totožné aplikace, které mají stejné tagy. V testu se provádí 250 požadavků na formulář s periodou 1s. Výsledkem provedených testů jsou průměrná doba odezvy a směrodatná odchylka. Směrodatná odchylka říká, jak moc se liší jednotlivé naměřené odezvy od průměru. Čím menší směrodatná odchylka je, tím jsou si naměřené hodnoty odezev podobnější. Ve statistice je směrodatná odchylka nepoužívanější mírou variability.

Jelikož Aspect Faces používá cache pro jednotlivé grafické widgety, tak na první dotaz se musí provést inspekce zdrojového kódu a na základě této inspekce daný fragment vygenerovat. Pro další dotazy se již tato inspekce neprovádí a grafické prvky se čtou z AF cache paměti. První dotaz má tedy delší čas odezvy než ostatní následující.

	Průměrný čas odezvy [ms]	**Směrodatná odchylka**	**Δ [%]**
EDITAČNÍ FORMULÁŘ			
manuálně (JSF)	545	47	-2
Aspect Faces	539	41	
FORMULÁŘ PRO ČTENÍ			
manuálně (JSF)	445	38	4
Aspect Faces	467	11	

Tabulka 7.5: Srovnání pro oddělené tagy a widgety.

Pro bohaté uživatelské prostředí (editační formulář) se povedlo dosáhnout zlepšení odezvy oproti klasické JSF aplikaci o 2%. Na první pohled se toto zrychlení zdá jako velice překvapující, ale dáno tím, že AF používá cache, kde jsou tagy uloženy a čtou se z `InputStream` objektu a nemusí se číst XHTML soubory jako v případě běžné JSF aplikace (zobrazovací technologie facelet).

V případě formuláře určeného pouze pro čtení jsou odezvy také srovnatelné. Aplikaci s použitím techniky automatického generování fragmentů UI byla naměřena odezva pouze o 4% pomalejší. Formulář pro čtení není tvořen pomocí složitých komponent, jedná se pouze o textový výstup, čemuž odpovídá kratší průměrný čas odezvy cca. o 100ms pro Aspect Faces i JSF aplikaci.

V rámci prováděných testů jsem vytvořil vlastní AF šablony, které slučují tagy a widgety do jednoho souboru. Odpadá tím dvě fáze, kde se tag dekoruje dalšími XHTML soubory. Sloučením tagů a widgetů lze dosáhnout výrazného zlepšení časů, viz tabulka 7.6. Toto sloučení jsem podal jako další návrh na vylepšení AF frameworku.

	Průměrný čas odezvy [ms]	**Směrodatná odchylka**	**Δ [%]**
EDITAČNÍ FORMULÁŘ			
manuálně (JSF)	370	53	-5
Aspect Faces	355	49	
FORMULÁŘ PRO ČTENÍ			
manuálně (JSF)	330	57	2
Aspect Faces	337	61	

Tabulka 7.6: Srovnání pro sjednocené tagy a widgety.

Zvýše uvedeného vyplývá, že se povedlo ověřit předpoklad, že odezva AF aplikace se od běžné JSF aplikace bude lišit v jednotkách procent. V případě bohatého uživatelského prostředí se dokonce podařilo dosáhnout rychlejšího průměrného času odezvy než u běžného přístupu. Dále z měření vyplývá, že čím méně budou komplikované šablony pro jednotlivé grafické elementy, tím bude doba odezvy aplikace kratší.

Kapitola 8

Závěr

V rámci této diplomové práce jsem navrhl a implementoval serverovou aplikace eForm, která slouží k vytváření formulářů, které jsou vyžadovány při zakládání nové společnosti pro podání do obchodního rejstříku. V rámci projektu byla nasazena technologie automatického generování fragmentů uživatelského prostředí z datového modelu a jeho metadat.

Framework Aspect Faces umožňuje automaticky generovat fragmenty UI na základě inspekce zdrojového kódu aplikace. Díky inspekci dochází k redukci duplicity informací v UI, které jsou již obsaženy v jiných vrstvách aplikace. Díky této redukci lze podstatně snížit úsilí na vývoj a následnou údržbu aplikace. V dokumentaci Aspect Faces se uvádí úspora až 25% při vývoji a ještě více při následné údržbě aplikace. V kapilote 7 provádím vlastní srovnání manuálního přístupu s použitím Aspect Faces, ze kterého plyne úspora při vývoji 22% LOC[1] pro editační formulář (bohaté UI) a 18% pro formulář určený pouze ke čtení. Dále byla srovnávána následná údržba aplikace, ve které po přidání nového atributu do datového modelu bylo uspořeno dokonce 73% LOC oproti manuálnímu přístupu a 75% pro modifikaci stávajícího atributu entitní třídy.

Dále je korektní uvést srovnání aplikací po výkonové stránce. V dokumentaci Aspect Faces frameworku se uvádí, že doba odezvy aplikace bude horší maximálně o jednotky procent. V kapitole 7.2 uvádím výsledky vlastního měření odezev. Pro editační formulář implementovaný pomocí AF vychází dokonce o 2% lépe než manuálně vytvořený JSF formulář. To je dáno tím, že AF používá cache, kde jsou tagy uloženy a čtou se z `InputStream` objektu a nemusí se načítat XHTML soubory jako v případě běžné JSF aplikace (zobrazovací technologie facelet). Pro formulář určený pouze ke čtení se AF přístup jeví pouze o 4% pomalejší. V rámci prováděných testů jsem dále vytvořil vlastní AF šablony, které slučují tagy a widgety do jednoho souboru. Odpadají tím dvě fáze, kde se tag dekoruje dalšími XHTML soubory. Sloučením tagů a widgetů lze dosáhnout výrazného zlepšení času odezvy (cca. 100ms). Toto sloučení jsem podal jako podmět pro vylepšení AF frameworku.

Jedním z hlavních cílů této práce byla také spolupráce při hledání možných vylepšení AF frameworku. Pro potřeby aplikace eForm jsem navrhl a implementoval rozšiřující knihovny, které zobecňují generování UI a umožňují automatizovaně generovat PDF a XLS formuláře z datového modelu a jeho metadat. Knihovny PDF Extension a XLS Extension rozšiřují AF

[1] Lines of Code

o další anotace pro podporu tisku. Dále byla implementovaná knihovna pro podmíněnou validaci formulářů, která umožňuje provádět nebo přeskočit validační fázi formuláře na základě vyvolané akce. Knihovna JSF on-demand validation rozšiřuje AF o tuto funkcionalitu, která nemá v současné verzi JSF 2 podporu.

Všechna implementovaná rozšíření mají otevřený zdrojový kód a jsou tak volně dostupná komunitě AF. V průběhu práce s AF jsem nalezl problém se zpracováním kolekce základních datových typů, jako je String nebo Integer. AF prování generování fragmentů UI pomocí inspekce zdrojového kódu, která je implementována pomocí Java Reflection API. Inspektor tak čte všechny getter a setter metody daného objektu, a ne pouze jeho hodnotu. V současné verzi AF (1.2.1) je tak nutné vytvářet objekty, které tyto typy zapouzdří, např. StringWrapper. Tento návrh na vylepšení byl opět podán tvůrcům frameworku. Výsledky své práce jsem prezentoval v rámci sezení *Towards a Smart, Automated User Interface* na vývojářské konferenci DevConf 2013[2], kterou pořádá společnost RedHat[3] v Brně.

[2] http://www.devconf.cz
[3] http://cz.redhat.com

Literatura

[1] *Grid and Cloud Computing.* Springer Berlin Heidelberg, 2010. ISBN 9783642052149.

[2] CERNY, T. – SONG, E. A profile approach to using UML models for rich form generation. In *Information Science and Applications (ICISA), 2010 International Conference on.* IEEE, 2010.

[3] CERNY, T. – SONG, E. UML-based enhanced rich form generation. In *Proceedings of the 2011 ACM Symposium on Research in Applied Computation.* ACM, 2011.

[4] CERNY, T. – CHALUPA, V. – DONAHOO, M. J. Towards Smart User Interface Design. In *Information Science and Applications (ICISA), 2012 International Conference on*, s. 1–6. IEEE, 2012.

[5] CERNY, T. et al. Machine-driven code inspection to reduce restated information. *Lecture Notes in Information Technology.* 2012.

[6] ED BURNS, C. S. *JavaServer Faces 2.0, The Complete Reference.* 1. Columbus, OH 43272 : Mc Graw Hill, 1th edition, 2009.

[7] ELRAD, T. – FILMAN, R. E. – BADER, A. Aspect-oriented programming: Introduction. *Communications of the ACM.* 2001, 44, 10.

[8] SYSTEMS, C. – ENGINEERING, S. Unified CodeCount (UCC). `http://sunset.usc.edu/research/CODECOUNT/`, stav z 15. 4. 2013.

[9] GONCALVES, A. *Beginning Java^{TM} EE 6 Platform with GlassFish^{TM} 3: From Novice to Professional.* Novice to Pro Series. Apress, 2009. ISBN 9781430219545.

[10] HEFFELFINGER, D. *JasperReports for Java Developers.* From technologies to solutions. Packt Publishing, Limited, 2006. ISBN 9781847190734.

[11] KEITH, M. – SCHINCARIOL, M. *Pro JPA 2: Mastering the Java^{TM} Persistence API.* Apresspod Series. Apress, 2009. ISBN 9781430219569.

[12] KRASNER, G. E. – POPE, S. T. et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming.* 1988, 1, 3.

[13] MATT STEPHENS, D. R. *Testování softwaru řízené návrhem.* 1. Holandská 3, Brno 639 00 : Cppress, 1th edition, 2011.

[14] MYERS, B. – ROSSON, M. Survey on user interface programming. 1992.

[15] PANKRAZ, A. JSF 2 - Conditionally Skip Validation.
http://javalabor.blogspot.cz/2012/02/jsf-2-conditional-validation.html/,
stav z 1. 3. 2013.

[16] PIPES, J. – KRUCKENBERG, M. *Pro MySQL.* Expert's voice in Java. Apress, 2005. ISBN 9781430200482.

[17] RUBINGER, A. – BURKE, B. *Enterprise JavaBeans 3.1.* O'Reilly Media, 2010. ISBN 9781449396961.

[18] SELIC, B. The pragmatics of model-driven development. *Software, IEEE.* 2003, 20, 5.

[19] W3C. CSS Generated Content for Paged Media Module.
http://www.w3.org/TR/css3-gcpm/, stav z 3. 3. 2013.

[20] web:Apache Isis. Project Pages.
http://isis.apache.org, stav z 1. 4. 2013.

[21] web:Arqullian. Arquillian - Reference Guide.
https://docs.jboss.org/author/display/ARQ/Introduction, stav z 3. 3. 2013.

[22] web:css. CSS Print Profile.
http://www.w3.org/TR/css-print, stav z 3. 3. 2013.

[23] web:DAO. Core J2EE Patterns - Data Access Object.
http://www.oracle.com/technetwork/java/dataaccessobject-138824.html//,
stav z 3. 3. 2013.

[24] web:glasfish-embedded. Oracle GlassFish Server Embedded Server Guide.
http://docs.oracle.com/cd/E26576_01/doc.312/e24932/embedded-server-guide.htm,
stav z 3. 3. 2013.

[25] web:infobp. K336 Info — pokyny pro psaní bakalářských prací.
https://info336.felk.cvut.cz/clanek.php?id=504, stav ze 4. 5. 2009.

[26] web:infodp. K336 Info — pokyny pro psaní diplomových prací.
https://info336.felk.cvut.cz/clanek.php?id=400, stav ze 4. 5. 2009.

[27] web:java, pdf. iText - project page.
http://itextpdf.com/, stav z 8. 3. 2013.

[28] web:java, pdf, transformation. Welcome to Flying Saucer.
https://code.google.com/p/flying-saucer/, stav z 8. 3. 2013.

[29] web:JMeter, performance analysis. The Apache JMeter.
http://jmeter.apache.org/, stav z 14. 3. 2013.

[30] web:MDA. MDA - The Architecture Of Choice For A Changing World.
http://www.omg.org/mda/, stav z 14. 3. 2013.

[31] web:MetaWidget. Project Pages.
`http://metawidget.org/`, stav z 1. 4. 2013.

[32] web:OmniFaces. What is OmniFaces?
`https://code.google.com/p/omnifaces//`, stav z 1. 3. 2013.

[33] web:xhtml java parser. JSoup: Java HTML Parser.
`http://jsoup.org/`, stav z 11. 3. 2013.

[34] web:xls java. Java Excel API - A Java API to read, write, and modify Excel spreadsheets.
`http://jexcelapi.sourceforge.net/`, stav z 14. 3. 2013.

[35] web:xlsx. XML SpreadsheetML File Format.
`http://msdn.microsoft.com/en-us/library/dd922181(v=office.12).aspx/`, stav z 11. 3. 2013.

[36] web:ZFO struktura. Specifikace formátu 602XML formulářů.
`http://www.602.cz/datainc/602xml/technical/602XML_form_popis.pdf`, stav z 1. 5. 2010.

Příloha A

Seznam použitých zkratek

AF Aspect Faces

AGPL Affero General Public License

AOP Aspect Oriented Programming

API Application Programming Interface

AS Application server

CSS Cascading Style Sheets

DAO Data Access Object

DOM Document Object Model

EJB Enterprise Java Beans

GPL General Public License

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

JDBC Java Database Connectivity

JPA Java Persistence API

JSF Java Server Faces

JSP Java Server Pages

JXL JExcelApi

J2EE Java 2 Platform Enterprise Edition

LGPL Lesser General Public License

LLOC Logical Lines of Code

LOC Lines of Code

MDA Model Driven Architecture

MDD Model Driven Development

OMG Object Management Group

OOP Object Oriented Programming

ORM Object Relational Mapping

PaaS Platform as a Service

PLOC Physical Lines of Code

SaaS Software as a Service

SLOC Source Lines Of Code

SOAP Simple Object Access Protocol

UC Use Case

UCC Unified Code Count

UI User Interface

UML Unified Modeling Language

XHTML extensible hypertext markup language

XML Extensible Markup Language

Příloha B

Ukázka vyexportovaných formulářů

V této příloze naleznete ukázku výsledných formulářů automaticky vygenerovaných z datového modelu a jeho metadat. Nejprve je nutné definovat jednu facelet šablonu, ze které bude automatizovaně vytvořen HTML, PDF a XLS formulář.

```
<ui:define name="content" >
  <div id="header-center">
    <h:outputText value="III - 2 PŘEDMĚT NÁVRHU na zápis SPOLEČNOSTI
                         S RUČENÍM OMEZENÝM"/>
  </div>

  <h2>#{caption}</h2>
  <h:panelGroup layout="block" class="form-block">
    <h:outputText value="Obchodní firma" class="block-caption" />
    <util:ui instance="#{lcf.form.name}" prefix="name" edit="false"
                layout="printable/simple-form-item-layout.xhtml" />
  </h:panelGroup>

  <h:panelGroup layout="block" class="form-block">
    <h:outputText value="Adresa sídla" class="block-caption" />
    <util:ui instance="#{lcf.form.address}" prefix="address1" edit="false"
                        layout="printable/simple-form-item-layout.xhtml" />
  </h:panelGroup>

  ...
</ui:define>
```

Nyní již následují ukázky výsledných formulářů ve všech podporovaný formátech.

Předmětem návrhu na zápis SPOLEČNOSTI S RUČENÍM OMEZENÝM jsou následující skutečnosti:

Obchodní firma		
	Zapsat ke dni:	
Název společnosti:		
Adresa sídla		
	Zapsat ke dni:	
Město:		
Městská část:		
Ulice:		
Č.p.:		
Č.o.:		
Č.e.:		
PSČ:		
Stát:		

Obrázek B.1: HTML náhled na formulář.

III – 2 PŘEDMĚT NÁVRHU na zápis SPOLEČNOSTI S RUČENÍM OMEZENÝM

Předmětem návrhu na zápis SPOLEČNOSTI S RUČENÍM OMEZENÝM jsou následující skutečnosti:

Obchodní firma		
	Zapsat ke dni:	
Název společnosti:		
Adresa sídla		
	Zapsat ke dni:	
Město:		
Městská část:		
Ulice:		
Č.p.:		
Č.o.:		
Č.e.:		
PSČ:		
Stát:		

Obrázek B.2: Náhled na PDF formulář

	A	B	C	D	E	F	G
1	**Obchodní firma**						
2	Zapsat ke dni:						
3	Název společnosti:						
4	**Adresa sídla**						
5	Město:						
6	Městská část:						
7	Ulice:						
8	Č.p.:						
9	Č.o.:						
10	Č.e.:						
11	PSČ:						
12	Stát:						

Obrázek B.3: Náhled na XLS formulář.

Příloha C

Instalační příručka

Aplikace eForm je serverová aplikace, která je postavena na specifikaci Java EE6. Je nabízena zdarma v distribučním modelu Software as a Service (SaaS). Tento distribuční model přesně vystihuje potřeby aplikace eForm. Kvůli vytvoření několika formulářů, si uživatel nemusí instalovat žádný software, ale využije eForm jako službu na internetu. Aplikace je dostupná na veřejné adrese https://eform-cvut.rhcloud.com/.

Aplikace eForm je nasazena v cloud prostředí OpenShift[1] od společnosti RedHat[2]. OpenShif jsem zvolil z důvodu, že nabízí JBoss AS 7.1 v distribučním modelu Plarform as a Service (PaaS). Jelikož se jedná o službu zdarma, tak není garantována její dostupnost a v době velkého vytížení serverů bude pozastavena. Pro lokální nasazení aplikace na vlastní aplikační server se musí nejdříve projekt sestavit. Jelikož se jedná o Maven projekt, tak stačí mít nainstalovaný Maven[3] a spustit příkaz pro sestavení.

```
mvn clean install
```

Všechny závislé knihovny se automaticky stáhnou do lokálního repozitáře a projekt se sestaví do výsledného form.war souboru. Nyní již zbývá soubor nasadit na libovolný aplikační server, který splňuje specifikaci Java EE 6. V této ukázce popíšu postup pro volně dostupný aplikační server JBoss AS 7.1, který splňuje EJB 3.1 specifikaci.

Jako perzistentní vrstvu volím MySQL relační databázi. MySQL je velice populární opensource relační databáze. MySQl není tak robustním systémem jako např. Oracle databáze, proto je poměrně rychlá a méně náročná na hardware. Další velkou výhodou je podpora pro celou škálu programovacích jazyků, jako jsou např. Java, PHP, Python, C++ a .NET. Pro platformu Java existuje JDBC ovladač JConnector[4].

Na aplikačním serveru je nejdříve nutné nastavit datový zdroj aplikace. Toto nastavení se provede pomocí přidání následující konfigurace do souboru standalone.xml resp. domain.xml a vyplní se server, port, jméno databáze, uživatelské jméno a heslo k databázovému serveru MySQL. Dále na serveru musí být JDBC ovladač JConnector.

[1] https://openshift.redhat.com/
[2] https://www.redhat.com
[3] http://maven.apache.org/
[4] http://dev.mysql.com/downloads/connector/j/

```
<subsystem xmlns="urn:jboss:domain:datasources:1.0">
  <datasources>
    <datasource jndi-name="java:jboss/MySqlDSJNDI"
                pool-name="MySqlDS"
                enabled="true"
                use-ccm="false"
                jta="true">

    <connection-url>jdbc:mysql://server:port/DB_name</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <driver>mysql-connector-java-5.1.22-bin.jar</driver>
    <security>
       <user-name>username</user-name>
       <password>password</password>
    </security>
    <validation>
      <validate-on-match>false</validate-on-match>
      <background-validation>false</background-validation>
    </validation>
    <statement>
      <share-prepared-statements>false</share-prepared-statements>
    </statement>
  </datasource>
</subsystem>
```

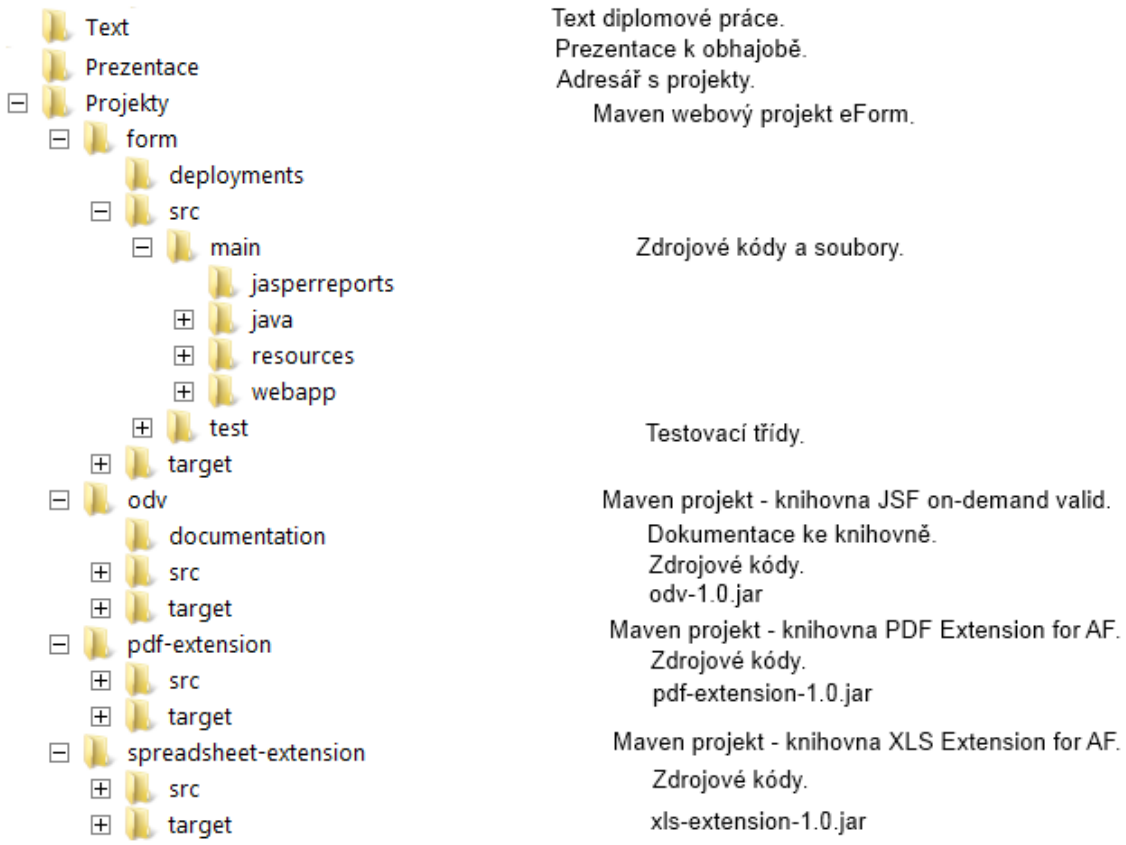
Nyní již lze aplikaci nasadit na aplikační server. Nejdříve server spustíme a následně provedeme další Maven příkaz, který provede nasazení form.war na JBoss As 7.1 server.

```
mvn jboss-as:deploy
```

Více o modelu nasazení naleznete v kapitole 4.4.

Příloha D

Obsah přiloženého CD



Obrázek D.1: Obsah přiloženého CD