Masarykova univerzita
Fakulta informatiky



Aplikace pro správu kvalifikačních prací průmyslového partnera Red Hat

Bakalářská práce

Jakub Čecháček

Brno, podzim 2013

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Jakub Čecháček

Vedoucí práce: Mgr. Jiří Kolář

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Jiřímu Pechancovi a Mgr. Marku Grácovi za poskytnuté konzultace a zpětnou vazbu. Dále bych rád poděkoval zbylým členům vývojového týmu, Pavlu a Václavovi Dedíkovým.

Shrnutí

Práce se v první části zabývá architekturou a vývojem aplikací s využitím webového rámce Grails. Část druhá se zaměřuje na vývoj informačního systému pro správu závěrečných prací, popis využitých technologií a vybrané implementační detaily.

Klíčová slova

informační systém, správa diplomových prací, správa souborů, Grails, MongoDB, MVC

Obsah

1 **Úvod** . . . 3
2 **Informační systém pro správu diplomových prací** . . . 4
2.1 *Diplomová práce* . . . 4
2.2 *Informační systémy* . . . 4
2.3 *Návrh systému pro správu diplomových prací* . . . 5
2.4 *Použité technologie* . . . 5
2.4.1 Groovy . . . 5
2.4.2 Aplikační rámec Grails . . . 6
2.4.3 MongoDB . . . 6
2.4.4 Technologie prezentační vrstvy . . . 7
3 **Groovy Grails** . . . 8
3.1 *Technologie v pozadí* . . . 8
3.1.1 Aplikační rámec Spring . . . 8
3.1.2 Groovy . . . 8
3.1.3 Hibernate . . . 9
3.1.4 Groovy Server Pages a SiteMesh . . . 9
3.2 *Convention over configuration* . . . 10
3.3 *Ekosystém zásuvných modulů* . . . 10
3.3.1 Instalace zásuvných modulů . . . 11
3.4 *Podobné platformy* . . . 11
3.4.1 Ruby on Rails . . . 11
3.4.2 Django . . . 12
4 **Architektura aplikací na platformě Grails** . . . 13
4.1 *Adresářová struktura* . . . 14
4.2 *Aplikační model* . . . 15
4.2.1 Doménové třídy . . . 15
4.2.2 Mapování doménových tříd do databáze . . . 15
4.2.2.1 **Persistenční logika** . . . 16
4.2.3 Servisní třídy . . . 16
4.3 *Prezentační vrstva* . . . 17
4.3.1 Knihovny značek . . . 17
4.4 *Komponenta controller* . . . 18
5 **Práce se soubory v systému pro správu diplomových prací** . . . 20
5.1 *Použité technologie* . . . 20
5.1.1 Fine Uploader . . . 20
5.1.2 Mongo GridFS . . . 21
5.1.2.1 **Kolekce `files`** . . . 21

5.1.2.2 **Kolekce `chunks`** . . . 22
5.1.3 Dostupné moduly pro práci s Mongo GridFS . . . 22
6 **Implementace správy souborů** . . . 24
6.1 *Archivace souborů* . . . 24
6.1.1 Servisní třída MongoFile . . . 24
6.1.1.1 **Uložení souboru do databáze** . . . 25
6.1.1.2 **Získání souboru z databáze** . . . 26
6.1.2 Zasílání souborů klientovi . . . 27
6.1.2.1 **Knihovna značek pro práci s GridFile** . . . 28
6.2 *Nahrávání souborů* . . . 28
6.2.1 Přijetí a zpracování nahrávaných dat . . . 28
6.2.1.1 **Příjem souborů** . . . 29
6.2.1.2 **Knihovna značek** . . . 30
7 **Nasazení systému a budoucí vývoj** . . . 31
7.1 *Budoucí vývoj* . . . 32
8 **Závěr** . . . 34
A **Doplňující informace** . . . 36
A.1 *Rozšíření pro práci s Mongo GridFS* . . . 36
B **Diagramy** . . . 37
B.1 *Domain Model* . . . 37
C **Odkazy** . . . 38
C.1 *Zdrojový kód projektu* . . . 38
C.2 *Adresa nasazeného systému* . . . 38

1 Úvod

Bakalářské a diplomové práce představují tradiční součást akademické sféry. Nejspíše každý z nás s nimi má zkušenost jako student, někteří jako zadavatelé či vedoucí. Ne každý si však uvědomuje složitost agendy související s jejich správou a množství subjektů, které se celého procesu účastní, obzvláště v případě, kdy je do procesu zahrnuto více univerzit.

Cílem této práce je vytvoření informačního systému pro správu diplomových a bakalářských prací pro společnost Red Hat, průmyslové partnera Fakulty informatiky Masarykovy univerzity. Systém nahradí stávající agendu správy závěrečných prací prostřednictvím firemní wiki stránky, která je s rostoucím počtem vypisovaných témat nedostačující. Hlavním přínosem nového systému oproti stávající agendě by mělo být značné zjednodušení a zpřehlednění distribuce vypisovaných témat jednotlivým univerzitám.

První část práce se věnuje vysvětlení základních pojmů a představení použitých technologií. V následující části je čtenář blíže seznámen s rámcem Grails, architekturou a vývojem aplikací na této platformě. Poslední dvě části popisují vybrané implementační detaily práce se soubory ve vyvíjeném systému a jeho nasazení.

2 Informační systém pro správu diplomových prací

2.1 Diplomová práce

Diplomová práce představuje finální studii, jejíž vypracování je požadováno pro získání vysokoškolského diplomu. Většina studentů si oblast své diplomové práce vybírá přibližně ve druhé polovině studia. Celý proces obvykle vypadá nějak takto:

1. Student si vybere pro něj zajímavé téma diplomové práce.

2. Student kontaktuje vedoucího práce a po domluvě s ním se k danému tématu přihlásí.

3. Student pracuje na své závěrečné práci a průběh konzultuje s vedoucím.

4. Jsou vypracovány dva posudky hodnotící kvalitu práce. Autorem prvního je vždy vedoucí. Druhý posudek je pak vypracován nezávislou osobou

5. Studen musí práci obhájit před komisí, která s přihlédnutím k posudkům udělí známku.

Obvykle existuje několik zdrojů, ze kterých mají studenti možnost vybírat téma své budoucí diplomové práce. První možností je vybrat si ze seznamu témat dostupných na dané univerzitě, v takovém případě se většinou jedná o oblast blízkou vedoucímu daného tématu. Další možností je vymyslet si téma vlastní a domluvit se s některým z vyučujících na jeho oficiálním vypsání. V neposlední řadě si pak studenti mohou vybrat téma nabízené nějakou třetí stranou – například společností Red Hat. V tomto případě se často do výše zmíněného procesu zapojuje druhý vedoucí, například člověk z firmy zadávající dané téma, který přebírá roli konzultanta.

2.2 Informační systémy

Informační systém představuje soubor lidí, prostředků a procesů pro sběr, přenos, uchování a zpracování dat[1]. Ačkoliv příkladem takového systémů může být i dobře vedená kartotéka, v kontextu této práce uvažuji vždy informační systémy elektronické.

2.3 Návrh systému pro správu diplomových prací

Návrh informačního systému pro správu bakalářských prací probíhal ve spolupráci se zástupci společnosti Red Hat. V průběhu prvních setkání byly určeny funkční požadavky na systém a vypracován základní doménový model celého systému, který byl následně během vývoje upravován. Mezi hlavní funkční požadavky patří:

- uživatelské účty s různou úrovní oprávnění
- správa závěrečných prací dle definovaného procesu
- možnost nahrávat přílohy
- diskuse v podobě komentářů
- nástroje pro univerzitní vedoucí

Dále byly na základě požadavků společnosti Red Hat a zkušeností vývojářů stanoveny nefunkční požadavky. Mezi nejdůležitější patří:

- Grails jako vývojová platforma.
- nasazení systému na OpenShift
- otevřený zdrojový kód

Pro bližší informace o návrhu systému a použitých procesech doporučuji přečíst bakalářskou práci Václava Dedíka[2].

2.4 Použité technologie

2.4.1 Groovy

Groovy je dynamický objektově orientovaný programovací jazyk pro platformu Java, který přináší spoustu vlastností a konstrukcí známých z jazyků jako Python, Ruby a Smalltalk. Jednou z řady výhod jazyka Groovy patří díky totožné syntaxi téměř nulová učící křivka pro stávající Java programátory[3]. Mezi nejvýznamnější charakteristiky tohoto jazyka patří[4]:

- rozumné výchozí hodnoty a nastavení
- uzávěry
- syntaxe pro podporu návrhového vzoru Stavitel

- podpora doménových jazyků

Díky přímé kompilaci do java byte kódu mohou programy napsané v Groovy využívat řadu existujících knihoven napsaných v jazyce Java a běžet všude tam, kde je k dispozici Java Runtime Environment[3].

2.4.2 Aplikační rámec Grails

Grails je aplikační rámec a platforma pokrývající všechny aspekty vývoje webových aplikací využívající programovací jazyk Groovy[5]. Díky systému zásuvných modulů lze aplikace vyvíjené na této platformě relativně snadno obohatit o nové funkce nebo upravit výchozí chování aplikačního rámce. Detailněji se celou platformou zabývají následující kapitoly.

2.4.3 MongoDB

MongoDB je dokumentově orientovaná databáze, a jedna z předních NoSQL databází na trhu, s otevřeným zdrojovým kódem[6]. V dokumentově orientovaných databázích se jednotlivé záznamy ukládají do souborů, které jsou následně seskupovány do takzvaných kolekcí. Pokud bychom se pokusili o přirovnání k relační databázi, pak kolekce představuje tabulku a dokumenty jednotlivé řádky. V případě MongoDB jsou data ukládána ve formátu BSON[1] a mohou být plně dynamická – nemusí dodržovat žádné definované schéma. NoSQL databáze nabývají v posledních letech na popularitě právě díky možnosti ukládat dynamická data a jejich vysoké škálovatelnosti. Mezi nejvýznamnější vlastnosti této databáze patří:

- převod objektových struktur do jazyka JSON je přirozenější než jejich mapování do tabulek relační databáze
- implementace návrhového vzoru Map/Reduce pro zpracování velkého množství dat
- dotazovací jazyk
- GridFS

Systém pro správu závěrečných prací využívá MongoDB primárně ke dvěma účelům. Prvním je ukládání dynamických dat (jako například konfigurace systému). Druhým případem je využití GridFS pro ukládání souborů nahrávaných studenty k jednotlivým diplomovým pracím a podobenek uživatelů.

1. Binary JavaScript Object Notation

2.4.4 Technologie prezentační vrstvy

Prezentační vrstva je implementována za použití řady moderních značkovacích a stylovacích jazyků. Dynamická části jsou skriptovány pomocí jazyka JavaScript a technologie AJAX. Mezi nejvýznamnější použité technologie, nástroje a jazyky dále patří:

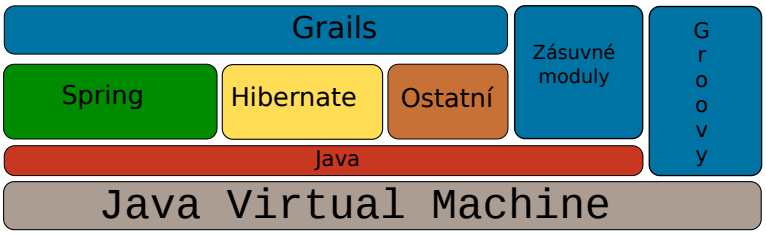
- Groovy Server Pages (GSP)
- HTML a HTML 5
- CSS a LESS
- Twitter Bootstrap

Pro bližší informace o použitých technologiích a návrhu uživatelského rozhraní doporučuji přečíst bakalářskou práci Pavla Dedíka[7].

3 Groovy Grails

3.1 Technologie v pozadí

Platforma programovacího jazyka Java je s námi již skoro 15 let. Pravděpodobně nikoho proto nepřekvapí, že během této doby vznikla spousta kvalitních knihoven řešících různé potřeby softwarového inženýrství. Této skutečnosti si byli dobře vědomi v roce 2005 i vývojáři vůbec první verze rámce Grails. Místo startu na zelené louce a vynalézání kola proto Grails staví na existujících a v praxi prověřených technologiích[8].



Obrázek 3.1: Platforma Grails

3.1.1 Aplikační rámec Spring

Aplikační rámec Spring tvoří základ platformy Grails. Jedná se o rámec poskytující prostředky pro vývoj moderních enterprise aplikací, jehož první verzi vyvinul Rod Johnson jako součást své knihy „Expert One-on-One J2EE Design“ pojednávající o návrhu a principech vývoje Java EE aplikací. Primárním cílem tohoto rámce je maximálně usnadnit vývoj enterprise aplikací. Díky tomu se dnes jedná o jednu z nejrozšířenějších platforem pro vývoj Java EE aplikací a alternativu k tradičním Enterprise Java Beans[9]. Grails využívá podstatnou část funkcionality poskytované platformou Spring – počínaje „Inversion of Control“ kontejnerem, přes integrační vlastnosti Springu, až po Aspektově orientované programování a podporu transakcí. S Grails lze navíc velmi snadno použít i libovolný z modulů rozšiřující jádro platformy Spring, jako je například Spring Security.

3.1.2 Groovy

V kontextu platformy Grails představuje Groovy nejen jazyk používaný k vývoji aplikací. Jedná se také o pomyslné „lepidlo“ spojující použité tech-

nologie a knihovny. Vlastnosti programovacího jazyka Groovy, jako například „buildery“, jej dělají ideálním prostředkem pro vývoj doménově specifických jazyků (dále jen DSL). Díky tomu vytváří Groovy uživatelsky přívětivá rozhraní zastřešující jednotlivé části platformy. Velmi časté je například využití DSL v konfiguračních souborech, jenž umožňuje uživatele odstínit od komplexity celé konfigurace a minimalizuje množství opakujícího se kódu.

```
<beans xmlns="http://www.springframework.org/schema/
    beans">
     <bean id="mongoProvider" class="com.redhat.theses.
         config.MongoProvider/>
     <bean id="configuration" class="com.redhat.theses.
         config.Configuration">
          <property name="povider" ref="mongoProvider"/>
     </bean>
</beans>
```

```
beans = {
     configuration(Configuration) {
          povider = ref("mongoProvider")
     }
}
```

Příklad 1: definice Spring bean – XML (nahoře) a Grails DSL (dole)

3.1.3 Hibernate

Hibernate je populární knihovna pro objektově-relační mapování (dále jen ORM), řešící problém rozdílné reprezentace dat v objektovém světě a relačních uložištích. Hibernate se také snaží odstínit uživatele od rozdílných dialektů jazyka SQL používaných v různých databázích. V kontextu platformy Grails představuje Hibernate součást řešení vrstvy persistence dat poskytované v základní distribuci.

3.1.4 Groovy Server Pages a SiteMesh

Groovy Server Pages (dále jen GSP) představují značkovací jazyk určený pro vývoj aplikací prezentační vrstvy. Obdobně jako syntakticky podobné Java Server Pages (dále jen JSP) umožňují GSP kombinovat statický a dynamický obsah. GSP však přináší drobná vylepšení vyplývající z dynamické podstaty jazyka Groovy. Lze tak například využít speciální textové řetězce

(`GString`) vyhodnocované přímo groovy interpretem, což minimalizuje potřebu definovat knihovny značek.

SiteMesh pak představuje flexibilní šablonovací systém založený na návrhovém vzoru „dekorátor“. Mezi nevýhody samostatného použití šablon SiteMesh patří zejména nutnost deklarovat každou šablonu, takzvaný „dekorátor“, v konfiguračním XML souboru. Grails použití tohoto šablonovacího systému usnadňuje pomocí přidaných značek pro práci se šablonami a automatickým generováním konfiguračního souboru.

3.2 Convention over configuration

Princip „Convention over configuration“ reprezentuje myšlenku poskytnutí rozumných výchozích hodnot a konfigurace za účelem maximálně usnadnit práci vývojáře. Cílem je jednak udělat vývoj aplikací zábavnějším a hlavně umožnit vývojáři aby se soustředil na samotnou funkcionalitu vyvíjeného systému místo přípravy prostředí. Jak jsem již zmínil, platforma Grails staví na řadě jiných aplikačních rámců, z nichž většina vyžaduje určitou formu konfigurace – ať už je to nastavení prostřednictví XML souboru nebo využití anotací. Dodržování stanovených zásad, jako například jména proměnných, proto značným způsobem sníží množství potřebné konfigurace. Ačkoliv princip využívání konvencí je nedílnou součásti Grails, samotný rámec však vývojáře nijak k jejich dodržování nenutí a možnosti konfigurace nejsou nijak omezeny [10].

3.3 Ekosystém zásuvných modulů

Existuje mnoho typů aplikací, určených pro různé oblasti naší společnosti, které mají mnohdy velice rozdílné požadavky. Ačkoliv se Grails snaží poskytnout komplexní řešení pro vývoj informačních systémů a webových aplikací, nelze uspokojit požadavky na vývoj všech možných projektů. Místo rozšiřování základní distribuce proto Grails nabízí možnost do vyvíjených aplikací doinstalovat zásuvné moduly, které rozšíří možnosti rámce o požadovanou funkci. Současné verze platformy Grails se snaží o modularizaci základní distribuce a vlastnosti, které nejsou nezbytně nutné, přesouvat právě do zásuvných modulů. Příkladem tohoto vývoje je plánované přesunutí podpory Hibernate do zásuvného modulu, související s rozvojem používání NoSQL databází. Základní distribuce platformy Grails aktuálně obsahuje 23 zásuvných modulů – 16 modulů tvořících jádro rámce a 7 doplňkových modulů, které lze v případě potřeby odinstalovat. Průměrně

používá každá aplikace postavená na této platformě 30 rozšíření, včetně modulů jádra. [10]

3.3.1 Instalace zásuvných modulů

V současné době lze do aplikace vyvíjené na platformě Grails nainstalovat zásuvný modul dvěma způsoby. První způsobem je použití příkazu `grails install-plugin <jmémo modulu>`. V aktuální verzi platformy je však tento způsob označen jako zastaralý a do budoucna se počítá s jeho odstraněním. Preferovaným způsobem je nyní přidat definici rozšíření do sekce `plugins` v konfiguračního souboru `BuildConfig.groovy`[11]. Ačkoliv se toto může zdát krokem zpět, důvodem k této změně je sjednocení instalace zásuvných modulů a definice ostatních závislostí projektu.

```
1  plugins {
2      build ":tomcat:$grailsVersion"
3      compile ":platform-core:1.0.RC5"
4      compile ":markdown:1.1.1"
5      compile ":mail:1.0.1"
6  }
```

Příklad 2: instalace zásuvného modulu

3.4 Podobné platformy

3.4.1 Ruby on Rails

Ruby on Rails je webovým rámcem, který začal jako první silně propagovat princip „Convention over configuration“. Jak název napovídá, jedná se o rámec pro programovací jazyk Ruby a právě Rails je příčinou dnešní popularity tohoto dynamického jazyka. Vlastnostmi se Ruby on Rails velmi podobá rámci Grails, neboť bylo jeho předlohou – v obou případech se jedná o vysoce produktivní webové rámce využívající konvence a rozumné výchozí hodnoty.

Hlavním rozdílem mezi oběma rámci je přístup k doménovým třídám a databázi. Grails využívá princip objektově-relačního mapování založeného na návrhovém vzoru „Data Mapper“, který kromě doménových tříd využívá i takzvaný „mapovací objekt“ zodpovědný za mapování dat z databáze do konkrétní instance doménové třídy. Rails naproti tomu řeší persistencí pomocí návrhového vzoru „Active Record“, který Martin Fowler definuje jako „Objekt, který obaluje řádek v databázové tabulce, zapouzdřuje

přístup k databázi, a přidává doménovou logiku“. [12] Z pohledu vývojáře představuje „Active Record“ transparentnější spojení objektu a databáze, cenou je však nižší flexibilita takového mapování.

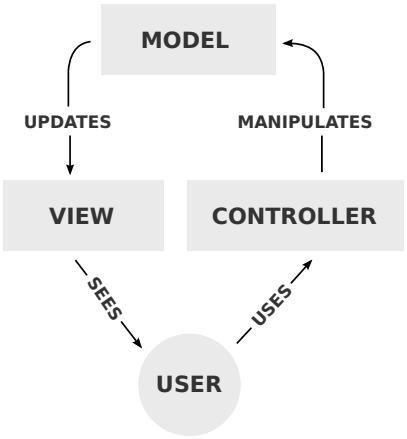
3.4.2 Django

Django představuje alternativu ze světa jazyka Python. Stejně jako v případě Grails a Rails se jedná o webový rámec s vysokou produktivitou. Co se technických možností týče, je Django srovnatelné s oběma předchozími. Hlavní odlišnosti této platformy jsou především filosofické a vycházejí přímo z návrhu samotného Pythonu. Django se tak snaží mnohem méně využívat takzvané „magie“ v podobě implicitně definovaných atributů, metod a výchozích hodnot. V důsledku tohoto chování jsou aplikace vyvíjené s použitím tohoto rámce přehlednější. Právě přehlednost je Django komunitou vyzdvihována jako velká výhoda této platformy, která je však vykoupena nutností napsat více kódu.

4 Architektura aplikací na platformě Grails

Grails, stejně jako řada jiných rámců pro vývoj aplikací obsahujících nějakou formu uživatelského rozhraní, využívá návrhového vzoru Model, View, Controller[1] (dále jen MVC). Ačkoliv jednotná definice tohoto vzoru prakticky neexistuje a mírně se liší mezi jednotlivými implementaci, obecně se vždy jedná o formu oddělení aplikační a prezentační logiky. Jak je patrné z názvu, aplikace postavené na architektuře MVC lze rozdělit do tří komponent (vrstev):

- **Model**
  Zahrnuje aplikační logiku systému včetně doménových tříd.
- **View**
  Tvoří základ prezentační vrstvy. Hlavním účelem této komponenty je zobrazení informací uživateli.
- **Controller**
  Zajišťuje komunikaci s uživatelem a prostřednictvím view zobrazuje požadované informace.



Obrázek 4.1: Architektura MVC

Poprvé tento návrhový vzor popsal v roce 1979 norský profesor Trygve Reenskaug. O rok později vznikla jako součást programovacího jazyka Smalltalk-80 první implementace této architektury. Specifické pro tuto implementaci bylo spojení komponent view a controller. Tento typ MVC se i

1. V průběhu práce se uvádí anglická terminologie, běžně rozšířená v oboru.

dnes používá u rámců pro GUI, kde ovládací prvky nejen zobrazují informace, ale také přijímají uživatelský vstup. Rozdělení všech tří komponent je pak běžnou praxí u rámců pro vývoj webových aplikací, jakým je například Grails.

4.1 Adresářová struktura

Stejně jako v aplikační rámec Spring umožňuje i Grails definovat jednotlivé komponenty aplikace více způsoby. Dostupná je tedy možnost přímé registrace komponenty v konfiguračním souboru nebo anotace jednotlivých tříd, avšak tyto postupy nejsou příliš používané. V předchozí kapitole byl popsán princip „Convention over Configuration“ a jedním ze způsobu uplatnění tohoto přístupu je dodržování stanovené adresářové struktury. Všechny aplikace na této platformě proto využívají jednotnou adresářovou strukturu s adresáři vyhrazenými pro příslušné komponenty. Následující výčet uvádí vybrané adresáře.[11]

- **grails-app/conf**
  Adresář `conf` je určen pro veškeré konfigurační soubory aplikace.

- **grails-app/controllers**
  Adresář `controllers` je určen pro controllery.

- **grails-app/domain**
  Adresář `domain` je určen pro doménové třídy

- **grails-app/i18n**
  Adresář `i18n` je určen pro soubory s texty jednotlivých jazykových mutací aplikace.

- **grails-app/services**
  Adresář `services` je určen pro servisní třídy

- **grails-app/taglib**
  Adresář `taglib` je určen pro knihovny značek.

- **grails-app/views**
  Adresář `views` je určen pro pohledy prezentační vrstvy.

4.2 Aplikační model

Aplikační model představuje jednu ze tří komponent MVC. V kontextu platformy Grails se jedná o souhrnné označení doménových a servisních tříd.

4.2.1 Doménové třídy

Doménové třídy představují v rámci aplikace reprezentaci doménového modelu. Doménový model popisuje jednotlivé entity vyvíjeného systému a vztahy mezi nimi. Novou doménou třídu lze založit buď to manuálním vytvoření `*.groovy` souboru v adresáři `grails-app/domain` nebo pomocí grails příkazu `create-domain-class` z příkazové řádky. V případě první možnosti je potřeba ručně vytvořit také případnou hierarchii balíčků. Systém pro správu diplomových prací obsahuje celkem 14 doménových tříd, za „základní“ lze označit tyto entity:

- **User**: uživatel registrovaný v systému.
- **University**: univerzita evidovaná v systému.
- **Topic**: téma závěrečné práce.
- **Thesis**: konkrétní práce konkrétního studenta.
- **Application**: přihláška studenta k tématu závěrečné práce.

Detailní popis celého doménového modelu je obsažen v práci Václava Dedíka[2], v příloze lze najít UML diagram modelu[2].

4.2.2 Mapování doménových tříd do databáze

Platforma Grails využívá pro persistenci doménových tříd do databáze nástroj GORM (Grails Object Relational Mapping). Označení „objektově-relační mapování“ v názvu je lehce matoucí, neboť dnes existují i implementace GORM API pro řadu nerelačních databází. Systém pro správu diplomových prací využívá rozhraní GORM pro práci s databázemi MongoDB a PostgreSQL.

Ačkoliv všechny implementace rozhraní GORM poskytují stejné API, liší se v rozsahu a poskytovaných funkcích (v případě NoSQL databází například nemá smysl konfigurace tabulek). V rámci této kapitoly proto uvažuji výchozí implementaci určenou pro relační databáze, kam je mapována

i převážná většina doménových tříd ve vyvinutém systému. V souladu s filosofií platformy Grails o využívání existujících knihoven se ani v případě GORM nejedná o zcela nové řešení a základ výchozí implementace je tvořen aplikačním rámcem Hibernate.

4.2.2.1 **Persistenční logika**

Jedním z principů popisovaným návrhovým vzorem „Data Mapper" je oddělení persistenční logiky a doménové třídy – tento návrh přispívá ke snížení závislosti na konkrétním řešení persistence. Ve srovnání s „Active Record", kde je persistenční logika přímo součásti doménových tříd, je však použití tohoto řešení méně přehledné a pohodlné. GORM, jakožto řešení využívající Hibernate je samozřejmě také založen na vzoru „Data Mapper", avšak díky dynamické podstatě programovacího jazyka Groovy umožňuje využit přístup podobný „Active Record". Persistenční CRUD operace jsou na jednotlivých entitách zpřístupněny pomocí „magických" metod, což jsou metody přidané objektu mimo definici třídy pomocí metaprogramování[2].

```
1       // ziskani tematu s id=1 z databaze
2       def topic = Topic.get(1)
3       // vyhledani tematu s nazvem foo
4       topic = Topic.findByTitle('foo')
5
6       // editace tematu
7       topic.title = "bar"
8       topic.save()
9       // smazani tematu
10      topic.remove()
11
12      // vyhledani vice zaznamu
13      def topics = Topic.findAllByTitleLike('buzz')
```

Příklad 3: Příklad CRUD operací

4.2.3 Servisní třídy

Účelem servisních tříd je zapouzdření komplexní aplikační logiky – tedy takové logiky, která jde nad rámec základních CRUD operací nebo pracuje s více zdroji zároveň. Při dodržení stanovených konvencí přináší servisní třídy několik funkcí, které běžné třídy neposkytují.

2. Princip úpravy kódu programu za běhu

- **Vkládání závislostí**
  V rámci servisních tříd funguje na základě jména třídy a jména atributu automatické vkládání závislosti.

- **Transakce**
  Jednotlivé metody probíhají vždy v rámci aktuální transakce. Pokud žádná při volání metody neexistuje, zahájí se transakce nová.

Servisní třídy jsou v rámci kódu vyvinutého systému rozděleny do několika balíčků:

- **com.redhat.theses**
  Práce s doménovými třídami.

- **com.redhat.theses.auth**
  Servisní třídy zajišťující autentizaci

- **com.redhat.theses.config**
  Servisní třídy obsahující logiku pro práci s konfigurací.

- **com.redhat.theses.listeners**
  Správa asynchronních událostí

- **com.redhat.grails**
  Logika práce se soubory.

4.3 Prezentační vrstva

Základem prezentační vrstvy jsou takzvaná views. Jedná se v podstatě o šablony, jejichž jediným účelem po naplnění daty je prezentace těchto data uživateli. Ačkoliv lze tuto komponentu v rámci Grails implementovat pomocí řady prezentačních rámců (Velocity, Freemarker, Thymeleaf), doporučuji použít výchozí řešení založeném na GSP a šablonách SiteMesh – důvodem je hluboká integrace s ostatními částmi platformy.

4.3.1 Knihovny značek

Jedním ze základních principů MVC je oddělení prezentační vrstvy a modelu. V případě implementací s oddělenou komponentou controller to znamená i omezení přímé komunikace mezi modelem a view – ideálně by měla být všechna potřebná data předána view z controlleru. V praxi je však

striktní oddělení view a modelu velmi obtížné, neboť v moderních aplikacích jsou jednotlivá view často dynamická. Důsledkem tak může být přenášení nepotřebných dat nebo zanášení aplikační logiky do implementace daného view. Volání metod obsahujících aplikační logiku přímo z view však také není žádoucí, neboť na prezentační vrstvě často pracují programátoři zaměření na prezentační vrstvu, bez znalosti aplikačního modelu.

Grails řeší tento problém možností definovat doplňující knihovny značek (anglicky „tag libraries“), které umožní používat požadovanou logiku prostřednictvím GSP značek. Novou knihovnu lze vytvořit manuálním založením třídy v adresáři `grails-app/taglib` nebo pomocí příkazu grails `create-tag-lib`

V rámci vyvíjeného systému vzniklo těchto knihoven několik, následující seznam uvádí některé z nich (jednotlivé položky seznamu jsou ve tvaru „tříd – jmenný prostor“)

- **Ajax4GSPTagLib – a4g**
  Knihovna značek pro práci s technologií AJAX.

- **RichGSPTagLib – richg**
  Knihovna rozšiřující standardní sadu GSP značek.

- **NotificationTagLib – notification**
  Značky pro práci se systémovými notifikacemi.

- **GridFileTagLib – file**
  Knihovna pro přímý přístup k souborům uloženým v databázi MongoDB.

- **UploaderTagLib – u**
  Knihovna pro nahrávání souborů

4.4 Komponenta controller

Jak jsem již zmínil, controller slouží k propojení view a příslušného modelu. Tato Komponenta však zajišťuje také komunikaci uživatele s modelem. V případě webových rámců je tato funkce zajištěna pomocí mapování metod controlleru, nazývaných akce, na jednotlivé URL aplikace. Obdobně jako servisní třídy, controllery nabízí možnost automatického vkládání závislostí při dodržení konvencí. Příklad níže uvádí akci dostupnou na URL `/topic/list` pomocí metody `GET` protokolu HTTP.

```
1  class TopicController {
2      // pristup pouze pres GET
3      static allowedMethods =  [list: "GET"]
4
5      def list() {
6          ...
7          // predani dat prislusnemu view
8          [topicInstanceList: topics]
9      }
10 })
```

Příklad 4: Definice controlleru

Systém pro správu diplomových prací obsahuje řadu controllerů rozdělených do několika balíčků:

- **com.redhat.theses**
  Obsluha požadavků pracujících se základními doménovými třídami.
- **com.redhat.theses.auth**
  Obsluha požadavků souvisejících s uživateli a autentizací – například přihlášení.
- **com.redhat.theses.ajax**
  Obsluha požadavků zaslaných pomocí technologie AJAX.
- **com.redhat.grails**
  Obsluha požadavků souvisejících se správou souborů.

5 Práce se soubory v systému pro správu diplomových prací

Výstupem bakalářské práce bývá zpravidla zpráva v podobně textového dokumentu (nejčastěji ve formátu PDF), často doplněna o množství příloh jako implementace konkrétního řešení nebo vzorek použitých dat. Jedná se o důležitou součást hodnocení práce a například konkrétní technické řešení mnohdy bývá hlavním důvodem vypsání práce. Tyto soubory je proto nutné archivovat. V této kapitole popisuji použité technologie a technické řešení nahrávání a archivace souboru v systému pro správu závěrečných prací.

5.1 Použité technologie

5.1.1 Fine Uploader

Fine Uploader je moderní nástroj pro nahrávání souborů prostřednictvím webových aplikací. Jedná se o projekt s otevřeným zdrojovým kódem, implementovaný v jazyce JavaScript bez jakýchkoliv závislostí[13]. K dispozici je také verze využívající široce využívanou JavaScriptovou knihovnu jQuery, jež poskytuje výrazně kompaktnější syntaxi. Právě tato varianta byla použita i v systému pro správu diplomových prací.

```
<body>
    <!-- Div obalujici uploader-->
    <div id="upload_wrapper"></div>

    <!-- Vytvoreni uploaderu-->
    <script>
      $(document).ready(function () {
        $('#upload_wrapper').fineUploader({
          request: {
            endpoint: '/upload'
          }
        });
      });
    </script>
</body>
```

Příklad 5: základní použití nástroje Fine Uploader

Soubory jsou na server zaslány zakódované v těle vícedílných požadavků HTTP[1] protokolu s využitím technologie AJAX. Fine Uploader poskytuje široké možnosti konfigurace a několik rozšiřujících bodů v různých fázích procesu nahrávání souboru.

5.1.2 Mongo GridFS

Pro ukládání souborů ve vyvinutém systému byla použita databáze MongoDB. Využití databáze oproti přímému ukládání na disk přináší možnost zaznamenat spolu se souborem i metadata poskytující doplňující informace. GridFS je specifikace popisující způsob ukládání a získávání souboru z databáze MongoDB, které přesahují maximální povolenou velikost BSON dokumentu, stanovenou na 16 MB[14]. Data ukládána do hierarchie kolekcí ve tvaru „`<bucket>.<kolekce>.<dokument>`“, kde:

- `<bucket> (kyblíček)` představuje kořenovou kolekci pro daný GridFS
- `<kolekce>` jedna z kolekcí `files` nebo `chunks` (viz. následující podkapitoly)
- `<dokument>` samotný dokument

5.1.2.1 Kolekce `files`

Dokumenty v kolekci `files` reprezentují jednotlivé soubory v úložišti GridFS. Obsah těchto dokumentů tvoří jednak základní informace o souboru (jméno, typ, kontrolní součet, čas a datum uložení) a dále také souborová metadata poskytující doplňující informace.

```
1  {
2      "_id" : ObjectId("5285fc9493f47d2c77e4c2db"),
3      "chunkSize" : NumberLong(262144),
4      "length" : NumberLong(11448),
5      "md5" : "109dcd47fadda95f6b9ad112fa006ad1",
6      "filename" : "avatar.png",
7      "contentType" : "image/png",
8      "uploadDate" : ISODate("2013-11-15T10:51:00.821Z"),
9      "aliases" : null
10 }
```

Příklad 6: dokument v kolekci `files`

1. Hypertext Transfer Protocol

5.1.2.2 Kolekce `chunks`

Samotná data tvořící soubor jsou rozdělena na bloky a následně po částech ukládána v kolekci `chunks`. Díky tomuto rozdělení lze číst pouze část uložených dat, čehož lze využít při čtení velkých souborů – jako je například streamování videa.

```
1  {
2       "_id" : ObjectId("5285fc9593f47d2c77e4c2dc"),
3       "files_id" : ObjectId("5285fc9493f47d2c77e4c2db"),
4       "n" : 0,
5       "data" : <data>
6  }
```

Příklad 7: dokument v kolekci `chunks`

Na příkladu výše lze vidět, že každý dokument uložený v této kolekci obsahuje kromě standardního atributu `_id` a samotných dat také atributy s pořadovým číslem bloku a id „rodičovského“ souboru v kolekci `files`.

5.1.3 Dostupné moduly pro práci s Mongo GridFS

Repozitář zásuvných modulů platformy Grails nabízí hned několik implementací technického řešení komunikace s databází MongoDB potažmo s její nadstavbou GridFS. Pro výběr vyhovujícího řešení jsem proto stanovil následující požadavky:

- Údržba modulu ze strany autora, alespoň v podobě opravy chyb.
- Alespoň stručná dokumentace k aktuální verzi modulu.
- Nejlépe žádné změny v entitách mapovaných na relační databázi.
- Zvolený zásuvný modul by měl řešit pouze ukládání souborů a to dostatečně pružně pro budoucí rozvoj systému.

Hned první dva z požadavků však dokázaly vyřadit většinu dostupných modulů. Ukázalo se, že velká část z nich postrádá dokumentaci, případně to jsou jednorázově vyvinuté projekty bez jakékoliv údržby. Další požadavky pak spolehlivě vyřadily i zbylé kandidáty. Seznam zvažovaných možností lze najít v příloze, za zmínku stojí zejména dva z dostupných modulů. MongoDB GORM je, jak název napovídá, implementací rozhraní GORM pro databází MongoDB. Tento modul se však hodí spíše, ob-

dobně jako výchozí ORM implementace, pro persistenci kompletních doménových tříd. Mongo GORM jsem se proto rozhodl použít k persistenci objektů dynamického charakteru, jakým je například konfigurace systému, ukládaných do databáze MongoDB. Druhým modulem, který se ukázal být přínosným, je projekt Mongodb Gridfs. Ačkoliv tento modul obsahuje spoustu drobných chyb a jeho vývoj byl s největší pravděpodobností ukončen, zdrojový kód se ukázal jako výborný studijní materiál pro vývoj vlastní aplikační logiky řešící ukládání souborů do GridFS.

6 Implementace správy souborů

6.1 Archivace souborů

Jelikož žádný z dostupných modulů pro práci s Mongo GridFS nevyhovoval ani základním stanoveným požadavkům, vyvinul jsem jako součást systému pro správu bakalářských prací s využitím získaných poznatků rozhraní vlastní – pracovně nazvané MongoFile. Aktuálně se jedná o součást vyvinutého informačního systému, avšak implementace umístěná v balíku `com.redhat.grails.mongodb` počítá s budoucím osamostatnění projektu v podobně zásuvného modulu platformy Grails.

6.1.1 Servisní třída MongoFile

Hlavní část implementace je tvořena servisní třídou `GridFileService` poskytující rozhraní pro manipulaci se soubory. Při návrhu MongoFile jsem zohlednil dva typy souborů:

- Soubory vázané na doménovou třídu
- Soubory bez doménových vazeb

Pro rozlišení mezi těmito soubory využívá navržené rozhraní možnost definovat několik kořenových kyblíčku GridFS v rámci jedné databáze. Soubory jsou ukládány do kyblíčku odvozeného z názvu vázané doménové třídy, případně lze toto jméno změnit definováním statického atributu `bucketMapping` v dané doménové třídě. Soubory bez doménové vazby jsou ukládány do kyblíčku výchozího, nazvaného `fs`. Vazba na konkrétní instanci doménové třídy je tvořena pomocí následujících atributů v souborových metadatech:

- `id` slouží jako identifikátor vázané entity.
- `group` klasifikuje soubory v rámci jedné doménové třídy.

Jako identifikátor může sloužit hodnota libovolného atributu vázané entity. Ve výchozím nastavení je použita hodnota atributu `id`.

```
1
2  {
3       "metadata" : { "group" : "avatar", "id" :
           NumberLong(6) }
4  }
```

Příklad 8: metadata souboru v kolekci `user.files`

6.1.1.1 Uložení souboru do databáze

Rozhraní MongoFile poskytuje několik metod pro uložení souboru do databáze. Základem je metoda `save` obsahující hlavní logiku vkládání nového souboru. Při volání této metody je potřeba uvést následující parametry:

1. `content`
   Obsah ukládaného souboru. Zdrojem dat může být instance třídy `File`, `MultipartFile`, `InputStream` nebo pole bytů.

2. `id`
   Identifikátor vázané entity nebo `null` v případě souboru bez vazby.

3. `bucket`
   Kořenový kyblíček použitého GridFS. V případě hodnoty `null` se použije výchozí kyblíček.

4. `group`
   Zařazení souboru do skupiny.

5. `metadata`
   Doplňující informace o souboru.

6. `filename`
   Jméno ukládaného souboru. Pokud je to možné, je jako výchozí hodnota použito jméno původního souboru.

7. `delete`
   Způsob jakým postupovat v případě již existujícího souboru se stejným `mongoId` nebo kombinací atributů `id` a `group`.
   V případě hodnoty `true` budou existující soubory smazány.

8. `mongoId`
   Unikátní identifikátor dokumentu uloženého v kolekci `files`.

Z množství parametrů metody `save` je zřejmé, že její použití není příliš pohodlné. Metoda navíc umožňuje libovolnou kombinaci hodnot a uživatele nijak nenutí dodržovat stanovené konvence pro jména kyblíčků či nastavení metadat. Z tohoto důvodu obsahuje třída `GridFileService` také přetíženou variantu `save` umožňující použití jmenných parametrů v kombinaci s výchozími hodnotami.

```
1  def thesis = Thesis.get(1)
2  def file = new File('/home/jcechace/file.zip')
3
4  // ulozeni souboru bez domenove vazby
5  gridFileService.save(file, null, null, null, null, "
       standalone_file.zip", false, null)
6
7  // s pouzitim jmenych atributu
8  gridFileService.save(content: file, filename: "
       standalone_file.zip")
9
10
11 // ulozeni souboru s domenovou vazbou
12 gridFileService.save(file, 1, "thesis", "archives",
       null, "bound_file.zip", false, null)
13
14 // s pouzitim jmenych atributu
15 gridFileService.save(content: file, filename: "
       bound_file", object: thesis)
```

Příklad 9: uložení souboru do databáze

6.1.1.2 **Získání souboru z databáze**

Rozhraní MongoFile umožňuje uložené soubory z GridFS získat buď to na základě informací uložených v metadatech, nebo přímo pomocí `_id` dokumentu v kolekci `files`.

```
1  def thesis = Thesis.get(1)
2
3  // ziskani souboru z kyblicku "thesis" podle _id
       dokumentu
4  def file = gridFileService.getFileByUniversalMongoId(
       mongoId, "thesis")
5
6  // ziskani souboru z kyblicku podle metadat
7  def file = gridFileService.getFile(['id': 1, group: "
       archives"], "thesis")
8
9  // misto konkreniho kyblicku lze uvest domenovou tridu
10 def file = gridFileService.getFile(['id': 1, group: "
       archives"], Thesis.class)
```

Příklad 10: získání souboru z databáze

V případě vyhledávání souboru na základě metadat vrátí volání metody `getAllFiles` pouze jednu instanci `GridFSDBFile` reprezentující soubor s nejnižším `_id`. Pro získání všech vyhovujících souborů jsou k dispozici přetížené varianty vracející seznam objektů typu `GridFSDBFile`.

6.1.2 Zasílání souborů klientovi

Poslední, avšak neméně důležitou, schopností MongoFile je zaslání získaného souboru klientské aplikaci. Metoda `serveFile` třídy `GridFileService` nastaví potřebné hlavičky, jako například `content-type`, a zapíše požadovaný soubor do výstupního proudu HTTP odpovědi.

```
1  // ziskani souboru z databaze
2  def file = gridFileService.getFile(['id': 1, group: "
       archives"], Thesis.class)
3
4  // zapsani souboru do http odpovedi
5  gridFileService.serveFile(response, file, true)
```

Příklad 11: zaslání souboru klientovi

Na příkladu lze vidět, že volání metody `serveFile` vyžaduje tři parametry:

1. `Response`: objekt reprezentující odpovědi HTTP protokolu.
2. `File`: instanci třídy `GridFSDBFile` zasílanou klientovi.

3. `Save`: vynucení stažení souboru webovým prohlížečem.

Pravděpodobně nejčastější použití této metody bude jako součást akce v controlleru. Proto jsem v rámci implementace MongoFile vytvořil jednoduchý controller `GridFileController`, schopný při volání akce `serveFile` zaslat soubor s požadovaným identifikátorem.

6.1.2.1 Knihovna značek pro práci s GridFile

Pro usnadnění přístupu k souborům z prezentační vrstvy jsem implementoval také knihovnu značek `GridFileTagLib`. Knihovna využívá jmenný prostor `grid` a obsahuje modifikované verze značek `link` a `createLink` ze standardní knihovny Grails. První značka slouží pro vytvoření hypertextového odkazu na soubor, druhá pak pouze pro sestavení adresy využitelné například v html značce `<img>`.

6.2 Nahrávání souborů

Implementace funkcionality pro nahrávání souborů se skládá ze dvou částí. Klienta umožňujícího zaslat soubory na server, který následně interpretuje výsledek provedené operace v uživateli přívětivé podobně. Právě tuto část řeší použití nástroje Fine Uploader[13]. Implementací serverové části, zajišťující zpracování přijatých dat, a její spolupráci s klientskou části aplikace se zabývají následující podkapitoly.

6.2.1 Přijetí a zpracování nahrávaných dat

Celý proces nahrání souboru od uživatele na server lze rozdělit do 4 kroků

1. Uživatel vybere soubor.
2. Klientská aplikace zašle data na server.
3. Server příjme zaslaná data, zpracuje je a zašle zpět odpověď.
4. Uživateli je zobrazen výsledek.

Jak bylo zmíněno na začátku kapitoly, Fine Uploader se z vetší části postará o všechny body s výjimkou čísla tři. Příjem a zpracování dat se při použití architektury MVC běžně provádí vytvořením akce v příslušném controlleru. V informačních systémech je však nezřídka potřeba zpracovávat soubory určené pro různé části systému různým způsobem. Definování

akce pro každý druh souboru však často povede k množství opakujícího se kódu.

```
1  def uploadToMongo() {
2      try {
3          MultipartFile file = ((
               MultipartHttpServletRequest) request).
               getFile('qqfile')
4
5          // ulozeni souboru
6          def saved = gridFileService.save(file,object:
               springSecurityService.currentUser, group: '
               files')
7
8          // vraceni vysledku
9          return render(text: [success: true] as JSON,
               contentType: 'text/json')
10     } catch (Exception e) {
11         // nastala chyba
12         log.error("Error␣during␣file␣upload", e)
13         def errorMessage = message(code: 'uploader.
               error.upload')
14
15         return render(text: [success: false, message:
               errorMessage] as JSON, contentType: 'text/
               json')
16     }
17 }
```

Příklad 12: akce pro zpracování přijatého souboru

Na příkladu výše lze vidět, že samotné uložení souboru do databáze tvoří jen minimální část celé akce. Zbytek kódu se proto bude nejspíš opakovat ve většině podobných metod. V systému pro správu diplomových prací minimalizuji množství takového kódu pomocí návrhového vzoru Pozorovatel a reaktivního přístupu[1].

6.2.1.1 **Příjem souborů**

V rámci vyvinutého systému jsem vytvořil controller `com.redhat.grails.upload.UploadController`, jehož akce `upload` slouží jako jediný kon-

1. Různé akce vyvolávají v systému tzv. události (anglicky events). Reakce na tyto události je pak přesunuta do speciálních metod nazývaných pozorovatel (anglicky observer).

cový bod pro příjem souborů v celém systému. Místo zpracování přijatého souboru však tato akce za použití servisní třídy `UploadService` v systému vyvolá událost obsahující data potřebná ke zpracování. Konkrétní vyvolané události lze specifikovat pomocí parametru `topic` zaslaného jako součást HTTP požadavku spolu se souborem. Logika pro zpracování přijatého souboru je potom součástí metody reagující na vyvolanou událost. Systém pro správu diplomových prací používá události `avatar` a `thesis`. První slouží k archivaci profilových obrázků uživatelů, druhý k archivaci souborů asociovaných s pracemi studentů. Obsluhu těchto událostí lze najít ve třídě `com.redhat.theses.listeners.UploadListenerService`

6.2.1.2 **Knihovna značek**

Pro snadnější použití nástroje Fine Uploader a eliminaci redundantního kódu jsem vytvořil knihovnu značek `UploaderTaglib`, která slouží k zapouzdření konfigurace a integraci s platformou Grails – včetně podpory internacionalizace nástroje.

```
1  <u:uploader topic="thesis" params="${[id:
       thesisInstance.id]}">
2       <u:trigger id="triggerUpload" class="tms-btn">
3           <i class="icon-upload"></i>
4           <g:message code="uploader.text.startUpload.
                button"/>
5       </u:trigger>
6  </u:uploader>
```

Příklad 13: použití nástroje Fine Uploader pomocí definovaných značek

7 Nasazení systému a budoucí vývoj

Po ukončení vývoje základních funkcí stanovených funkčními požadavky byl systém pro správu diplomových prací nasazen do cloudového prostředí OpenShift. OpenShift je „Platform as a Service“ řešení vyvíjené společností Red Hat, které umožňuje nasazení webových aplikací bez nutnosti správy vlastních serverů. Základní stavební jednotkou v rámci prostřední OpenShift je takzvaný „gear“, reprezentující množství zdrojů (hardwarových i softwarových) dostupných konkrétní aplikaci. V současné době jsou k dispozici tři druhy gearů, v základu všechny poskytují 1GB diskového prostoru[15]:

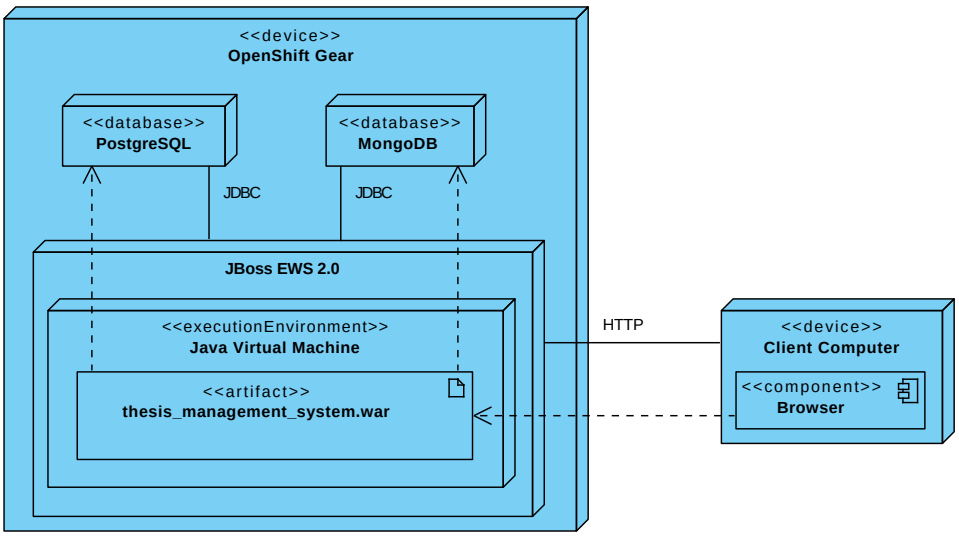
- **Small gear** (512 MB RAM): vhodný pro skriptovací jazyky
- **Medium gear** (1 GB RAM): vhodný pro JVM a provoz databází
- **Large gear** (2 GB RAM): vhodný pro vysoce náročné aplikace

V rámci jednotlivých gearů pak lze provozovat takzvané „cartridges“, což jsou nástroje připravené k nasazení v prostředí OpenShift – v podstatě se jedná o ekvivalent linuxových balíčků. K dispozici je řada připravených technologií – například aplikační servery či různé databáze.

Pro nasazení vyvinutého systému byl použit střední gear s diskovou kapacitou navýšenou na hodnotu 10 GB, aplikace běží na JBoss EWS 2.0[1] a využívá také cartridge s databázemi PostgreSQL a MongoDB. Celý proces nasazení aplikace je velmi jednoduchý a zahrnuje tyto kroky:

1. Vytvoření cartridge pro Jboss EWS a databáze.
2. Přidání vzdáleného OpenShift repozitáře do nastavení lokálního git repozitáře
3. Vytvoření souboru `.myenv` a export proměnných prostředí s nastavením emailového serveru
   ```
   export OPENSHIFT_EMAIL_HOST="smtp.example.com"
   export OPENSHIFT_EMAIL_PORT=465
   export OPENSHIFT_EMAIL_USERNAME="e@exmpl.com"
   export OPENSHIFT_EMAIL_PASSWORD="password"
   ```
4. Synchronizace vzdáleného a lokálního repozitáře příkazem `git push`

1. Jboss Enterprise Web Server



Obrázek 7.1: Diagram nasazení[2]

7.1 Budoucí vývoj

Vývoj systému pro správu diplomových prací jeho nasazením neskončil. Testovací provoz systému v semestru podzim 2013 již přinesl hodnotnou zpětnou vazbu zástupců firmy Red Hat i vedoucích z Masarykovy univerzity a Vysokého učení technického v Brně. Mezi nejčastější požadavky zástupců obou stran patří

1. seznam vypsaných témat určený k tisku
2. export vypsaných témat
3. podpora školních projektů

První položku seznamu se podařilo implementovat již v průběhu testovacího provozu. Export vypsaných témat předneslo jako návrh několik zástupců MU a VUT. Tento požadavek je pochopitelný, jelikož možnost importovat data (nebo automatizace jejich zadávání) zásadním způsobem sníží množství práce vedoucích. V následujícím semestru proto plánujeme vyvinutý systém rozšířit o export témat ve formátu XML[2] a CSV[3].

2. Extensible Markup Language
3. Comma-separated values

Kromě spolupráce v oblasti diplomových prací vyučuje společnost Red Hat na partnerských univerzitách několik kurzů[4]. Požadavek na rozšíření systému o správu těchto kurzů byl očekávaný, avšak implementace požadované funkcionality bude časově náročná. Správa kurzů se totiž od diplomových prací výrazně liší.

V rámci budoucího vývoje je v plánu také vyčlenit funkcionalitu pro práci s GridFS a nahrávání souborů do samostatných zásuvných modulu. Zatímco v případě GridFS se jedná pouze o „formality" v podobě vyčlenění kódu,v případě modulu pro nahrávání souborů je situace složitější. Během uplynulého půl roku došlo k vydání nové verze nástroje Fine Uploader, jenž rozšiřuje možnosti konfigurace a řeší problém oddělení prezentační vrstvy, se kterým se tento nástroj dlouhodobě potýkal. V důsledku těchto změn si tak vyčlenění modulu vyžádá rozsáhlý refaktoring.

Rád bych také zmínil chystanou změnu webové adresy, na které je systém dostupný. Současná adresa[5] je příliš komplikovaná a nepřehledná. Aktuálně proto připravujeme přesunutí systému pod doménu společnosti Red Hat, na adresu `https://diplomky.redhat.com`.

4. Na FI se jedná například o kurz programovacího jazyka Python nebo enterprise Java technologií
5. https://thesis-managementsystem.rhcloud.com

8 Závěr

Hlavním cílem této práce bylo implementovat nový systém správy diplomových prací pro průmyslové partnera Fakulty informatiky, společnost Red Hat. První část tohoto dokumentu poskytla základní informace o vyvinutém systému a stručně představila použité technologie. Následovalo podrobnější seznámení s aplikačním rámcem Grails, architekturou a vývojem aplikací na této platformě. Třetí část byla věnována práci se soubory ve vyvinutém systému se zaměřením na detaily vlastní implementace. V závěru jsem popsal nasazení vyvinuté aplikace do cloudu OpenShift.

Systém se podařilo implementovat a úspěšně nasadit do produkčního prostředí. Aktuálně tak probíhá první semestr testovacího provozu, v rámci kterého se zaměřujeme na připomínky uživatelů, primárně studentů a vyučujících z jednotlivých univerzit. Spolu s odstraňováním případně nalezených chyb je systém průběžně upravován podle získaných ohlasů.

Literatura

[1] J. A. O'Brien, *Introduction to Information Systems*, eleventh ed. McGraw-Hill, June 2003.

[2] V. Dedík, *Thesis management system for industrial partner Red Hat*, Brno, 2013.

[3] D. Koenig, A. Glover, P. King, G. Laforge, J. Skeet, and J. Gossling, *Groovy in Action*. Manning Publications, January 2007.

[4] Groovy homepage. [Online]. Available: http://groovy.codehaus.org

[5] Grails homepage. [Online]. Available: http://grails.org

[6] Mongodb homepage. [Online]. Available: http://www.mongodb.org/

[7] P. Dedík, *Grafické uživatelské rozhraní systému pro správu závěrečných prací*, Brno, 2013.

[8] G. Smith and P. Ledbrook, *Grails in Action*. Manning Publications, May 2009.

[9] Spring framwork homepage. [Online]. Available: http://spring.io/

[10] D. Koenig, A. Glover, P. King, G. Laforge, J. Skeet, and J. Gossling, *Programming Grails*. Manning Publications, May 2013.

[11] Grails framework. [Online]. Available: http://grails.org/doc/latest/

[12] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, November 2002.

[13] Fineuploader homepage. [Online]. Available: http://fineuploader.com/

[14] Mongodb. [Online]. Available: http://docs.mongodb.org/manual/

[15] Openshift online pricing. [Online]. Available: https://www.openshift.com/products/pricing

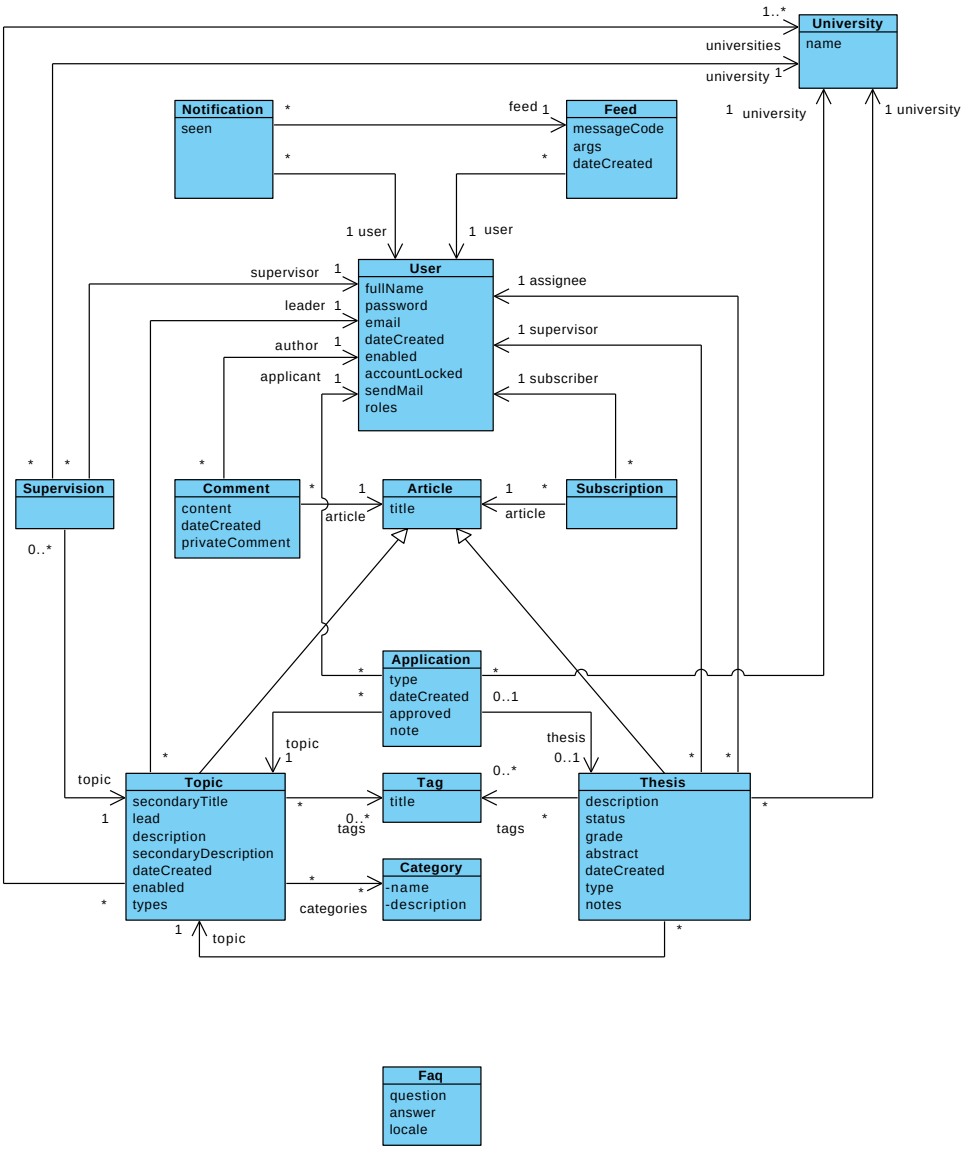
A Doplňující informace

A.1 Rozšíření pro práci s Mongo GridFS

- MongoDB Tools
- Mongodb Gridfs Plugin
- MongoDB GORM
- MongoDB GORM – Morphia
- MongoFile Plugin

B Diagramy

B.1 Domain Model



C Odkazy

C.1 Zdrojový kód projektu

Repozitář se zdrojovými kódy projektu (Thesis Management System) je hostován službou GitHub:
`https://github.com/jcechace/Thesis-management-system`

C.2 Adresa nasazeného systému

`https://thesis-managementsystem.rhcloud.com/`